Comparison of Rubik’s Cube Solution Softwares with SWOT Analyses for the Input-Output Process Modeling

Csaba Fogarassy
Szent István University Faculty of Economics and Social Sciences
Institute of Regional Economics and Rural Development
Gödöllő, Hungary
E-mail: fogarassy.csaba@gtk.szie.hu

***Abstract* -** The Cube Explorer was the first Rubik’s Cube solution program which was able to solve a cube from any starting position using around 30 rotations. After this first software, and also using it as a basis, began the different personal developments for the different solution programs all around the world. In order to view the connection network of Rubik’s Cube software development, and modeling of input-output process based on Rubik’s Cube’s solution algorithms, I will do the SWOT analysis for three different development routes or software methods (Ruwix program - Kociemba Cube explorer development; Solution Searching LBL software developed by Gábor Nagy; Rubiksolve Program developed by Eric Diec). During input-output process analysis, the goal is to make the analysed development or investment process faster and simpler, even with the use of software. The role of the software can be important if after assigning the attributes to the cube’s respective sides, we can define the starting state of the project even with the unassortedness state of Rubik’s Cube.

***Keywords* -** *Rubik’s Cube solutions, Rubik’s Cube software, solution algorithms, cube explorer, layer-by-layer method, input-output process analysis, Ruwix program, Rubiksolve program.*

*****

I. Introduction

Scientists basically thought that a maximum of 18 steps are required to solve the cube, however, Michael Reid mathematician created a mathematical formula that made it obvious that the cube can’t be solved starting from any given state, with a rotation which consists of less than 20 steps. This means that the cube can only be solved with at least 20 rotation steps according to theoretical calculations. Rokicki (2008) and his affiliates divided all the starting configurations using the technique derived from group theory (Davis, 2006). This meant 2.2 billion groups, each of which consisted of 19.5 billion configurations. The grouping was dependent on the reaction of the configurations to 10 possible rotation movements. The mathematicians working on the project, using the different symmetries of the cube, successfully reduced the groups to 56 million. This reduction was made possible through a very simple methodology, since if we turn the given cube upside down, or to each side, the solution won’t get any more complicated, therefore making these equal ’combinations’ outright unnecessary (Fogarassy, 2014). This means that the newly created algorithm was able to match movements with the correct starting state at an incredible speed, making the solution of a 19,5 billion series possible in a mere 20 seconds, which may seem like an astounding speed, but still would’ve required 35 years for an ordinary computer to complete the entire task. In order to shorten the time required, they were searching to an especially efective method. During the process of problem solving, it was quite fortunate that the work was followed by John Dethridge, one of Google’s engineers, and offered the free capacities of his IT systems to aid the research. With using the free capacity of the PC empire, he managed to solve the problem in a few weeks. The result of the astounding and persistent research spanning 15 years therefore proved the assumption made and supported by mathematicians for a long time, that to solve the 3×3×3 Rubik’s Cube from any given starting state, no more than 20 moves is required. The basic rule is that during the arrangement of the cube, our goal is to move the small cubes into a different location, or leave them in place, but at a different angle (f.e. let a cornercube do a 120° turn, or rotate an edge cube with its colour) while everything else remains untouched. To solve the 3×3×3 cube, many different methods were made independent of each other in the last few decades, one of which is the very popular *layer by layer method* designed by David Singmaster, which was published in „Notes on Rubik’s ’Magic Cube’” in 1981. Using another general solution, named corner first method, the speed of solution can go well below a minute. Obviously, the speed is dependent on the number of required rotations. The corner first method is the basis of one of the fastest, Gilles Roux’s method. The point is that as a first step, all corners must be arranged to their position and proper angle. After this, all mid rows can be freely moved in a way that the corners remain intact. With this method, we have a much wider margin of freedom on the cube, compared to the layer by layer method (Doig, 2000). A very widely known and used method among „cubers” is the Fridrich method. The method was developed by Jessica Fridrich, which is very similar to the layer by layer method, but uses a high number of algorithms for the solution. With this method, and lots of practice, the cube can usually be solved in 17 seconds, whith is why most of the world’s „speedcubers” use this method. As a general assumption, the n x n x n, n=3 Rubik’s Cube can be solved with Θ(n2 / log(n)) rotations (Fogarassy et al., 2014). The optimalisation of the 3×3×3 cube, and reaching the minimum amount of rotations began with the discovery of group theory using computers, the basis of which was laid down by Morwen Thistlethwaite in 1981. The basis of the Thistlethwaite method was to divide the problem into subproblems, meaning searching for the solution by dividing the cube into subgroups. The tools of group theory can simplify the calculations of the process of software development by defining subgroups of the hundreds, or millions of layouts, which have shared mathematical characteristics. Herbert Kociemba German mathematician used a cunning method to decrease the 43 quintillion possible rotations of the cube in 1992 (Ajay, 2011). Kociemba had a different approach to the mathematical relations of the cube, compared to the usual method of basing it on fix combinations – he made a subgroup, which was based on 10 out of the 18 possible rotations of the cube. With the combination of these 10 rotations, he found out that he can reach 20 billion different configurations from a solved cube. This is an important step, because this subgroup is small enough to fit an ordinary PC-s memory. Kociemba also developed a program for this, named Cube Explorer, which was further developed by American mathematician Michael Reid in 1995, and used to estimate the minimum required rotations to solve the cube at 30. Cube Explorer was the first Rubik's Cube solution program which was able to solve a cube from any starting position using around 30 rotations. Thus, after this first software, and also using it as a basis, began the different personal developments for the different solution programs all around the world. In order to view the connection network of Rubik's Cube software development, and modeling of input-output process based on Rubik's Cube's solution algorithms, I will do the SWOT analysis for three different development routes. During input-output process analysis, the goal is to make the analysed development or investment process faster and simpler, even with the use of software. The role of the software can be important if after assigning the attributes to the cube's respective sides, we can define the starting state of the project even with the unassortedness state of Rubik's Cube.

