Empirical Benchmarks of a Genetic Algorithm Incorporating Human Strategies

Markus Borschbach and Christian Grelle
Faculty of Computer Science
University of Applied Sciences
51465 Bergisch Gladbach, Germany
++492202 9527369
Markus.Borschbach@fhdw.de

Technical Report no. 2009/01
Faculty of Computer Science, University of Applied Sciences, Bergisch Gladbach, Germany

ABSTRACT
This work outlines the incorporation of human strategies in a genetic algorithm. Human competence and machine intelligence are merged creating symbiotic human-machine intelligence, which is called HuGO!, the Human strategy based Genetic Optimizer. HuGO! emerged from and is applied to the restoration problem of Rubik's Cube and successfully solves this task. A competition between HuGO! and human Rubik's Cube contestants demonstrates that the incorporated human strategies improved the genetic solver's performance to become human-competitive.

1. INTRODUCTION
The simulation of human intelligence is a central goal of artificial intelligence. Genetic algorithms are admittedly nature inspired optimization techniques, however basically ignoring human problem solving strategies. This work illustrates the incorporation of human strategies in a genetic algorithm and therefore introduces a method of collaboration and knowledge exchange. Human competence and machine intelligence are merged creating symbiotic human-machine intelligence, which is called HuGO!, the Human strategy based Genetic Optimizer.

As use case, a both simply working and complex mathematics invoking application is analyzed. Rubik's Cube is widely known and allows a vivid problem embodiment. This three-dimensional puzzle has undergone comprehensive research in the past. The results are used to develop a thoroughly formal problem description and solve the task of exploiting human knowledge.

There are five distinct sections. Section 2 introduces Rubik's Cube. Beginning with the historical roots, the cube's physical structure is explained and specific terminology is introduced to allow clear communication throughout the work. The section comprises an insight into the mathematical group theory and treats certain subgroups of the cube that are important to realize a human strategy. Belonging to the survey of related work are scientific achievements regarding optimal solution sequences for a scrambled cube. Corresponding work on upper and lower bounds for necessary turn numbers is presented completed with an overview on existing evolutionary solving approaches.

From the introduced fundamentals of Rubik's Cube, Section 3 derives the idea of HuGO! and describes the developed algorithm in detail.

Section 4 comprises tests on the introduced algorithm. First, integrity tests provide information on the correctness of the results. Then performance tests demonstrate the efficiency of HuGO!. The dramaturgical climax is reached with a comparison of the algorithm's and real human capabilities to solve the cube that indicates the human competitiveness of HuGO!.

The work is brought to a close with the conclusion of Section 5 where results are summarized, problems are reflected and an outlook is given including suggestions for further research as well as potential application areas.

2. RUBIK'S CUBE

2.1 History
ERNO RUBIK, a Hungarian architect and professor at the University of Budapest presented the first prototype of Rubik's Cube in 1974. RUBIK developed the cube as a teaching aid for recognizing three-dimensional spatial relationships. Already in the beginnings of its history, the cube became interesting for scientific research. The first significant public attention outside of Hungary was caused by the mathematician DAVID SINGMASTER. He conducted analyzes of the cube mathematics, which led to an article in *Scientific American* by DOUGLAS HOFSTADTER in 1979.[9] Today, Rubik's Cube as discrete optimization problem is a testing ground for scientific questions. It allows researchers from different disciplines to compare their methods on a single, well-known and vivid problem. Popular application areas are mathematical group theory (Section 2.3) or search and enumeration, incorporating disciplines like artificial intelligence.[14]

2.2 Structure and terminology
Rubik's Cube consists of 26 smaller pieces, which are called *cubies* (see Figure 1). There are three different types of cubies. Eight corner cubies are located in the corners of the cube. They have three visible surfaces that are called *facelets*. 12 cubies have two facelets. They fill in the space along an edge between two corner cubies and are therefore called edge cubies. The third type of cubie only has one facelet. These cubies are located in the center of each side (*face*) of the cube and are consequently called center cubies. There are six center cubies whose facelet colors determine the cube face.[22] In the standard Rubik's Cube the

possible colors are white, yellow, orange, red, green and blue, whereas distinct center cubie color pairs are always on opposite faces: white and yellow, orange and red as well as green and blue.[23] Each of the 6 faces of the entire cube is made up of nine facelets. Thus there are $6 \cdot 9 = 54$ facelets on the cube.

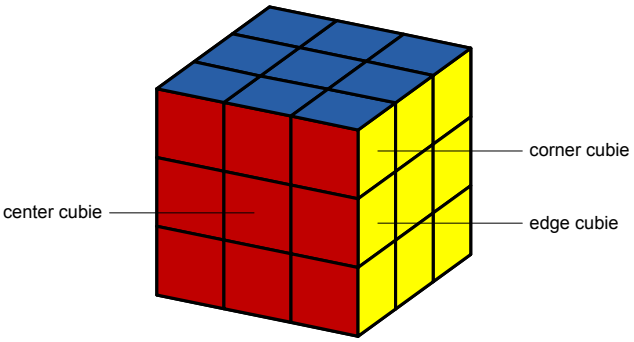


Figure 1. Structure of Rubik's Cube

By rotating different faces of the cube, the cubies can be moved. Each cubie of the turned face, except the center cubie, moves to a location vacated by another cubie. These locations are called *cubicles*. No matter how faces are rotated, corner cubies always move from one corner cubicle to another corner cubicle. Edge cubies move from one edge cubicle to another edge cubicle. Center cubies have a fixed location relative to the other center cubies. They only can be spun in place.[5]

Besides, the center cubie of each face determines the only color to which this face can be restored. Therefore it is possible to define the only cubicle in which each cubie can be placed to restore the cube. For example, if the two facelets of an edge cubie are red and yellow, then that cubie must be placed in the unique edge cubicle between the red center cubicle and the yellow center cubicle (see Figure 1). Furthermore, the cubie must be placed in that cubicle so that its red facelet is next to the red center cubie and the yellow facelet is next to the yellow center cubie. Similarly, it is possible to determine the corresponding corner cubicles of each corner cubie, except that there are 3 facelets to consider.

Since different Rubik's Cube manufacturers use different colors, each of the faces is named based on its position relative to the person holding the cube.[23] The six faces have the names *Front*, *Up*, *Right*, *Back*, *Down* and *Left*. These faces are designated by their initials:

Table 1. Faces abbreviations

Face	Abbreviation
Front	F
Up	U
Right	R
Back	B
Down	D

On each rotation exactly 20 facelets are moved. Rotations are described by the face initials (see Table 1)

F, U, R, B, D and L.

A single initial indicates a *clockwise quarter turn* of the corresponding face while viewing the face from that side of the cube. Figure 2 provides a graphical demonstration of all possible clockwise quarter turns. A *half turn* of any face is two quarter turns of that face. The following notation is used:

F2, U2, R2, B2, D2 and L2.

