Development of an Evolutionary Filter Optimization Routine (EFOR)

John C. Boyington

Mechanical and Nuclear Engineering, Kansas State University, Manhattan, KS
mjcb@ksu.edu

INTRODUCTION

Recent efforts at Kansas State University (KSU) have been made to optimize a multi-material neutron-beam filter capable of discriminating noise from gamma-rays and thermal neutrons to provide a source for testing fast neutron detectors. The goal of the project was to find a permutation of materials capable of maximizing the fast-to-total neutron (FT) ratio, maximize the neutron-to-gamma (NG) ratio, and also maximize the fast neutron flux (0.5 MeV cutoff) streaming through the filter. The initial approach solved the problem by fixing the filter length and slab discretization and then used Monte Carlo N-Particle Transport Code (MCNP) [1] to calculate the FT and NG ratios for filters of each permutation of a given set of materials. The permuatative approach required every potential solution within the problem space be calculated and this caused the problem run time to be too high for the available computational resources. To minimize the computational cost, the problem space was reduced to 3 filter lengths, 3 materials, and a discretization of 10 slabs. Although this approach found an 'optimum' filter, a desire to incorporate a larger number of materials and variable slab length, as well as to further reduce run-time led to the implementation of a formal optimization technique, a genetic algorithm (GA) to solve the problem. Described in this work is the development of EFOR, as well as an investigation of its effectiveness and ability to be parallelized.

THEORY

A GA uses an evolutionary approach to converge on an optimum solution or set of optimum solutions [2]. Initially, a randomly generated population (collection of potential solutions) of individuals (potential solutions) is created. Each individual can be represented by a chromosome (collection of genes), where the genes represent the value of each parameter unique to the individual. Using a user-defined objective function, or 'fitness' function, the fitness of each individual in the population can be calculated. Then, a selection process chooses fit individuals from the initial population to 'breed' and the resulting individual will contain genes from both of the parents. Repeating this process for several generations, while eliminating the least fit individuals from each generation will eventually cause the population to converge on an optimum solution. To avoid convergence on a local optimum, each new generation will contain a number of individuals with a random mutation in one of their genes.

DESCRIPTION OF EFOR

EFOR is a python-based, object-oriented, genetic filter optimization algorithm for use on single- or multi-core computers or computing clusters. A general layout of the code is presented in Fig. 2. The code begins with a driver file that starts the cycle routine on one core and if ran in parallel, sends each other core to wait within the slave script to receive work. The cycle script is in charge of driving the cyclic behavior of the population as it moves towards a solution using functions with in population class. It begins by initializing a population of randomly generated filters. After that, the fitness of each of those individuals are calculated. The data is then stored by increasing fitness and stored, and the the process is looped until a user-defined number of cycles, or generations, is complete. This script is also responsible for the output of run-related information, such as the total time of the calculation or the average time to run one filter.

The population object houses all of the population-level functions used in the cycle script, as well as the filter data for each population. Self-contained is a function that creates a number of random chromosomes and creates the current generation of filter objects with those. This object will send work chunks (the number of filters in that generation // number of cores - 1) to each of the slave tasks and then assign whatever filters remain to the master core (the modulus of the above operation, 100% of the work if ran in serial). Also contained within this object is the selection algorithm. After ordering the filters by increasing fitness, a cumulative distribution function is generated based on each filter's fitness. Based on random number generation (RNG) filters that have a higher fitness will more likely be selected than those with lower fitness; however, there is still a chance that any filter from the population can be chosen, as not to too quickly converge on a solution. After each sorting and selection iteration, the population information is written to a file, along with the data from the filter with the highest fitness.

The filter object contains everything specific to an individual filter. Initialized with a chromosome and using template files, it can write the MCNP files for both neutron and gamma-ray source terms. It also drives the MCNP process and can read output files to collect information on the FT and GN ratios. The fitness function script houses the fitness function that is read by the filter object to calculate it's fitness. Each filter is given a unique ID number which is contained within the MCNP file names and simplifies the process of creation of, data extraction from, and deletion of the MCNP files.

