Choosing Representation, Mutation, and Crossover in Genetic Algorithms

Alexander Dockhorn
Leibniz University Hannover

Simon Lucas
Queen Mary University of London

***Abstract*—This paper aims to provide an introduction to evolutionary algorithms and their three main components, i.e., the *representation of solutions* and their modification through *mutation* and *crossover operators*. Given its introductory nature, it is specifically designed for newcomers to this exciting research area. We would like to note that this short paper just represents a summary of the full paper found online. The latter provides interactive components for a hands-on exploration of the covered material and can be found under: https://aiexplained.github.io/**

I. Introduction

Evolutionary algorithms (EA) can be used to solve plenty of tasks using the power of evolutionary optimization. One of the key tasks in the algorithm design process is the choice of representation for a candidate solution. While the phenotype describes a solution that is directly applicable to the task, the genotype is its computational counter-part which is used during the optimization. Given the task of optimizing a car for speed, the phenotype is represented by the actual car on the road, while the genotype represents the car's configuration, e.g. the type of motor and the tires used. During optimization, the computer will modify a candidate's genotype, whereas the phenotype will be built according to the genotype's description for evaluation.

In this paper, we are going to review some of the most prominent choices for mutation and crossover of matrix and vector-based genomes. Those will be studied in the context of the 8-Queens Puzzle to be introduced in Section II, for which we will discuss several types of representations in Section III. Furthermore, we will analyze how the choice of representation guides the development of respective mutation and crossover operators in Section IV and Section V.

II. 8-Queen Puzzle

The mapping between genotype and phenotype plays an important part in the algorithm design process. In this article, we will present EA design concepts based on a discussion of the 8-Queens Puzzle. This seemingly simple Chess problem requires you to position eight queens on an 8×8 chessboard, so that no two queens threaten each other. Thus, no two queens share the same row, column, or diagonal. Figure 1 shows an exemplary solution to the puzzle. In total, there are 92 unique solutions to the puzzle, coming down to 12 when removing rotations and reflections.

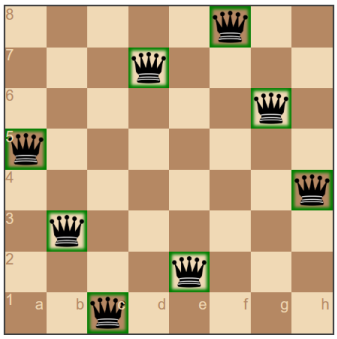

Fig. 1. An exemplary solution of the 8-Queens Puzzle

III. Representation

To allow for an efficient search and optimization of solutions, we first need to define how a solution candidate should be encoded. We have chosen the 8-Queens puzzle because it allows for multiple simple but intuitive types of representation. Thinking about the basic properties of our task and its likely solution can serve as a guideline for our design process. Therefore, we first summarize the underlying requirements:

1) a chessboard offers 64 fields arranged in an 8×8 grid.
2) the solution consists of exactly 8 queens.
3) each row can include only one queen, otherwise, they would threaten each other.
4) each column can include only one queen; otherwise, they would threaten each other.
5) each diagonal can include only one queen; otherwise, they would threaten each other.

We can create different representations of solutions by taking a varying set of requirements into account. Just considering the first one, we can encode a chessboard as an 8×8 *binary matrix* in which a 0 encodes an empty field and a 1 the position of a queen. Such a simple representation would allow us to encode all valid solutions, but also many invalid candidates of which we already know cannot be valid due to requirements 2-5. More specifically, it allows placing more or less than 8 queens on a board. Since we already know that we need to position exactly 8 queens, we can simply constrain the binary matrix to include exactly 8 queens.

So far, we have incorporated the first two requirements in our encoding. We can do even better by adding the constraints on rows, columns, and diagonals to it. Let's consider an even

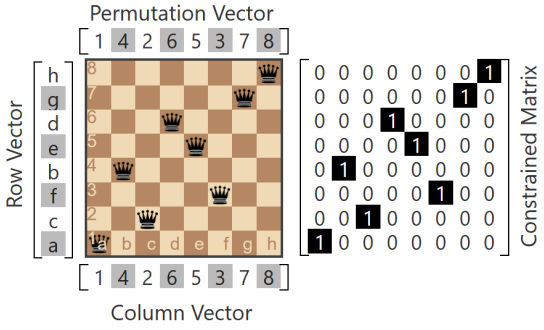

Fig. 2. Comparing representations of the 8-Queens Puzzle.

more condensed representation using the third requirement. Since each row can contain only a single queen, we can use a vector of size 8 to encode the column each queen is placed at. The same can be done by using forth the requirement, by representing a solution in terms of a vector in which each cell encodes the row of the respective queen. Hence, we will call the former solution the row vector representation and the latter the column vector representation.