II. Material and Methods

To analyse the software aimed at solving Rubik's Cube, I used a SWOT analysis during my research, and to evaluate the processes of the Rubik's Cube solution algorithms. SWOT analysis is a strategic planning tool which helps evaluating strenghts, weaknesses, opportunities and threats, which may come up in case of corporate or personal decisions concerning a product, project, or business venture, or any other goal. SWOT analysis includes the measurement of the system, the person, or the inner and outer environment of the business, thereby helping the decision maker to concentrate only on the most important topics (Fogarassy, 2014).

The answers we seek with the analyses:
Strengths:

- What pros does the analysed system have in input-output process analysis, analysation of internal attributes?
- What does it do better compared to the other system?
- What's the hearsay about the system, its strengths?

Weaknesses:

- What parts could be improved?
- What should be avoided?
- What's the hearsay about the system, its weaknesses?

Opportunities:

- What opportunities does it have in the future?
- What trends, market tendencies are known to it?

Threats:

- What problems may surface during its use?
- What are the competitors doing?
- Are unfavourable changes visible in the operation environment?

The above defined questions are answered in the evaluation chart below, by giving short answers to them.

	POSITIVE TRAITS	NEGATIVE TRAITS
Internal traits	Strengths	Weaknesses
External traits	Opportunities	Threats

In the case of the solution-searching software applications which were examined, the goal of the SWOT analysis is to determine if the functions of each software are applicable to the input and output system attributes of the project evaluation model, and if they satisfy the user expectations.

The SWOT analysis offers a good opportunity to create an overview comparison, which has no exact attributes definable in easily comparable dimensions. In itself, the SWOT analysis has no meaning, however, if it's part of a complex analysis, it can sufficiently facilitate thought process.

III. Discussion

The tools of group theory can simplify the calculations of the process of software development by defining subgroups of the hundreds, or millions of layouts, which have shared mathematical characteristics. Herbert Kociemba german mathematician used a cunning method to decrease the 43 quintillion possible rotations of the cube in 1992 (Ajay, 2011). The mathematical basis of the calculation (according to group theory) was how we calculate the variation possibilities, in other words, how many different samples can we observe on the cube:

– 8 corners = 8! positions / each have 3 possible orientations = 3^8
– 12 edges = 12! positions / each have 2 possible orientations = 2^{12}
– Impossibilities:
  - no element substitution (2),
  - no edge orientation (2),
  - no corner orientation (3).
– Meaning 2x2x3 = divided by 12, which totals for = ($8! \times 3^8 \times 12! \times 2^{12}$) / 12 ~= 4.3×10^{19}

Kociemba had a different approach to the mathematical relations of the cube, compared to the usual method of basing it on fix combinations – he made a subgroup, which was based on 10 out of the 18 possible rotations of the cube. With the combination of these 10 rotations, he found out that he can reach 20 billion different configurations from a solved cube. This is an important step, because this subgroup is small enough to fit an ordinary PC-s memory. Kociemba also developed a program for this, named Cube Explorer, which was further developed by American mathematician Michael Reid in 1995, and used to estimate the minimum required rotations to solve the cube at 30. Theoretical scientists alredy considered 20 to be „God’s number” (the minimum required rotations), but the proof would’ve required a supercomputer. Finally, the proof of „God’s number” being 20 only happened in July 2010, when Thomas Rokicki, Herbert Kociemba, Morley Davidson and John Dethridge (Rokicki et al., 2010) proudly declared to the world that it’s proven – „God's Number for the Cube is exactly 20”.

