

Cube Explorer 5.01

If you like Cube Explorer you can show your appreciation.



Please note: Cube Explorer is not derived from, is not associated with and is not endorsed or sponsored by the owner of the RUBIK'S CUBE Trademark. This owner is Seven Towns Limited, the manufacturer and worldwide distributor of the RUBIK'S CUBE three dimensional puzzle and provider of an electronic version of the puzzle via its [official web site](#). Cube Explorer is a non-commercial, educational product. It is freeware and the result of scientific research.

© 2013  [Herbert Kociemba](#)

The Two-Phase-Algorithm

The following description is intended to give you a basic idea of how the algorithm works.

The 6 different faces of the Cube are called U(p), D(own), R(ight), L(eft), F(ront) and B(ack). While U denotes an Up Face quarter turn of 90 degrees clockwise, U² denotes a 180 degrees turn and U' denotes a quarter turn of 90 degrees counter-clockwise. A sequence like U D R' D² of Cube moves is called a maneuver.

If you turn the faces of a solved cube and do not use the moves R, R', L, L', F, F', B and B' you will only generate a subset of all possible cubes. This subset is denoted by $G1 = \langle U, D, R^2, L^2, F^2, B^2 \rangle$. In this subset, the orientations of the corners and edges cannot be changed. That is, the orientation of an edge or corner at a certain location is always the same. And the four edges in the UD-slice (between the U-face and D-face) stay isolated in that slice.

In phase 1, the algorithm looks for maneuvers which will transform a scrambled cube to $G1$. That is, the orientations of corners and edges have to be constrained and the edges of the UD-slice have to be transferred into that slice. In this abstract space, a move just transforms a triple (x, y, z) into another triple (x', y', z') . All cubes of $G1$ have the same triple (x_0, y_0, z_0) and this is the goal state of phase 1.

To find this goal state the program uses a search algorithm which is called iterative deepening A* with a lowerbound heuristic function (IDA*). In the case of the Cube, this means that it iterates through all maneuvers of increasing length. The heuristic function $h_1(x, y, z)$ estimates for each cube state (x, y, z) the number of moves that are necessary to reach the goal state. It is essential that the function never overestimates this number. In Cube Explorer 2, it gives the exact number of moves which are necessary to reach the goal state in Phase 1. The heuristic allows pruning while generating the maneuvers, which is essential if you do not want to wait a very, very long time before the goal state is reached. The heuristic function h_1 is a memory based lookup table and allows pruning up to 12 moves in advance.

In phase 2 the algorithm restores the cube in the subgroup $G1$, using only moves of this subgroup. It restores the permutation of the 8 corners, the permutation of the 8 edges of the U-face and D-face and the permutation of the 4 UD-slice edges. The heuristic function $h_2(a, b, c)$ only estimates the number of moves that are necessary to reach the goal state, because there are too many different elements in $G1$.

The algorithm does not stop when a first solution is found but continues to search for shorter solutions by carrying out phase 2 from suboptimal solutions of phase 1. For example, if the first solution has 10 moves in phase 1 followed by 12 moves in phase 2, the second solution could have 11 moves in phase 1 and only 5 moves in phase 2. The length of the phase 1 maneuvers increase and the length of the phase 2 maneuvers decrease. If the phase 2 length reaches zero, the solution is optimal and the algorithm stops.

In the current implementation the Two-Phase-Algorithm does not look for some solutions that are optimal overall, those that must cross into and back out of phase 2. This increases the speed considerably. Use the Optimal Solver, if you want to prove some maneuver to be optimal.

Two-Phase-Algorithm and God's Algorithm:

God's number is 20

The algorithm which gives an optimal solution in the sense that there is no shorter solution is called God's algorithm. There are cube positions (for example the superflip which flips all 12 edges), which are known to have a shortest maneuver length of 20 moves to be solved. After 30 years it has finally been shown in July 2010, that all cube positions can be solved within 20 moves or less.

God's Number is 20

An overview is given on the webpage <http://cube20.org/>.

Our [proof](#) was recently published in SIAM Journal on Discrete Mathematics (Volume 27, Issue 2).

A tried to make a few webpages, which give more detailed information about the proof without being too technical. Some subgroups and their cosets play an important role here, and if you have some basic knowledge of group theory and combinatorics you should be able to follow the train of thoughts.

1. [The subgroup H and its cosets](#)
2. [The subgroup Q and its cosets](#)
3. [Solving a reduced set cover problem](#)
4. [A fast coset solver](#)

The Two-Phase-Algorithm gives near optimal solutions

I generated 1 million random cubes on a 3 GHz Pentium 4 PC, trying to find a cube which was not solvable within 20 moves with the Two-Phase-Algorithm.

