Novel Genetic Algorithms for Loading Pattern Optimization Using State-of-the-Art Operators and a Simple Test Case

Ella Israeli
The Unit of Nuclear Engineering,
Ben-Gurion University of the Negev,
Beer-Sheva 84105, Israel
e-mail: ellaisra@post.bgu.ac.il

Erez Gilad[1]
The Unit of Nuclear Engineering,
Ben-Gurion University of the Negev,
Beer-Sheva 84105, Israel
e-mail: gilade@bgu.ac.il

Novel genetic algorithms (GAs) are developed by using state-of-the-art selection and crossover operators, e.g., rank selection or tournament selection instead of the traditional roulette (fitness proportionate (FP)) selection operator and novel crossover and mutation operators by considering the chromosomes as permutations (which is a specific feature of the loading pattern (LP) problem). The algorithm is applied to a representative model of a modern pressurized water reactor (PWR) core and implemented using a single objective fitness function (FF), i.e., k_{eff}. The results obtained for some reference cases using this setup are excellent. They are obtained using a tournament selection operator with a linear ranking (LR) selection probability method and a new geometric crossover operator that allows for geometrical, rather than random, swaps of gene segments between the chromosomes and control over the sizes of the swapped segments. Finally, the effect of boundary conditions (BCs) on the symmetry of the obtained best solutions is studied and the validity of the "symmetric loading patterns" assumption is tested.
[DOI: 10.1115/1.4035883]

1 Introduction

The vast majority of nuclear reactors are operated in cycles, i.e., they must be periodically refueled due to fuel depletion during normal operation. This refueling outage is a complicated and expensive procedure that usually necessitates halting the reactor and opening the reactor pressure vessel. The fuel depletion is not homogeneously distributed throughout the core, and usually, the most depleted fuel assemblies (FAs) are replaced (typically 30% for power reactors) in each refueling. The loaded fresh FAs, together with the remaining depleted FAs, are rearranged to form a new core configuration (loading pattern, or LP). The new core configuration must maximize the energy production until the subsequent refueling outage (long cycle) while still satisfying all safety limitations and operational constraints. For example, the core excess reactivity should be maximized to ensure a long cycle and high fuel burn-up, while maintaining the ability to control and shut down the reactor within the required safety margins [1].

This optimization problem is characterized by a huge search space and is a multiobjective, nonlinear, nonconvex, NP-hard combinatorial problem [1,2]. In a standard pressurized water reactor (PWR) nuclear power plant (NPP) core, there are typically between 150 and 250 FAs. Of those Fas, there are many types that differ in material composition (enrichment, burn-up degrees, burnable poison configuration, etc.). For example, a core of 200 FAs of ten different types (20 FAs per type, for simplicity) has approximately $200!/20! \times 10 \approx 10^{355}$ different possible core configurations. Suppose the calculation and evaluation of one core takes only 1 s, going through the entire search space in search of the best configuration would take approximately 10^{348} yrs. The age of the universe, for comparison, is $\sim 10^{10}$ yrs.

Therefore, one must find a more economical search method. This study deals with the implementation of genetic algorithms (GAs) for solving the optimization problems of FA LPs in the reactor core.

The GA is a well-known method used for addressing the optimization problem of in-core fuel [3,4]. However, many studies dealing with this problem use fairly basic and traditional implementations of the GA, disregarding the geometrical structure of the core and imposing symmetry restrictions on the problem, e.g., Refs. [5–20]. A good example of this approach is the use of the fitness proportionate (FP) roulette wheel (RW) instead of tournaments and linear ranking (LR) for the selection, neglecting the permutationlike nature of the core LP representation vector, etc.

In this work, novel and modern GA methods are developed, implemented, and evaluated using different case studies, which account for the geometric structure of the core. The effect of boundary conditions (BCs) on the symmetry of the core configuration is also studied.

This study is of interdisciplinary nature, and it includes both an algorithmic approach rooted deep in computer sciences and an applicable approach for evaluating the algorithms performances using more realistic cases. In order to develop and test new genetic operators, the nuclear part of the problem is greatly simplified, thus reducing the complexity and noise which are *not related* to the algorithmic study. This approach enables one to better understand and characterize the newly developed algorithms. This paper summarizes the algorithmic part of the research and hence focuses on the *algorithmic* approach rather than on realistic implementation. It should be noted that the second part of this study adopts a much more realistic approach by considering a well-characterized PWR core and including the power peaking factor (PPF) as an additional optimization objective. These results will be published elsewhere.

2 Methodology

2.1 LPs and k_{eff}. The effective neutron multiplication factor, k_{eff}, is the average number of neutrons generated from a single fission event that eventually induce another fission event. The remaining neutrons are either absorbed in nonfission reactions or leave the system (leak) without being absorbed. NPPs are operated (most of the time) in a critical state that sustains the nuclear chain reaction, i.e., $k_{eff} = 1$.

In practice, due to other factors affecting the core's criticality, and especially the need for long irradiation cycles, the core design should possess higher k_{eff} values. One way of achieving high k_{eff}

[1]Corresponding author.

Manuscript received October 30, 2016; final manuscript received December 26, 2016; published online May 25, 2017. Assoc. Editor: Ilan Yaar.

values is by clustering the most highly enriched fuel together, surrounding it with the next level enrichment fuel, and so on, in a concentric circles pattern of decreasing enrichment. This spatial arrangement of the FAs increases k_{eff} since a substantial quantity of highly enriched fuel packed in close proximity causes the neutrons emitted from fission to "see" more fuel, thus increasing the likelihood of inducing more fissions—effectively increasing k_{eff}.

This theoretical pattern is the result of the strong effects the spatial arrangement of the different FAs in the reactor core have on the probabilities of fission neutrons to engage in different nuclear reactions. When such an arrangement increases the *relative* probability of a fission neutron to induce a fission, the k_{eff} of the core increases. As an example, consider a bare reactor core with void BC, i.e., a neutron that crosses the core's boundaries to the outside does not return to the core and does not contribute to the fission chain. Using our intuition as core physicists, the spatial arrangement of fuel assemblies that provides the highest k_{eff} is the one that minimizes the neutron leakage. This implies the positioning of as much fissile material as possible away from the core boundaries, i.e., in its center. In this example, human intuition works well, due to the problem's simplicity.

An example of a random LP, representative of the first generation in the evolutionary process, is shown in Fig. 1, where the different locations in the core indicate the different enrichment levels of the FAs, with central (peripheral) locations indicating higher (lower) enrichment.

2.2 Core #1: Simplified PWR Core. The nuclear reactor core considered in this work, a.k.a core #1, is a simplification of a typical advanced (Gen III+) PWR, e.g., advanced pressurized water reactor (APWR) [21], with 17×17 rectangular lattice containing 257 fuel assemblies of three different ^{235}U enrichment levels, i.e., 3.1 w/o, 2.4 w/o, and 1.6 w/o. The axial composition of an FA is assumed to be homogeneous and all FAs are assumed to be fresh. Radial BC are assumed to be black absorber, whereas the axial BC are assumed to simulate the axial reflector. The number of fuel assemblies of each type is assumed to be constant, so the optimization is performed only on the LP and not on the number of FAs of each type. A schematic view of a typical initial core layout is given in Fig. 2.