Therefore, Kociemba’s Cube Explorer was the first Rubik’s Cube solution program which was able to solve a cube from any starting position using around 30 rotations. Thus, after this first software, and also using it as a basis, began the different personal developments for the different solution programs all around the world. In order to view the connection network of Rubik’s Cube software development, and the output-input project development methodology based on Rubik’s Cube’s solution algorithms, I will do the SWOT analysis for three different development routes. During output-input project development, the goal is to make the analysed development or investment process faster and simpler, even with the use of software. The role of the software can be important if after assigning the attributes to the cube’s respective sides, we can define the starting state of the project even with the unassortedness state of Rubik’s Cube. If we define the unassortedness with the cube’s state, the solution program can easily inform the user how he can reach various levels of assortedness. The solution search using software raises one simple question: is the route appropriate, and can the process of solution search abide by the various professional requirements (Global best practice for innovation ecosystems), which lead to the basis of successful project development?

The goal of the detailed introduction of the SWOT analysis in the methodology section was to make it clear to me, if the functions of the software are applicable to project the evaluation model’s input and output requirements. The analysis was done by classic SWOT rules, the details of which won’t be shown, only the results. For the sake of understanding them, I’ll give short descriptions on the various software applications.

Software evaluated using SWOT analyses:

- ✓ RUWIX PROGRAM (KOCIEMBA CUBE EXPLORER DEVELOPMENT)
- ✓ SOLUTION SEARCHING LBL SOFTWARE (GÁBOR NAGY)
- ✓ RUBIKSOLVE PROGRAM (ERIC DIEC)

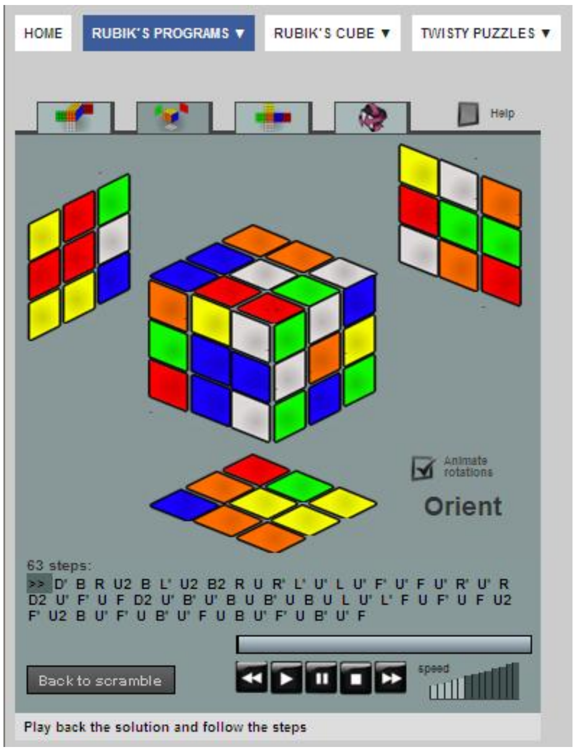
IV. RESULTS AND CONCLUSIONS

SWOT analysis of the Ruwix program

The complex solution and demonstration program was developed by Hungarian Ferenc Dénes, using Kociemba’s 2005 solver program as a basis. The software chooses the shortest possible solution from any given starting combination. The average number of rotations is 50-60, which does not prefer *layer by layer* algorithms. In this case, the developers uploaded a lot more algorithms into the optimal solution search program, which finds more right solutions during optimalisation. The online solution software shares all important information with the user, and it’s very spectacular (Illustration 1.).

Ruwix is an online Java-based web application, which doesn’t use any support platforms. It was applied with necessary functions by the developer, in order to help users learn Rubik’s Cube, and the various solution methods for it. Its suitable f.e. to animate the process of solution step by step, and display it to the user.

The solution search engine can animate the solution and rotation moves from any given combination, which is preferred by users training for Rubik’s Cube solving competitions. Using the Ruwix program, users can play with different Rubik products online (2×2×2 cube, 3×3×3 cube, 4×4×4 cube, 5×5×5 cube, etc.), which offer a pleasing game experience in 3D.



63 steps:
>> D' B R U2 B L' U2 B2 R U R' L' U' L U' F' U' F U' R' U' R D2 U' F' U F D2 U' B' U' B U B' U B U L U' L' F U F' U F U2 F' U2 B U' F' U B' U' F U B U' F' U B' U' F

Play back the solution and follow the steps

Illustration 1.: Visual style and shortest solution formula of Ruwix program
Source: Dénes, T. (2005) Ruwix.com

The SWOT evaluation of Ruwix (Chart 1.), in accordance to the project evaluation model’s input and output requirements:

Chart 1.: Ruwix program SWOT chart

	POSITIVE TRAITS	NEGATIVE TRAITS
INTERNAL TRAITS	**STRENGHTS** Exceptional graphics and visual details, some mention it as the world's most advanced solution software. Offers solutions not only to Rubik's Cube, but many other logical games.	**WEAKNESSES** Presently not compatible, since it uses different, faster algorithms than the layer by layer solution, which aren't the best for development solutions.
EXTERNAL TRAITS	**OPPORTUNITIES** Because of its strenghts, and the applicability, it would be beneficial to develop output-input specifications as well.	**THREATS** Since the program runs in an online format, it isn't possible to add special data to it. Even in case of a output-input specification, syncing the free software with the pay-to-use SMART add-on makes it difficult to use.