But the Two-Phase-Algorithm solved all generated random positions within 20 moves. More precisely, it solved about 30000 random cubes per hour and the final distribution of the maneuver length was:

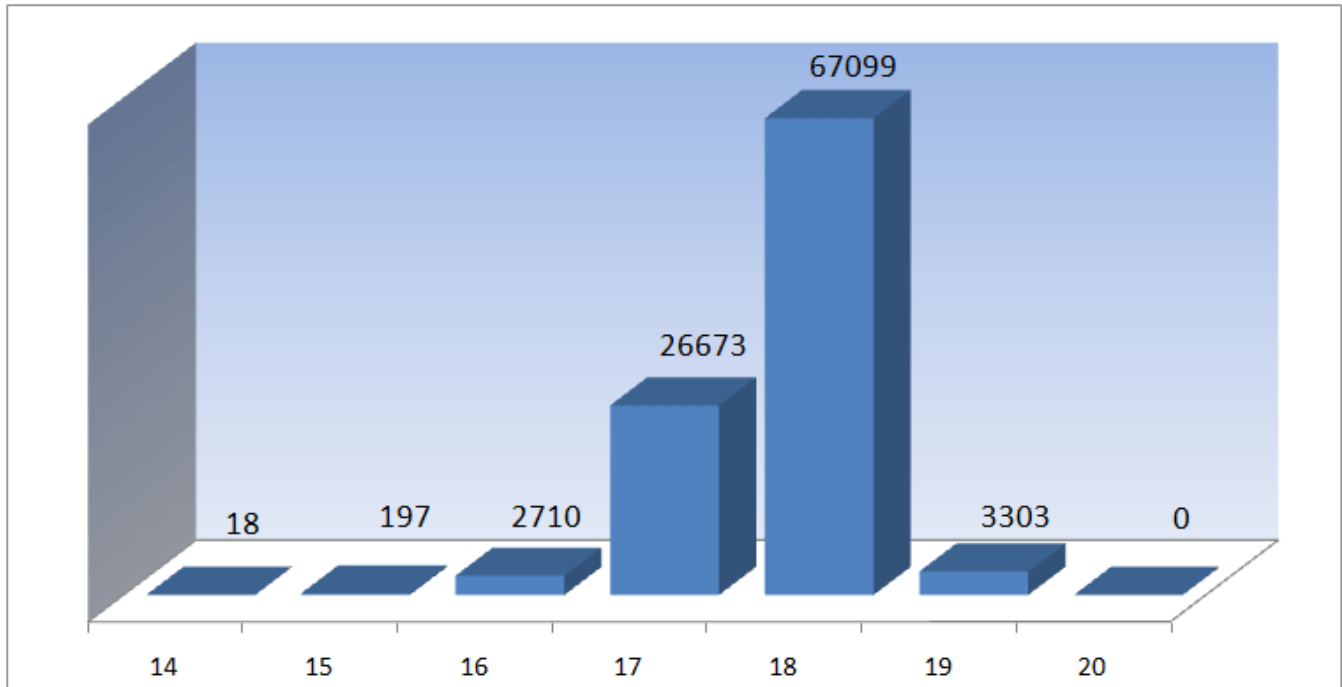
13: 4, **14:** 18, **15:** 81, **16:** 609, **17:** 3893, **18:** 23411, **19:** 141366, **20:** 830618.

Nevertheless the computation of God's number showed, that there some very rare positions which are quite hard to solve within 20 moves with the two-phase algorithm, for example the cube generated by the maneuver

L R2 U2 B' D2 L D2 F' U' R U2 L' F' D' R' B2 D2 R' U' F2 . On my machine this one takes 16 minutes (in triple search mode).

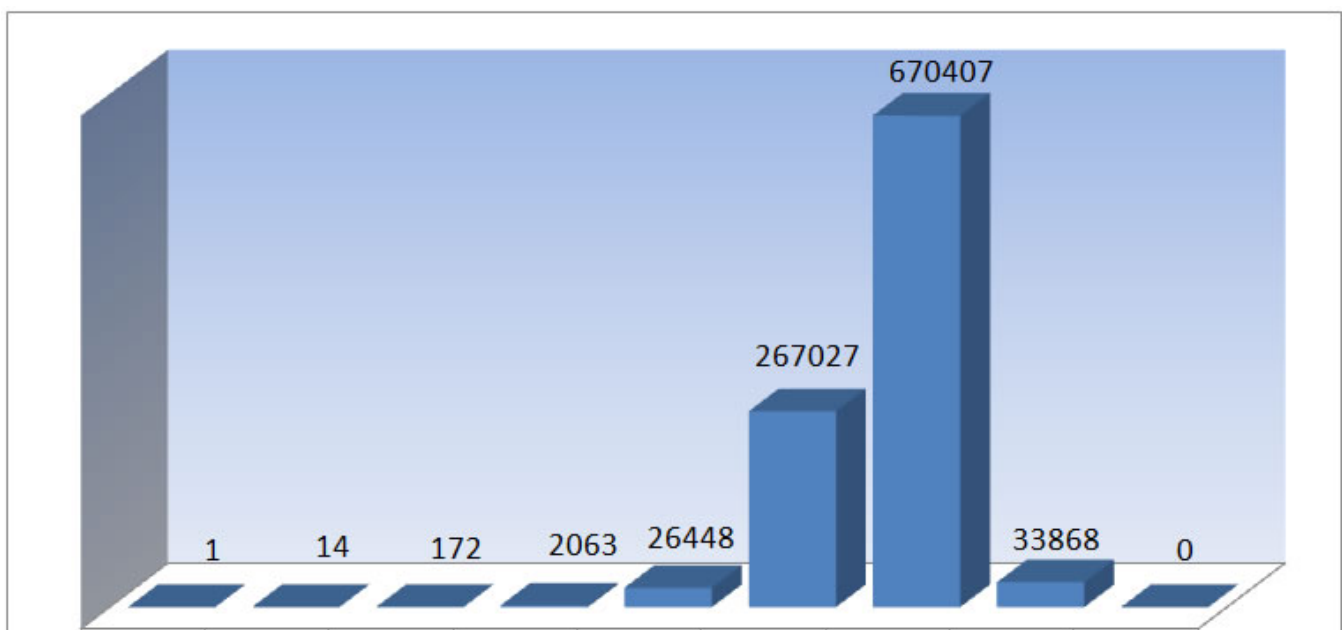
Optimal Cube Solver

In 2009 I used Cube Explorer 4.64s and an Intel Core i7 920 CPU machine (Vista 64 bit with 6 GB of RAM) to solve 100000 random positions **optimally** in parallel on 8 cores (4 physical and 4 virtual cores).