Counter-clockwise quarter turns are denoted by

F', U', R', B', D' and L'.[1][5]

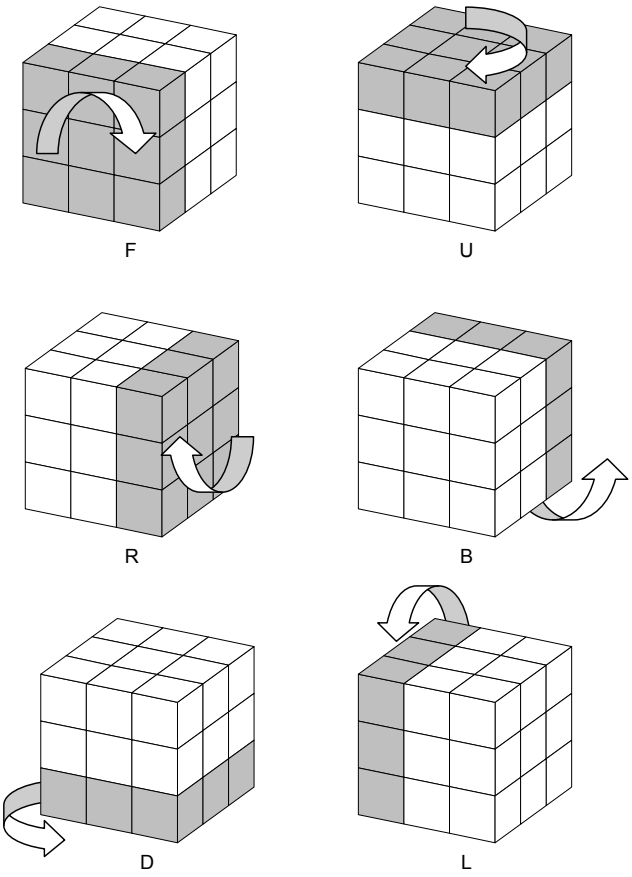


Figure 2. Clockwise face turns

To describe a sequence of turns, the turns are listed from left to right. For example, FR' means apply F first and then apply R'. Any sequence of turns is called a *process*.[23]

2.3 Group Theory

Each process on the cube generates a permutation of the cubies. Additionally, if one process is followed by another, the processes form a new process that generates another permutation. This is the first requirement for a *group*. Group theory is the mathematical foundation of the study of symmetry. The concept has many applications in art, physics, chemistry and biology. It is a way to study the solvability of polynomial equations and the structure of geometric and topological objects. So group theory is one of the basic subjects of mathematics. On Rubik's Cube, the processes and permutations form groups. Since they are vividly embodied on the cube, it is often used as a concrete example of group theory concepts.[5]

---

[1] The presented notations X, X2 and X' are type-friendly modifications of the original intentions X, X^2 and X^{-1}, whereas the exponent denotes direction and number of turns.

There are several subgroups of the cube group like the two-squares group, the slice group or the two-generator group.[2] The two-generator is an important group in the progress of this work and shall therefore be explained. The cube group is the group of all states that are accessible by using all faces of the cube for rotations. The total number of cube permutations, which denotes the order of the cube group, is $4.3 \cdot 10^{19}$. The two-generator, however, is the group of all states that can be accessed by just turning two adjacent faces of the cube, for example, U and B. Significant characteristics of the two-generator are that edge cubies cannot be changed in orientation, i.e. flips are not possible, and that corner cubies are incapable of ceteris paribus swapping two corner cubies. The total number of permutations and therefore the order of the two-generator group is $73,483,200$, which is significantly lower than the order of the cube group.

2.4 Survey of Related Work

2.4.1 An optimal Solution

In 1992, the mathematician HERBERT KOCIEMBA developed CubeExplorer, a program incorporating an algorithm that, after some improvements, is expected to calculate a shortest solution to any scrambled cube provided.[10] Therefore the tools of group theory were used, which can simplify the calculations by defining groups of different cube configurations that share mathematical properties.

2.4.2 An upper bound for the worst case

One of the most fundamental questions about Rubik's Cube is the question after the number of turns necessary to solve it in the worst case. Even after more than 30 years since its introduction, the answer to this question remains unknown.

However, several approaches have been used to find an upper bound for the worst case. These approaches are gradually shifting the upper bound on the diameter of the cube group closer towards the expected number of 20 turns. Combined with the estimates of a lower bound, which has been shown to be at least 20 turns[16] in a configuration called *superflip*, this allows to determine the real number. Basically, most upper bound approaches are based on defining a suitable way-station configuration and then optimally solving it. After working out how many turns it takes at least to get to the way station from any random configuration, the sequence lengths of both ways having the way station in between are summed up to receive the solution for an upper bound. Rather than using a single configuration, it is often more efficient to exploit symmetries when dealing with several way stations in a single calculation.

MORWEN THISTLEWAITE presented an algorithm traversing four specifically defined groups that requires a maximum of 52 turns in 1981.[25] KOCIEMBA's algorithm (Section 2.4.1) is actually an improvement of THISTLEWAITE's algorithm and was used by MICHAEL REID to show that 29 turns suffice to solve the cube.[17] In 2007, SILVIU RADU reduced the upper bound to 27 by generalizing REID's method.[15] In the same year, DANIEL KUNKLE and GENE COOPERMAN could reduce the upper bound even further to 26 turns.[11] They devised a way to construct 1.5 trillion groups called cosets of about 660,000 configurations each. Regarding symmetries, they identified 15,000 unique configurations in each coset and determined the maximum distance to the solved cube. Besides, they introduced a method to transform all coset configurations into one of the 15,000. So solving just one of the 15,000 configurations equals solving the whole coset. The calculations required seven terabytes of computer memory and 8,000 hours processing time before the upper bound could be lowered down to 26. In 2008, TOMAS ROKICKI devised a computational proof that all unsolved cubes can be solved in 25 turns or fewer.[18] Using the same algorithm, but more computational capacity, this was later reduced to 23 turns.[19] In August 2008, ROKICKI announced that calculations are brought forward to the current best found upper bound of 22 turns.[20]

ROKICKI's algorithm works by dividing up the problem into two billion cosets, each containing around 20 billion related configurations. The program then works through one coset at a time, building a list of the turns that bring each of its 20 billion configurations into one of those in KOCIEMBA's subgroup used in CubeExplorer, until all of the configurations in the coset have been solved at least once. The longest sequence found by the program is the upper bound for the whole coset. The calculations are highly capacity-intensive and were conducted on a computer grid of Sony Pictures Imageworks. Theoretically, on even faster computers, ROKICKI could try to reach lower worst case boundaries with his algorithm. These computers like Blue gene/L, based at the Lawrence Livermore National Laboratory in California, however, are not affordable to ROKICKI so that at the present time, he is trying to improve the algorithm.[13][18]

2.4.3 Evolutionary approaches

In 1994, MICHAEL HERDY and GIANNINO PATONE solved the cube using evolution strategies.[8] Therefore, they introduced a quality function for the evaluation of the cube state. The quality function to be minimized consists of three parts, Q1, Q2 and Q3, combined by addition. Q1 is increased for a wrong facelet while Q2 and Q3 penalize wrong positioned edge- and corner cubies. The 10 different mutations are realized using swaps and turns of individual cubies. On the one hand this allows a rapid solution search because dependencies are minimized. But on the other hand the results will be fairly long solution sequences, because accomplishing a single swap already incorporates around 10 cube rotations. Since HuGO! is supposed to find short solutions, HERDY and PATONE's approach is inappropriate for this work's goals. Admittedly, CYRIL CASTELLA built an evolutionary approach to solve the cube, aiming short solution sequences, that is much more convenient to be exploited.[3] The program uses a genetic algorithm, which is based on a one-point crossover, omitting a selection operator and mating pool. Unfortunately, the approach suffers from missing integrity as yet so that no performance comparison could be conducted. More precisely, the solution output seems to fail if the cube is already in the two-generator group. However, it incorporates some useful functions that are enhanced and applied by HuGO! (Section 3.2).

[2] For details on the two-squares group and the slice group cf. [5], p. 112 ff.