2.3 Reference Core #1. The reference case for core #1 is intended to test the algorithm performances in the case of a single objective fitness function (FF) which maximizes k_{eff}. The reference case is constructed as our educated guess for the core configuration that results in the highest possible k_{eff}, and is shown in Fig. 3. The different colors indicate the different enrichment levels of the FAs, with red (green) indicating highest (lowest) enrichment. As discussed earlier, given void BCs, the idea is to minimize neutron leakage by concentrating as much fissile material as possible in the center of the core, away from its boundaries and in proximity of more fissile matter. This LP is characterized by $k_{\text{eff}} = 1.338932$.

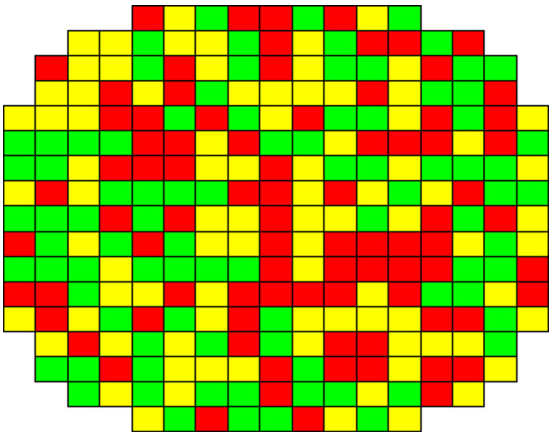

Fig. 1 A random LP representative of some first generation of an evolutionary process

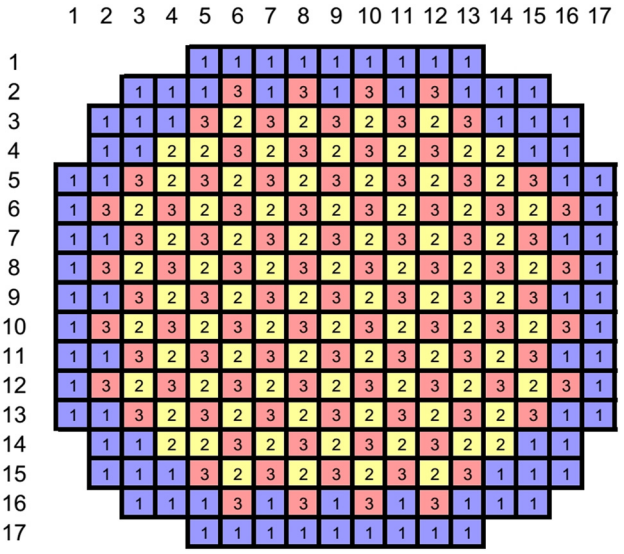

Fig. 2 A schematic layout of core #1 fuel assemblies typical initial LP. Fuel type 1/2/3 represent 3.1/2.4/1.6 w/o ^{235}U enrichment, respectively.

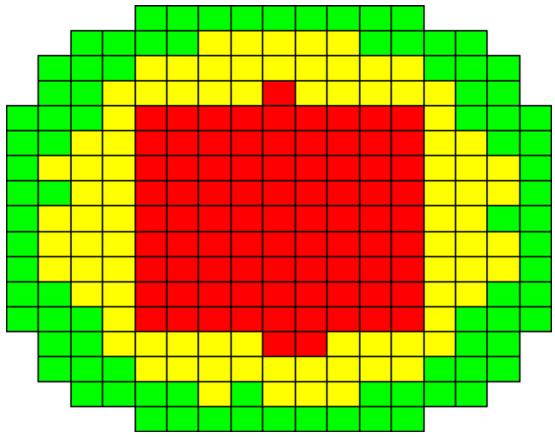

Fig. 3 A rough estimation (pre-GA) for the highest k_{eff} core LP for core #1 (Sec. 2.3). The different locations in the core indicate the different enrichment levels of the FAs, with central (peripheral) locations indicating higher (lower) enrichment.

2.4 The Core Simulator. The core simulator used is DYN3D [22], which is a three-dimensional core model, developed at Helmholtz Zentrum Dresden-Russendorf (HZDR), for dynamic and depletion calculations in light water reactor cores with quadratic or hexagonal FA geometry. The two- or multigroup neutron diffusion equation is solved by nodal expansion methods. A thermal-hydraulic model (FLOCAL) of the reactor core and a fuel rod model are implemented in DYN3D. The reactor core is modeled by parallel coolant channels which can describe one or more fuel elements. Starting from the critical state, the code allows the simulation of the neutronic and thermal-hydraulic core response to reactivity changes caused by control rod movements and/or changes of the coolant core inlet conditions. Cross section libraries generated by different lattice codes for different reactor types are linked with DYN3D. In this work, the code is used only for static neutronic (i.e., eigenvalue) calculations.

This core is a simplification of a typical APWR with 17×17 rectangular lattice containing 257 FAs of three different ^{235}U

enrichment levels, 3.1 w/o, 2.4 w/o, and 1.6 w/o. The axial composition of an FA is assumed to be homogeneous and all FAs are assumed to be fresh. Axial BCs are assumed to simulate the axial reflector whereas the radial BCs are either void or reflective. The number of FAs of each type is assumed to be constant, so the optimization is performed only on the LP and not on the number of FAs of each type. No symmetry constrains are enforced on the LPs and the thermal-hydraulic feedback calculations of DYN3D are turned off (purely neutronic calculations).

The FA level calculations for the generation of few-group cross section data sets were performed by HZDR with the commercial lattice code HELIOS 1.9 [23], which is a neutron and gamma transport and depletion code developed by Studsvik Scandpower, Kjeller, Norway. The cross section libraries employed by HELIOS are based on ENDF/B-VI evaluated data files. The full core nodal diffusion calculations employed cross sections collapsed into two energy groups with assembly discontinuity factors (ADFs) set to unity.

2.5 GAs. The GA is a search tool, adapted for optimization problems over huge search spaces, such as the one we are concerned with, that are extremely hard or impossible to search in more direct ways. The GA is based on a concept of the Darwinian evolution—survival of the fittest. The concept that underlies the basic working method of the GA consists of taking a group of solutions (a population of individuals), which covers a portion of the search space, and performing on it a process of "evolution," thus moving the group through the search space in search of optimal solutions [3,4].

3 Algorithm

The GA developed in this study is based on a standard GA with the required modifications. The essentials of the basic GA are summarized in Algorithm 1 and illustrated in Fig. 4.

Algorithm 1 basic GA

```
1: procedure GA
2:    Generation zero: gen = 0
3:    Create an initial random population pop of size N
4:    Calculate its variance popVar
5:    Calculate fitness for every individual
6:    while popVar > threshold AND gen < maxGen do
7:       Store the best individual for later reinsertion
8:       Select N/2 pairs of individuals for crossover
9:       Crossover chosen pairs to generate N offspring
10:      Randomly mutate a fraction of the population
11:      Reinsert previous generation's best individual
12:      Store new population as newPop
13:      gen = gen + 1
14:      Calculate the newPop variance popVar
15:      Calculate fitness for every individual
16:   end while
17: end procedure
```

3.1 Representation. A solution in the GA is an LP of the core, i.e., a spatial arrangement of the FAs in the core. Some solutions are better for the purposes of the optimization, and some are worse. A good solution in GA is defined by a high FF value, which in this study is based on k_{eff} values of the LP, as subsequently explained. In this study, a restriction is imposed on the allowed solution LPs, i.e., they are required to maintain the original fuel bank in the initially given LP. That is, all solutions must be permutations of one original given LP.