On average the program solved about 7000 cubes/day ! In this sense Cube Explorer's implementation of God's Algorithm does a decent job.. The average optimal solving length is ~17.7 . If you want to download the 100000 optimally solved cubes for some reason, you can do this [here](#).

In January 2010 Tomas Rokicki even solved 1 million random cubes optimally and got the following distribution, which is very close to the distribution I got:



12 13 14 15 16 17 18 19 20

As you can see, it is unlikely to find a cube position which really needs 20 moves by random. Presumably the chance is less than 10^{-11} , see [here](#) for details.

The first cube proved in 1995 to need 20 moves was the [superflip](#), a highly symmetric cube position.

Theoretical Probability Distribution

Though we now know, that all positions can be solved within 20 moves, the theoretical distribution for the maneuver length of God's Algorithm for Rubiks Cube is only known for maneuver lengths less than 16.

Because there are 1, 18, 243, 3240, 43239, 574908, 7618438, 100803036, 1332343288, 17596479795, 232248063316, 3063288809012, 40374425656248, 531653418284628, 6989320578825358, 91365146187124313 positions which have a shortest maneuver length of n moves for $n = 0$ to 15 (see [sequence A080601](#) in the [On-Line Encyclopedia of Integer Sequences](#)) and there are 43252003274489856000 different cube positions, we get for the probability P to solve a random cube optimally in n moves:

Maneuver Length n	Probability P	Branching Factor B
0	$2.31203 \cdot 10^{-20}$	18
1	$4.16166 \cdot 10^{-19}$	13.5
2	$5.61824 \cdot 10^{-18}$	13.3333
3	$7.49098 \cdot 10^{-17}$	13.3454
4	$9.99699 \cdot 10^{-16}$	13.2961
5	$1.32921 \cdot 10^{-14}$	13.2516
6	$1.76141 \cdot 10^{-13}$	13.2315
7	$2.3306 \cdot 10^{-12}$	13.2173
8	$3.08042 \cdot 10^{-11}$	13.2072
9	$4.06836 \cdot 10^{-10}$	13.1986
10	$5.36965 \cdot 10^{-9}$	13.1897
11	$7.08242 \cdot 10^{-8}$	13.1801
12	$9.33469 \cdot 10^{-7}$	13.1681
13	$1.22920 \cdot 10^{-5}$	13.1464
14	$1.61595 \cdot 10^{-4}$	13.0721
15	0.0021124	~12.8 (simulation)
16	simulation result: $\sim 0.0264 \pm 0.0003^*$	~10.1 (simulation)
17	simulation result: $\sim 0.267 \pm 0.0009^*$	~2.51 (simulation)
18	simulation result: $\sim 0.6704 \pm 0.0009^*$	~0.051 (simulation)
19	simulation result: $\sim 0.03387 \pm 0.0004^*$	probably below $3 \cdot 10^{-10}$
20	$P > 0$, but probably below 10^{-11}	0
> 20		-

	*: 95% confidence interval	
--	----------------------------	--

Important milestones on the path towards God's number

Phase 1 of the Two-Phase-Algorithm needs at most 12 moves and phase 2 needs at most 18 moves. Michael Reid showed in 1995, that the 18 moves case for phase 2 always can be avoided. So all cubes can be solved within [29 moves](#).

Gene Cooperman and Dan Kunkle claim in [this paper \(2007\)](#) to have proven that [26 moves](#) suffice, but there is yet a gap in the paper. This gap seems to be fixed meanwhile (August 2007, see Kunkles comment in the [Domain of the Cube Forum](#) but the corrected paper is not available.

Silviu Radu proved in [this paper](#) (also 2007) that [27 moves](#) suffice.

In May 2008 Tomas Rokicki proved, that [23 moves](#) suffice, analyzing more than 200000 cosets of the phase 2 subgroup of the Two-Phase-Algorithm ([Domain of the Cube Forum](#)). The method is similar to the method Rokicki describes in [this paper](#) (also 2008) for 25 moves.

In August 2008 Tomas Rokicki reduced the upper bound to 22 moves after having analyzed 1.28 million cosets within 50 core-years of CPU time ([Domain of the Cube Forum](#)). See [this well written paper](#) for details.

In July 2010 Morley Davidson, John Dethridge, Tomas Rokicki and me proved that [God's Number for the Cube is exactly 20](#).

Symmetric Patterns in Detail

You can search for cubes of all symmetry types with the Symmetry Editor module of Cube Explorer.

[Look here first](#) for the mathematical background of symmetric patterns and an explanation of the pictograms.

An external page with good information about the schoenflies symbols can be [found here](#).

We know God's algorithm for all the 164,604,041,664 symmetric cubes which exist. The following table gives the distribution:

Distance	Number	Distance	Number
0f	1	11f	9,732,164
1f	18	12f	35,024,904
2f	51	13f	122,054,340
3f	312	14f	436,197,214
4f	1,335	15f	1,763,452,505
5f	4,380	16f	8,035,307,127
6f	17,782	17f	37,542,012,922
7f	70,188	18f	95,387,902,305
8f	229,336	19f	21,267,102,443
9f	851,139	20f	1,091,994
10f	2,989,204	21f	0

Reducing the 1,091,994 symmetric cubes with 20 moves by symmetry and antisymmetry we find exactly 32,625 essentially different symmetric cubes which need 20 moves to be solved. They are included in the file [20moves.zip](#).