Source: self-made (based on Fogarassy, 2014)

SWOT analysis of Solution Searching LBL software for Rubik's Cube

To introduce the Rubik's Cube solution software, I will mostly use a domestic development made by IT technician and engineer Gábor Nagy's (University of Debrecen) description and methodology guide: „Solution searching methods", which is unique because of its *status space* representation, which was used to work out the problem of multi-level solution search. The other important thing to note about the choice of software was that it prefers the *layer by layer(LBL)* solution, and as far as I know, this is the only application which uses only this method, because it's considered „too slow". (On another note, any solution search could implement the *layer by layer method*, were it coded with it in the first place.)

The program was developed in 2008, using Java language, and NetBeans IDE 6.1 development platform. To make the structure of the program clear, we have to understand the respective structures of two packs – the *status space* and *cube* packs.

The pack named *Status Space („Allapotter")* contains two abstract classes, and an interface, which save the exact, various elements and attributes of the status spaces. During the main problem's implementation, these elements are concretised by the program to fit the representation of the *status space*. The program checks (for each different status) if a given status is the goal, or not. According to the developer's manual, the heuristic result is ensured by the interface named *Heuristic Status („HeurisztikusAllapot"),* which needs to be implemented in the program from the get-go. In the case of the solution search program, we define the *Cube Status („KockaAllapot")* class, or the *cube* pack as the start, the elements of which describe a given element of the *status space.* This class contains the constructors not included in the 54-element byte packets, which record the various states of Rubik's Cube, and all the methods applicable for the different statuses. The *objective status checking function* checks the 3D parts of the cube, and if it finds a colour out of place, returns a „false" message, while if it doesn't, the cube is solved, and every colour is in its place.

In the program's description, there's also mention that the status of the cube is marked with 54 number, which are selected from the [0,5] interval, and the colours symbolise the various colours (based on Nagy, 2008):

$$H = \{(0,0,\ldots\ldots,0),(1,0,\ldots\ldots,0),\ldots\ldots,(5,5,\ldots.,5)\}$$

A ≠ H, since not all elements of H can be real statuses.

$$A = \{a | a \in H_1 \times \ldots . \times H_n\}$$

Description of Cube Pack

Using the classes and interface of the *„Status Space Pack***"**, the created classes arecategorised into the „Cube Pack", which are closely related to Rubik's Cube and its structure. The examples of the „Cube Status" class are defining the various statuses of the status space, but the class also contains the constructors not included in the 54-element byte packets, which record the various states of Rubik's Cube, and all the methods applicable for the different statuses, which are as follows (based on Nagy, 2008):

- ✓ „Objective status checking function", which has a return value of either true or false. Using three For-loops integrated into each other, it analyses the 3D block that defines the status of the cube, and if it finds a colour out of place, returns a „false" message, while if it doesn't, the cube is solved, and every colour is in its place.
- ✓ „Operator" – a function that checks the application master, and analyses if the operator condition is applicable to the given status. This also has a logical return value, which is – for Rubik's Cube – always true.
- ✓ *„Apply function*", which contains the operator for the given status as a parameter, and it's return value is the function of the resulting status. It creates a copy of the cube's status, executes the value copying abiding by the operators, and returns with the copy.
- ✓ The function that benchmarks the given status against a different status, which is the result of a parameter. Has a logical return value, which is true if all elements of the statuses of benchmarked cubes are identical. Otherwise, its value is false.
- ✓ An evaluation function which is exceptionally important for our research.
- ✓ Method to access the „data tags" which register the various states of the cube.
- ✓ Methods related to imaging and burning.

Layer by layer method, and the evaluation function

Due to the developer's choice, the program uses a *AVID* search engine (greedy search) to solve the cube, therefore, the evaluation function consists only of the heuristic function, which is implemented by the „Cube Status" class' „Heuristic Method", alredy mentioned above. The method evaluates and scores the various statuses by the sequential row by row, in other words, the *layer by layer* method. Therefore, due to the impact of the heuristic pack, the program uses the *layer by layer* method to find the solution, meaning row by row, though it's a known fact that this isn't the fastest, and most effective way to produce the result in solution search. The program doesn't analyse the starting state, since the optimalisation of the starting side would require a complex evaluation function's implementation, which was deemed unnecessary for this program by the developer, so the program always starts with the yellow side. In terms of the method, we're talking easily checkable layers, or in other words, levels, meaning the heuristic function also begins by the check of this so-called level, to avoid checks which are not important on the actual level, but may be on lower levels (Molnár, 1994, Nagy, 2008).