3.2 The Core Vector. In this study, the building blocks of the LPs, the FAs, are vertical rods. In the first stage of the algorithm, the changes from LP to LP are radial only. There is no vertical difference in between LPs with respect to the calculations in the DYN3D simulator. Hence, an LP, and subsequently the entire core, can be represented by an upper view of itself. The core's upper view roughly takes the shape of a rectangular two-dimensional matrix with rounded corners, according to the core structure and geometry. Essentially, it is an array of core cells and can be represented as such, where each of the cells contains different FA types. Hence, the simplest representation of an LP can be an array of cells each containing a different FA type.

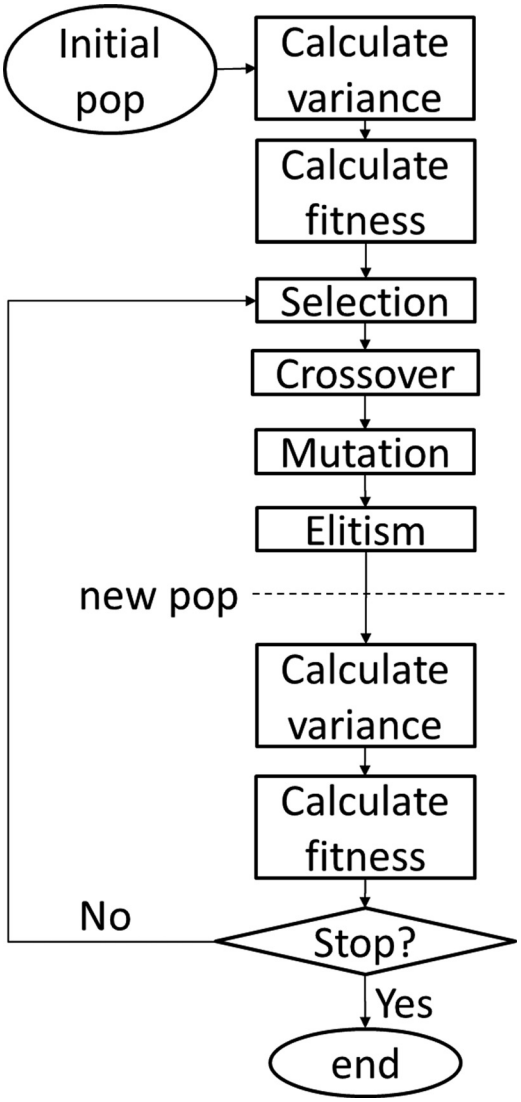

Fig. 4 Algorithm flow chart

For reasons of computational convenience, the LP is represented first by a core vector. The core vector is of length $n_I \times n_J$, where n_I and n_J are the number of rows and columns in the core matrix, respectively. The cells in the core vector correspond to the core's cells: The first vector cell represents the top left core cell, the second one to the cell on its right and so forth from left to right, top to bottom. The core vector's entries are integer numbers from zero to the number of fuel types (n). Each number represents the corresponding fuel type, with the exception of zero. Zero cells are fixed in location and are merely structural aids. A visualization of the core vector for core #1 (Sec. 2.2) is shown in Fig. 5.

3.3 The Chromosome. The solutions in the GA, i.e., the individuals of the population, are represented by chromosomes. The chromosomes can, but do not need to, resemble the solutions they represent. A chromosome only needs to supply the information needed to construct its corresponding solution. In order to represent the LPs, the core vector described in Sec. 3.2 can be used. However, since all solutions are spatial permutations of an initial

0	0	0	0	1	2	3	4	5	6	7	8	9	0	0	0	0
0	0	10	11	12	13	14	15	16	17	18	19	20	21	22	0	0
0	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	0
0	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	0
53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86
87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103
104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137
138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154
155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171
172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188
189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205
0	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	0
0	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	0
0	0	236	237	238	239	240	241	242	243	244	245	246	247	248	0	0
0	0	0	0	249	250	251	252	253	254	255	256	257	0	0	0	0

Fig. 5 The core vector data structure

core structure (hence the number of FAs of each type is constant), the core vector data structure is found to be unfit to handle this restriction.

In order to preserve the original number of FAs of each type, our chromosome is defined in a way that inherently does so. Let n_f designate the number of FAs in the core; hence, the chromosome is an n_f long vector. It is logically divided into n parts, where n is the number of fuel types in the core. Each part is as long as the number of FAs of that type. The lengths of the vector parts are constant throughout the evolution and are determined by the user via an input initial LP.

All LPs must adhere to the original fuel inventory and thus are permutations of the original LP. For this purpose, a chromosome data structure was designed that maintains the fuel inventory: The chromosome is a permutation of the indices of the core vector (without zeros). The location of a core index in the chromosome determines what fuel type it holds. The core indices in the first part of the chromosome contain the first fuel type, the ones in the second part contain fuel of type two, and so on. This way the fuel inventory remains unchanged.

3.4 Initialization. In order to begin the evolutionary process, an initial population of solutions is needed. This initial population is created randomly in order not to affect the search with unintentional bias. As mentioned in Secs. 3.2 and 3.3, the solutions are permutations of the original LP input, and the chromosomes corresponding to those are permutations of the n_f core cells. To create the initial population, random permutations of the sequence $[1\ldots l]$ are thereby created, where l is chromosome length. The population's size, which is the number of permutations generated to create the population, is a predetermined (constant throughout the evolution) variable of the algorithm.

3.5 Termination Criteria. The GA, being an iterative stochastic processes, requires termination criteria, which specify the conditions in which the search should be terminated. A natural termination point is when the search is no longer making any tangible progress toward better solutions. This happens either when the population is mostly converged to one solution, good or otherwise, or when, for whatever reason, it wanders through the search space without converging.

When the population converges, most of its individuals are very similar to one another and its variance is very low. When the population's variance is low, there is not much chance for genetically different solutions to be generated since there is no genetic diversity from which that difference might emerge. In other words, searches that reach low variance populations are futile since their stochastic journeys have no means of escaping the search space regions in which they are stuck.

The population's variance is calculated using a function that estimates how different the chromosomes in the population are from one another. It counts chromosomal differences throughout the population, i.e., for every chromosome in the population it counts the number of differences from subsequent chromosomes.

Example: chromosome 1 (c_1) differs from chromosome 2 (c_2) by 51 genes, so the temporary count is 51. c_1 differs from c_3 by 76 genes so 76 is added to the count and so forth until the last chromosome in the population. Then the same process is carried out for c_2 and so on to the last chromosome. Then the sum of differences is normalized by dividing it with the number of genes compared

$$\text{population variance} = \frac{\text{number of differences}}{\text{number of genes compared}}$$

Ideally, the search should end upon finding an optimal solution to the problem at hand. However, being a stochastic process, not every evolutionary search ends with convergence to an optimal solution. In order to terminate unsuccessful searches that do not converge, an additional, synthetic, stopping criterion is added. Thus, the GA is also terminated if the population does not converge to a solution. This is done by limiting the number of generations. That is, the algorithm halts if the population's variance drops below a predetermined threshold, or after a specified number of generations if the variance threshold has not been reached.

3.6 The FF. The FF is a crucial part of the GA, and its definition has an immense impact on the performance of the algorithm. An FF whose definition is not well suited for the problem does not steer the search in fertile directions while an algorithm sporting a well-defined FF is more likely to result in a successful search. In the code, different FFs are defined and their effect on the algorithm's success is studied. All of the FFs rely on extracting core values from the output file of the core simulation code DYN3D (Sec. 2.4). The different FFs studied are hereby introduced.