The details for the different symmetry types can be found below.

	Schoenflies-	Number of	Number of cubes	Shortest generator for exactly this	More
--	--------------	-----------	-----------------	-------------------------------------	------

type	Symbol	Symmetries	having at least this symmetry	symmetry	Information
	O_h	48	4	do nothing	yes
	O	24	4	---	yes
	T_d	24	4	---	yes
	T_h	24	24	U2 L2 F2 D2 U2 F2 R2 U2	yes
	T	12	72	B F L R B' F' D' U' L R D U	yes
	D_{3d}	12	16	U L D U L' D' U' R B2 U2 B2 L' R' U'	yes
	C_{3v}	6	48	U L' R' B2 U' R2 B L2 D' F2 L' R' U'	yes
	D_3	6	432	D B D U2 B2 F2 L2 R2 U' F U	yes
	S_6	6	7776	B' D' U L' R B' F U	yes
	C_3	3	3,779,136	L' R U2 R2 D2 F2 L R D2	yes
	D_{4h}	16	128	U2 D2	yes
	D_4	8	512	U D	yes
	C_{4v}	8	1024	D2	yes
	C_{4h}	8	1536	U D'	yes
	C_4	4	147456	U	yes
	S_4	4	442368	U R2 L2 U2 R2 L2 D	yes
	D_{2d} (edge)	8	3072	U F2 B2 D2 F2 B2 U	yes
	D_{2d} (face)	8	512	U R L F2 B2 R' L' U	yes
	D_{2h} (edge)	8	2048	U R2 L2 D2 F2 B2 U	yes
	D_{2h} (face)	8	12288	B2 D2 U2 F2	yes
	D_2 (edge)	4	98304	U F2 U2 D2 F2 D	yes

	D_2 (face)	4	294912	R2 L2 F B	yes
	C_{2v} (a1)	4	65536	U R2 L2 U2 F2 B2 U'	yes
	C_{2v} (a2)	4	1,179,648	R2 L2 U2	yes
	C_{2v} (b)	4	98304	B2 R2 B2 R2 B2 R2	yes
	C_{2h} (a)	4	589824	U' D F2 B2	yes
	C_{2h} (b)	4	98304	U R2 U D R2 D	yes
	C_2 (a)	2	15,288,238,080	L R U2	yes
	C_2 (b)	2	2,548,039,680	U R2 D' U' R2 U'	yes
	C_s (a)	2	18,345,885,696	F2 R2	yes
	C_s (b)	2	424,673,280	U B2 U D B2 D'	yes
	C_i	2	45,864,714,240	U D' R L'	yes
	C_1	1	43,252,003,274,489,856,000	U R	yes

G

Symmetry and Antisymmetry

You can search for cubes of all types of symmetry/antisymmetry types with the Symmetry Editor module of Cube Explorer.

[Look here first](#) for the mathematical background of symmetric patterns and an explanation of the pictograms.

An external page with good information about the schoenflies symbols can be [found here](#).

We use the symmetrygroup M of the cube with 48 elements to construct a group with 96 elements using the direct product $M \times C_2$, where $C_2 = \{1, a\}$ is the cyclic group of order 2. Now we are able to extend the concept of symmetry to the concept of antisymmetry.

While applying $(m, 1)$ to a cube c just means that we apply the symmetryoperation m to to cube, (m, a) means that we apply m first and then take the inverse of this cube. We call a cube c which is invariant under (m, a) antisymmetric. This is equivalent to the statement that applying m gives the inverse of the cube.

In a mathematically precise sense we define a group action of $M \times C_2$ on the set of cubes by

$(m, 1).y = m y m^{-1}$ and $(m, a).y = (m y m^{-1})^{-1}$ for any cube y . All cubes in the same orbit are related by conjugacy by whole-cube symmetry or/and inversion. In particular all cubes in an orbit share the same optimal maneuver length.

The inverse of a cube has nothing to do with the point reflection at the center of the cube which also is called an inversion. With the inverse of a cube c we mean the inverse regarding the permutation. The inverse of a cube generated for example with the maneuver $R U' L2$ then is $L2 U R'$.

While M has 33 different subgroups up to conjugation $M \times C_2$ has 131 different subgroups. This can be verified easily with [GAP](#):












```
M:= Group((1,2,3)(4,6,5),(2,3,5,4),(1,6)(2,5)(3,4));
M1:=DirectProduct(M,Group((1,2)));
Size(ConjugacyClassesSubgroups(M));
33
Size(ConjugacyClassesSubgroups(M1));
131
```

Each of the 131 subgroups of $M \times C_2$ defines an unique type of symmetry/antisymmetry. There are 3 different types of subgroups:
























1. If H is any of the 33 essentially different subgroups of M , $(H,1)$ is a subgroup of $M \times C_2$ which is isomorphic to H .
2. If H is a subgroup of M , $(H,1) \cup (H,a)$ also is a subgroup of $M \times C_2$. This gives 33 additional cases.
3. If H_1 and H_2 are subgroups of M with $H_1 \subset H_2$ and $|H_2| = 2|H_1|$, then $(H_1,1) \cup (H_2 \setminus H_1, a)$ is a subgroup of $M \times C_2$. $H_2 \setminus H_1$ denotes the difference set of H_2 and H_1 . We have 65 different types of this most interesting case.