These levels are as follows (Illustration 2):

Level 0.:	Cube doesn't abide by level 1's requirements.
Level 1.:	Edges which also have yellow are in position, with proper orientation, meaning „yellow cross" is complete.
Level 2.:	Corners which also have yellow are in position, with proper orientation, meaning „upper row" is complete.
Level 3.:	Mid row is complete.
Level 4.:	Edges which also have yellow are in position, with proper orientation, meaning „white cross" is complete.
Level 5.:	Corners which also have yellow are in position, with proper orientation, meaning the cube is in its finished state.

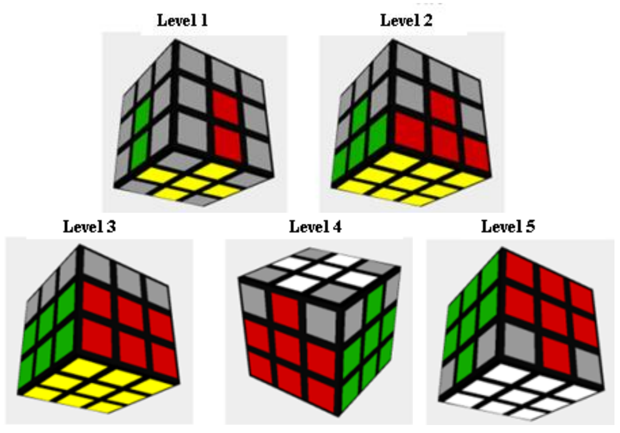


Illustration 2.: Levels of Layer by layer method in the program
Source: self-made (based on Nagy, 2008)

According to the developer's description, we may not be able to continue without heuristics, or breaking the level. In this case, the so-called solution algorithms may help when used for the correct statuses, which are series of steps that, though degrade the heuristics at first, but get closer to the goal in the end, compared to where we stood before applying them. The first level (solution of first row) may be reached even without algorithms, but this is the part of the heuristic function which is implemented with the greatest hardship. According to Nagy, the reason for this is that unlike on higher levels, where we primarily use algorithms apart from 1-2 rotations, at first, we use steps which are simple, but numerous, and give a lot of various alternatives, so translating human knowledge for the program becomes difficult. On higher levels, use of the heuristic algorithm becomes much less of a problem, we can assign a few fixed algorithms for virtually any status, we only have to decide wich to implement first.

With the heuristics of a status, the programmer defines the return value of the heuristic function, in other words, the „correctness" of the status. His idea was that while we're on lower levels, the heuristics of the status starts from a higher value, while the farther the next level seems during the appropriate checks for each level, the more its value increases. Therefore, the rate of increase is dependent on the positions and/or orientations of the edges and corners required to complete the level. Each of these edges or corners raises the value of the heuristics more or less. The scale therefore depends on how far it is from its proper position, or a position from which it can be moved to its proper position using an algorithm. According to the developer, within a single level, the value of heuristics will never raise so much, that a lower level's heauristics is lower as well. This condition is necessary for the search engine to find the shortest route to the solution, based on the method. One of the consequences for this is that if we reach a certain level with the program, it's sufficient to do the checks only for that given level, since all the others either alredy stand true, or aren't needed yet. According to this, the scoring in the program is as follows (based on Nagy, 2008):

- ***Determining the level*** is the first step of evaluation/scoring. The higher the level we're on, the lower the number will be. The starting value of heuristics on level 5's evaluation function is "**0".**
- ***On level 0:*** An edge in its place with proper orientation barely raises heuristics, while the ones far from their position raise it according to their exact „misplacedness". If we have at least two edges in the right position and with proper orientation, we can allow the use of algorithms, but this causes the edges to raise heuristics less, if they're close to being put in their proper position using an algorithm. These algorithms consist of only 3-5 steps, but have other extra effects. For each side, we have to check using three of these algorithms. The reason for this is that the software interprets operators from a fixed point of view, with the yellow side always being on top, and the blue side in the front. Because of this, the same sequence of rotations may be built with different operators for the various sides, but we have to be able to choose the correct one. A good example for this would be for us to check three different positions for

the yellow-blue edge, from where only an algorithm can put it in its proper position (Illustration 3.).

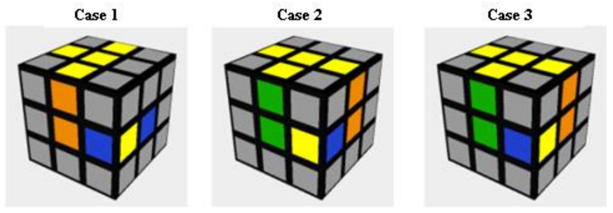


Illustration 3.: Edges only solvable through algorithms
Source: self-made (based on Nagy, 2008)

Algorithm 1.: UR, LB, UL.
Algorithm 2.: UR, LF, UL.
Algorithm 3.: UR, UR, RR, UL, UL.