In order to test the algorithm, single objective FFs are used, with a simple objective for which there is an estimated optimal solution. Thus, the proximity to a successful and well-functioning algorithm can be assessed. The objective set for the algorithm is the maximization of k_{eff} It is, as requested, a simple objective that allows determining the effectiveness of the code in finding good LPs. As explained in Sec. 2.3, an intuitive guess for the optimized LP has been constructed, which is shown in Fig. 3.

The first form of the FF defined is the simplest form possible—the k_{eff} value itself, so

$$\text{FF} = k_{\text{eff}} \tag{1}$$

This form has the advantage of being an objective parameter value of the LP. It is not relative to the current population. But, there are problems with this simple form. If the k_{eff} values of the population are too close to each other, the FF does not have enough pressure to pull the search toward good LPs, when using the FP selection method.

The second FF definition takes the form of

$$\text{FF} = \frac{1}{\max(k_{\text{eff}}) - k_{\text{eff}} + \text{FFparameter}} \tag{2}$$

where FFparameter is used to regulate the scale of FF. Higher FFparameter values result in weaker dominance of the best chromosomes.

The last purely k_{eff} based FF replaces $\max(k_{\text{eff}})$ in the former FF (Eq. (2)) with a constant that represents an estimated upper limit of the k_{eff} value. This keeps the FF values relative to the constant k_{eff} limit instead of the population dependent $\max(k_{\text{eff}})$, thus solving the objectivity and dominance problems of the two former FF definitions. It minimizes the FF's relative relation to the current population and gives a more objective value for FF. In this study, the constant value of 1.5 is used, so the FF takes the form of

$$\mathrm{FF} = \frac{1}{1.5 - k_{\mathrm{eff}} + \mathrm{FFparameter}} \tag{3}$$

3.7 Elitism. In the process of generating the next population, the LPs of the current one are not safe from changes, be it good changes or otherwise. The process of crossover essentially destroys the parent solutions to create the offspring, which are not automatically better. So the best solutions of the population are sure to disappear from the next ones unless actively preserved. In order to protect them, the elitism strategy is introduced to the algorithm.

With the elitism strategy, the algorithm finds and stores the best chromosome in the population for later re-insertion into the new population. The best chromosome is not removed from the population; it still takes part in the selection, crossover, and mutation and is most likely to parent offspring that might improve on it. Even if they do not, though, with the elitist algorithm its genetic data is still preserved and passed on the next generation.

4 Genetic Operators

4.1 Selection. In the selection process, pairs of chromosomes are chosen for crossover. The chromosomes are selected according to their fitness values (Sec. 3.6). The chromosomes with the better fitness values are more likely to be selected as parents. Every chromosome receives a selection probability, which determines its probability to be chosen as a parent in the selection process. The selection probability can be calculated in many varied ways. In this work, it is calculated either by the FP method or by our modification of the LR method.

4.1.1 Selection Probability. When using the FP method, a chromosome's selection probability is calculated using the formula $P(c) = \mathrm{FF}(c)/\mathrm{sum}(\mathrm{FF})$, where c designates the chromosome. The consequence of this probability equation is that each chromosome gets a selection probability proportional to its FF value relative to the current population. The weakness in this method lies in the selection pressure it causes. Since the selection probability the FP selection method gives to different solutions is proportional to their respective FFs, the selection pressure is also dependent upon the relative differences between those FFs; big differences cause complete convergence of the population. If the high FF solution is not necessarily the best solutions possible but only better relative to the current population, this convergence is thus premature. This effect results from the fact that solutions that have very high FF values, relative to the current population, are much more likely to be selected as parents for the next generation and thus take over the gene pool and cause convergence, while "weaker" solutions are not selected at all and disappear from subsequent generations. On the other hand, differences that are too small do not impress upon the population enough selection pressure for a progression in any direction and result in search divergence. That is, in a population comprised of solutions of very similar FF, as is usually the case in the first generation of the evolution, the better (albeit slightly) ones do not receive any substantial selective advantage and are given a selection probability very close to others', a situation that renders the evolutionary process powerless to make any progress.

A possible solution to this problem is assigning selection probabilities more equally, independent of the fitness values themselves. One way of implementing this is the LR method, in which the chromosomes are ranked according to their relative FFs and given a selection probability based on their relative rank, rather than their FF values. This ranking procedure detaches the selection pressure from the exact value of the FFs and the differences between them. On the one hand, this allows the algorithm sufficient driving force toward the better solutions, in the case of very little difference in the FFs. On the other, it relieves the selection pressure off the leading chromosome in the case of great FF differences, allowing other chromosomes participation in the evolution and preventing premature convergence.

The implementation of the LR method in this study consists of ranking the chromosomes according to their FF values and giving each a selection probability linearly, according to its relative rank. The selection probability for every chromosome c is calculated using a parameter, expVal, that represents the expected number of copies of c in the selection table. The expected number expVal is calculated using the following formula:

$$\mathrm{expVal}(c) = 2 - m + \frac{2(m-1)(\mathrm{rank}-1)}{\mathrm{groupSize}-1} \tag{4}$$

where m is the maximum expected amount of copies for the *best* individual, and rank is the chromosome's relative rank in the group. The chromosome's selection probability is then $P(c) = \mathrm{expVal}(c)/\mathrm{groupSize}$. The variable m is predetermined and has a value within the range of $1 \leq m \leq 2$. Higher values of m result in greater selection pressure to the best solution, and vice versa. For example, when $m = 1$, the expected amount of copies for each chromosome in the group is 1, so they all receive the same probability to be selected. When $m = 2$, the best chromosome in the group gets a "double" probability to be selected, i.e., its selection probability is twice the probability of the middle ranked chromosome; and the worst chromosome, ranked 1, has zero probability to be selected. So the probability of a chromosome to be selected is proportional to the expected number of its copies in the selection table, which with LR is proportional to its relative rank in the population. Note that the equation for expVal is linear. It is maximized when rank = groupSize and minimized when rank = 1, and equals m and $2 - m$ for those cases, respectively.

4.1.2 Selection Methods. The selection methods chosen for this work are either the classic RW method or the tournament method. In the RW method, the chromosomes' selection probabilities are cumulatively summed up (up to 1, since they are probabilities) and a random number r between 0 and 1 is chosen, through which a chromosome for the selection table is selected. The cumulative sums are the "pockets" of the RW; the larger the probability the larger the "pocket"; and the chromosome of the first pocket larger than the randomly manufactured r is chosen.

Another method is the tournament, in which the selection pressure can be adjusted more easily, allowing for a higher degree of control over the algorithm's behavior. High selection pressure leads to preconvergence of the population, while low selection pressure does not result in any convergence to an optimal solution. In the tournament method, "tournaments" are held—choosing random groups of chromosomes from the population and picking, with some probability, the best chromosome in the group to enter the selection table. Tournament size is adjustable and is one of the parameters that allow control over the selection pressure. Larger tournament groups increase the pressure while smaller ones decrease it. It is also possible to control the tournament size through the evolution, allowing for another measure of control. Inside a tournament group, a "winner" can be determined in one of two ways: either by choosing the best chromosome or by choosing the RW method. If the RW method is chosen, the chromosome's probability to be chosen is calculated either by FP or LR, as described above.

4.2 Crossover. The crossover is the genetic operator responsible for the creation of new solutions out of the selected parent solutions. It mixes the genetic data of said parents, thus generating new solutions. It does so by swapping a segment of genes in between the chromosomes of two solutions. The segment's geometry has never before (to the best of our knowledge) been used as a variable of the algorithm, even though it may have significant

impact due to the geometric nature of the problem. Instead, crossover operators used thus far for the in-core fuel management problem have been standard "off the shelf" operators, which do not account for geometric effects.

