Hindawi
International Journal of Computer Games Technology
Volume 2023, Article ID 2445335, 14 pages
https://doi.org/10.1155/2023/2445335

Research Article

Solving Full $N \times N \times N$ Rubik's Supercube Using Genetic Algorithm

Robert Świta and Zbigniew Suszyński

Faculty of Electronics and Informatics, Koszalin University of Technology, Koszalin, Poland

Correspondence should be addressed to Robert Świta; robert.swita@wp.pl

Received 1 October 2022; Revised 6 December 2022; Accepted 26 December 2022; Published 14 January 2023

Academic Editor: Fabrizio Balducci

The article presents an algorithm that uses an evolutionary approach to the problem of solving the Full Rubik $N \times N \times N$ Supercube, i.e., the orientation of all cubies, including the internal ones, not only according to face colors but to the same orientation in 3D space. The problem is formally defined by the matrix representation using affine cubies transforms. The Full Supercube's solving strategy uses a series of genetic algorithms that try to find a better cube configuration than the current one. Once found, movements are made to change the current configuration. This strategy is repeated until the cube is solved. The genetic algorithm limits the movements to the current cluster by solving the cube in stages, outwards from the center of the cube. The movements that solve the clusters are saved as macros and used to train and speed up the algorithm. The purpose of the presented algorithm is to minimize the solution time and not necessarily the number of moves.

1. Problem

The problem under consideration concerns solving new type of the N-segment Rubik's $N \times N \times N$ supercube, which includes orientation of all internal cubies, using the genetic algorithm. This problem is much more general than solving traditional cubes due not only the need to correct orientation of the off-border cubies (and not only to the appropriate face color) but also to orient the inner cubies. To our knowledge, such structures, though being natural generalizations of Rubik's cube, were not yet investigated.

The traditional cube has three layers, it does not contain inner cubies, and the orientation of the middle cubies is usually not important. The first solving algorithms were developed in the early 1980s, and the most popular is Thistlethwaite's [1], based on the division of the task into 4 stages, in which the number of possible moves is gradually reduced. These restrictions apply to the possible angles of rotation for certain segments to only 180°. In the last step, all segments are subject to this limitation. For cubes larger than traditional ones, it is more important to limit the movements of the increasing number of segments, rather than types of rotation axis or values of rotation angles.

The problem of arranging a traditional cube with the use of the shortest possible sequence of movements is often solved by algorithms for iterative A* searching of the state tree, down to a specific depth (e.g., IDA *). A typical example is Kociemba's algorithm [2], which also breaks the problem into substages but uses state tree search and heuristic functions. The most common approach, however, is to use special macromovement sequences (macros) that change the orientation of a small number of cubies, smaller than that resulting from a single movement [3]. Recent research has allowed to determine the minimum number of moves necessary to arrange $N \times N \times N$ standard cubes (without internal cubies) from any configuration to $O(N^2/\log N)$, although the optimal algorithm is not yet known [4, 5]. Already in 1994, Herdy and Patone solved the cube using a genetic algorithm that used macromovements as genetic operators [6]. Later studies tried to develop strategies without using expert knowledge [7, 8] or to use already known heuristic algorithms [9, 10].

The aim of the presented work was to develop the most effective and stable algorithm to solve the presented problem in the shortest possible time (number of GA), regardless of the number of movements made. With the problem posed in this way, the presented solution is based on the execution of a sequence of genetic algorithms improving the configuration of the cube in accordance with the adopted evaluation function and defined genetic operators. The solution found by the genetic algorithm is used by executing movements written in the chromosome of the best individual only if it improves the cube configuration; otherwise, the attempt is repeated. The solving algorithm is divided into stages, and the sequences that complete the stages are memorized and used for teaching and speeding up of the algorithm.

2. Cubies Orientation

Rubik's $N \times N \times N$ cube has been defined in a coordinate system whose center coincides with geometric center of the cube. Each cubie of a cube is related to its local coordinate system defined in relation to the parent system of the entire cube. Movements of cube slices, in such defined system, result in additional rotations R of the cubies around their selected *axis*. Each cubie from the slice changes its origin position from p to p' according to the linear formula:

$$p' = Rp. \tag{1}$$

After transformation, cubies swap their position and change orientation; therefore, configuration of the cube can be represented as a 3D array of cubies composite transformations. All cubies that can occupy the same position as a result of rotation, belong to the same set, called a cluster.

Positions p of the cubies do not have to be stored in the matrix, because they are dependent on their indices v in the 3D matrix. Since the indices are always nonnegative in the range $[0 : N-1]$, the cubie coordinates p can be defined, e.g., by shifting their indices v by the indices of the cube center C, so that a shifted center will move to the origin of the cube coordinate system:

$$\begin{aligned} p &= v - C, \\ p' &= v' - C, \end{aligned} \tag{2}$$

where $C = [(N-1)/2, (N-1)/2, (N-1)/2]$.

Positions p would be then origins of the cubies in the range $[-(N-1)/2, (N-1)/2]$ in each dimension, with an offset step of 1 from one another. The matrix of their transformation is then a combination of the translation to the center of the cube's system, the orthogonal rotation matrix, and inverse of this translation:

$$v' = R(v - C) + C. \tag{3}$$

The orientation and position of the cubies are thus clearly defined by the hitherto position of the cubie p and the rotation matrix R, which can be expressed as the composition of basic transformation matrices of the cube slice rotation in the planes of its system, around the X, Y, or Z axis by 90°. In column notation, these matrices are defined by

$$\begin{aligned} R_X &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}, \\ R_Y &= \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}, \\ R_Z &= \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \end{aligned} \tag{4}$$

Performing successive rotations in the same axis results in the combination of the same transformations, i.e., the exponentiation of these matrices. Obviously, their fourth powers are identity matrices (full angle rotations), hence,

$$R_{\text{axis}}^{\varphi} = R_{\text{axis}}^{\varphi \pm 4}. \tag{5}$$

Since four orientations with the face in front are possible for each face of the cube (four multiples of the angle of 90° around the vector perpendicular to the face), the total number of possible cubie orientations (as well as the maximum number of different positions of the cubie p' or the maximum number of cubies in a cluster) is 24. Therefore, these orientations can be written shortly as a 5-bit number, but the most convenient way is to define them by using the Euler angles, presenting rotation R as a combination of elementary rotations around the cube axis in the order X, Y, and Z, respectively, by the angles $[\alpha', \beta', \gamma'] = \pi/2[\alpha, \beta, \gamma]$, where $\alpha, \beta, \gamma = 0 : 3$.

$$\begin{aligned} R = R_Z^{\gamma} R_Y^{\beta} R_X^{\alpha} = &\begin{bmatrix} \cos\gamma' & -\sin\gamma' & 0 \\ \sin\gamma' & \cos\gamma' & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta' & 0 & \sin\beta' \\ 0 & 1 & 0 \\ -\sin\beta' & 0 & \cos\beta' \end{bmatrix} \\ &\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha' & -\sin\alpha' \\ 0 & \sin\alpha' & \cos\alpha' \end{bmatrix}. \end{aligned} \tag{6}$$

The determination of orientation by means of the Euler angles is not unequivocal. Different combinations of angles may specify the same orientations. For any rotation Q, a conjugation, like

$$Q^{-1} R_{\text{axis}} Q = R_{\text{axis}'} \tag{7}$$

changes only rotation axis from axis to $\text{axis}' = Q^{-1}\text{axis}$. A real eigenvector axis' corresponds to the eigenvalue $\lambda = 1$

Table 1

φ	0	1	2	3
φ'	0	90	180	270
$\sin\varphi'$	0	1	0	-1
$\cos\varphi'$	1	0	-1	0

```
function φ := GetAngle(sinφ', cosφ')
    φ := 0
    if (cosφ' > ε)   φ := 0 endif
    if (sinφ' > ε)   φ := 1 endif
    if (cosφ' < −ε) φ := 2 endif
    if (sinφ' < −ε) φ := 3 endif
endfunction
```

Figure 1: Function for the determination of rotation angle.