The operations object is used by the population object, and houses the functions for the splicing and mutation operations. The slicing operation employs a technique called uniform crossover, which compares two given parent chromosomes one gene at a time and grants each a 50% change they'll be passed onto the child. The mutation operation takes an individual filter, picks one random gene from its chromosome and randomly alters it.

A materials library object holds all of the material information for the problem, including compositions and density,

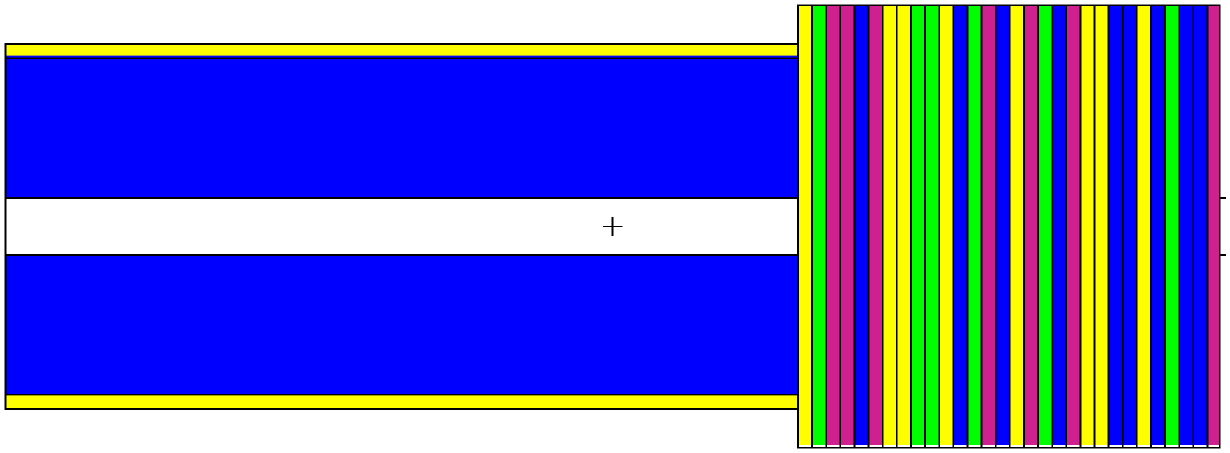


Figure 1: Cross-sectional view of the TRIGA Penetrating Beam Port Collimator and a randomly generated filter.

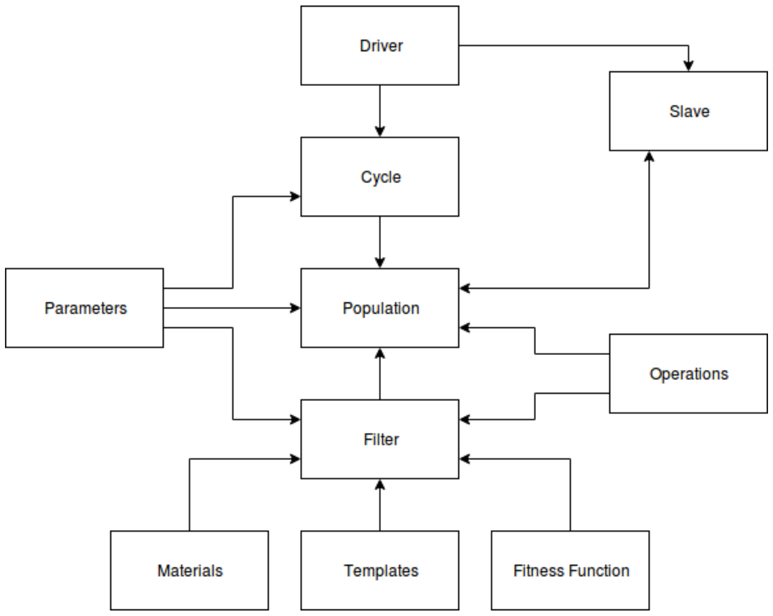


Figure 2: Layout of the EFOR algorithm

and a parameters script is where a user would define certain parameters, such as bounds on the filter length, number of individuals per generation, number of generations to be ran, etc.