The crossover operator used in this study is of geometrical nature and consists of a geometric crossover mechanism. This mechanism was developed to manipulate the genetic data given by the chromosome, in a way that allows control over the swapped core segment's shape (note that the genes in the chromosome are not identical to the core cells, but are a part of the core's mathematical representation. Hence, control over the geometry of the *core* segment swapped requires some thought as to how to control it through the chromosome, which, as opposed to the core, can be directly manipulated by the code). Using this geometric crossover mechanism, the segment's shape is controlled. It is chosen out of the following options:

(1) Chromosome consecutive segment—The core's indices contained in the chromosome between two randomly chosen cut points. Since the order of the cells in the chromosome is not consistent with the locations in the core, it is a collection of randomly placed cells in the core.
(2) Chromosome consecutive segment with size control—Same, but allows for control over the segment's size. That is, the number of cells being swapped is also under control. The segment size is determined by the following linear formula, which defines seg_size as a linear function of generation number, between the maximum and minimum sizes permitted

$$\text{seg\_size} = \text{floor}\left[\frac{(g - \text{crossGen}) \cdot (\text{minSeg} - \text{maxRec})}{\text{maxGens} - \text{crossGen}} + \text{maxRec}\right] \tag{5}$$

where minSeg and maxRec are the minimum and maximum segment sizes, respectively, and are determined as a parameter of the algorithm, crossGen is the generation in which the segment size starts to be limited, g is the current generation number, and maxGens is the maximum number of generations in the evolution process.
(3) Core vector consecutive segment—The segment of core's indices between two randomly chosen ones. This means swapping consecutive cells in the core; two random cells in the core are chosen and the segment swapped consists of all the consecutive cells between them.
(4) Rectangle of core neighbors—The rectangle of core indices defined by two randomly chosen indices.
(5) Square of core neighbors with size control not dependent on decrease starting generation—This is a correction for previous options, in which the starting generation of the segment size decrease has an influence over the rate of decrease. In order to change that and create a segment size formula that looks the same for different starting generations, i.e., the segment size decreases at the same rate without starting generation impact, we introduce the next formula. It depends not on the starting generation but on the difference $g - \text{crossGen}$

$$\text{recSide} = \text{floor}\left[\text{maxRec} \cdot \frac{I+J}{2} \cdot \frac{\text{crossDecRate}}{\text{crossDecRate} + \text{genDepend}(g - \text{crossGen})}\right] \tag{6}$$

where maxRec is the maximum portion of the core diameter size that the segment square side can reach, genDepend is a boolean parameter that determines if the segment's maximum size decreases over the generations or remains constant, and crossDecRate is a parameter that influences the decrease rate. The larger crossDecRate, the slower the decrease.

Option 1 is the standard crossover with the standard crossover segment used, while options 2–5 are novel and allow for different measures of control over the segment shape. Some of the options also allow the segment size to be controlled; decreased as a function of generation. The large segment swaps at beginning of evolution allows the weakly optimized chromosomes of the early populations to exchange large segments of genetic information, in search of the best solution. It also allows the gradual decrease of segment size to facilitate finer genetic alterations in the good solutions found.

4.2.1 Fixing the Chromosome After Crossover. For every GA whose solutions are of permutational nature, a simple segment swapping crossover is likely to result in invalid chromosomes that are no longer permutations.

As mentioned in Sec. 3.1, the genes in the chromosomes are indices in the core and chromosomes are permutations of those indices. So any chromosomal recurrences must be fixed. That is, there cannot be a core cell denominator that appears both in the first and second parts of the chromosome, since its counterpart in the respective LP should then need to contain two different FAs at the same time.

The correction is done by a separate function which, outside the chosen segment, swaps recurring indices between the two chromosomes. A demonstration of the operation of the crossover operator and the fixing process can be seen in Figs. 6–9.

4.3 Mutation. The evolutionary search is limited to the regions it is able to reach with the population's existing genetic pool. If a way out of these regions is not accessible, the algorithm's reach is limited to the local optima of those regions only. When the search is stuck on such local optima, it is said to be stagnated. The mutation operator is one measure used to avoid this stagnation. It does so by changing the chromosomes and introducing new genetic data into the population, thus reaching untapped regions of the search space.

Mutation on an individual is swapping two random genes within the chromosome. Since in this study the location of the genes in the chromosome is the trait that carries genetic significance, this action introduces new information into the genetic pool. For example, if none of the chromosomes contains the gene "*a*" in location "*A*" then due to the nature of the crossover operator none of the offspring will.

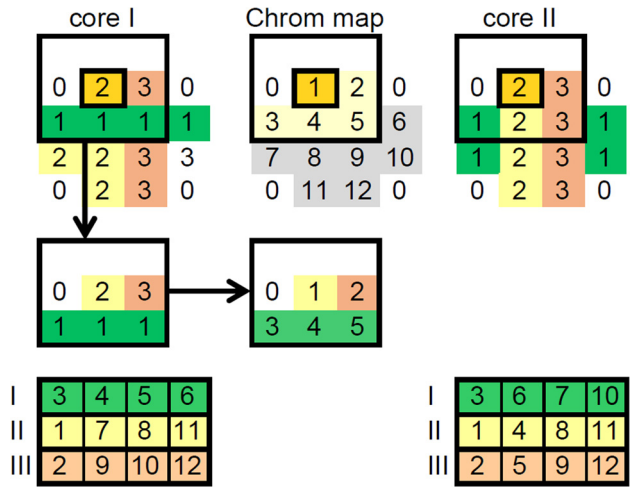

Fig. 6 Top—the original cores and the mapping of the chromosome to the core. Bottom—the corresponding chromosomes, with each fuel type, e. g., I, II, III, represented by different shade. The cell randomly chosen for crossover is marked with bold border in the upper panel. The randomly chosen neighborhood size is 3 × 3. Segment parts that are not in the core are omitted.

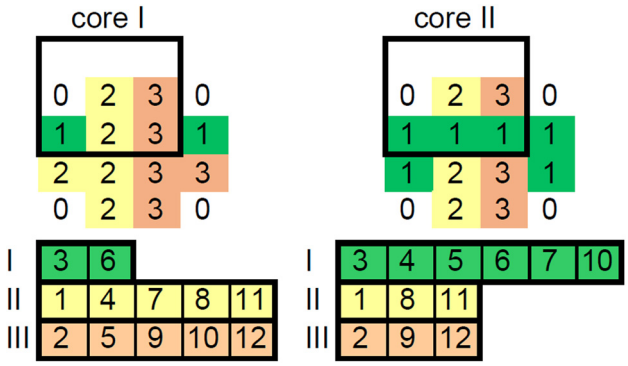

Fig. 7 The cores after segments swap. Notice that the number of FAs of each type is not preserved.