3. HUGO!

3.1 Boundary Conditions

The task to solve Rubik's Cube can be considered as a very special optimization task. In ordinary optimization problems that are appropriate to be solved by genetic algorithms, the goal is to reach a comparatively good solution after a certain termination condition is fulfilled. In contrast to this, the cube optimization process can only be finished, if all facelets are fully restored, i.e. the cube is solved. However, a cube solving turn sequence can be differentiated in fitness by means of turn numbers. Characterizing, the cube optimization is a discrete optimization problem. Discrete optimization, also called integer programming, can be described by the usage of restricted variables in the objective function as being exclusively receptive for discrete values. So the examined problem is of discrete nature, because the determinants of the cube fitness are face turns, which cannot be conducted partially.

The most obvious way to solve the cube by evolutionary means is to use a trivial fitness function that compares the scrambled cube to the solved one and counts the number of consistent facelets. Tests showed that this approach turns out to be inefficient. While the algorithm quickly reaches around 70% consistent facelets, further calculations only provide marginal improvements. This is due to $4.3 \cdot 10^{19}$ interdependent permutations as possible states of the cube group, which lead to an enormously jagged fitness landscape containing lots of local optima. Consequently, the algorithm repeatedly becomes trapped in local optima, extending calculation time tremendously.

The integration of a common human solving strategy for Rubik's Cube, called two-generator method, provides a solution to this problem. The two-generator method is inefficient for fast solution generating, but often used by contestants of fewest moves challenges. This method restores a scrambled cube by transforming it into the two-generator subgroup first and then solving the cube in this group. As described in Section 2.3, the two-generator subgroup encompasses a number of different states that is only about 73,483,200 and therefore much smaller than the entire cube group. Thus, it is promising to split up the cube-solving search algorithm into part solutions.

In the first phase a 2x2x3 subcube, i.e. the entire cube except two adjacent layers, is solved in one of the twelve possible locations. The two remaining layers are left scrambled. Since not all cube states that only have two adjacent layers unsolved are automatically in the two-generator, the second phase transforms the cube into this group. For this purpose tests are necessary that check, whether the edge cubie orientations and corner cubie permutations stick to the states achievable in the two-generator. In the third phase, the remaining two layers are turned until the entire 3x3x3 cube is completely restored. The three phases of the algorithm realizing this human strategy are three independent algorithms that are based on the common canonical genetic algorithm and share a lot of analogies. The individual components, as well as their interaction in the composed workflow are explained in detail in the succeeding sections.

3.2 Components

3.2.1 Solution Representation

According to these suggestions, the discrete characteristics of Rubik's Cube as optimization process allow a rather direct incorporation of the search space into the solution representation. All three phases of the algorithm have in common that instead of binary strings, each gene of a solution contains a number ranging from 0 to 17 representing one of the 18 potential cube turns introduced in Section 2.2.[3] Tab. 4.1 depicts the exact allocations used:

Table 2. Turn allocations

Clockwise quarter turns		Half turns		Counter-clockwise quarter turns	
F	0	F2	6	F'	12
U	1	U2	7	U'	13
R	2	R2	8	R'	14
B	3	B2	9	B'	15
D	4	D2	10	D'	16
L	5	L2	11	L'	17

It becomes obvious that the search space equals the representation space, except that the growth function maps numbers to the literal turn abbreviations. In this way genes do not need to be merged to decision variables that they already are. Fig. 4.4 illustrates an example of the resulting genotypical solution representation:

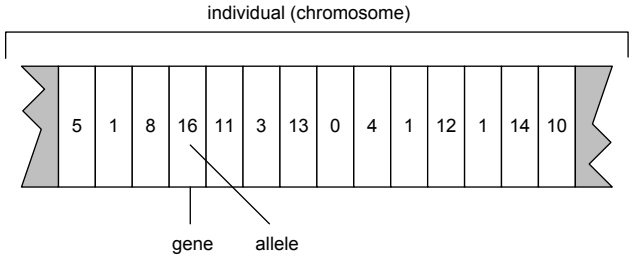


Figure 3. A coded representation of an individual

The alleles of the solution compose a process as defined in Section 2.2. Equally, substrings of the process can be seen as potential solutions. If l determines the length of an individual, then this solution representation leads to a search space of 18^l possibilities in every phase. Section 3.2.2 describes in detail how evaluation of the solution representation is proceeded. Deviating approaches regarding turn allocations excluding half turns or using binary representations are still to be compared in further investigations.

3.2.2 Fitness Function

To successfully apply a GA to a given optimization problem, an adequate representation of the problem must be developed. In the case of Rubik's Cube, the scrambled cube represents the environment and the solution string represents the individual that is customized to fit to the environment. The fitness function assigns a fitness value to the individual judging the quality of adaptation, i.e. ability to solve the cube. To test this ability, the particular solution operates on the scrambled cube and the resulting cube undergoes evaluations that determine the individual's fitness. It becomes clear that the implementation of the cube has to fulfill two main capabilities. First it must be receptive to the individual's modifications, i.e. turns. Then the

resulting cubes must be distinguishable in quality, respecting the number of turns applied.

The cube representation, as introduced in Section 2.2, allows a color independent modification and identification of individual cubies and therefore supports the two mentioned conditions. To simplify implementation, the facelets of the cubies are labeled with integer numbers.[3] Since there are 54 facelets on the cube, the numbers from 0 to 53 are used to identify them. Allocated row by row on the faces in the order F, U, R, B, D, L, the entire cube is encompassed. Fig. 4.5 shows the internal representation of the cube:

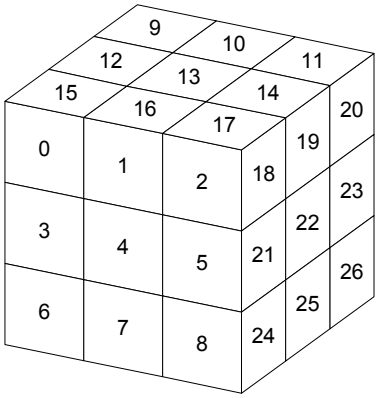


Figure 4. Internal representation of Rubik's Cube

During evaluation, the individuals are traversed successively from gene to gene (see Figure 3) while on each step the substring's (first to current gene) fitness is determined. The best fitness and the step number, i.e. number of turns, are stored. The overall fitness of the potential solution is determined by calculating the fitness and subtracting the number of needed steps. The fitness describes the progress of restoring the cube. The counted number of steps allows to differentiate between longer and shorter turn sequences of cube states of the same fitness. The distinct determination of the best fitness is different in each algorithm phase due to different goals of the phases.[3]

In the first phase, the *solve 2x2x3 cube phase*, the fitness is determined by counting the number of facelet pairs of the same color on the 2x2x3 surface. Particularly, the pairs are corner-edge pairs and center-edge pairs as displayed in Figure 5:

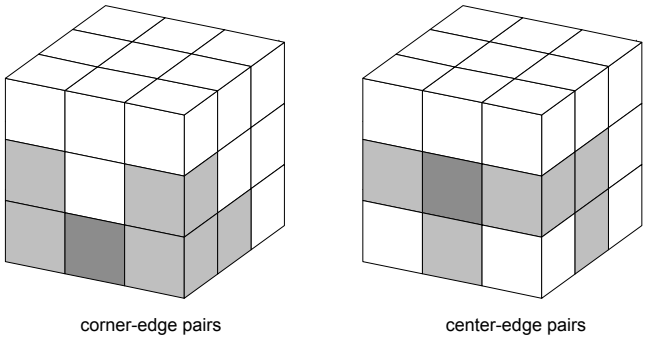


Figure 5. Corner-edge pairs and center-edge pairs on the 2x2x3 surface

A *corner-edge pair* is the conformance of the corner cubie facelets and the adjacent edge cubie facelets of a corner cubie and an edge cubie. Thus there exist six corner-edge pairs on the 2x2x3 surface. A *center-edge pair* is the conformance of a center cubie facelet and an edge cubie facelet. There are 10 center-edge pairs on the 2x2x3 surface, adding up to a total number of 16 pairs that constitute the maximum fitness value when the entire 2x2x3 subcube is solved. If the two cubes of Figure 5 are placed one upon the other, it becomes obvious that the pair technique encompasses the entire 2x2x3 surface and no facelet is ignored.[3]