Abbreviations are from initials:
F (Front)
B (Back)
U (Up)
D (Down)
L (Left)
R (Right)

- ***On level 1:*** On this level, we can use almost only algorithms to solve a corner. Heuristics may further increase due to the corners' distance of their „algorithm possibilities", apart from the basic increase of the level. On this level, we have to watch 5 different algorithms. Let's go through the blue-yellow-orange corner's five different algorithms via the examples on Illustration 4. below:

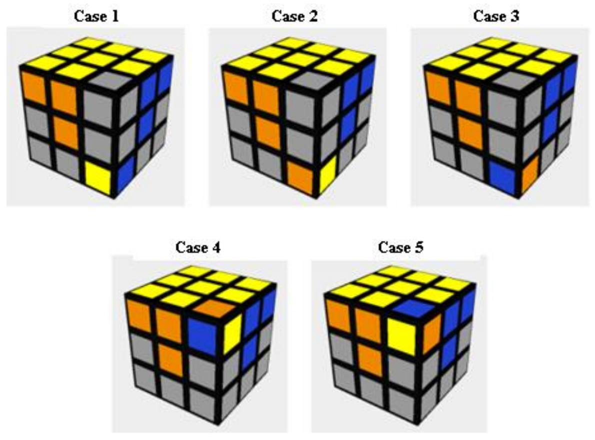


Illustration 4.: Positions of corners defineable via algorithm
Source: self-made (based on Nagy, 2008)

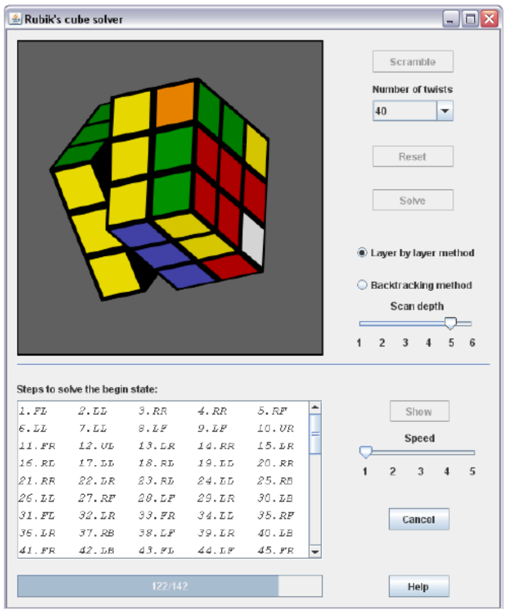
Algorithm 1.: LF, LL, LB.
Algorithm 2.: FL, LR, FR.
Algorithm 3.: LF, LR, LB.
Algorithm 4.: FL, LL, FR.
Algorithm 5.: LF, LL, LB.

The solution search program therefore uses the above mentioned seven levels' *AVID* search to solve the cube (Chart 2). During the evaluation of the above defined methodology guide, it's obvious that the program is able to solve Rubik's Cube from virtually any starting combination using *Layer by layer* method. The number of required rotations is dependent on the base combination, but usually needs more than 70 rotations. However, in case of a simpler starting combination, this can decrease to 40-45 rotations (Illustration 5.).

Chart 2.: Layer by layer solution algorithms for 3×3×3 Rubik's Cube using software and *AVID* search engine (on levels 2., 3., 4., 5.)

Level	Phase	Algorithms
2.	Positions defineable with algorithms for second row edges	Algorithm 1.: FL, LL, FR, LB, FR, LF, FL. Algorithm 2.: LF, LR, LB, FR. LB. FL LF. Algorithm 3.: LF, LL, LB, LR.
3.	State fit for edge switch, edge switch on sealing side	Algorithm 1.: LF, LL, LL, LB, LR, LF, LR, LB, LR.
	Edge rotation, rotating sealing side to match colours	Algorithm 1.: LB,RB, FL, LF, RF, LR, LB,RB,FL,LF,RF, LR, LB,RB,FL,LF,RF,LR
4.	Corner switch	Algorithm 1.: LB, LL, RB, LR, LF, LL. RF, LR. Algorithm 2.: FR, LR, RR LL, FL, LR EL, LL
5.	Rotating corners to match colours, correction of misplaced corners	Algorithm 1.: RB, LL, RF, LL, RB, LR, LR, RF, LB, LR, LF, LR, LB, LR, LR, LF. Algorithm 2.: LB, LL, LL, LF, LL, LB, LL, LF, RB, LL, LL, RF, LR, RB, LR, RF.