and does not change its direction during rotation $Q^{-1}R_{\text{axis}}Q$, and hence, it is an axis of this rotation:

$$Q^{-1}R_{\text{axis}}Q\bullet Q^{-1}\text{axis} = Q^{-1}\text{axis}. \tag{8}$$

Using as a Q, e.g., 90° rotation about the X axis, changes rotation axis from the Z axis to the Y axis, because

$$R_X^{-1}[0\,0\,1]^T = [0\,1\,0]^T. \tag{9}$$

Therefore, the following equations are true:

$$\begin{aligned} R_Y^{-1}R_X^{\gamma}R_Y &= R_Z^{\gamma},\\ R_X^{-1}R_Z^{\beta}R_X &= R_Y^{\beta},\\ R_Y^{-2}R_X^{\gamma}R_Y^{2} &= \mathrm{R}_{-X}^{\gamma} = R_X^{-\gamma},\\ R_Z^{-2}R_X^{\gamma}R_Z^{2} &= R_{-X}^{\gamma} = R_X^{-\gamma}. \end{aligned} \tag{10}$$

For α, β, and γ taking values in the range [0: 3], all possible cubie orientations can be obtained with just two movements; i.e., one of the angles can be considered to be equal zero:

$$R_Z^0R_Y^{\beta}R_X^{\alpha} = \begin{bmatrix} \cos\beta' & \sin\alpha'\sin\beta' & \cos\alpha'\sin\beta' \\ 0 & \cos\alpha' & -\sin\alpha' \\ -\sin\beta' & \sin\alpha'\cos\beta' & \cos\alpha'\cos\beta' \end{bmatrix},$$

$$R_z^{\gamma}R_Y^0R_X^{\alpha} = \begin{bmatrix} \cos\gamma' & -\sin\gamma'\cos\alpha' & \sin\gamma'\sin\alpha' \\ \sin\gamma' & \cos\gamma'\cos\alpha' & -\cos\gamma'\sin\alpha' \\ 0 & \sin\alpha' & \cos\alpha' \end{bmatrix},$$

$$R_z^{\gamma}R_Y^{\beta}R_X^0 = \begin{bmatrix} \cos\beta' cos\gamma' & -\sin\gamma' & \sin\beta'\cos\gamma' \\ \cos\beta'\sin\gamma' & \cos\gamma' & \sin\beta'\sin\gamma' \\ -\sin\beta' & 0 & \cos\beta' \end{bmatrix}. \tag{11}$$

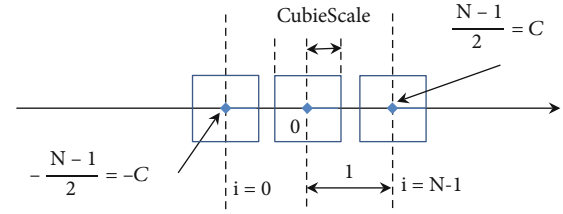

Figure 2: Arrangement of cubies in the cube coordinate system.

Which orientation definition can be used may be determined, e.g., by the position of the zero element in the rotation matrix (in bold in equation (11)). Due to the adopted order of transformations, rotations in the form $R_z^{\gamma}R_X^{\alpha}$ and $R_Y^{\beta}R_X^{\alpha}$ or $R_z^{\gamma}R_X^{\alpha}$ and $R_z^{\gamma}R_Y^{\beta}$ are enough to determine any orientation. We chose the first option. Using equation (10), it can be proved that for every possible value of β, the matrix R can be represented as a combination of just such rotations:

$$\begin{aligned} R(\beta=0) &= R_Z^{\gamma}R_X^{\alpha},\\ R(\beta=1) &= R_Z^{\gamma}R_YR_X^{\alpha} = R_Y^{-1}R_X^{\gamma}R_Y^2R_X^{\alpha} = R_Y^{-1}\left(R_X^{\gamma}R_Y^2\right)R_X^{\alpha}\\ &= R_Y^{-1}\left(R_Y^2R_X^{-\gamma}\right)R_X^{\alpha} = R_YR_X^{\alpha-\gamma},\\ R(\beta=2) &= R_Z^{\gamma}R_Y^2R_X^{\alpha} = R_Z^{\gamma}\left(R_X^{-1}R_Z^2R_X\right)R_X^{\alpha} = R_Z^{\gamma}\left(R_Z^{-2}R_X\right)R_X^{\alpha+1}\\ &= R_Z^{\gamma-2}R_X^{\alpha+2},\\ R(\beta=3) &= R_Z^{\gamma}R_Y^3R_X^{\alpha} = \left(R_Y^{-1}R_X^{\gamma}R_Y\right)R_Y^3R_X^{\alpha} = R_Y^3R_X^{\alpha+\gamma}. \end{aligned} \tag{12}$$

Since the angles are multiples of 90°, it is easy to determine them from the signs of the sine and cosine functions (Table 1, with $\varphi = \alpha, \beta, \gamma$), which are elements of the rotation matrix (Figure 1).

$\varepsilon$is some small nonnegative value for taking account of float-point errors (smaller than cubieScale from Figure 2).

From the point of view of the movements performed, more favorable orientations are those that are composed of fewer moves. Any orientation can be defined by two or less moves. The orientations obtained as a result of a single movement must have one positive value in the cubie rotation matrix (equal to one in the absence of cubie scaling) on its diagonal. This number can also be determined directly from the values of γ, β, and α. Accurate orientation information is also important as it will allow evaluation function to recognize and remember cube configurations. The cubie's orientation vector, briefly referred to as its state, was encoded on 8 bits. Each of the consecutive 2 bits represents the number of turns (see Table 2):

$$\begin{aligned} \text{moveCount} &= \text{sign}\,(\gamma) + \text{sign}\,(\beta) + \text{sign}\,(\alpha),\\ \text{state} &= \text{moveCount} \ll 6|\gamma \ll 4|\beta \ll 2|\alpha. \end{aligned} \tag{13}$$

Orientation $R_z^2R_X^2$ can be more preferably achieved with one move. In this case, it is worth to change its definition to R_Y^2. Extraction of the Euler angles from cubie composite affine transform R and definition of its orientation state are presented also in a form of pseudocode (Figure 3).

Table 2: Possible orientations of the cubie in the form of Euler angles (α', β', and γ'), matrix R, coded *state*, and cubie positions p' after transformation by rotation R.