Incorporating both of the former requirements yields a permutation vector in which the column vector representation is constrained to include every row number only once. While each permutation vector is also a valid column vector, there can be column vectors that are not valid permutation vectors.

As we have seen above, the type of representation defines the search space of our optimization problem. Figure 2 and Table I summarize introduced encodings and their resulting search space size. By including a larger number of requirements in our representation, we can reduce the search space to a feasible size, and therefore, speed up the optimization process.

IV. Mutation

Mutation is an evolutionary operator that modifies a single individual of our population. It is often used for a local search trying to further improve an already promising solution candidate. This is achieved by adding small changes to a candidate solution, such that the resulting individual is slightly different from its parent.

For binary encodings, we can use the bit (or value) mutation, in which we randomly flip (replace) one or multiple bits (values) of the individual. In terms of our matrix representation, this would either introduce a new queen at the flipped position or removing a queen from the board, respectively. Applying this to the constrained matrix has the potential to violate the representation's requirements. More specifically, we need to assure that the number of queens on the board remains constant

To achieve this, we can make use of swapping or displacement mutations in which two or more elements of the encoding are swapped. Simple forms include swapping two positions of the board or inverting the whole sequence.

V. Crossover

The crossover is an evolutionary operator that uses two or more candidate solutions to create a new one. The original

TABLE I
Size of the search space per representation type.

Representation	Size of an Individual	Size of the Search Space
Binary Matrix	$8 \times 8 = 64$	$2^{64} \approx 1.8 \cdot 10^{19}$
Constrained Matrix	$8 \times 8 = 64$	$\binom{64}{4} = 4426165368$
Row Vector	8	$8^8 = 16777216$
Column Vector	8	$8^8 = 16777216$
Permutation	8	$8! = 40320$

candidate solutions will be called parents and the resulting individuals will be named children, respectively. Similar to the mutation operator, a crossover needs to fit the underlying representation of an individual.

In terms of vector and matrix-based representation, the 1-point crossover is one of the most simple to implement. Therefore, we first choose a cut-off point along with one of the axes of our underlying representation. Thereafter, we swap the two ends of the parents to produce two children, which each consist of at least one element of their parents. The same can be done for multiple cut-off points. Instead of choosing a cut-off point, we can define a probability for swapping some of the parents' elements. This makes it especially simple to extend the crossover to multiple parents.

Once again, individuals produced by the two previously discussed crossover types have the potential to violate constraints of our representation. All the crossover methods we have seen so far can be applied to the constrained matrix and vector-based representations. However, they are not suitable for permutations. Specialized methods, such as the cycle crossover can keep permutations intact.

VI. Conclusion

In this paper, we introduced several representations and their respective mutation and crossover operators for solving the 8-Queens Puzzle. We have shown how the choice of representation impacts the size of the search space and the choice of genetic operators. This short paper has been limited in the breadth and depth of the discussion. A larger selection of mutation and crossover operators and their effects on the board can be found in the full version of the paper online. Furthermore, introductory literature on evolutionary algorithms [1] as well as more detailed selections of mutation [2] and crossover operators [3] can help to better understand the EA design process.

References

[1] R. Kruse, C. Borgelt, C. Braune, S. Mostaghim, and M. Steinbrecher, *Computational Intelligence*. Springer London, 2016.
[2] T. Bäck, D. B. Fogel, D. Whitley, and P. J. Angeline, "Mutation operators," *Evolutionary computation*, vol. 1, pp. 237–255, 2000.
[3] A. Umbarkar and P. Sheth, "Crossover Operators in Genetic Algorithms: a Review," *ICTACT Journal on Soft Computing*, vol. 06, no. 01, pp. 1083–1092, Oct. 2015.