EXPERIMENTS

Although the formal filter optimization is not considered in this work, a number of experiments were conducted to evaluate the potential for this algorithm to be an effective optimization tool.

The first of which was simply to see if the algorithm would move a population towards an optimum solution. Using a the fitness function:

$$fitness = 10FT + 0.5NG + N_{remaining},$$

where FT is the fast-to-total ratio, NG is the neutron-to-gamma ratio and $N_{remaining}$ is the remaining fraction of neutrons, 9 generations of 50 filters each were simulated. A weighted sum approach to calculating fitness was used and preference was given to the FT value.

Follow that, an experiment was done to test how well the algorithm scaled on multiple cores. Using the same fitness function and 3 total generations of 25 filters each, the simulation was run using 1, 2, 5, 10, 20, and 26 cores.

A third experiment studied the convergence of the algorithm. Keeping a total number of filters calculated constant, 225, two runs were done, one using an initial population of 25 with 9 generations total, and one with an initial population of 75 with 3 generations total.

Finally, an experiment was done with the same parameters as the first experiment, but the fitness function:

$$fitness = FT + 0.5NG + N_{remaining},$$

was used instead. This function now prefers the NG ratio much more and will show the effects that changing this weighting has on the algorithm.

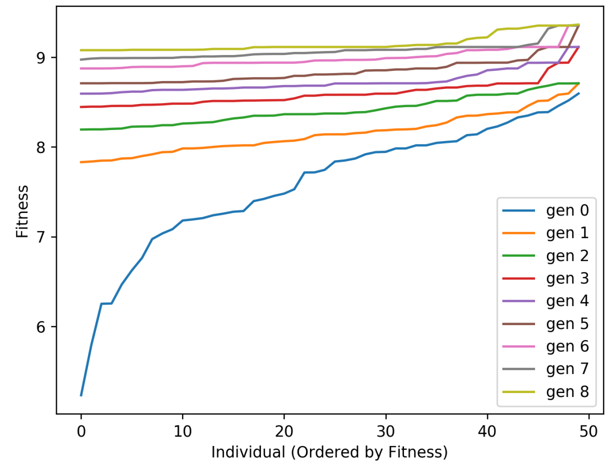
Materials used for each of these experiments were borated polyethlyene, lead, tungsten, and cadmium. Lead and tungsten were chosen for their high Z-numbers, being notoriously good at gamma-ray shielding and thus improving the NG ratio. Cadmium and boron have especially high thermal neutron absorption qualities and help maximize the FT ratio.

RESULTS AND DISCUSSION

Studying Fig. 3 shows the results of the first experiment. The fitness plot [top] clearly shows the fitness of each generation increasing, which suggests that the algorithm is working as intended. Generation 0 has a wide fitness spread due to the random nature of its initialization; however, for each following generation, the difference in fitness between the fittest and least fit individual is greatly reduced and the population becomes much more specialized. The ratio plot [bottom] demonstrates the trend of each successive generation as they try to optimize the objective function. Recall that the weighting assigned to the FT ratio was much higher than any other ratio for this experiment. The population seems to propagate towards the right side of the graph, seeking to increase its overall FT ratio.

Figure 4 contains information related to the parallel performance of the algorithm. The average time to calculate one filter is plotted as a function of number of cores [top]. The time per filter trends $\frac{1}{n}$ with increasing number of cores, which correlates to the seemingly linear speedup. At 20 cores, there is a slight drop in speedup observed, which is possibly explained by uneven work distribution. If the generation size is not divisible by the number of cores used in the run, a portion of the cores with have an extra filter to calculate, so on small scale calculations such as this one, the effect is noticeable. Because the number of filters in a generation remains constant in any simulation, this effect can propagate if the number of generations is increased, causing large inefficiencies; however, if the population size is increased, this effect is diminished.