Symmetries/Antisymmetries of type 3

In the animated gifs below, the blinking elements are those of $H_2 \setminus H_1$. Applying these elements of M give the inverse cube, while applying the not blinking elements of H_1 leave the cube unchanged.

	Type H	H1	H2	Subgroup size H	Size of conjugacy class	Number of cubes mod $M \times C_2$ with exactly this symmetry	Shortest generator for exactly this antisymmetry
3.1		O	O_h	48	1	0	---
3.2		T_d	O_h	48	1	0	---
3.3		T_h	O_h	48	1	4	B2 L R2 B2 F2 D2 U2 R' F2 D U2 B2 F2 L2 R2 U' (16f*)
3.4		T	O	24	1	6	B F L' R' B F D' U' L R D' U' (12f*)
3.5		T	T_d	24	1	6	B F L R B' F' D' U' L R D U (12f*)
3.6		C_{3v}	D_{3d}	12	4	8	D U R2 B D2 U2 L' D2 U2 R B' R2 D' U' (14f*)
3.7		D_3	D_{3d}	12	4	88	D B D U2 B2 F2 L2 R2 U' F U (11f*)
3.8		S_6	D_{3d}	12	4	1338	B' D' U L' R B' F U (8f*)
3.9		T	T_h	24	1	0	---
3.10		C_3	C_{3v}	6	4	43740	D' B2 D L2 D2 B2 U B2 U' (9f*)
3.11		D_4	D_{4h}	16	3	80	U D (2f*)







3.12		D_{2d} (face)	D_{4h}	16	3	80	$U' L R B_2 F_2 L' R' U'$ (8f*)
3.13		C_{4v}	D_{4h}	16	3	176	$F_2 R_2 D_2 F_2 L R U_2 F_2 L' R'$ (10f*)
3.14		C_{4h}	D_{4h}	16	3	448	$U D'$ (2f*)
3.15		D_{2h} (edge)	D_{4h}	16	3	0	---
3.16		D_{2d} (edge)	D_{4h}	16	3	496	$B F L_2 R_2 B' F' U_2$ (7f*)
3.17		D_{2h} (face)	D_{4h}	16	3	60	$B_2 L R_2 B_2 F_2 D_2 U_2 R' F_2 D' B_2 F_2 L_2 R_2 U'$ (15f*)
3.18		C_3	D_3	6	4	8780	$B_2 D' U' R_2 B_2 U_2 F_2 D U'$ (9f*)
3.19		D_2 (edge)	D_4	8	3	64	$L R B' F' U_2 B' F' U_2 L' R'$ (10f*)
3.20		C_4	D_4	8	3	2080	$U B_2 F_2 L_2 R_2 D'$ (6f*)
3.21		D_2 (face)	D_4	8	3	490	$B' F' L R$ (4f*)
3.22		C_{2v} (a1)	D_{2d} (face)	8	3	32	$L_2 B_2 U_2 B_2 R_2 U_2 L_2 D' U R_2$ (10f*)
3.23		D_2 (face)	D_{2d} (face)	8	3	490	$B F L R$ (4f*)
3.24		S_4	D_{2d} (face)	8	3	6144	$D' B_2 F_2 U_2 B_2 F_2 U'$ (7f*)
3.25		C_{2v} (a1)	C_{4v}	8	3	32	$U B_2 F_2 D_2 L_2 R_2 U'$ (7f*)
3.26		C_4	C_{4v}	8	3	4192	U (1f*)
3.27		C_{2v} (a2)	C_{4v}	8	3	1136	$L_2 R_2 D_2 B_2 F_2$ (5f*)
3.28		C_4	C_{4h}	8	3	448	$B F U B F L R D L R$ (10f*)
3.29		S_4	C_{4h}	8	3	1568	$B' F' D' B' F' L R D L R$ (10f*)
3.30		C_{2h} (a)	C_{4h}	8	3	0	---
3.31		C_{2v} (a1)	D_{2h} (edge)	8	3	4544	$B_2 F_2 U R_2 B_2 F_2 R_2 U' L_2 R_2$ (10f*)
3.32		C_{2v} (b)	D_{2h} (edge)	8	6	12800	$D B_2 D' U' B_2 D_2 U'$ (7f*)