In the second phase, the *transformation to two-generator phase*, fitness is determined by the number of corner-edge pairs and center-edge pairs as in the first phase, as well as the fact, whether the cube is in the two-generator group. The check for the pairs is done because in this phase, the cube is allowed to temporary leave the solved 2x2x3 state. Only in this way it is possible to manipulate the two not yet solved layers beyond the turns allowed in the two-generator to enter the two-generator subgroup. So the maximum fitness value achievable is 17 and signifies that the cube is successfully transformed to the two-generator group. To determine, whether the cube is in the two-generator, the cubies that are in the edge cubicles and in the corner cubicles of the not yet solved layers are requested. The center cubies are not directly involved in this transformation process because their position is fixed a priori.

The third phase, the *solve two-generator phase*, only allows turns of the two still scrambled sides, so that the cube remains in the two-generator subgroup.

This phase performs the integrity determination, which equals the fitness value, for the entire cube surface. Integrity is always at least 16 because the already solved 2x2x3 cube stays unaffected. The maximum value the integrity can achieve is 48 pairs, which means that the two-generator is solved and the cube is completely restored.

3.2.3 Selection

As selection operator, stochastic universal sampling is used to propagate individuals to the mating pool. The selection operator incorporates two selection points to reduce spread. The best solution is maintained as elite. Elitist strategies link the lifetime of individuals to their fitness. They are techniques to keep good solutions in the population for longer than one generation. The use turns search more exploitative rather than explorative. Elitist strategies are guessed to be necessary when genetic algorithms are used as function optimizers and the goal is to find a global optimal solution as it is the case in the cube optimization process regarding integrity restoration.[21]

3.2.4 Crossover

The genetic operations in the reproduction stadium are slightly adapted standard operators. The crossover operator is realized by a uniform crossover, eliminating positional bias. Table 3 depicts an exemplary application of the uniform crossover operator during the algorithm execution with two children. As explained in Section 3.2.1 and shown in Figure 3, the gene values are, in contrast to binary values, composed of numbers 0 to 17:

Table 3. Application of uniform crossover during algorithm execution

Individuals 1 and 2																								
17	14	0	2	17	10	17	12	1	12	13	9	1	12	1	9	13	5	12	14	0	1	8	12	3	5
17	12	11	4	3	17	4	1	11	6	11	8	0	16	3	4	9	5	14	6	15	17	7	0	14	17
Random bit pattern																									
0	1	1	0	0	1	1	1	1	1	1	1	0	1	0	0	0	0	1	0	0	1	0	0	1	0
Individuals 2 and 3																									
17	12	11	2	17	17	4	1	11	6	11	8	1	16	1	9	13	5	14	14	0	17	8	12	14	5
17	14	0	4	3	10	17	12	1	12	13	9	0	12	3	4	9	5	12	6	15	1	7	0	3	17

In the implementation of HuGO!, for each crossover operation, a new random bit pattern is created.

3.2.5 Mutation

A standard mutation is performed after the crossover using the adjustable mutation probability p_m. Table 4 depicts an exemplary application of the mutation operator during the algorithm execution:

Table 4. Application of mutation during algorithm execution

Individual before mutation																									
17	14	0	4	3	10	17	12	1	12	13	9	0	12	3	4	9	5	12	6	15	1	7	0	3	17
Individual after mutation																									
17	14	0	8	3	10	17	12	1	12	13	9	0	12	3	4	9	5	12	6	15	1	7	0	3	17

At position 4 of the individual, a mutation occurs so that the gene value 4 is replaced by the gene value 8. In contrast to binary string individuals, where during mutation a gene value is just swapped from 0 to 1 or from 1 to 0, here the genes have values from 0 to 17. So if a gene is chosen to become mutated, the gene value will be replaced by a random integer number between 0 and 17.

3.2.6 Optimization

An additional genetic operator, proposed by CASTELLA, is the optimization operator.[3] The optimization operator eliminates redundant turns so that gene strings are reduced in length. For example, the sequence F R B B' D2 F' is reduced to F R D2 F' so that B B' is cut out by the optimization operator, because a B-turn followed by a B'-turn results in the same state of the cube as it has been before these two turns.

Keeping in mind the internal representation of turns outlined in Table 2, the optimization process is done for all consecutive pairs of turns.

3.3 WORKFLOW

3.3.1 Solve 2x2x3 cube phase

The three successive phases are genetic algorithms itself and in the end composed to HuGO!, the algorithm that completely restores Rubik's Cube.

In the first phase, a 2x2x3 subcube is solved. There are 12 edges on the entire cube, which constitute the 12 possible locations for 2x2x3 cubes. For this reason, 12 populations are created, one for each possible location. The populations work on their allocated 2x2x3 location one after the other until a population produces an individual that solves the 2x2x3 cube. So the algorithm starts with the selection of a 2x2x3 location. In fact, in the implementation, populations are working on the same spatial edge of the cube, whereas the cube itself is rotated by horizontal and vertical rotations so that all 2x2x3 locations can be treated successively. Then it is checked, whether the algorithm has already been working on this location beforehand. If so, no new population is created, but the already existing is loaded from the population storage. So the algorithm does not have to develop the individuals from initialization, but can work on with already advanced individuals as proposed by CASTELLA.[3] On the other hand, if it is the first run through this 2x2x3 location, a new population is created and the algorithm proceeds. The following steps consist of the operations presented in Section 3.2. The individuals are optimized, the fitness becomes evaluated by the fitness function and then the individuals are sorted by their fitness for the selection step. After selection, the genetic operators crossover and mutation are applied to the individuals. These operations are done until the number of set generations is reached. Then the current population is stored in the population storage so that it can be loaded in the next run on this location. Besides, it is tested, whether all 12 locations are processed. If not, the next location is selected and the operations are repeated. If all locations are processed, the solution containing the best fitness is selected and tested, whether it can solve the 2x2x3 cube. If it cannot solve the 2x3x3 cube, the locations are processed again, but if it can solve the 2x2x3 cube, the solve 2x2x3 cube phase stops and the second phase is allowed to start.

3.3.2 Transform to two-generator phase

The second phase is the *transform to two-generator phase*. The phase receives a solved 2x2x3 cube from the first phase and manipulates the two remaining unsolved layers so that they can be solved in the two-generator group. The phase starts by creating a population that contains the potential solutions and then traverses the same genetic operations as the first phase does. It differentiates from the 2x2x3 operations by the fitness evaluation that is handled according to the description of the second phase's fitness function in Section 3.2.2. The consecutive steps are optimization, fitness evaluation, sorting, selection, crossover and finally mutation. The steps are repeated until the set number of generations is reached. Then it is tested, whether the cube entered the two-generator. If it did not enter the two-generator, the algorithm is repeated for the set number of generations. If the cube entered the two-generator group, this phase is finished and the last phase can start.