(γ', β')	$(0, 0)$	$(90°, 0)$	$(180°, 0)$	$(270°, 0)$	$(0, 90°)$	$(0, 270°)$
$\alpha' = 0$	$(0, 0, 0)$	$(90°, 0, 0)$	$(180°, 0, 0)$	$(270°, 0, 0)$	$(0, 90°, 0)$	$(0, 270°, 0)$
	I	R_z^1	R_z^2	R_z^3	R_Y^1	R_Y^3
R	$\begin{bmatrix} + & 0 & 0 \\ 0 & + & 0 \\ 0 & 0 & + \end{bmatrix}$	$\begin{bmatrix} 0 & - & 0 \\ + & 0 & 0 \\ 0 & 0 & + \end{bmatrix}$	$\begin{bmatrix} - & 0 & 0 \\ 0 & - & 0 \\ 0 & 0 & + \end{bmatrix}$	$\begin{bmatrix} 0 & + & 0 \\ - & 0 & 0 \\ 0 & 0 & + \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & + \\ 0 & + & 0 \\ - & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & - \\ 0 & + & 0 \\ + & 0 & 0 \end{bmatrix}$
state	00000000	01010000	01100000	01110000	01000100	01001100
p'	$[x, y, z]$	$[-y, x, z]$	$[-x,-y, z]$	$[y,-x, z]$	$[z, y,-x]$	$[-z, y, x]$
$\alpha' = 90°$	$(0, 0, 90°)$	$(90°, 0, 90°)$	$(180°, 0, 90°)$	$(270°, 0, 90°)$	$(0, 90°, 90°)$	$(0, 270°, 90°)$
	R_X^1	$R_z^1 R_X^1$	$R_z^2 R_X^1$	$R_z^3 R_X^1$	$R_Y^1 R_X^1$	$R_Y^3 R_X^1$
R	$\begin{bmatrix} + & 0 & 0 \\ 0 & 0 & - \\ 0 & + & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & + \\ + & 0 & 0 \\ 0 & + & 0 \end{bmatrix}$	$\begin{bmatrix} - & 0 & 0 \\ 0 & 0 & + \\ 0 & + & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & - \\ - & 0 & 0 \\ 0 & + & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & + & 0 \\ 0 & 0 & - \\ - & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & - & 0 \\ 0 & 0 & - \\ + & 0 & 0 \end{bmatrix}$
state	01000001	10010001	10100001	10110001	10000101	10001101
p'	$[x,-z, y]$	$[z, x, y]$	$[-x, z, y]$	$[-z,-x, y]$	$[y,-z,-x]$	$[-y,-z, x]$
$\alpha' = 180°$	$(0, 0, 180°)$	$(90°, 0, 180°)$	$(180°, 0, 180°)$	$(270°, 0, 180°)$	$(0, 90°, 180°)$	$(0, 270°, 180°)$
	R_X^2	$R_Z^1 R_X^2$	$R_z^2 R_X^2 = R_Y^2$	$R_z^3 R_X^2$	$R_Y^1 R_X^2$	$R_Y^3 R_X^2$
R	$\begin{bmatrix} + & 0 & 0 \\ 0 & - & 0 \\ 0 & 0 & - \end{bmatrix}$	$\begin{bmatrix} 0 & + & 0 \\ + & 0 & 0 \\ 0 & 0 & - \end{bmatrix}$	$\begin{bmatrix} - & 0 & 0 \\ 0 & + & 0 \\ 0 & 0 & - \end{bmatrix}$	$\begin{bmatrix} 0 & - & 0 \\ - & 0 & 0 \\ 0 & 0 & - \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & - \\ 0 & - & 0 \\ - & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & + \\ 0 & - & 0 \\ + & 0 & 0 \end{bmatrix}$
state	01000010	10010010	01001000	10110010	10000110	10001110
p'	$[x,-y,-z]$	$[y, x,-z]$	$[-x, y,-z]$	$[-y,-x,-z]$	$[-z,-y,-x]$	$[z,-y, x]$
$\alpha' = 270°$	$(0, 0, 270°)$	$(90°, 0, 270°)$	$(180°, 0, 270°)$	$(270°, 0, 270°)$	$(0, 90°, 270°)$	$(0, 270°, 270°)$
	R_X^3	$R_z^1 R_X^3$	$R_z^2 R_X^3$	$R_z^3 R_X^3$	$R_Y^1 R_X^3$	$R_Y^3 R_X^3$
R	$\begin{bmatrix} + & 0 & 0 \\ 0 & 0 & + \\ 0 & - & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & - \\ + & 0 & 0 \\ 0 & - & 0 \end{bmatrix}$	$\begin{bmatrix} - & 0 & 0 \\ 0 & 0 & - \\ 0 & - & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & + \\ - & 0 & 0 \\ 0 & - & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & - & 0 \\ 0 & 0 & + \\ - & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & + & 0 \\ 0 & 0 & + \\ + & 0 & 0 \end{bmatrix}$
state	01000011	10010011	10100011	10110011	10000111	10001111
p'	$[x, z,-y]$	$[-z, x,-y]$	$[-x,-z,-y]$	$[z,-x,-y]$	$[-y, z,-x]$	$[y, z, x]$

Table 2 presents possible cubie orientations in the form of its rotation matrix. For any cube configuration, this matrix can be recreated on the basis of a 3×3 submatrix of cubie's affine transformations—without taking into account its position (translation). Because each cubie is also scaled by positive valued cubieScale (Figure 2), only meaningful signs of matrix elements are presented.

The cube configuration can be saved as the orientation table of all cubies. Due to the small number of orientations of a single cubie (i.e., 24), they can be encoded as alphanumeric characters and the cube configuration as a string. This makes it easy to compare different cube configurations.

For a cubie $p = [x, y, z]$, the remaining cubies in the cluster are p' cubies with coordinates Rp given in Table 2. Solving the cube corresponds to the situation when for all cubies their rotation matrices are identity matrices—if the cubie is correctly oriented, it must be in the correct position.

Since the orientation of the inner cubies is important and the arrangement of the cubies begins from its center, all cubies are separated from each other by a certain constant spacing and scrambled cubies are presented as partially transparent (Figure 2).

3. Solution

3.1. Algorithm. Cube movements were defined by means of three parameters: rotation *axis* X, Y, and Z [0 : 2], index of the *slice* [0:N-1], and the number of 90° slice rotations *angle*

```
state := function GetState(R)
  γ = GetAngle(R[1,0], R[0,0]);
  if (γ = 0) then
      β = GetAngle(-R[2,0], R[0,0]);
      α = GetAngle(-R[1,2], R[1,1]);
  else
      β = 0;
      α = GetAngle(R[2,1], R[2,2]);
  endif
  moveCount := sign(α) + sign(β) + sign(γ);
  state := moveCount * 64 + γ * 16 + β * 4 + α;
  if α = 2 and γ = 2 state := 72; endif
endfunction
```

Figure 3: Definition of the cubie's orientation state (pseudocode).

[0 : 2]. The number of different possible moves is therefore equal to 9N. Movements are coded to the values of linear indices in the range [0 : 9N-1] and thus represented as genes in the chromosome containing the solution:

$$\text{move}(\text{slice}, \text{axis}, \text{angle}) = 9\text{slice} + 3\text{axis} + \text{angle}. \tag{14}$$

The gene values can then also be easily decoded into these 3 parameters, determining the movement.

The cube configuration can be represented as a block vector $\mathbb{p}$, elements of which are the cubies rotation (orientation) matrices. For a single movement, permutation of the cubie p in the block vector can then be described as

$$\mathbb{p}'(Mp) = M\mathbb{p}(p), \tag{15}$$

where the matrix M is the transformation of the rotation by the *angle* around the *axis* for cubies in the *slice*, and the identity matrix for the others:

$$M = \begin{cases} R_{\text{axis}}^{\text{angle}+1}, & \text{for } p(\text{axis}) = \text{slice}, \\ I, & \text{otherwise}. \end{cases} \tag{16}$$

Since all cubies in the *slice* segment are rotated (permutated) in the cube vector, the cubie's transformation can be determined based on the M transformation combined with the transformation stored in the cube vector at the position from inversed transformation performed on that position:

$$\mathbb{p}'(p) = M\mathbb{p}\left(M^{-1}p\right). \tag{17}$$

By saving the cube configuration as a vector, it is also possible to define a transformation that performs any permutation of the elements of this vector (cubies in a cube), composite with the M transformation, and present the movement of the cubie slice as a matrix operation:

$$\mathbb{p}' = \mathbb{R}_{\text{axis, slice}}^{\text{angle}+1}\mathbb{p}. \tag{18}$$

Matrix $\mathbb{R}$ has the size $N^3 \times N^3$, and its elements are the M matrices of the cubies. There is one nonzero M transformation at the position specified by undoing the M transformation at the original cubie position and converting it to a linear index in each row and column. Knowing the current cubies orientations, it is possible to construct such a matrix $\mathbb{R}$, which is a composite of all previous cubie's transformations and represents cube configuration (examples for $N = 3$ are presented in Figure 4). For example, for a cube with size $N = 2$ and unfolding to the vector $\mathbb{p} = [\mathbb{p}_{XYZ}]$, the single rotations of the slice = 0 around the Z and Y axes are represented by matrices (dots represent zero matrices):

$$\mathbb{R}_{Z,0} = \left[\begin{array}{cccc|cccc} \cdot & \cdot & R_Z & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & I & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & R_Z & \cdot \\ \cdot & \cdot & \cdot & I & \cdot & \cdot & \cdot & \cdot \\ \hline - & - & - & - & - & - & - & - \\ \hline R_Z & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & I & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & R_Z & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & I \end{array}\right],$$

$$\mathbb{R}_{Y,0} = \left[\begin{array}{cccc|cccc} \cdot & \cdot & \cdot & \cdot & R_Y & \cdot & \cdot & \cdot \\ R_Y & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & I & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & I & \cdot & \cdot & \cdot & \cdot \\ \hline - & - & - & - & - & - & - & - \\ \hline \cdot & \cdot & \cdot & \cdot & \cdot & R_Y & \cdot & \cdot \\ \cdot & R_Y & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & I & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & I \end{array}\right], \tag{19}$$

and commutator $\mathbb{K} = \mathbb{R}_{Y,0}^T\mathbb{R}_{Z,0}^T\mathbb{R}_{Y,0}\mathbb{R}_{Z,0}$:

$$\mathbb{K} = \left[\begin{array}{cccc|cccc} \cdot & \cdot & R_Z & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & I & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ R_X & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & I & \cdot & \cdot & \cdot & \cdot \\ \hline - & - & - & - & - & - & - & - \\ \hline \cdot & \cdot & \cdot & \cdot & \cdot & R_X & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & R_Y^T & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & I & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & I \end{array}\right]. \tag{20}$$