Source: self-made (based on Nagy, 2008)



Number of twists: 40
(● Layer by layer method)

Steps to solve the begin state:

```
1.FL    2.LL    3.RR    4.RR    5.RF
6.LL    7.LL    8.LF    9.LF    10.UR
11.FR   12.UL   13.LR   14.RR   15.LR
16.RL   17.LL   18.RL   19.LL   20.RR
21.RR   22.LR   23.RL   24.LL   25.RB
26.LL   27.RF   28.LF   29.LR   30.LB
31.FL   32.LR   33.FR   34.LL   35.RF
36.LR   37.RB   38.LF   39.LR   40.LB
41.FR   42.LB   43.FL   44.LF   45.FR
```

122/142

Illustration 5.: Evaluation screen of Solution Searching LBL software for Rubik's Cube
Source: Solution Searching LBL software for Rubik's Cube

The reason I found showing the Solution Searching LBL software for Rubik's Cube in this much detail is that during the process of solution, it follows rotations by hand almost completely, and uses each algorithm of the *layer by layer* method, but doesn't implement any other methods.

The SWOT evaluation of Solution Searching LBL software for Rubik's Cube (Chart 3.), in accordance to the project development model's input and output requirements:

Chart 3.: Rubik's Cube Solution Search program SWOT chart

	POSITIVE TRAITS	NEGATIVE TRAITS
INTERNAL TRAITS	**STRENGHTS** The steps of conceptual and practical solutions are the same The layer by layer solution is followed through in the program Uses obvious advancement and correction steps Because of the easy programming, it's also easy to develop Every algorithm is also defineable in the steps of the input-output project evaluation model as well.	**WEAKNESSES** The visual interface is not up-to-date Slightly slow processing Not available in online format As of now, it can only solve the 3×3×3 Rubik's Cube.
EXTERNAL TRAITS	**OPPORTUNITIES** Visual interface Easy to sync with the SMART evaluation software plugin The definition of input-output model domain requires no additional development on the software Because of the easy programming, it may prove to be cheap to be a newcomer on the market.	**THREATS** Quite an old development The program may seem slow, because it can't be accelerated properly because of a set of certain configurations „Easy to copy".

Source: self-made (based on Fogarassy, 2014)

SWOT analysis of Rubiksolve program

One of the most well-known solution software on the web. The developer, Eric Dietz has been interested in the mathematics and programming opportunities of Rubik's Cube since his childhood. His first program that solves Rubik's Cube was published, and shared with the members of the Rubik „fun" community in 2002. In 2005, he used Kociemba's 3×3×3 method to popularise his own online program. In 2007, he developed a solver program which he further developed by lowering the amount of required rotations, using newer algorithms. The one that's currently running was finalised in 2010, which uses Kociemba's algorithm, meaning it needs less than 25 rotations to finish the cube from any given starting combination. Eric Dietz always used Kociemba's algorithms for the solution, two of which can be seen on Illustrations 8. and 9., or by clicking the link below (Dietz, 2010).
The program only handles 2×2×2, 3×3×3 and 4×4×4 cubes' solution algorithms, its portfolio has no other Rubik games. It illustrates every detail in 2D, and offers no special visual enjoyment either. The illustrations that explain rotations can be interpreted easily, therefore, in the last few decades, tens of thousands of players learned to solve Rubik's Cube with this program's guides.

Because of the reduced number of algorithms, we won't find the same levels as for the previously introduced Solution Searching LBL software. The progrem doesn't implement the *layer by layer method* as a solution process, but some algorithms of the various methods are the same, meaning the same algorithms are sometimes used in different solution searching programs. The program works quite fast, only needs a few seconds to display the solution formula for the combinations put in. As a comparison, Ruwix and Solution Search need several tens of seconds, or even minutes to display the solution formula (Illustration 6.).

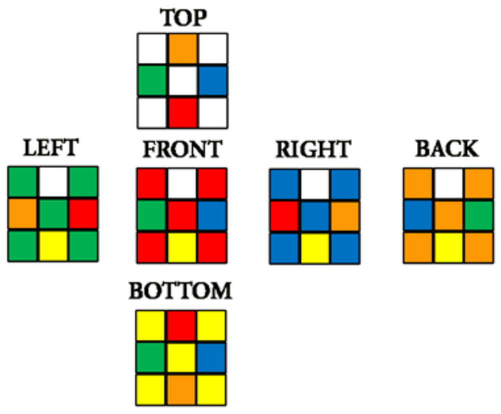


Illustration 6.: Notations of sides on the program's solution interface (flip state)
Source: based on Dietz, 2010

The SWOT evaluation of Rubiksolve, in accordance to the project develpment model's input and output requirements we can follow on the Chart 4.