3.3.3 Solve two-generator phase

The last phase is the *solve two-generator phase*, where the two remaining layers are rotated until the entire cube is totally restored. The solve two-generator algorithm is similar to the transform to two-generator algorithm. After creating a population, the algorithm traverses the operations optimization, fitness evaluation, sorting, selection, crossover and mutation until the predetermined number of generations is reached. The difference to the second phase lays in the fitness function, which evaluates the individuals regarding the integrity until a maximum value of 48 according to Section 3.2.2. The processing of generations is repeated until the two-generator is solved and the last phase stops.

3.3.4 Human strategy based Genetic Optimizer algorithm

The entire composed Human strategy based Genetic Optimizer is depicted in Figure 6. Three columns of workflows become obvious that represent the just explained three phases of the algorithm. Before each phase, the algorithm checks, whether the

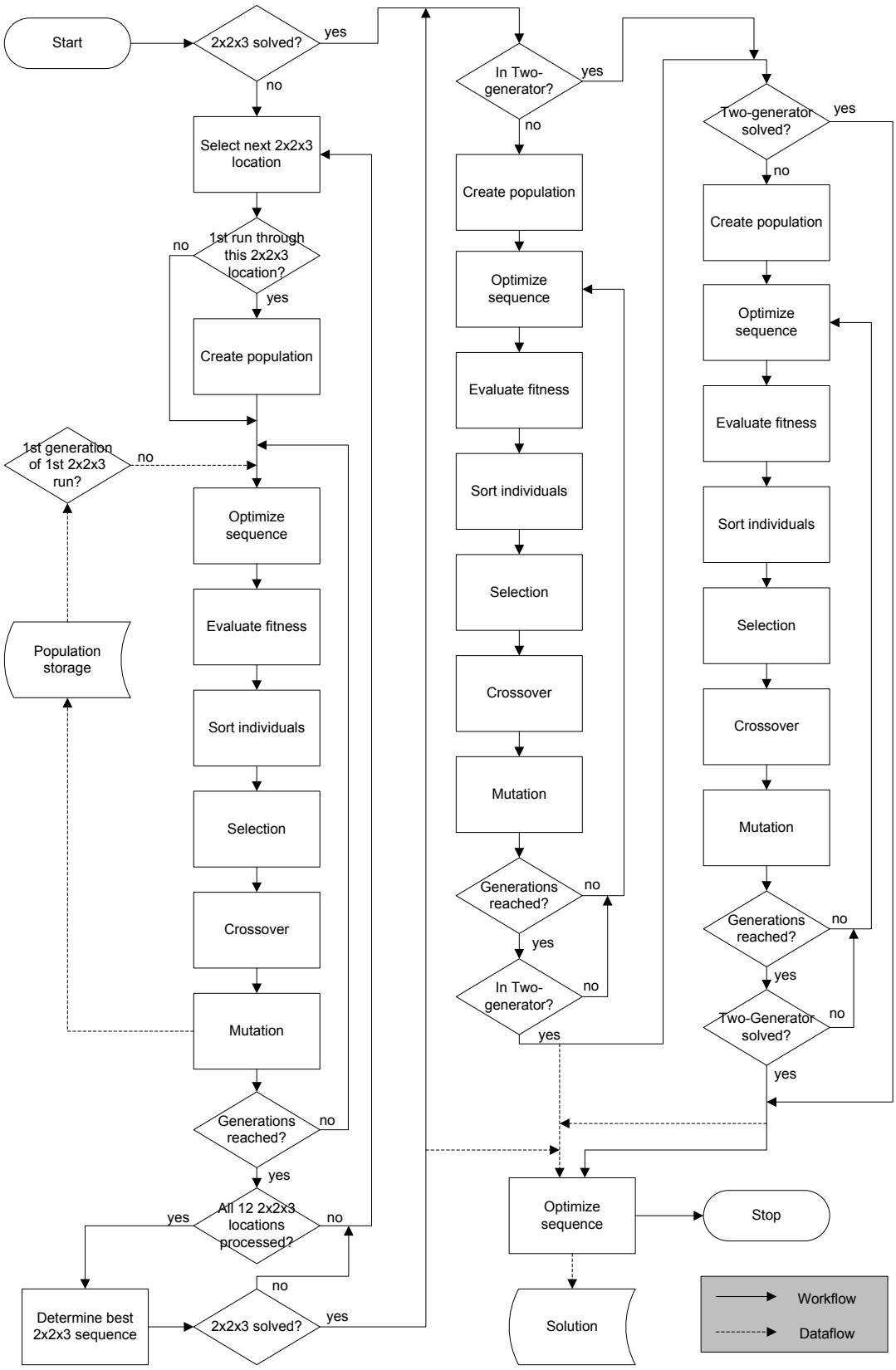


Figure 6. HuGO! algorithm

goal of the current phase is already reached and skips this phase if possible. The resulting turn sequence of each phase is composed to the solution sequence that restores the scrambled cube. Before put out, the solution sequence is treated by an adapted optimization operator to cut out redundant turns that occur during composition. This operator repeats optimization until no further abbreviation of the sequence is possible. For example, the process F R B2 B2 R' D needs two optimization steps for a satisfying abbreviation. In the first run, the sequence is transformed to F R R' D by cutting out the redundant turns B2 and B2. In the second run, the sequence is reduced to F D, because the newly succeeding turns R and R' are redundant as well. So the complete composed algorithm starts with a scrambled cube, passes the phases solve 2x2x3 cube, transform to two-generator as well as solve two-generator and stops when the cube is completely restored. The resulting individual turn sequences are composed and in the end, the optimized final solution can be displayed.

4. TEST RESULTS

4.1 Test Objectives and Resources

Three kinds of tests are performed to evaluate the characteristics of HuGO!. First, *integrity tests* provide answers to the question, whether the algorithm is able find a solution for all possible scrambles of Rubik's Cube. These extensive tests are also used to investigate the interior properties and interdependencies between the different phases to find appropriate parameter configurations. Then *performance tests* are conducted that give information about the efficiency of the algorithm. For this purpose, a measure called *complexity* is introduced with the help of the program CubeExplorer (Section 2.4.1) to classify a scrambled cube according to the difficulty of restoring it. For each complexity class, the algorithm's performance is worked out. The third kind of tests is the *HuGO!-human competition* that tests the algorithm's and human capabilities to solve the cube in the fewest turns possible. On the basis of data, evaluated from real human competition databases, a comparison is conducted that indicates the human competitiveness of HuGO!. All tests are carried out on comparable hardware, using Pentium 4 processors as well as one gigabyte of main memory.

Despite the best settings seem hard to find, some recommendations exist that suggest starting values for the search of good parameter settings. A population size of 50 individuals is chosen as suggested by DE JONG.[4] Besides, an individual size of 30 genes demonstrates to be sufficient to solve each tested scramble. Generation numbers vary from phase to phase due to different goals. The basic configuration is 10, 50 and 50 generations for phase 1 to 3. The 2x2x3 phase is assigned with fewer generations than the other phases because the restoration of the 2x2x3 cube shows to be the shortest part of the algorithm. If a phase is not finished after the set generations, the generation number automatically increases linearly for phase 1 and 2, as well as progressively for phase 3, which indicates to be the longest part. For crossover a standard probability p_c of 0.6 is chosen as suggested by GREFENSTETTE.[6] The mutation probability is calculated by $p_m = 1/l$ with l being the individual length. This treatment is recommended by BÄCK and leads to a value of around 0.03.[1][2][12]

4.2 Integrity Tests

It is not possible to test, whether the algorithm can solve all $4.3 \cdot 10^{19}$ different scrambles of the cube group in a brute force way due to calculation capability restrictions. So, some scramble sequences are chosen to be representative test scrambles. First, all scrambles containing three turns are tested. This includes $8^3 = 5832$ different turn sequences.