It follows that only 4 cubes changed their transformations, but they swapped their places $\mathbb{p}_{000} - \mathbb{p}_{010}$ and $\mathbb{p}_{100} - \mathbb{p}_{101}$ and are transformed by a single move. Executing a

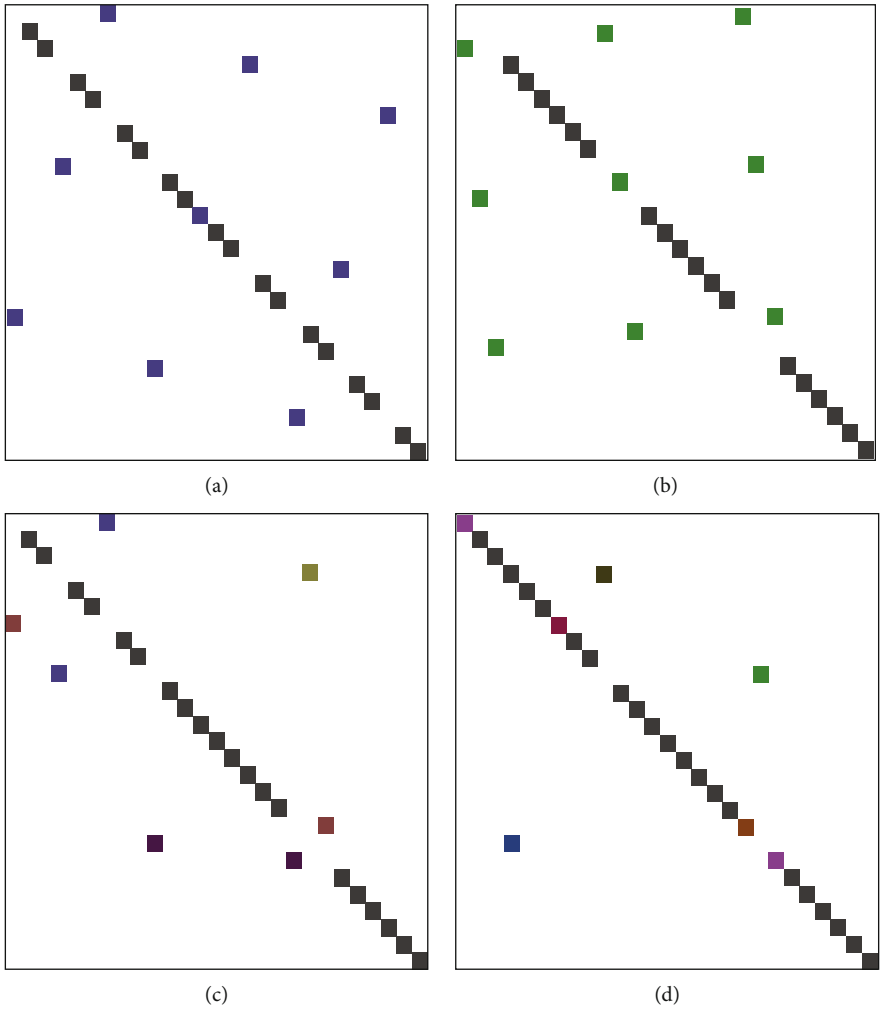

Figure 4: Examples of rotation matrices for a $3 \times 3 \times 3$ cube: (a) $\mathbb{R}_{Z,0}$; (b) $\mathbb{R}_{Y,0}$; (c) $\mathbb{K}$; (d) $\mathbb{K}^2$.

double commutator will result in the absence of a permutation of the positions of these cubies with composite transformation from 2 movements:

$$\mathbb{K}^2 = \text{diag}\left(R_Z R_X, I, R_X R_Z, I, R_X R_Y^T, R_Y^T R_X, I, I\right). \quad (21)$$

Thanks to this, the movements of commutators can be used to position the corner cubies of the cube (see also Figure 4).

The cube configuration changes through consecutive movements:

$$\mathbb{p}' = \mathbb{R}_{\text{move}_k} \cdots \mathbb{R}_{\text{move}_2} \mathbb{R}_{\text{move}_1} \mathbb{p} = \mathbb{R}\mathbb{p}. \quad (22)$$

Finding the movements that reverse these transformations, that is, solving the cube, is related to the decomposition of the matrix $\mathbb{R}^T$ into the sequence of the cube movements:

$$\mathbb{p} = \mathbb{R}^T \mathbb{p}' = \mathbb{R}_{\text{move}_1}{}^T \mathbb{R}_{\text{move}_2}{}^T \cdots \mathbb{R}_{\text{move}_k}{}^T \mathbb{p}'. \quad (23)$$

Each of these transformations is orthogonal to a given segment. As in the one-side Jacobi transform used to derive the SVD of a matrix by means of a $Q^T R Q$ transformation to a similar matrix (maintaining the same eigenvalues), such a sequence in the case of a cube transformation, called rotation conjugate of R, also changes only a limited number of elements. Presented algorithm will also use a strategy similar to Gaussian elimination or "chasing zero" in QR iterations, by solving the cube, cubie by cubie in a specific order, so as not to cause deterioration of the existing configuration.

The number of possible configurations of a cube with N segments can be estimated, by assuming that each cubie can be in any orientation, on

$$c = 24^{N^3}. \quad (24)$$

This number therefore increases exponentially very quickly, and the chromosome evaluation function must take into account not only the orientation of the N^3 cubies themselves but also their position, changed as a result of rotation.

Due to the possibility of repeating the same cubie position coordinates during rotation, there are 6 types of

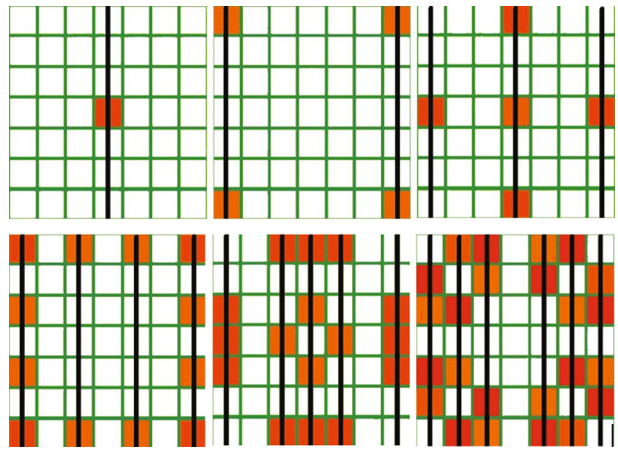

Figure 5: Six types of clusters $C_k k = 1..6$ defined by the number of segments (black lines) that affect those clusters (orange cubies are father away in this projection).

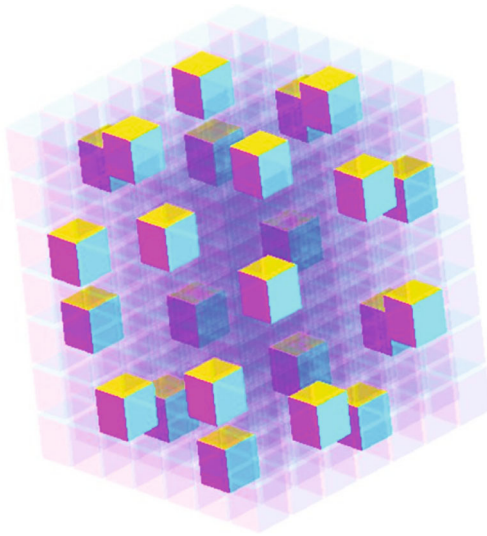

Figure 6: Typical cluster, containing 24 cubies.

clusters. They differ in the number of segments that can affect the position of cubies of a given cluster type. For each axis and selected angle, the index of the segment influencing the cubie in the cluster can take the value of the cubie coordinate $p^{'}$ after transformation, i.e., the cubie coordinate x, y, z of p or the opposite number (Table 2), which gives a maximum of 6 options: slice $= (x, N-1-x, y, N-1-y, z, N-1-z)$.

Apart from the possibility of changing the *axis* and the *angle* of rotation for each segment, the type of the C_k cluster can be determined precisely by the number of $k = 1..6$ segments affecting it. Figure 5 shows all possible cubies positions from given C_k cluster as red squares in a transparent cube. Vertical lines mark segments for chosen *axis*. If we try to arrange a cubie or cubies only from a given cluster, then the values of genes in the chromosome can be restricted to movements that affect these cubies. The number of such movements for any cube size, taking into account the possibility of choosing one of the three axes and one of the three angles, is 9k (max. 54).