The results from the constant total filters experiment are shown in Fig. 5. The general behavior of the two plots is similar; however, there are some notable differences between the two. The FT ratios (and as a result, the fitnesses) of the final generation of the nine-generation group are higher than those of the of the three-generation group. The points in the nine-gen plot are also much more densely packed together. The reason for this is likely due to the higher number of generations. Every time another generation is calculated, the population takes a step towards an optimum solution. The more steps, the more closely all of the individuals in that population will resemble the optimum. The trade-off is that with a lower starting population, less of the problem space is sampled and there is an increased chance of arriving at a local optimum solution, instead of a true optimum solution. The three-gen plot demonstrates this healthy diversity, being slower to converge to a solution, but having a higher likelihood that the solution it finds will be the best.



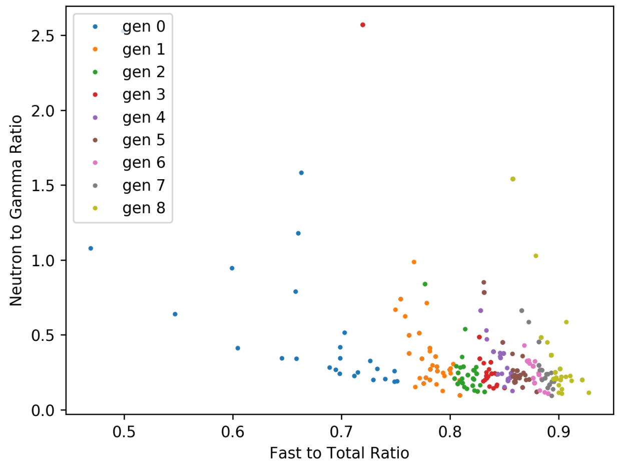


Figure 3: The fitness of each individual in each generation of the first study [top] and the FT and NG ratios of each of those individuals [bottom]. Energy integrated fluxes are included in the legends.

An alteration of the fitness function led to the results plotted in Fig. 6. Here, much more weight is given to the NG ratio and the generational movement now seems to be climbing toward an optimum with a large NG ratio instead of a large FT ratio. A stratification similar to that in Fig. 3 can be seen in this plot, too, but there is a much larger spread for the NG dominant plot's final generation. This suggest a slower convergence to a NG dominated solution, and that possibly more filter traits work to maximize the NG ratio than do the FT ratio.

FUTURE WORK

In the near future, work will be done to actually optimize a filter using EFOR, since it is shown to work. This will require the generation of an objective function whose maximum truly represents an optimum filter. The material library will be