3.33		C_{2h} (D)	U_{2h} (edge)	8	6	12800	$D' R2 D' U' R2 U'$ (6f*)
3.34		D_2 (edge)	D_{2h} (edge)	8	3	6688	$D' F2 D2 U2 F2 U'$ (6f*)
3.35		C_{2h} (a)	D_{2h} (edge)	8	3	10176	$R2 D2 U2 R2 D U'$ (6f*)
3.36		D_2 (edge)	D_{2d} (edge)	8	3	144	$L2 F' L2 D2 U2 R2 B' R2 D U$ (10f*)
3.37		S_4	D_{2d} (edge)	8	3	6224	$U L R U2 L R U$ (7f*)
3.38		C_{2v} (a2)	D_{2d} (edge)	8	3	192	$D' L2 B2 U2 B2 R2 U2 L2 D' U R2 U'$ (12f*)
3.39		C_3	S_6	6	4	364	$U' F2 R2 D U2 F2 D2 L2 U' B2 R2 U'$ (12f*)
3.40		D_2 (face)	D_{2h} (face)	8	1	2288	$D B2 D2 U2 F2 U$ (6f*)
3.41		C_{2h} (a)	D_{2h} (face)	8	3	14144	$D' B2 D2 U2 F2 U$ (6f*)
3.42		C_{2v} (a2)	D_{2h} (face)	8	3	29232	$U2 L2 B2 F2 R2$ (5f*)
3.43		C_s (b)	C_{2v} (a1)	4	6	481856	$B' L B L' R F' R' F$ (8f*)
3.44		C_2 (a)	C_{2v} (a1)	4	3	1,776,960	$B F U L R$ (5f*)
3.45		C_s (b)	C_{2v} (b)	4	6	481856	$R D' R' D B R'$ (6f*)
3.46		C_2 (b)	C_{2v} (b)	4	6	2,024,512	$U R2 F2 D U' R2 U$ (7f*)
3.47		C_s (a)	C_{2v} (b)	4	6	3,539,648	$R2 B2$ (2f*)
3.48		C_s (b)	C_{2h} (b)	4	6	481848	$U' R2 D B2 R2 B2 D B2 U'$ (9f*)?
3.49		C_2 (b)	C_{2h} (b)	4	6	2,024,424	$D' R2 D2 B2 D2 R2 U'$ (7f*)
3.50		C_i	C_{2h} (b)	4	6	3,695,238	$L' R B' F$ (4f*)
3.51		C_2 (b)	D_2 (edge)	4	6	592064	$U R2 U2 F2 D2 R2 U'$ (7f*)
3.52		C_2 (a)	D_2 (edge)	4	3	514384	$L R D' U B' F'$ (6f*)
3.53		C_2 (a)	C_4	4	3	10832	$B' F' U2 L R$ (5f*)
3.54		C_2 (a)	D_2 (face)	4	3	512816	$D L2 R2 U'$ (4f*)
3.55		C_2 (a)	S_4	4	3	3264	$U B' F' L' R' D$ (6f*)

3.56		C_2 (a)	C_{2h} (a)	4	3	1,765,392	D L R U (4f*)
3.57		C_s (a)	C_{2h} (a)	4	3	1,731,088	L2 B2 L2 D2 U2 (5f*)
3.58		C_i	C_{2h} (a)	4	3	1,863,344	D2 L2 B2 L2 U2 (5f*)
3.59		C_2 (a)	C_{2v} (a2)	4	3	1,766,720	D L R D (4f*)
3.60		C_s (a)	C_{2v} (a2)	4	6	3,474,976	F2 U2 B2 (3f*)
3.61		C_1	C_s (b)	2	6	108,272,809,188	R B (2f*)
3.62		C_1	C_2 (b)	2	6	21,419,485,172	R B' (2f*)
3.63		C_1	C_2 (a)	2	3	10,677,084,112	F U2 B' (3f*)
3.64		C_1	C_s (a)	2	3	54,180,798,352	U R D (3f*)
3.65		C_1	C_i	2	1	18,059,430,572	R U R L D L (6f*)

Symmetries/Antisymmetries of type 2











Cubes having this symmetry/antisymmetry are those of antisymmetry type 1 with the additional requirement that they are selfinverse.

	Type H	Schoenflies-Symbol	Subgroup size H	Size of conjugacy class	Number of cubes mod $M \times C_2$ with exactly this symmetry	Shortest generator for exactly this antisymmetry
2.1		O_h	96	1	4	Solved Cube (0f*)
2.2		O	48	1	0	---
2.3		T_d	48	1	0	---
2.4		T_h	48	1	2	U2 L2 F2 D2 U2 F2 R2 U2 (8f*)
2.5		T	24	1	0	---
2.6		D_{3d}	24	4	12	U L D U L' D' U' R B2 U2 B2 L' R' U' (14f*)



2.29		C_2 (b)	4	6	592040	U R2 U2 F2 U2 R2 U' (7f*)
2.30		C_s (a)	4	3	1,731,088	R2 F2 R2 (3f)
2.31		C_s (b)	4	6	481848	B2 D' F2 L2 B2 U' R2 (7f*)
2.32		C_i	4	1	616848	R2 U2 F2 U2 R2 (5f)
2.33		C_1	2	1	3,558,670,020	R U2 R' (3f)

Symmetries/Antisymmetries of type 1

These symmetries/antisymmetries directly correspond to the 33 symmetry types of the cube given [here](#). So it might be better to call cubes which are invariant under one of these subgroups of $M \times C_2$ symmetric instead of antisymmetric. But be aware that there is a subtle difference between a cube having the symmetries of a subgroup H of M and a cube having the symmetries of a subgroup $(H,1)$ of $M \times C_2$. For example entry 1.5 tells us that there are no cubes with symmetry $(T,1)$. But there are 12 cubes which exactly have the symmetry of the subgroup T of M if we ignore antisymmetries.