Clusters C_2 consist of 8 corner cubies for cube or for inner cubes. For odd-sized cubes, the C_1 cluster constitutes one, middle cube, and the C_3 clusters consist of 6 cubies located in the middle of the cube segments (except for the middle segments) or 12 corner cubies for the middle segments of the cube or inner cubes. The remaining types of clusters contain maximal number of cubies, i.e., 24 (presented in Figure 6). Even N size cubes do not contain odd cluster types.

A set of cubies in a cluster can be created on the basis of a selected cubie p by applying to it any possible rotation R and obtaining positions (or orientations) of cubies p', which belong to the same cluster (Table 2 and Figure 7).

Based on restricted *slice* indexes and any possible value of *axis* and *angle*, list of restricted moves is created. It serves as a gene pool for creating initial population of chromosomes in genetic algorithm. The chosen genetic operators will not violate this restrictions, and whole algorithm will run on restricted list of acceptable moves. Implementation example of function retrieving restricted list of moves is presented in Figure 8.

Because in the definition of movement, as the size of a cube N increases, only the number of segments increases; the number of possible moves for a cube should be limited by blocking the movements of specific slices. Additionally, when the slice is rotated, cubies do not change their distance to the cube center. Movements of the slices in planes that are more distant from the center than r do not affect the position of cubies in planes less distant. Thus, these cubies can be arranged only by means of slice movements that are distant from the center by less than r. The adopted strategy solves cube by orienting cubies one by one, preserving orientation of the cubies previously arranged, whereby selected (active) unoriented cubie is always that, which is closest to the middle of the inner cube of the smallest possible size. This way, the algorithm will try to arrange the inner cubies

```
function cluster := GetActCluster(actCubie)
  v = [actCubie.X,actCubie.Y,actCubie.Z];
  for α = 0:3
    for β = 0:3
      for γ = 0:3
        p := v − C
        p′ := R_Z^γ R_Y^β R_X^α · p
        v′ := p′ + C
        cubie = Cubies[v′.X, v′.Y, v′.Z];
        if (not cubie in cluster)
          cluster.append(cubie);
        endif
      endfor
    endfor
  endfor
endfunction
```

Figure 7: Implementation of function retrieving actual cluster (containing active cubie) (pseudocode).

```
function freeGenes := GetFreeGenes(actCubie)
  v := [actCubie.X,actCubie.Y,actCubie.Z];
  for axis = 0:2
    for coord = 0:2
      for side = 0:1
        if (side = 0)
          slice = v[coord];
        else
          slice = N − 1 - v[coord];
        endif
        gene = 9 * slice + 3 * axis;
        if (not gene in freeGenes)
          for angle = 0:2
            freeGenes.append(gene + angle);
          endfor
        endif
      endfor
    endfor
  endfor
endfunction
```

Figure 8: Implementation of function retrieving restricted list of genes (moves) influencing cubies in active cluster (pseudocode).

from inside to the outside of the cube. The algorithm tries to correctly orientate active cubie by limiting the movements to the movements influencing its cluster. By arranging the cube from inside, one can gradually limit the movements of the interior slices. The last clusters of the cube (and also inner cubes) are always the clusters containing the corner cubies—most distant from the center. The movements of these clusters are restricted to only movements of the 6 outermost walls of a given cube. The consecutive stages of solving $5 \times 5 \times 5$ supercube are presented in Figure 9.

The whole process of solving the $3 \times 3 \times 3$ cube, in form of ever changing configuration matrix, is presented in Figure 10. White squares represent zero matrices, and black ones represent identity matrices. Identity block matrix is a configuration of a solved cube.

3.2. Fitness Function. A chromosome containing a sequence of cube movements may contain movements that can be combined into a single movement or that are completely self-canceling. Prior to the evaluation of the chromosome, a correction procedure can be performed that combines the movements of the same slice of the cube into one movement if they are not separated by movements in different planes, or to remove both genes when movements are cancelling each other.

The fitness function evaluates the cube's alignment status for each subsequence of movements in the chromosome starting with the first move. The first movement on a chromosome should change the orientation of the active cubie, changing the number of the movement's *slice* to the segment containing the active cubie, taking into account the axis of rotation specified in the gene. This will increase the probability of finding the active cubie orienting sequence.

First, a copy of the cube is created, which will be subject to movements written in the chromosome. The chromosome genes are processed sequentially. The gene is decoded to the movement and performed on a copy of the cube, which is then evaluated. The sequence of movements is successively supplemented with successive movements defined in the genes of the chromosome, creating an ever longer substring all the way to the entire chromosome. The length of the substring with the best score is remembered, and its score becomes the overall chromosome score. In this way, the possibility of finding solutions with a different number of movements using chromosomes of a fixed length has been effectively implemented.

The cube is assessed on the basis of the sum of the assessments of individual cubies belonging to the selected set of cubies. This collection consists of two groups. Group A consists of all cubies belonging to solved clusters. Position of cubies belonging to this group cannot be changed by the algorithm; therefore, they gain a weight greater than the sum of the assessment of all cubies from group B, which are cubies belonging to the current active cubie cluster. The value of 0 is the best fitness of the cube arrangement and individual cubies. The fitness of the $X = \mathrm{A}$ or B group is the sum of the scores of nonoriented cubies, each in the range [0, 2) and defined as

$$f(X) = \sum_{X_i.\mathrm{state} \neq 0} 1 + X_i.\mathrm{state}/255, \tag{25}$$

where X_i denotes orientation of the i-th cubie, encoded as its state (see equation (13) and Table 2) in A or B group.

Cube fitness error is a combination of both group fitnesses:

$$\mathrm{error} = f(A)\ \max\left(f(B)\right) + f(B), \tag{26}$$

where the maximum value of group B's fitness is the double number of cubies in this group (according to equation (25)).

$$\max\left(f(B)\right) = 2 \bullet \mathrm{count}(B). \tag{27}$$

By taking into account the exact orientations of the cubies, the fitness function causes the genetic algorithm to prefer placing cubies in a certain order, always bringing the