The introduced Ruwix Solver and Rubiksolve applications are both the further developed versions of Kociemba's Cube Explorer, which was the basis of most Rubik's Cube fans' software development work and ideas since 2005. After reviewing the different solution programs, we can say that there is an option to bring in technically any new algorithm, but of course, the goal of all the developers was to give the competitors a program that offers the solutions with the highest possible procession speed, and lowest number of combinations necessary. In the case of the Rubiksolve program, this is below 25 steps.

Chart 4.: Rubiksolve program SWOT chart

	POSITIVE TRAITS	NEGATIVE TRAITS
INTERNAL TRAITS	**STRENGHTS** Fast, constantly developed, can use layer by layer method	**WEAKNESSES** 2D, can’t interpret layer by layer logic at the input, other user functions are missing.
EXTERNAL TRAITS	**OPPORTUNITIES** Easy plugin options offer good compatibility with model usage.	**THREATS** Since it focuses on fast solutions, not all details can be understood by the users.

Source: self-made (based on Fogarassy, 2014)

The Rubik’s Cube Solution Search program completes the cube with the seven solution levels defined by AVID’s search engine. During the evaluation of the methodology manual, we made it clear that this one is able to get to the completed stage, meaning the one side – one colour state from any starting stage with the layer by layer method. Also, the process may be stopped at any given stage. The number of rotations varies by the starting stage, but usually it takes more than 70 rotations to complete the cube. However, from an easier starting point, it can reduce to a mere 40-45 rotations. Also, by analysing the SWOT evaluations, it can be said that the swift strenghts/weaknesses/opportunities/threats chart prefers the hungarian-developed Rubik’s Cube Solution Search program, which was optimised for the layer by layer algorithms.

This Java-based application proved to be best in its functionality for the project evaluation model’s input and output expectations, also noted by the structural trait that the software’s „State Area” pack designates almost the same solution levels, that the hand-solved algorithms do. (The other evaluated softwares designate almost completely different levels.)

REFERENCES

[1] Ajay, J. (2011) Rubik’s Cube Model of Software Engineering for Incremental and legacy projects. Journal of Computing, Volume 3. Issue 2. Februar 2011 pp. 99-101.
http://journalofcomputing.org/volume-3-issue-2-february-2011/

[2] Davis, T. (2006) Group Theory via Rubic’s Cube. Geometer Org, http://geometer.org/rubik/group.pdf pp.10-12

[3] Denes, T. (2005) Ruwix program’s descriptions and programming. 2005 Source: Ruwix.com/Online Ruwix Cube Solver program p.1

[4] Dietz, E. (2010) About me. USA, Minnesota, 2007, pp. 4-5.

[5] Doig, A. (2000) Community planning and management of energy supplies - international experience. Renewable Energy. 2000; pp.325-331. https://sites.google.com/site/journalofcomputing/ www.journalofcomputing.org , pp. 99

[6] Fogarassy, C. (2012) Low-carbon economy (Karbongazdaság in Hungarian language). Monography. L’Harmattan Kiadó, Budapest, 2012, ISBN: 978-963-236-541-1 pp. 8-10

[7] Fogarassy, C. (2014) The Interpretation of Sustainability Criteria using Game Theory Models (Sustainable project development with Rubik’s Cube), Budapest; Paris: L' Harmattan Publisher, pp. 36-47
https://www.scribd.com/doc/250370912/Fogarassy-Rubik-Model-Eng-Harmattan-Publisher-2014

[8] Fogarassy, C. Borocz, M., Molnár, S. (2014) Process of sustainable project development with Rubik’s Cube using Game Theory interpretations. IJAIR, ISSN 2278-7844, Volume No. 03, Issue No.10, October-2014.
http://www.advanceresearchlibrary.com/temp/downloads/ijair/oct2014/t1.pdf

[9] Molnár, S. (1994) On the optimization of INPUT-OUTPUT systems cost functions, Pure Mathematics and Applications, Vol. 5. No. 4, 1994, pp. 404

[10] Nagy, G. (2008) Solution searching methods (on Hungarian). Scientific Paper. Debreceni University, Debrecen, 2008, pp.12-16, pp. 22-25, pp. 40-45

[11] Rokicki, T. (2008) Twenty-Five Moves Suffice for Rubik’s Cube. Source: http://tomas.rokicki.com/rubik25.pdf pp. 1-4

[12] Rokicki, T. et al. (2010) God's Number is 20. http://www.cube20.org/ pp.1

[13] Singmaster, D. (1981) Notes on Rubik's Magic Cube. Penguin Books. ISBN 0907395007). pp. 10-12.

[14] https://www.scribd.com/doc/58001400/Notes-on-Rubik-s-Magic-Cube

[15] START UP GUIDE (2012) Business advise for SMEs. (Üzleti tanácsok fejlődő kisvállakozók részére, Online publication) on Hugarian language, http://startupguide.hu , pp. 2.2