HuGO! provides a solution to all tested three-turns scrambles with an average solution length of 2.945 turns. Random samples also depict that the solutions are correct and restore the scrambled cube. The distribution of turn numbers to the phases is 0.723 to 0.202 to 2.050 before optimization. Consequently, phase 3 is the phase needing the most turns. Then a test with tripled generation numbers is conducted including 30, 150 and 150 generations. During this test, the average solution sequence length could be reduced to 2.872 turns. Regarding the test results, it becomes clear that a long solution for phase 2 results in an even longer solution for phase 3. This is due to the cubie mixing to transform the cube into the two-generator. The three initial turns to scramble the cube only have a limited influence on the cube's integrity so that it remains in a relatively ordered state. Many turns in the second phase, however, mix the cube further, even though the two-generator is reached. As a consequence, the generations for phase 2 are increased to 300. Phase 1 and 3 are reduced to 10 and 50 again, because the preceding improvement can be referred to the second phase's increase of generations and in this way, calculation speed is improved. The resulting average solution length using these settings is about 2.870 turns with a distribution of 0.724 to 0.193 to 1.992 turns before optimization.

Table 5 gives an overview about the average test results:

Table 5. Three-turns average results

Test	Min generations 2x2x3	Min generations 2GenTransform	Min generations 2GenSolve	Avg total turns	Avg turns 2x2x3	Avg turns 2GenTransform	Avg turns 2GenSolve	Avg total generations	Avg generations 2x2x3	Avg generations 2GenTransform	Avg generations 2GenSolve	Avg total time ms
1	10	50	50	2.945	0.723	0.202	2.050	2255	6	22	2227	12315
2	30	150	150	2.872	0.722	0.194	1.992	376	18	29	328	3141
3	10	300	50	2.870	0.724	0.193	1.992	506	6	39	462	3319

The derived parameter settings of the third test are the best found configuration for the three-turns tests and are therefore used for an integrity test of four turns. In this test, 6670 of the $8^4 = 104976$ different scrambles are calculated. This corresponds to 6.4% of all four-turns scrambles possible. The test shows that HuGO! provides a solution for all considered scrambles. Table 6 gives an overview about the average results obtained:

Table 6. Four-turns average results

Test	Min generations 2x2x3	Min generations 2GenTransform	Min generations 2GenSolve	Avg total turns	Avg turns 2x2x3	Avg turns 2GenTransform	Avg turns 2GenSolve	Avg total generations	Avg generations 2x2x3	Avg generations 2GenTransform	Avg generations 2GenSolve	Avg total time ms
1	20	300	50	4.924	1.237	0.724	3.042	13362	15	675	12672	106178

The resulting average solution length is 4.924 turns with a distribution of 1.237 to 0.724 to 3.042 turns in phase 1 to 3. 88% of the scrambles are solved in a maximum of four turns. The remaining 12% have an average solution length of 17.467 turns with the longest sequence being about 39 turns. Similarly, 84% of the scrambles are solved in a maximum of two seconds on the used hardware, whereas the left 16% need an average computing time of 11 minutes and 27 seconds, the longest duration being 11 hours, 29 minutes and 47 seconds. The exact overall time for the four-turns test is 210 hours, 11 minutes and 53 seconds for all 6670 scrambles to become solved. Thereby the 16% solutions of more than two seconds calculation time count for 99% of the overall time. Consequently the few long solutions constitute to a substantial part of the computing duration. One reason for outliers was found as a consequence of a certain state during restoration of the cube. If the complete cube is solved, except two opposite corner cubies that need to be swapped, the algorithm can need comparatively much time. This is due to the contrasting goals of exploitation and exploration in the restoring process. For two opposite corner cubies to become swapped, the almost solved cube must be severely decomposed. With the 6670 four-turns scrambles, the limit of calculation capacity for comprehensive integrity tests is reached, whereas all test scrambles could be solved. Scrambles of further lengths are tested as samples during the performance tests.

4.3 Performance Tests

The performance tests investigate the efficiency of HuGO!. To display the efficiency in terms of solution lengths, it is essential to differentiate between different difficulties of scrambled cubes. A slightly scrambled cube is likely to have a short solution while a heavily scrambled one should need more turns to become solved. To give a statement on the efficiency, the scrambled cube must be considered. A feasible way to gain information on the scrambled cube is to count the number of turns that led to the cube state. However, the number of turns is limited in information content. For example, a scramble sequence of 20 turns might need no turns as solution, if after 10 turns, the first 10 turns are inverted and repeated in reverse order. For this reason, a measure called *complexity* is introduced that does not consider the scramble sequence but orientates on the actual solution length. The actual solution length can be computed with a program generating optimal solutions to a given cube configuration. This is the case in CubeExplorer (Section 2.4.1). So, for the performance tests, the complexity of each cube configuration considered is derived from CubeExplorer beforehand. The resulting optimal solution length is used to develop classes. Since each cube should be solvable in up to 20 turns (Section 2.4.2), complexity has a range of 1 to 20, whereas any cube can be classified in one of the 20 complexity classes.

For each of the 20 classes, three random scrambles are solved 10 times resulting in 600 test runs that evaluate the performance of the algorithm. As generation numbers 200, 5000 and 50000 are used. This exceeds the number of generations used for the integrity tests, because in this case, the goal is to minimize the solution lengths. Additionally, in contrast to the integrity tests, the third phase is run with the highest amount of generations, because the third phase produces the longest part solutions and therefore has the highest potential for solution improvements as observed in Section 4.2.

To illustrate the test results, boxplot diagrams are used. Boxplots are an easy way to show differences in groups of data, i.e. the 20 classes. They indicate the degree of variability as well as asymmetry in the data and identify outliers. So trends and the limits of acceptable data are revealed. The graphical illustrations are the smallest observation, the lower quartile (25% of data), the median (50% of data), the higher quartile (75% of data) and the largest observation. The range between the lower and the higher quartile is called interquartile range (IQR). The smallest and largest observations are at the ends of the whiskers, which have a maximum size of 1.5 times the IQR. All data exceeding the smallest or largest observations are outliers and marked in the diagram as separate points.[24]

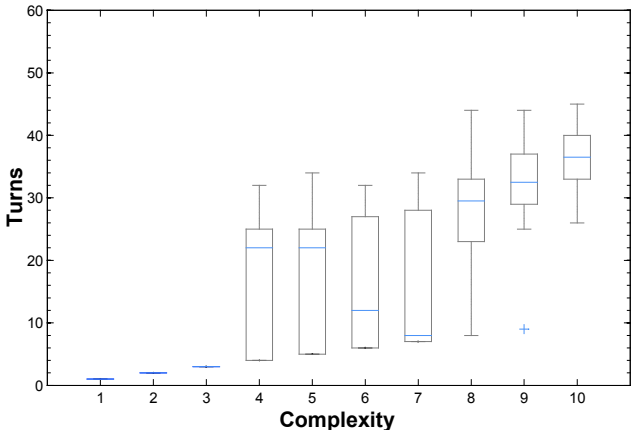


Figure 7. Complexity classes 1 to 10

Figure 7 illustrates the resulting boxplot diagram for the complexity classes 1 to 10. Regarding complexities 1 to 3, the algorithm provides the optimal solution, which is the complexity measure itself, in each test. From complexity 4, the solutions diverge in length so that an IQR appears. Up to complexity 7, the IQRs lie between the optimal solution and around 20 turns above it. Surprisingly, the median decreases in these complexities so that in complexity 4, the median is about 22 turns, while in complexity 7, the median is only about 8 turns. The reason becomes obvious when observing the raw data, which reveal that from complexity 4 to 7, the optimal solutions are increasingly reached. Since optimal solutions are the shortest possible, the median consequently becomes decreased. The same reason causes that in these cases no lower whiskers are visible. For complexity 8 to 10, the IQR reduces to around 8 turns, beginning at turn numbers 23, 29 and 33. So, from complexity 8, the solution lengths are heavily increasing, while classes 8 and 9 sometimes still met optimal solutions as smallest observation on the whisker or as outlier.