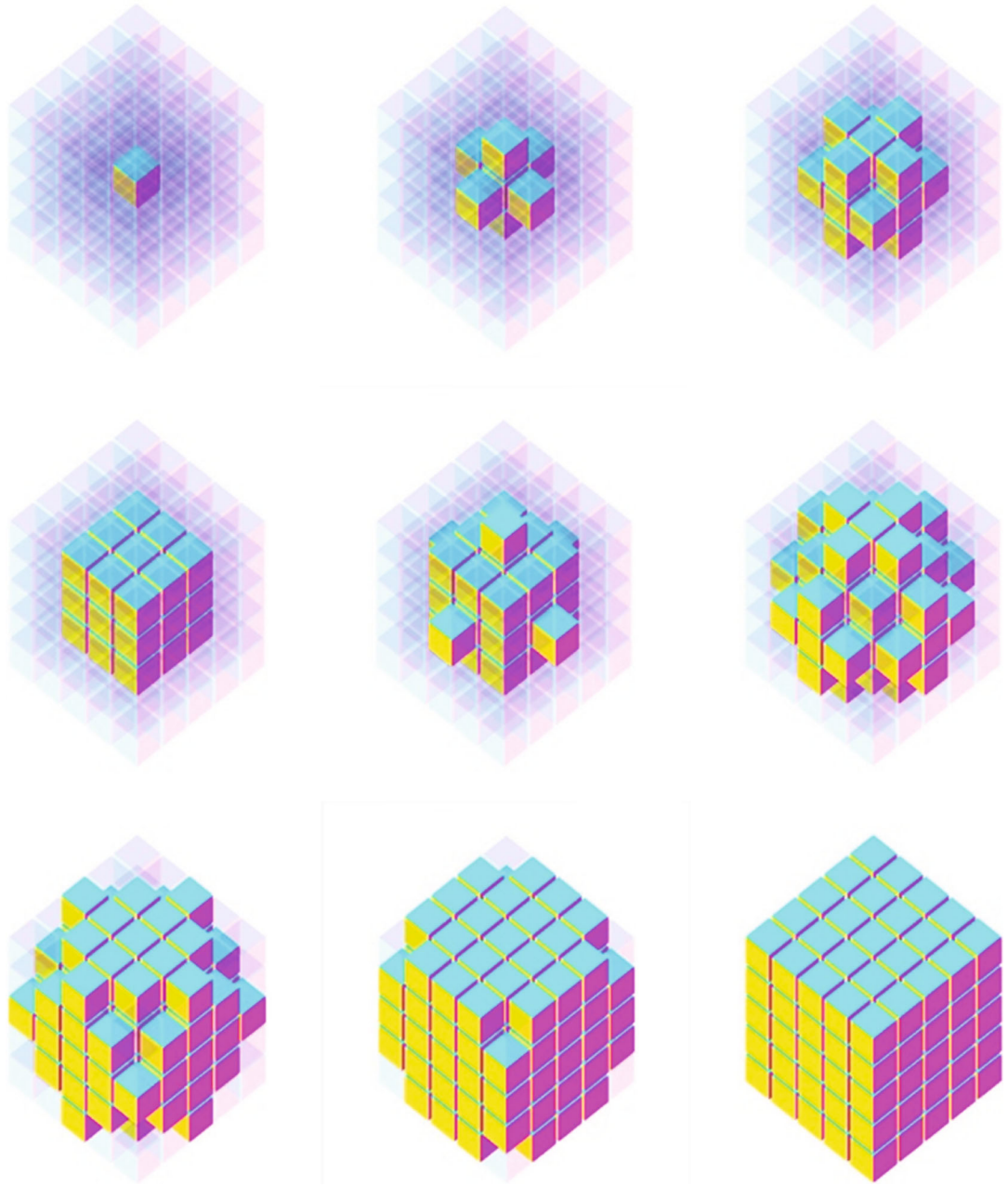

Figure 9: Stages of solving $5 \times 5 \times 5$ cube.

cube to similar configurations and reducing the number of cube layouts to a limited number with individual cubies not in their final position. The number of such configurations for k nonoriented cubies in the entire cube is less than

$$c_k = 24^k \binom{N^3}{k}. \tag{28}$$

After arranging cubies from clusters in groups A and B, the value of the cube fitness function drops to zero. A new active cubie is designated. If the cube is not yet solved, then such a cubie exists and it is located at a distance from the center of the cube greater than the cubies in groups A and B. The group B is then joined with A, and the cubies from the active cubie cluster form a new group B. This ends the cube building stage, and the solution that led to it, if not a sequence derived from a saved solution, is stored in the solution file and can be used to speed up the algorithm. The cube's fitness is set to its maximum value, and another genetic algorithm is executed.

3.3. Selection, Crossover, and Mutation. An important step in the genetic algorithm is selection. In the case of a Rubik's cube, the chromosomes will often differ in genes at distant positions in the chromosome, which will not affect the value of its fitness function. When selecting winners, a kind of ranking selection can be used, which omits individuals with

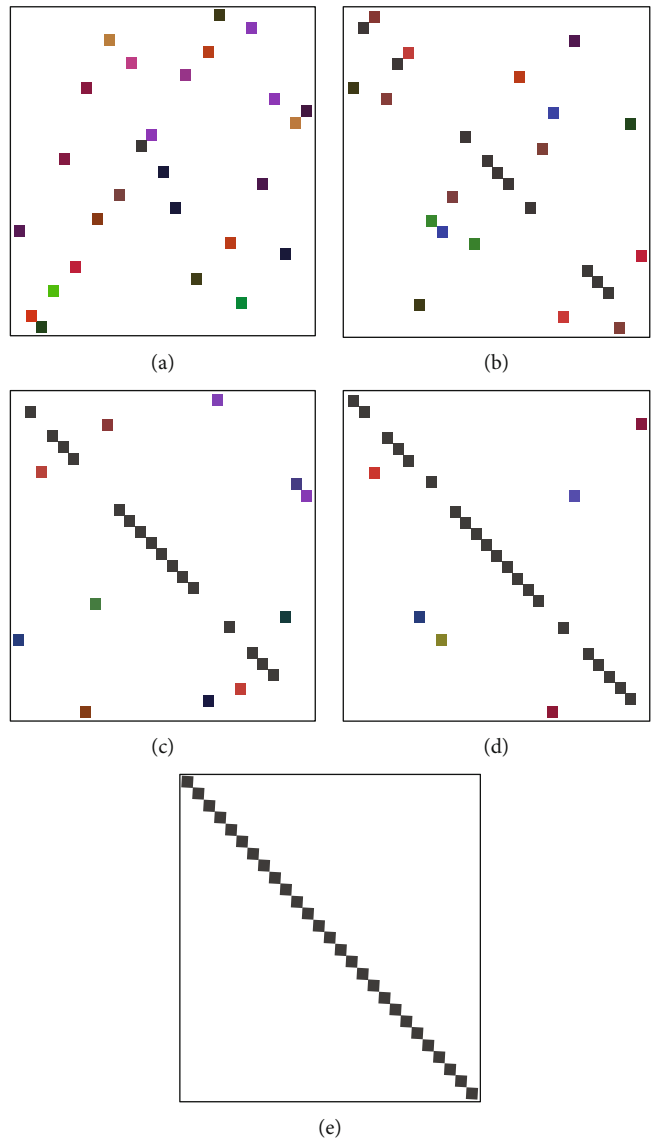

Figure 10: Example of solving $3 \times 3 \times 3$ cube presented in block matrix notation for GA iteration: 0 (a), 4 (b), 8 (c), 12 (d), and 16 (e).

Table 3: Number of used genetic algorithms with 100 macros.

Size	$2 \times 2 \times 2$	$3 \times 3 \times 3$	$4 \times 4 \times 4$	$5 \times 5 \times 5$	$6 \times 6 \times 6$
GA1	2	31	99	130	364
GA2	2	21	88	190	459
GA3	4	17	85	175	405
GA4	3	17	44	280	727
GA5	4	16	46	144	493
GA6	3	18	95	134	545
GA7	2	16	84	256	836
GA8	3	13	69	225	776
GA9	2	14	66	184	507
GA10	3	14	112	166	493

the value of the fitness function the same as the already selected winner. The advantage of this approach is also low computational complexity.

The most important thing in the algorithm, however, is the use of efficient genetic operators. While the crossover may be a simple one-point recombination of chromosomes, the mutation should allow finding a solution for a cube in a configuration of only a few cubies with wrong orientation, i.e., macros. Such R' sequences are in the form of a commutator or a conjugation in which the matrix Q can be composed of several movements around different axes:

$$R' = QR_{\text{axis}}Q^{-1}. \tag{29}$$

By limiting the movements to the movements that change the active cubie's cluster, it is much easier to find the sequence improving the cube's configuration in accordance with the fitness function. If the R_{axis} performs rotation on a slice more distant from the center of the cube than the inner slices of the already oriented cube, then it is an identity transformation for the cubies of the inner cube. This means

Table 4: Number of used genetic algorithms with 200 macros.

Size	2 × 2 × 2	3 × 3 × 3	4 × 4 × 4	5 × 5 × 5	6 × 6 × 6
GA1	3	16	55	124	369
GA2	4	15	62	192	304
GA3	3	13	102	289	711
GA4	4	19	87	172	441
GA5	3	55	72	116	487
GA6	3	15	65	178	428
GA7	3	16	88	114	494
GA8	3	17	87	241	490
GA9	3	20	69	87	507
GA10	4	15	67	141	445

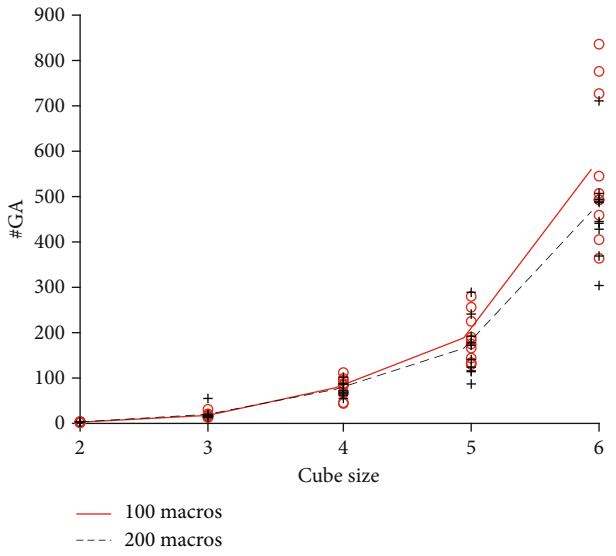

Figure 11: Function of GA count vs. cube's size.