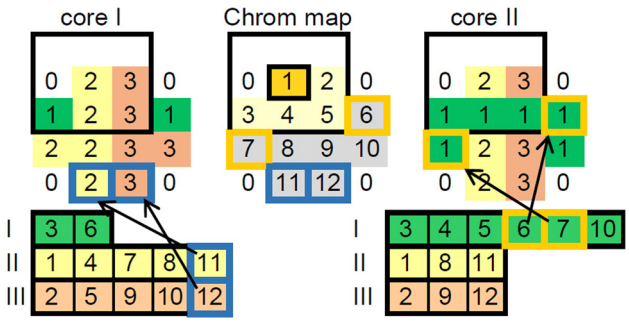

Fig. 8 Cells outside the selected segment are chosen to switch fuel type, in order to restore the original fuel inventory

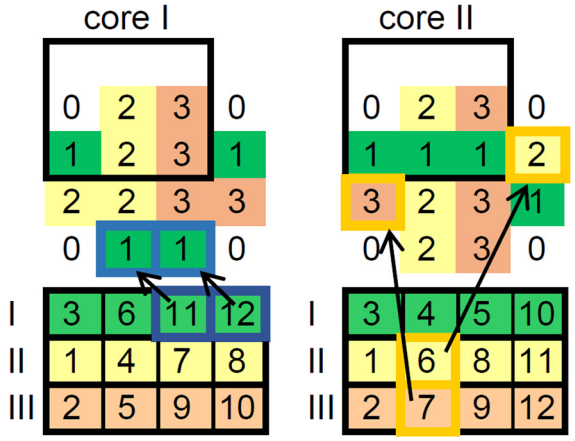

Fig. 9 The chosen cells are repositioned into the appropriate fuel type, resulting in two "legal" offspring cores

The basic mutation operator mutates a randomly chosen portion of the population. The portion size is normally distributed around an adjustable predetermined value that is an algorithm parameter, namely mutationRate. This parameter has a great effect on the search's stagnation. For example, if the mutation rate is increased dramatically (to a value greater than 1), most of the individuals in the population undergo mutation several times, thus creating a shuffling effect that allows the population to escape stagnation. Too high a mutation rate shuffles the genetic data in the population too much and distances the search from good solutions thus found; it does not allow the population to converge but keeps shuffling it. Alternatively, a mutation rate too low does not allow the population to escape stagnation and results in the search getting stuck on local optima.

There are three optional alterations added to the basic mutation. They are additional features of the algorithm's mutation operator and can be either switched on or off as pleased. First is the peak mutation in which the mutation rate is peaked every constant number of generations in order to shuffle the genetic pool of the population. This is done by increasing the mutation rate for one generation and then dropping it back to its original value.

The second variation of the mutation operator is the variance mutation. It adapts the mutation rate according to the population's variance. Low variance of the population too early in the evolution suggests premature convergence to local optima, while high variance indicates that the population has a difficulty in convergence. The variance mutation attempts to evade those problems by increasing the mutation rate when variance is too low and decreasing it when too high. However, a low population variance is not a globally harmful phenomenon. It is expected that the population should converge and display low variance at some point of the evolution, when it gets to an optimal solution. So, after a large enough number of generations the mutation rate tempering is stopped and the population is allowed to converge. This generation number is also a parameter of the algorithm.

The last variation is the decreasing mutation rate. The size of the population portion being mutated is decreased from a predetermined generation onward throughout the evolution. The idea behind this approach is letting the population wander through the search space in the beginning of the evolution and allowing it to converge as it progresses. The formula used to decrease the mutation rate depends on the difference between the current generation, g, and the generation in which we start to decrease, mutationGen. The pace of decrease is controlled by the mutDecRate parameter; high parameter values lead to a slower decrease pace

$$\text{mutationRate} = \text{mutationRate} \cdot \frac{\text{mutDecRate}}{\text{mutDecRate} + g - \text{mutationGen}} \tag{7}$$

5 Results and Discussion

Out of the many core parameters to be optimized, in the first stage of this project it was decided to optimize a single parameter of the reactor core—k_{eff}. The reason for choosing this objective is its relative simplicity, which allows the application of physical intuition based on reactor core physics. Because this objective is a simple one, it is possible to predict its expected optimal LP and see if the GA developed can produce an LP close to it, testing the algorithm's performance.

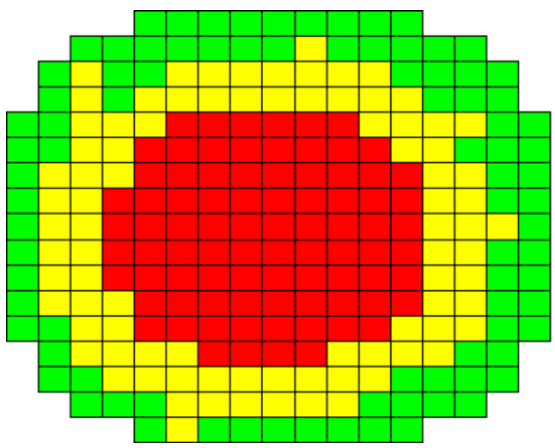

Fig. 10 The LP with the highest k_{eff} value produced by the GA algorithm with void BCs

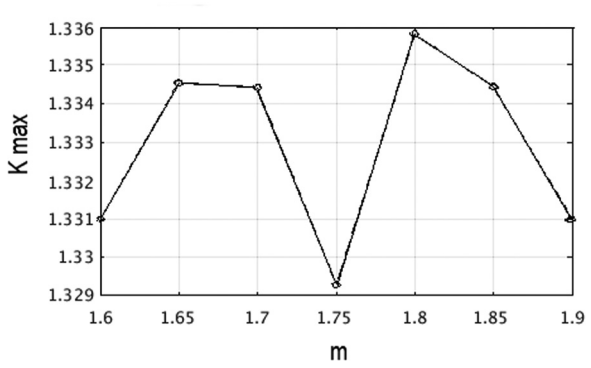

Fig. 11 k_{eff} as a function of maximum expVal with RW

5.1 Largest k_{eff} LP. In Sec. 2.1 the physical intuitive reasoning behind the construction of the estimated optimal LP for the single objective k_{eff} optimization problem is explained. It is presented in Fig. 3 and sports a k_{eff} value of 1.338932. The LP with the highest k_{eff} value produced by the GA algorithm is of $k_{eff} = 1.339627$ and is shown in Fig. 10. From the comparison between the two LPs, it is evident that the algorithm performs as required and produces an LP that outperforms the estimated one.

5.2 Selection—Maximum expVal Versus Tournament Size. When using the FP or LR selection methods with RW instead of tournament, good results appear around the value of expVal = 1.8, as can be seen in the graph in Fig. 11. The graph represents one set of comparable optimizations. Two such examples of relatively good LPs generated from optimizations with different parameter sets but with expVal = 1.8 can be seen in Figs. 12(*a*) and 12(*b*).

When using tournament, there seems to be a trade-off relationship between the maximum expVal parameter and the tournament size parameter. Higher maximum expVal values require bigger tournament groups to give good results, and vice versa. This phenomenon is probably due to the effect both parameters have on the convergence pressure—the higher the maximum expVal value, and the smaller the tournament, the higher the pressure. This correlation between the parameter and the convergence pressure is demonstrated very clearly for the case of the maximum expVal parameter in Fig. 13.