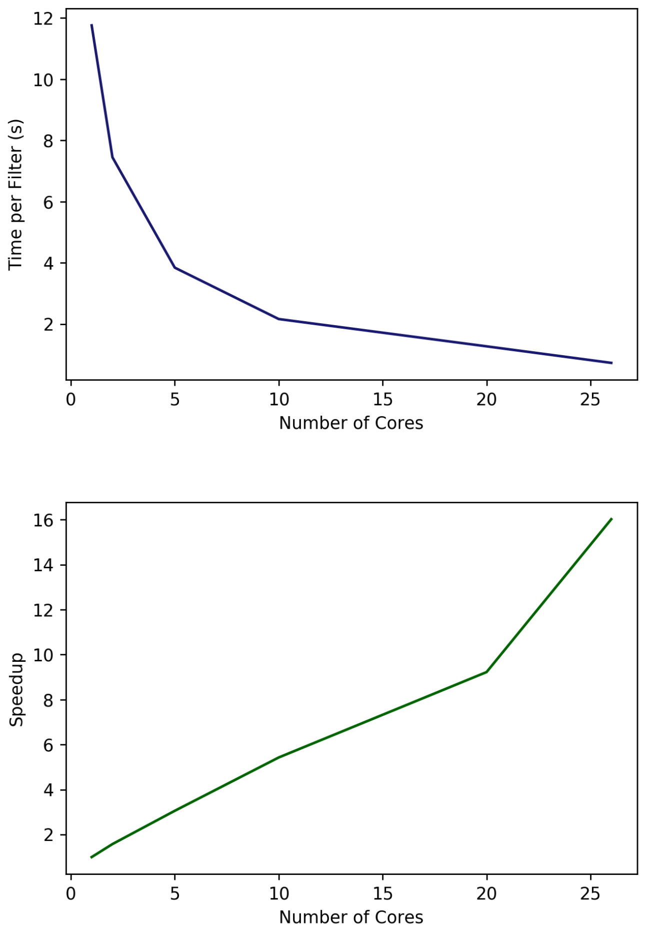


Figure 4: The average time to run one filter [top] and the parallel speedup [bottom].

greatly expanded to include many more potentially effective materials.

REFERENCES

1. J. E. SWEEZY, *MCNP-A General Monte Carlo N-Particle Transport Code*, Los Alamos National Laboratory.
2. E. K. BURKE, *Search Methodologies*, Springer (2005).

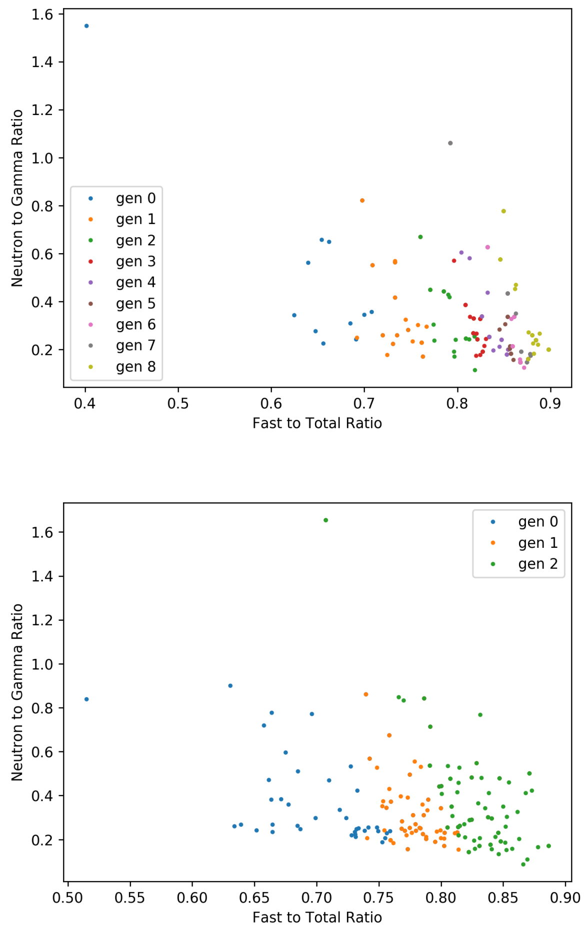


Figure 5: Ratios for population 25, generations 9 [top] and population 75, generations 3 [bottom].

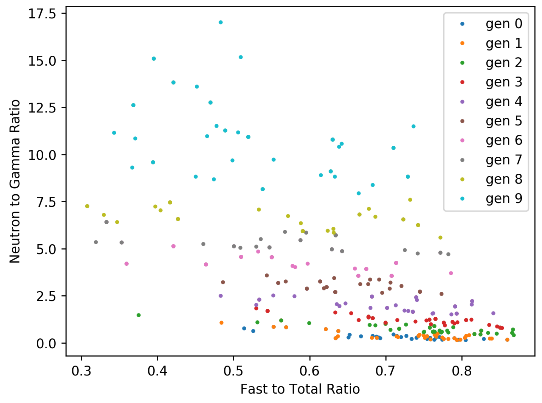


Figure 6: Cross-sectional view of the TRIGA Penetrating Beam Port Collimator and a randomly generated filter.