Table 5: Calculation time (s) with 100 macros.

Size	2 × 2 × 2	3 × 3 × 3	4 × 4 × 4	5 × 5 × 5	6 × 6 × 6
GA1	1	29	204	464	2000
GA2	1	18	181	655	2559
GA3	1	14	169	616	2279
GA4	1	12	72	1025	4181
GA5	1	10	80	481	2789
GA6	1	13	198	425	2933
GA7	1	11	177	897	4755
GA8	1	9	135	776	4400
GA9	1	9	132	657	3089
GA10	1	8	227	580	3205

that also the entire R' transformation is an identity transformation for these cubies and does not change their arrangement.

Table 6: Calculation time (s) with 200 macros.

Size	2 × 2 × 2	3 × 3 × 3	4 × 4 × 4	5 × 5 × 5	6 × 6 × 6
GA1	0	8	61	261	1252
GA2	0	6	74	389	1105
GA3	0	6	115	550	2193
GA4	1	9	98	345	1461
GA5	0	32	88	243	1608
GA6	1	7	74	368	1412
GA7	0	7	101	258	1639
GA8	1	6	98	478	1652
GA9	1	10	77	176	1668
GA10	0	7	77	304	1481

The proposed mutation procedure performs the following actions:

(1) Random selection of an *idx* less than half-length of the chromosome

(2) Genes are processed from the start of the chromosome to $idx - 1$, and inverse transformations to them are coded in genes from index $2 \bullet idx$ down to $idx + 1$

Due to the high efficiency of finding macro sequences, the mutation rate in the algorithm can be extremely high, within 100% of the population size.

3.4. Learning. The sequences found by the genetic algorithm that end with the arrangement of all cubies from the assessed set cause the change of position of only a few, $k \ll N^3$ cubies, not changing orientations of the rest. These are the key sequences that can be used not only in the last stage of solving the cube. The algorithm can remember such sequences, implementing the ability to learn and arrange the cube faster and faster.

The memorized sequences are used each time a better cube configuration is found to further improve it. The movements of these sequences are restricted to a certain cluster (maximum 6 segments) and can be mapped to active cubie cluster movements by replacing segments with active cluster segments. This enables, among others, using the saved movements from smaller cubes, not only to orient the inner cubes but for all clusters of the same type in the cube.

Movements that, after composing, can be represented in the form of an orthogonal matrix R can be additionally complemented by a conjugation to the form $Q^{-1}RQ$, in which case the matrix Q is a single rotation, selected optimally among all possible movements of the active cluster. This will enable a much wider use of memorized movements, taking into account similar patterns resulting from the cube symmetry.

4. Results

The way the algorithm works can be summarized in four main points:

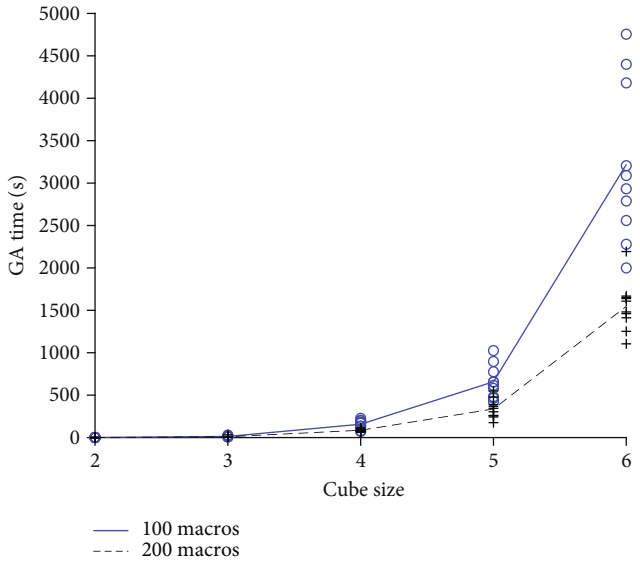

Figure 12: Function of calculation time vs. cube's size.

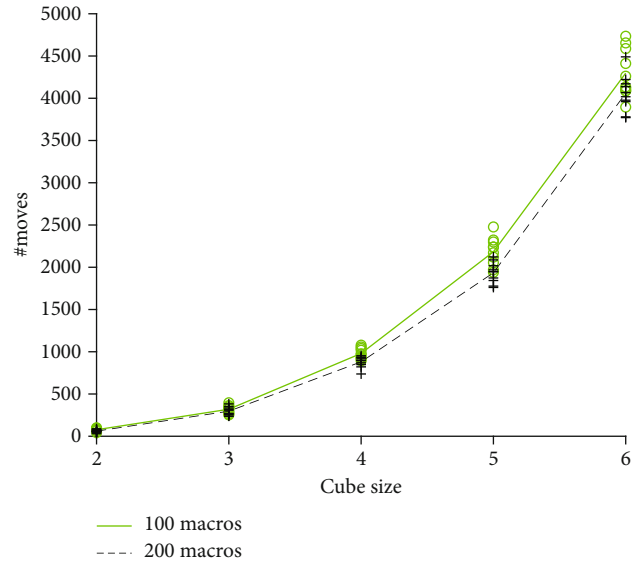

Figure 13: Solution length vs. cube's size.

Table 7: Solution lengths with 100 macros.

Size	2 × 2 × 2	3 × 3 × 3	4 × 4 × 4	5 × 5 × 5	6 × 6 × 6
GA1	75	396	1078	1970	4588
GA2	70	350	913	2239	4089
GA3	100	329	1018	2171	4134
GA4	87	333	935	2299	4116
GA5	87	325	926	2477	4261
GA6	82	363	1056	2239	4088
GA7	43	301	1038	2322	4655
GA8	76	253	976	2054	4734
GA9	68	282	967	1940	4412
GA10	82	265	893	2120	3896

Table 8: Solution lengths with 200 macros.

Size	2 × 2 × 2	3 × 3 × 3	4 × 4 × 4	5 × 5 × 5	6 × 6 × 6
GA1	51	308	737	1948	4137
GA2	59	270	898	2122	3977
GA3	55	238	934	2087	3780
GA4	89	383	917	1761	3960
GA5	76	260	848	1777	4490
GA6	30	334	895	1948	4018
GA7	58	263	953	2018	4222
GA8	76	238	821	1873	4171
GA9	47	353	869	1843	4067
GA10	67	318	934	1969	3771

(1) The algorithm arranges one cubie at a time, selecting as an active cubie that, which is closest to its middle and contained in the inner cube of the smallest possible size

(2) Gene values in chromosomes (movements) are limited to segments that influence the current cubie cluster, i.e., the current cluster

(3) Only cubies belonging to the current cluster are assessed. Cubies from oriented clusters cannot change orientation

(4) The algorithm has the ability to learn macros, which are the sequence of movements that complete orientation of the current cluster. Macros are mapped to the movements of the current cluster by changing segment indexes to segment indexes of the current cluster

The results of the algorithm's operation on full cubes with sizes from 2 to 6 are presented. Each cube was scrambled with a 100 random moves before the algorithm was started.

The parameters of the algorithm assumed the length of the chromosome equal to 27, and the number of selection winners was equal to 10% of the population, the population itself at the level of 100 individuals, and the number of generations at the level of 50 iterations, due to the attempt to find only the cube configuration better than the one found so far. For the duration of the experiment, the ability of the algorithm to learn was suspended. The algorithm has had already known and used 100 or 200 macro sequences. Tables 3 and 4 (Figure 11) and Tables 5 and 6 (Figure 12) show the number of genetic algorithms performed and the time it took to complete the cube in 10 consecutive experiments.

The total number of analyzed moves equals approximately to the product of number of genetic algorithms, population count, length of the chromosome, and number of generations (in our case, $135E3 \bullet \#GA$), and that is why we tried to minimize number of genetic algorithms (and not number of solution moves).

Tables 7 and 8 and Figure 13 show the number of moves, which led to a solution. Some of them could stand from learned macros.