The best results with tournament seem to appear around the values of $1.57 < \text{expVal} < 1.65$ with tournament size of about $1/30$ of the population size. These results can be observed in Fig. 14. It is surprising to see that without tempering with other parameters, the RW produces better results, as can be seen from comparing the LPs in Figs. 12(*a*) and 12(*b*), possessing values of $k_{eff} = 1.335648$ and $k_{eff} = 1.335812$, respectively, to the LP of Fig. 12(*c*) which has a k_{eff} value of 1.335222 and was produced

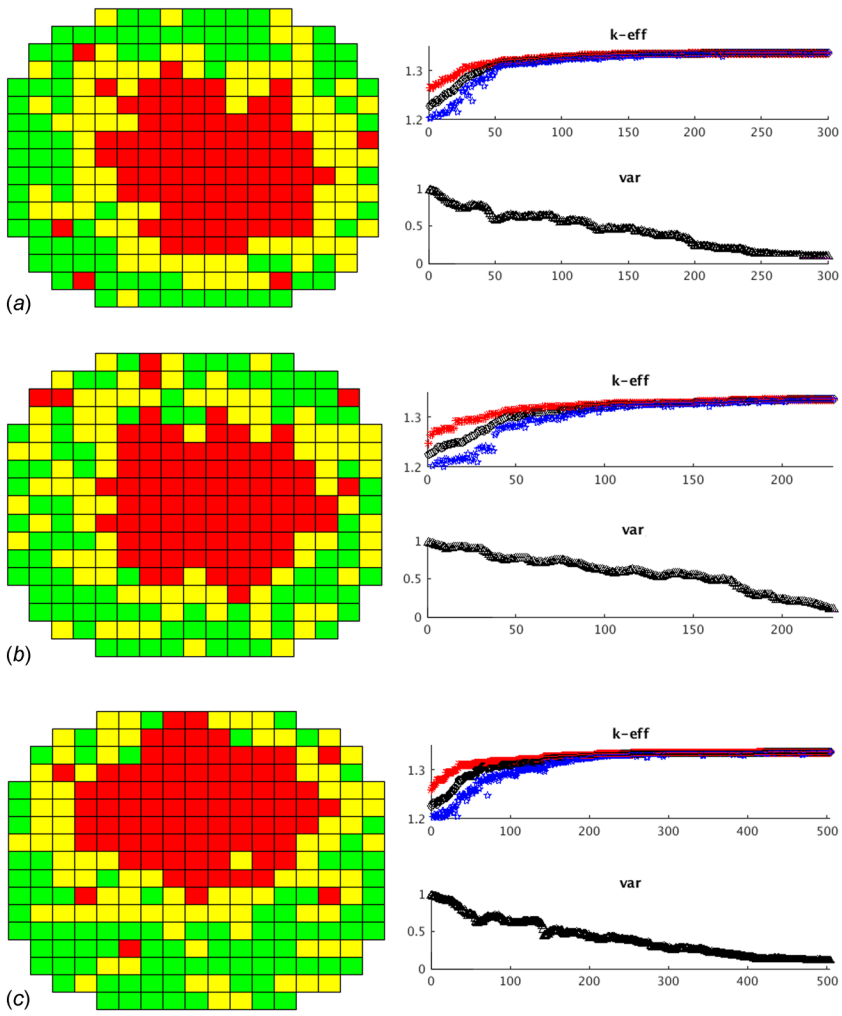

Fig. 12 LPs generated from optimizations with different parameter sets, all with maximum expVal = 1.8, alongside the evolution of their (lighter shades represent the maximum and minimum k_{eff} of each generation, whereas dark black represents the mean) and population variance

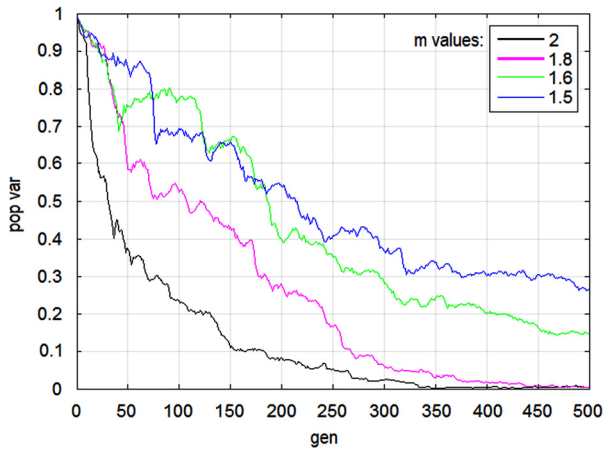

Fig. 13 The variance of the population as a function of generation number for different maximum expVal values (*m*) with RW selection

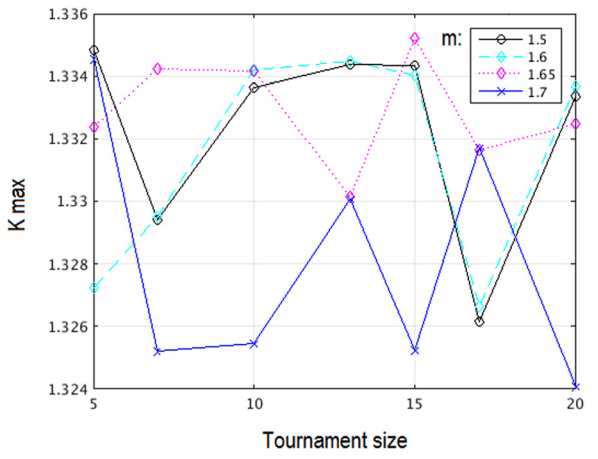

Fig. 14 k_{eff} as a function of tournament size for different maximum expVal values

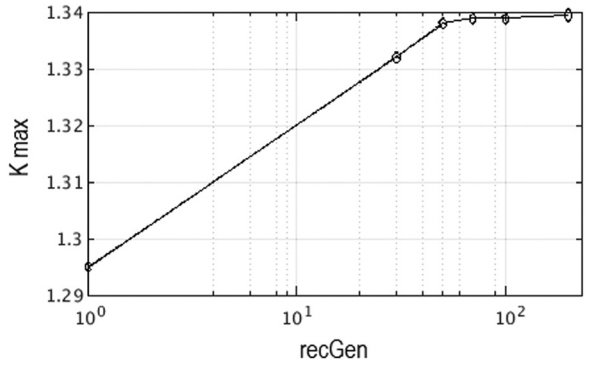

Fig. 15 k_{eff} versus recGen. recGen values are 1, 30, 50, 70, 100, and 200.

from an optimization with the parameter values of expVal = 1.65 and tournament size = 15.

5.3 Crossover Segment Size. Adding a geometric crossover that limits the segment's size from some generation on proved to be the most influential change at this stage of the study. The results improve dramatically when using the limited segment crossover. Below the value of crossGen = 50, populations converge too quickly to bad results. LPs with good k_{eff} values start appearing above crossGen = 50 and seem to peak in amount between the values of 200 and 350, as shown in Fig. 15.

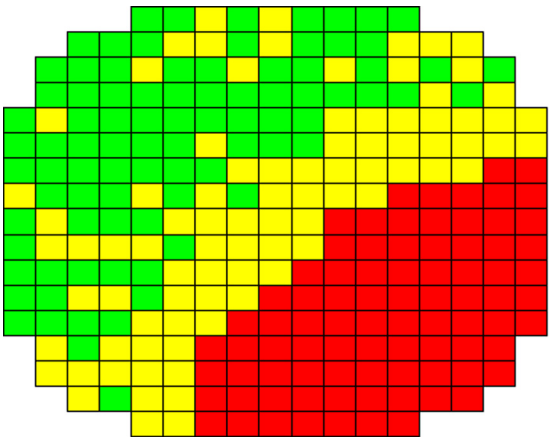

Fig. 16 The LP with the highest k_{eff} value produced by the GA algorithm with reflective BCs

5.4 Population Size. Different population sizes have been tested and show that bigger population results are better when the parameters of the algorithm are good. When the parameters are not good, a bigger population causes excessive variance that results in a smaller chance to converge to a good result.

5.5 Changing BCs. All results so far presented originate from optimization with void BCs. As explained, in this case the best LP tend to concentrate the high enrichment FAs in center of the core and the less enriched ones around in a concentric circular pattern of descending enrichment, as shown in Fig. 10.