	Type H	Schoenflies-Symbol	Subgroup size H	Size of conjugacy class	Number of cubes mod $M \times C_2$ with exactly this symmetry	Shortest generator for exactly this antisymmetry
1.1		O_h	48	1	0	---
1.2		O	24	1	0	---
1.3		T_d	24	1	0	---
1.4		T_h	24	1	2	R2 D2 R B2 D U F2 R U2 R2 D' F2 L' R' F2 U' (16f*)
1.5		T	12	1	0	---
1.6		D_{3d}	12	4	0	---
1.7		C_{3v}	6	4	0	---
1.8		D_3	6	4	48	R F R' U R D B D U2 B2 F2 L2 R U' (14f*)
1.9		S_6	6	4	1260	D' U' B2 L2 D' U R2 F2 U2 (9f*)
1.10		C_3	3	4	444024	D U D2 U D U D2 U D D (10f*)

1.10		C_3	3	4	444034	$U L D_2 L B U D_2 U B U (10f)$
1.11		D_{4h}	16	3	0	---
1.12		D_4	8	3	16	$D U B_2 F_2 L_2 R_2 (6f^*)$
1.13		C_{4v}	8	3	48	$U_2 B_2 F_2 L_2 R_2 (5f^*)$
1.14		C_{4h}	8	3	96	$R_2 B R_2 D' B_2 F_2 L_2 R_2 U' R_2 B' R_2 (12f^*)$
1.15		C_4	4	3	14496	$U F_2 B_2 R_2 L_2 (5f^*)$
1.16		S_4	4	3	47536	$U' F' B' U_2 R_2 L_2 F' B' U (9f^*)$
1.17		D_{2d} (edge)	8	3	320	$F_2 L_2 R_2 U_2 L_2 R_2 U_2 F_2 U_2 (9f^*)$
1.18		D_{2d} (face)	8	3	16	$D' U' R_2 B_2 F_2 U_2 B_2 F_2 U_2 R_2 (10f^*)$
1.19		D_{2h} (edge)	8	3	0	---
1.20		D_{2h} (face)	8	1	310	$B' L_2 R_2 D_2 U_2 F' U_2 L' B_2 F_2 D_2 U_2 R' U_2 (14f^*)$
1.21		D_2 (edge)	4	3	6512	$U' R' L' F' B' R L U (8f^*)$
1.22		D_2 (face)	4	1	9604	$R L U_2 D_2 (4f^*)$
1.23		C_{2v} (a1)	4	3	3200	$R_2 L_2 U' F_2 B_2 D' F_2 R_2 L_2 F_2 (10f^*)$
1.24		C_{2v} (a2)	4	3	115504	$U_2 L_2 R_2 (3f^*)$
1.25		C_{2v} (b)	4	6	11264	$B' U' F_2 D L_2 D R_2 U' B_2 F R_2 L_2 (12f^*)$
1.26		C_{2h} (a)	4	3	52688	$R_2 L_2 U D' (4f^*)$
1.27		C_{2h} (b)	4	6	11264	$B_2 R_2 U' F_2 L_2 B' D' B R_2 F' U F (12f^*)$
1.28		C_2 (a)	2	3	951,909,808	$U R L (3f^*)$
1.29		C_2 (b)	2	6	315,851,984	$U L_2 D U L_2 U' (6f^*)$
1.30		C_s (a)	2	3	1,141,184,640	$U D' R_2 (3f^*)$
1.31		C_s (b)	2	6	52,088,192	$D R_2 D U R_2 U' (6f^*)$

1.32		C_i	2	1	952,378,138	R L U2 F2 U2 R2 ($6i^*$)
1.33		C_1	1	1	450,541,590,977,171,858	U R2 ($2f^*$)

How to count the number of "essentially" different cubes up to symmetry and inversion

In 2005 Mike Godfrey and me [obtained this number using the Lemma of Burnside](#). A direct analysis of the 131 different subgroup types not only verifies the number 450,541,810,590,509,978 but furthermore gives the number of essentially different cubes having a specific symmetry type H.

For any cube y , the stabilizer subgroup $\text{Stab}(y)$ is the subgroup H of $M \times C_2$ that fixes y and hence defines the symmetry/antisymmetry of y .

The orbit $\text{Orb}(y)$ for any cube y is the set of all cubes we get by applying the group action of all 96 elements of $M \times C_2$ to y . All cubes in $\text{Orb}(y)$ are equivalent, in particular they have the same maneuver length.

Let $S(H)$ be the union of all cubes which have the stabilizer subgroup H or a stabilizer subgroup conjugate to H . Then the orbit $\text{Orb}(y)$ of any element of $S(H)$ is a subset of $S(H)$. Each Orbit has the same number of elements, namely $|M \times C_2| / |H| = 96/|H|$.

If we divide $|S(H)|$ by this number we get the number $O(H)$ of different orbits of $S(H)$. We can interpret $O(H)$ as the number of essentially different cubes which have the symmetry/antisymmetry of type H .

$$O(H) = |S(H)| / |H| / 96$$

Though it seems difficult to classify all 43,252,003,274,489,856,000 different cubes regarding their stabilizer subgroup on first sight, separating corners and edges and the fact that most of the cubes are of type 1.33 make it possible to do this classification within a couple of hours of CPU-time. $O(H)$ is given in the column "Number of cubes mod $M \times C_2$ with exactly this symmetry".

If we add the counts $O(H)$ for all 131 cases we exactly get 450,541,810,590,509,978 in accordance with the result of 2005.

Cubes with Twisted Centers



Usually you can ignore the twists of the center facelets of the cube. But if there are pictures on the facelets you will not get the desired result if you just solve the cube in the usual way. There are 2048 possible ways the center facelets still can be twisted.

I used Cube Explorer to show that all 2048 possible center twist can be solved within 21 moves or less and there is essentially only one situation which really needs 21 moves in the usual face turn metric. If slice moves are allowed (slice turn metric) all situations can be solved within 18 moves and there is only one case which really needs 18 moves.

Up to symmetry and inversion there are exactly 73 nontrivial cases. They are listed below.

U R L F2 B2 R' L' D' R L F2 B2 R' L' (14f*) u+ d-

means for example, that the given maneuver has an optimal solution of 14 moves and twists the U-center facelet 90 degree clockwise and the D-center facelet 90 degree counterclockwise.