From complexity 10 on, optimal solutions are not reached any more.

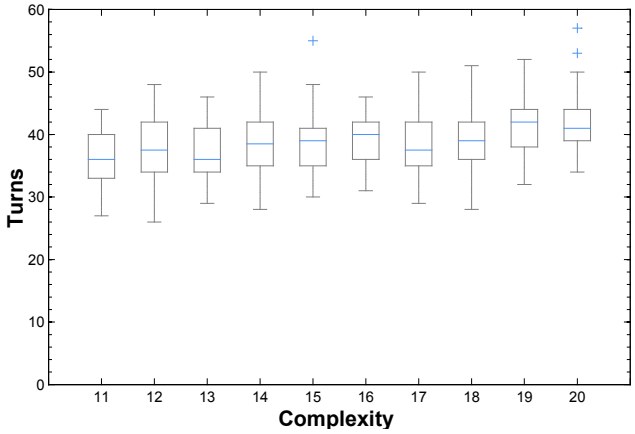


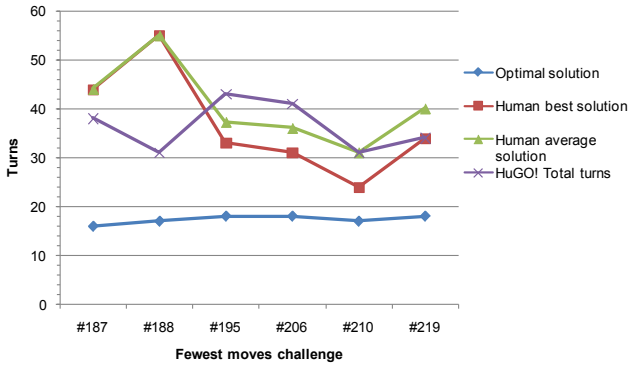
Figure 8. Complexity classes 11 to 20

Regarding the complexity classes 11 to 20, as illustrated in Figure 8, the needed numbers of turns are relatively constant in contrast to the complexities 1 to 10. The IQRs lie between around 33 turns and 44 turns in all classes. While no optimal solutions are reached any more, some outliers identify turn number of up to 57 turns.

To summarize, the performance tests reveal four distinct categories of complexity classes. The first category goes from complexity 1 to 3, where HuGO! reaches optimal solutions. In classes 4 to 7, optimal solutions are often reached, while a heavy variation occurs. From complexities 8 to 10, HuGO! loses capability of calculating the optimal solution, while solution sequences are rigorously lengthened. The last category goes from complexity class 11 to 20 and is characterized by a relatively equal amount of around 38 turns on average.

4.4 HuGO!-Human Competition

In the HuGO!-Human competition, the human competitiveness of the developed genetic algorithm is tested. Therefore, a new benchmark that represents the human performance is introduced. To obtain information on human capabilities, the databases of DAN HARRIS' and PER KRISTEN FREDLUND's *fewest moves challenge* was evaluated.[7] The fewest moves challenge is the leading provider of contests that aim to solve the cube using the fewest turns possible. Contestants meet regularly using different human strategies without technical support to compete in finding the best solution. The resulting data was used to create a benchmark of human capability that can now be compared to the results of HuGO!. This is especially of interest because the genetic algorithm incorporates a human strategy as well.[3] Arbitrary challenges of the fewest moves challenge, i.e. scrambles, are chosen and a solution is calculated by HuGO!. The algorithm's result is compared to the human best solution and the human average solution of a certain scramble, whereas the human results have a universe of one to seven solutions. The competition is repeated for six times so that possible outliers would be exposed. The used parameter settings are 200, 5000 and 50000 generations for phase 1 to phase 3 as set for the performance tests of Section 4.3. Figure 9 illustrates the results of the competition, while Table 7 provides background information on the genetic algorithm:



#187: B F2 D2 L B2 D' L B R2 U' B' F' L B2 L' R B' U2 F B' R D R' D' F' L D' U L'
#188: D L' R D' U R F D2 R L' F2 B2 L' U' F2 D U B U B' L2 F U R U2 L' B2 U F' D'
#195: F R' F2 L' D' R' D' R F' L2 R' B2 L2 R' F2 U' D' R' D R F' B2 D B2 F' L2 R2 U' B' D2
#206: D' F2 B' L R' U F U' D2 B R2 L2 D' B2 F R2 L2 D U2 B2 L2 D R D2 U L D R2 U' R'
#210: L' F L B' L' B' R' L' D' R L B2 R' D2 F2 R' D2 B R' L D' R2 U B' U' B R L' B2 L'
#219: D' L2 R' F' R B2 R2 F B' R D2 R D B' L' R U2 D L' R2 U D B L' F L2 U D B2 L

Figure 9. HuGO!-human comparison

Table 7. HuGO!-human comparison background information

Test	Fewest move challenge	Optimal solution	Human best solution	Human average solution	HuGO! total turns	Total generations	Total time	Generations 2x2x3	Turns 2x2x3	Generations 2GenTransform	Turns 2GenTransform	Generations 2GenSolve	Turns 2GenSolve
1	#187	16	44	44	38	1277400	02:12:42	2400	9	175000	8	1100000	21
2	#188	17	55	55	31	255800	00:26:09	800	10	5000	4	250000	17
3	#195	18	33	37.250	43	1225400	02:04:33	400	16	175000	9	1050000	18
4	#206	18	31	36	41	955400	01:18:36	400	12	5000	6	950000	23
5	#210	17	24	31	31	620200	00:50:26	200	7	20000	6	600000	18
6	#219	18	34	40	34	1131200	01:32:08	1200	11	280000	6	850000	17
Avg		17.333	36.833	40.542	36.333	910900	01:24:06	900	10.833	110000	6.500	800000	19.000

On the horizontal axis of Figure 9, there are the indices of the human challenges that provide the basis of the comparison. On the vertical axis the needed turns of HuGO! as well as the human best solution and the human average solution are recorded. As additional benchmark, the optimal solution calculated by CubeExplorer is displayed. Regarding the human best solution, beside one draw, human capability wins three times, while HuGO! wins two times. Concerning the human average solution, beside one draw, human capability wins two times and HuGO! wins three times. From Table 7, the average solution lengths become obvious. HuGO! has the best average with 36.333 turns per solution. Then the human best solution and the human average solution follow with values of 36.833 and 40.542.

The comparison between the algorithm's solutions and human solutions shows that the algorithm, using the selected settings and human capability are quite similar in performance. There are marginal advantages for HuGO! that cannot be generalized to determine an obvious winner. Consequently, HuGO! can be considered as human-competitive. However, compared to the optimal solution, there is still a lot of potential for performance enhancement. Keeping in mind that the algorithm will improve solutions having allowance for higher generation numbers, it becomes clear that human strategies can have a high significance

[3] From a philosophical point of view, the competition might be called an evolutionary duel between the creator and its creation. The creator's capability evolved about millions of years while the creation's performance is shaped in an evolutionary time lapse.

for the implementation of evolutionary processes and vice versa. If the human practice sticks to certain optimization ideas, evolutionary processes can imitate human behavior and improve optimization performance.