What happens when the BCs are changed while keeping the GA parameters constants? This is a purely physical effect. An example of a very simple problem, which is not so intuitive, can be considered by altering the problem described in Sec. 2.1 to completely reflective BCs, i.e., no neutron escapes the core. In that case, our intuition as core physicists often fails, as the LP of the highest k_{eff} core is an asymmetric one. The solution for this simple problem is shown in Fig. 16.

6 Summary and Conclusions

A complete calculation scheme was developed and studied for the application of GAs to the in-core fuel management single objective k_{eff} optimization problem. Very good results are displayed by the algorithm for the simple k_{eff}-based FF. Extensive research in the vast parameter space reveals a trade-off between maximum expVal and tournament size, probably because both affect the convergence pressure. For no tournament there seems to be a peak of results around max expVal = 1.8, and with tournament around 1.57 < max expVal < 1.65 and tournament size = 1/30 of the population size, depending on other parameters. Further research shows better results for LR with tournament.

Some different crossover operators were checked. The best applied a geometric crossover that decreases the limit size of the crossed segment after a predetermined generation number (crossGen). The results seem to be better for crossGen > 50, and the best ones appear around 200 < crossGen < 350.

Finally, very good results are obtained for changing the BCs of the core, without modifications to the code. This means that the FF was enough to inform the code of the changes to the core's physics and also that the code is not effected by hidden bias.

To conclude the first stage of the study, the evolutionary algorithm seems very much suited for solving the simplified single objective nuclear core loading problem presented. The results

obtained after much research of the parameter search space exceed the ones predicted. Moreover, the code seems to handle changes in the physics of the problem without difficulty and without modification, and produce the expected results. At the end of this stage, a well-functioning EA adapted to the simple single objective is obtained and ready for modification to add the minimization of the PPF value of the LP as an objective of the problem.

Nomenclature

ADF = assembly discontinuity factors
APWR = advanced pressurized water reactor
BC = boundary conditions
FA = fuel assembly
FF = fitness function
FP = fitness proportionate
GA = genetic algorithm
HZDR = Helmholtz Zentrum Dresden-Russendorf
LP = loading pattern
LR = linear ranking
NP = nondeterministic polynomial-time (in computational complexity theory)
NPP = nuclear power plant
PPF = power peaking factor
PWR = pressurized water reactor
RW = roulette wheel

References

[1] Turinsky, P. J., 2005, "Nuclear Fuel Management Optimization: A Work in Progress," Nucl. Technol., **151**(1), pp. 3–8.
[2] Turinsky, P. J., Keller, P. M., and Abdel-Khalik, H. S., 2005, "Evolution of Nuclear Fuel Management and Reactor Operational Aid Tools," Nucl. Eng. Technol., **37**(1), pp. 79–90.
[3] Holland, J. H., 1975, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI.
[4] Goldberg, D. E., 1989, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed., Addison-Wesley Longman Publishing, Boston, MA.
[5] Parks, G. T., 1996, "Multiobjective PWR Reload Core Design by Nondominated Genetic Algorithm Search," Nucl. Sci. Eng., **124**(1), pp. 178–187.
[6] Haibach, B. V., and Feltus, M. A., 1997, "A Study on the Optimization of Integral Fuel Burnable Absorbers Using the Genetic Algorithm Based Cigaro Fuel Management System," Ann. Nucl. Energy, **24**(6), pp. 439–448.
[7] Parks, G. T., and Miller, I., 1998, *Selective Breeding in a Multiobjective Genetic Algorithm*, Springer, Berlin, pp. 250–259.
[8] 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," Ann. Nucl. Energy, **26**(7), pp. 641–655.
[9] Toshinsky, V. G., Sekimoto, H., and Toshinsky, G. I., 1999, "Multiobjective Fuel Management Optimization for Self-Fuel-Providing LMFBR Using Genetic Algorithms," Ann. Nucl. Energy, **26**(9), pp. 783–802.
[10] 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 Nondominated Solutions," Ann. Nucl. Energy, **27**(5), pp. 397–410.
[11] Hongchun, W., 2001, "Pressurized Water Reactor Reloading Optimization Using Genetic Algorithms," Ann. Nucl. Energy, **28**(13), pp. 1329–1341.
[12] Gang, P., Feng, P., and Rong, F., 2002, "Application of Genetic Algorithm in Research and Test Reactor Core Loading Pattern Optimization," PHYSOR 2002, Seoul, Korea, Paper No. 8A-03.
[13] Erdogan, A., and Geckinli, M., 2003, "A PWR Reload Optimisation Code (XCore) Using Artificial Neural Networks and Genetic Algorithms," Ann. Nucl. Energy, **30**(1), pp. 35–53.
[14] Pereiraa, C. M., and Lapa, C. M., 2003, "Coarse-Grained Parallel Genetic Algorithm Applied to a Nuclear Reactor Core Design Optimization Problem," Ann. Nucl. Energy, **30**(5), pp. 555–565.
[15] Ortiz, J. J., and Requena, I., 2004, "An Order Coding Genetic Algorithm to Optimize Fuel Reloads in a Nuclear Boiling Water Reactor," Nucl. Sci. Eng., **146**(1), pp. 88–98.
[16] Alim, F., Ivanov, K., and Levine, S. H., 2008, "New Genetic Algorithms (GA) to Optimize PWR Reactors Part I: Loading Pattern and Burnable Poison Placement Optimization Techniques for PWRs," Ann. Nucl. Energy, **35**(1), pp. 93–112.
[17] Alim, F., Ivanov, K., Yilmaz, S., and Levine, S. H., 2008, "New Genetic Algorithms (GA) to Optimize PWR Reactors Part II: Simultaneous Optimization of Loading Pattern and Burnable Poison Placement for the TMI-1 Reactor," Ann. Nucl. Energy, **35**(1), pp. 113–120.
[18] Khoshahval, F., Minuchehr, H., and Zolfaghari, A., 2011, "Performance Evaluation of PSO and GA in PWR Core Loading Pattern Optimization," Nucl. Eng. Des., **241**(3), pp. 799–808.
[19] Rahmania, Y., Pazirandeh, A., Ghofrani, M. B., and Sadighi, M., 2013, "Using a Combination of Weighting Factors, Genetic Algorithm and Ant Colony Methods to Speed up the Reloading Pattern Optimization of VVER-1000 Reactors," *Transactions of the Conference Safety Assurance of NPP With WWER* Vol. 1, V. Mokhov, S. Sorokin, S. Titova, and E. Serdobintseva, eds., JSC OKB GIDROPRESS, Podolsk, Russia.
[20] Zameer, A., Mirza, S. M., and Mirza, N. M., 2014, "Core Loading Pattern Optimization of a Typical Two-Loop 300 MWe PWR Using Simulated Annealing (SA), Novel Crossover Genetic Algorithms (GA) and Hybrid GA(SA) Schemes," Ann. Nucl. Energy, **65**, pp. 122–131.
[21] Mitsubishi Heavy Industries, Ltd., 2013, "APWR Design Control Document (DCD)," Mitsubishi Heavy Industries, Ltd., Tokyo, Japan, Tier 2, Chap. 4, Rev. 4.
[22] Grundmann, U., Rohde, U., and Mittag, S., 2000, "DYN3D–Three-Dimensional Core Model for Steady State and Transient Analysis of Thermal Reactors," PHYSOR 2000, Pittsburgh, PA, Paper No. 155.
[23] Stammler, R. J., 2003, "HELIOS Methods," Studsvik Scandpower, Kjeller, Norway.