5. Conclusions

The structure of the Full Rubik's Supercube differs from the classic supercube [11]—and the existing algorithms (reduction and cage) and macromovements, which are used to solve it, may cause changes in the orientation of the inner cubies. This makes the puzzle much more difficult, and comparing the times or the number of moves needed to solve a full and classic supercube is not directly possible for $N > 3$. For $N = 3$, a full supercube differs from a supercube only in the need to orient one inner cubie. Comparison of the results, for example with the algorithms presented by El-Sourani et al. [9], shows a greater number of moves necessary to solve a cube in our algorithm, but much shorter times to find a solution (much more important when solving larger cubes). To our knowledge, the Full Rubik's Supercube, from the point of view of its solving, has not yet been analyzed. However, it is certainly possible to determine the strategy and sequences of macromoves for full supercube as well. This task is well fulfilled by the methodology presented in the article, which uses a sequence of genetic algorithms in combination with a simple technique of learning detected macros.

Presented method deals with a generic problem of solving Full Rubik's Supercube of any size, taking into account orientation of the off-border and internal cubies. The cube is represented as a model of the composite transformations of the block rotation matrices around X, Y, and Z axes. The resulting cube configurations are therefore always correct, and it is not necessary to color map the faces of cubies and select correct permutations to determine their orientation. The article presents also a way of formally writing the cube configuration as a combination of moves (transformations) in the form of a block matrix. This type of notation can be helpful especially for the analysis of generalizations of the Rubik's cube in multidimensional spaces. In n -dimensional space, the orientation of the cubies is given by $\binom{n}{2}$ Euler angles. As the number of dimensions increases, the number of possible movements therefore also increases, due to the increasing number of possible planes of rotation. Future research will focus on the greater use of cube symmetry and the problem of solving a multidimensional cube.

As we mentioned, the problem of solving the cube can be formulated as a decomposition of configuration block rotation matrix into orthogonal move block matrices (see also equation (23)). Such analysis can be treated as a generalization of the SVD decomposition for higher dimensions [12], and similarities of the presented algorithm to this algorithms are therefore highlighted. Solving strategy, which arranges cubies in the current cluster, one by one, allows for strong restriction of the possible moves (gene values in GA), which depends on the type of cluster, but is indifferent to the size of the cube (number of segments). The method also does not use prior expert knowledge of the existing macromoves until it finds them on its own.

The results of all experiments were given for algorithms with a small number of learned macromoves (100 and 200) and the learning module turned off, in order to demonstrate the ability of the system to learn quickly and to properly compare the times of solving cubes of different sizes.

The project available in the repository was written in C# language with the use of OpenGL functions and without the use of other programming libraries. It has the characteristics of a game, because it is very satisfying to watch how the algorithm learns to solve a full cube faster and faster. It can be easily supplemented with a learning module for manual supercube solving, based on commutator sequences learned by the system. Cubies with the wrong orientations can be rendered semitransparent, revealing internal structure of a Full Supercube.

Rubik's cube can fascinate, but it is not only a toy; it is also used in data encryption [13–16], search algorithms in 3D structures, or solving mechanical problems [17–22]. It is an inspiration to build scientific analogies in the theory of groups, permutations, cycles [23], but also in 3D graphics. Like fractals, multidimensional regular polyhedra, or block tensors, it is a gateway to new dimensions and discovering rules and beauty in mathematics.

Data Availability

The code and datasets are available in the GitHub repository (https://github.com/robertswita/RubikCube).

Disclosure

A preprint has previously been published [24].

Conflicts of Interest

The authors declare that there is no conflict of interest.

Authors' Contributions

RŚ prepared the concept, wrote the manuscript and code, and conducted the experiments. ZS adapted the genetic algorithms for solving supercube.

References

[1] M. Thistlethwaite, *The 52 Move Strategy*, 1981, http://www.jaapsch.net/puzzles/thistle.htm.

[2] H. Kociemba, "The Two-Phase-Algorithm," http://kociemba.org/twophase.htm.

[3] R. E. Korf, "Finding Optimal Solutions to Rubik's Cube Using Pattern Databases," in *AAAI-97 Proceedings*, pp. 700–705, Providence, Rhode Island, USA, July 1997.

[4] T. Rokicki, H. Kociemba, M. Davidson, and J. Dethridge, "*The diameter of the Rubik's cube group is twenty*," *SIAM Journal on Discrete Mathematics*, vol. 27, no. 2, pp. 1082–1105, 2013.

[5] E. Demaine, M. Demaine, S. Eisenstat, A. Lubiw, and A. Winslow, "Algorithms for solving Rubik's cubes," in

Algorithms – ESA 2011, C. Demetrescu and M. M. Halldórsson, Eds., vol. 6942 of Lecture Notes in Computer Science, pp. 689–700, Springer, Berlin, Heidelberg, 2011.

[6] M. Herdy and G. Patone, *Evolution Strategy in Action, 10 ES-Demonstrations*, Technical Report, International Conference on Evolutionary Computation, 1994.

[7] E. B. Baum and I. Durdanovic, "Evolution of cooperative problem-solving in an artificial economy," *Neural Computation*, vol. 12, no. 12, pp. 2743–2775, 2000.

[8] F. Agostinelli, S. McAleer, A. Shmakov, and P. Baldi, "Solving the Rubik's cube with deep reinforcement learning and search," *Nature Machine Intelligence*, vol. 1, no. 8, pp. 356–363, 2019.

[9] N. El-Sourani, S. Hauke, and M. Borschbach, "An evolutionary approach for solving the Rubik's cube incorporating exact methods," in *Applications of Evolutionary Computation. EvoApplications 2010*, vol. 6024 of Lecture Notes in Computer Science, pp. 80–89, Springer, Berlin, Heidelberg, 2010.

[10] M. Borschbach and C. Grelle, *Empirical Benchmarks of a Genetic Algorithm Incorporating Human Strategies*, Technical Report, University of Applied Sciences, Bergisch Gladbach, 2009.

[11] K. Fraser, "Implementing and solving Rubik's family cubes with marked centres," http://kenblackbox.com/cube/solving/markctr.pdf.

[12] A. Ishida, N. Yamamoto, J. Murakami, and N. Oishi, "Solving 3-D puzzles using tensor decomposition and application to education of multidimensional data analysis," *International Journal of Machine Learning and Computing*, vol. 8, no. 5, pp. 447–453, 2018.

[13] K. Loukhaoukha, J.-Y. Chouinard, and A. Berdai, "A secure image encryption algorithm based on Rubik's cube principle," *Journal of Electrical and Computer Engineering*, vol. 2012, Article ID 173931, 13 pages, 2012.

[14] T. T. Anusree and K. P. Swaraj, "*Rubik's cube encryption for securing cloud stored data*," in *Second International Conference on Computer Networks and Communication Technologies. ICCNCT 2019*, S. Smys, T. Senjyu, and P. Lafata, Eds., vol. 44 of Lecture Notes on Data Engineering and Communications Technologies, Springer, Cham, 2020.

[15] R. Vidhya and M. Brindha, "A chaos based image encryption algorithm using Rubik's cube and prime factorization process (CIERPF)," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 5, pp. 2000–2016, 2022.

[16] M. Balkrishna Salunke, P. Narendra Mahalle, and G. R. Shinde, "Rubik's cube encryption algorithm-based technique for information hiding during data transmission in sensor-based networks," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 10, no. 1s, pp. 429–439, 2022.

[17] S. McAleer, F. Agostinelli, A. Shmakov, and P. Baldi, "Solving the Rubik's cube without human knowledge," 2018, https://arxiv.org/abs/1805.07470.

[18] V. Mnih, A. P. Badia, M. Mirza et al., "Asynchronous methods for deep reinforcement learning," in *Proceedings of The 33rd International Conference on Machine Learning*, pp. 1928–1937, New York, NY, USA, June 2016.

[19] G. Tesauro, "Temporal difference learning and TD-gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[20] V. Mnih, K. Kavukcuoglu, D. Silver et al., "*Human-level control through deep reinforcement learning*," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[21] D. Silver, J. Schrittwieser, K. Simonyan et al., "*Mastering the game of Go without human knowledge*," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[22] D. Silver, T. Hubert, J. Schrittwieser et al., "Mastering chess and Shogi by self-play with a general reinforcement learning algorithm," 2017, https://arxiv.org/abs/1712.01815.

[23] W. D. Joyner, "Mathematics of the Rubik's cube," https://www.fuw.edu.pl/~konieczn/RubikCube.pdf.

[24] R. Świta and Z. Suszyński, "Solving Full NxNxN Rubik's Supercube using Genetic Algorithm," 2022.