5. CONCLUSION

The Human strategy based Genetic Optimizer is a collaboration of human proceeding and genetic optimization techniques. It is applied to the restoration problem of Rubik's Cube and successfully solves this task.

The foundation of the developed algorithm is built up in the introduction of the cube. The structure analysis of the applied problem instance provides a mathematically reasoned usage of the human two-generator approach for complexity reduction. The concepts of the used genetic operators are explained on the other hand. They provide the machine intelligence part that is guided by the human method after reception.

The introduction of HuGO! is not only theoretically reasoned but also empirically. Tests to solve the cube using a plain genetic algorithm show the inappropriateness of this optimization technique to the chosen discrete optimization problem. Only when a human strategy was implemented that splits up the process to distinct intermediate stages, the algorithm's solving capability became enabled. The strategy consists of three phases that allow the cube to be gradually solved. In the first phase, a 2x2x3 subcube is completed in one of the 12 possible locations. The second phase transforms the cube to the two-generator subgroup, which allows the cube to be solved by just turning the two adjacent layers that are not solved yet. In the third phase these two layers are turned until the entire 3x3x3 cube is completely solved. For the realization of the human strategy, several obstacles in solution representation, additional operators and fitness function had to be overcome. Finding of an appropriate fitness function will also be the most demanding task regarding other possible problem instances. The reasons to additionally use a human strategy and the corresponding overhead must be weighted to decide about an introduction.

The tests documented for this work are divided up into three categories. Integrity tests show that the algorithm is likely to find a solution to every input. For this assumption 5832 three-turns scrambles and 6670 four-turns scrambles were tested. Performance tests reveal four distinct efficiency characteristics depending on the introduced complexity measure. From complexity 1 to 3, HuGO! reaches optimal solutions. From 4 to 7, a heavy variation in solution lengths occurs. From 8 to 10, HuGO! loses the capability of optimal solutions and in the last category from 11 to 20, a relatively equal amount of around 38 turns establishes itself. Consequently, the algorithm performs well for lower complexities, while it shows disadvantages in the handling of complex scrambles. The final competition between evolution in a time lapse, the genetic algorithm, and the product of real evolution, the human capability, demonstrates the human-competitiveness of HuGO! in a draw on the given parameter setup.

An additional idea for algorithm improvement is the variation of the solution representation to test binary strings or exclude half turns to reduce the search space. Concerning the string optimization, solutions could be browsed for entire redundant turn sequences that can be cut out. From a more distanced viewpoint, other evolutionary or non-evolutionary optimization techniques could be tested for applicability regarding the covered or other problems with human intrusion.

To summarize, this work vividly shows, how a human strategy can be incorporated in a genetic algorithm. The goal was not to develop an outstanding fast or efficient algorithm, but to demonstrate the advantageous adaptation of genetic algorithms to human induced strategies. While an external strategy guides the algorithm, the evolutionary process produces solutions no human had thought of. Application fields of this technique could be, among others, evolutionary knowledge management with integration of knowledge from different sources, collaborative human-computer interaction and fields that facilitate user-centered design to integrate user preferences into problem solving processes. All in all, it must be stated that human- and artificial intelligence can complement each other, producing cooperative results and improving performance.

6. REFERENCES

[1] Bäck, T.: Optimal mutation rates in genetic search. In: *Proceedings of the 5th International Conference on Genetic Algorithms*. (1993), pp. 2-8.

[2] Bäck, T.: Mutation parameters. In: The *Handbook of Evolutionary Computation*. Eds.: T. Bäck, D. Fogel, Z. Michalewicz. Bristol 2002c, pp. E1.2:1-E1.2:7.

[3] Castella, C.: Rubik's Cube, méthodes pour tous. 2005. http://www.francocube.com. Call date 2009-01-20.

[4] De Jong, K.: *An analysis of the behaviour of a class of genetic adaptive systems*. PhD Thesis, University of Michigan, Michigan 1975.

[5] Frey Jr., A.; Singmaster, D.: *Handbook of Cubik Math*. New Jersey 1982.

[6] Grefenstette, J.: Optimization of control parameters for genetic algorithms. In: *IEEE Transactions on Systems, Man, and Cybernetics*. 16 (1986) 1, pp. 122-128.

[7] Harris, D.; Fredlund, P.: Fewest Moves Challenge. 2009. http://fmc.mustcube.net/. Call date 2009-01-18.

[8] Herdy, M.; Patone, G.: Evolution Strategy in Action – 10 ES-Demonstrations. 1994. http://www.bionik.tu-berlin.de/user/giani/esdemos/evo.html. Call date 2009-02-23.

[9] Jumbo Spiele GmbH: Geschichte des Zauberwürfels. 2008. http://www.zauberwuerfel.de/history/. Call date 2009-02-28.

[10] Kociemba, H.: CubeExplorer. 2009. http://kociemba.org/cube.htm. Call date 2009-02-21.

[11] Kunkle, D.; Cooperman, G.: Twenty-six moves suffice for Rubik's Cube. In: *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation*, (2007), pp. 235-242.

[12] Mühlenbein, H.: How genetic algorithms really work. I. Mutation and hillclimbing. In: *Proceedings of the 2nd International Conference on Parallel Problem Solving from Nature*. (1992), pp. 15-25.

[13] Palmer, J.: Cube routes. In: *New Scientist*. (2008) 2668, pp. 40-43.

[14] Physorg.com: Northeastern University researchers solve Rubik's Cube in 26 moves. 2008. http://www.physorg.com/news99843195.html. Call date 2009-02-28.

[15] Radu, S.: New Upper Bounds on Rubik's cube. 2007 http://www.risc.uni-linz.ac.at/publications/download/risc_3122/uppernew3.ps. Call date 2009-02-22.

[16] Reid, M. Superflip requires 20 face turns. 1995a. http://www.math.rwth-aachen.de/~Martin.Schoenert/Cube-Lovers/michael_reid__superflip_requires_20_face_turns.html. Call date 2009-02-22.

[17] Reid, M. New upper bounds. 1995b. http://www.math.rwth-aachen.de/~Martin.Schoenert/Cube-Lovers/michael_reid__new_upper_bounds.html. Call date 2009-02-22.

[18] Rokicki, T.: Twenty-Five Moves Suffice for Rubik's Cube. 2008a. http://arxiv.org/abs/0803.3435. Call date 2009-02-21.

[19] Rokicki, T.: Twenty-Three Moves Suffice. 2008b. http://cubezzz.homelinux.org/drupal/?q=node/view/117. Call date 2009-02-21.

[20] Rokicki, T.: Twenty-Two Moves Suffice. 2008c. http://cubezzz.homelinux.org/drupal/?q=node/view/121. Call date 2009-02-21.

[21] Sarma, J.; De Jong, K.: Generation gap methods. In: *The Handbook of Evolutionary Computation*. Eds.: T. Bäck, D. Fogel, Z. Michalewicz. Bristol 2002, pp. C2.7:1-C2.7:5.

[22] Seven Towns Ltd.: You can do the Rubik's Cube. 2008. http://www.youcandothecube.com. Call date 2009-01-13.

[23] Singmaster, D.: *Notes on Rubik's Magic Cube*. New Jersey 1981.

[24] The MathWorks Inc.: boxplot. 2009. http://www.mathworks.com/access/helpdesk/help/toolbox/stats/index.html?/access/helpdesk/help/toolbox/stats/boxplot.html. Call date 2009-02-27.

[25] Thistlethwaite, M.: Thistlethwaites's 52-move algorithm. 1981. http://www.geocities.com/jaapsch/puzzles/thistle.htm. Call date 2009-02-21.