From the 73 maneuvers below you only need a combination of the first few to solve your cube, the others are more or less of theoretical interest. If you want to load them directly into Cube Explorer to analyse for example the symmetries, use the files [centertwists_f.txt](#) and [centertwists_s.txt](#) for the face turn or the slice turn metric.

U R L U2 R' L' U R L U2 R' L' (12f*,12s*)	u++
U2 L' R B2 D2 F2 L R' D2 B2 (10f*) M U2 B2 D2 M' D2 B2 U2 (8s*)	u++ f++
U2 R2 F2 B2 L2 D2 R2 F2 B2 L2 (10f*) L2 S2 L2 U2 R2 S2 R2 U2 (8s*)	u++ d++
U R L F2 B2 R' L' D' R L F2 B2 R' L' (14f*) M' E2 M U' M E2 M' U (8s*)	u+ d-
U F B L R' U' D' F' U D L' R F' B' (14f*) S' E' S U' S' E S U (8s*)	u+ f-

U R U R2 U' R' B2 L2 F2 D R2 F2 L B2 L (15f*) M E2 L2 F2 R' L' F' M' S2 R2 U2 L' R' U' (14s*)	u+ d+
U F U' D' F2 L' R F B U F' B' L R' F2 D (16f*) D' F D U F2 U2 M E R E' M' U2 F2 U' (14s*)	u+ f+
R L D2 R' L' D F2 R2 D2 L2 B2 R2 U2 B2 L2 D (16f*) z2 S E F2 L2 B2 E' S U' B F U2 B' F' U' (14s*)	u++ d++ f++
U R U D F' B R' L U' D F2 D' R L' F B' D' (17f*) y L2 S' E' S E L2 D M E M' U' (11s*)	u+ f++ r+
U D F2 U D' F R2 U2 D2 L2 B' U2 B2 R2 D2 L2 F2 (17f*) M S2 M D' M U M2 S2 D M' U' (11s*)	u++ f+ b+
U R U R2 U' D' L' F2 D2 B2 R' D L U D' B2 D' (17f*) x' B' L' E R2 E' M' U2 F2 U2 R D U R2 U' (14s*)	u++ f++ r++
F U R L' U' D R' L F B' R' L F' U D' R L' B' (18f*) S' E' S U' M S' E S M' U (10s*)	u+ f+ b++
U F B' R2 F B L2 B2 D' F2 R2 D2 R2 F2 R2 D2 B2 L2 (18f*) y2 x' D2 M E R2 E' M D2 F' R2 E2 M' U2 L' R' U' (15s*)	u+ d- f++
L' D2 B2 R' U2 B2 D2 L' F2 L2 U L U' L' U' L' U' L U (19f*) z x' L' B2 E R L F' D2 S' E' R2 S' D U R' U' (15s*)	u+ f++ r-
U B U' B' U' B' U' B U F2 R2 U2 L2 F B2 R2 D2 L2 B2 (19f*) z S' M2 B2 U2 F2 M S' L' S M F2 U2 B' F' U' (15s*)	u+ d++ f-
U2 F2 L R' U2 F2 B2 U2 L' R F2 D2 (12f*) S2 D2 M' U2 S2 D2 M' U2 (8s*)	u++ d++ f++ b++
U F2 L2 U F2 R2 B2 D F2 L2 D R2 B2 L2 (14f*) y M' E2 B2 M' B2 E' M D2 M' D' U' (11s*)	u++ d++ f++ l++
U F L' R U' D F B' L' U D' L R' F' B D' (16f*) z' E S E' B' M' E' M E F (9s*)	u+ d- f+ l-
U F' B L' R U R' F B' U' D L R' F L' D' (16f*) y' x' E' S E B' R S M' S' L' U (10s*)	u+ f+ r- l-
U F2 U2 F2 U2 L2 R2 B2 D' F2 U2 B2 U2 L2 R2 B2 (16f*) S E2 S' U M S' E2 S M' U' (10s*)	u+ d- f++ b++
U B F R' L U' D' B F' U D R L' B' F' D' (16f*) S M S M' S' E S' E' (8s*)	u+ d- f- b+
U L2 F2 B2 R2 D' F B R2 U2 F B L2 U2 F2 L2 (16f*) y' S' E2 S D S U2 S E S' D2 S U' (12s*)	u+ d- f++ l++
U F B L' R U' D' F2 B' U D L R' F' B' D2 (16f*) S E' S' U M2 S E S' M2 U' (10s*)	u+ d++ f++ b-
U B F R L' U' D' B' F' U D R' L B' F' D (16f*) y' E' S' E S U' D' S' E' S U2 (10s*)	u+ d+ f- b-
U F2 R2 L2 B2 D F2 U2 B U2 R2 L2 D2 F D2 B2 (16f*) S' E S U D M S' E' S M' D' U' (12s*)	u+ d+ f+ b+
U R' F2 R L' U' D F B' R' L' U R' L F' B D' (17f*) y' x' E' S E B' R2 S M' S' L' R' U (11s*)	u+ f++ r++ l-
U F U B L2 B R2 F2 U F2 R2 B2 U' D F' U' F2 (17f*)	u+ d+ f++ l++

y x M B2 M' S' R2 U2 L2 B R2 E2 M' U2 L' R' U' (15s*)	u+ u+ l++ l++
U D' F' B' L2 B2 U F B' U' D' B' L2 R2 F L2 D2 B (18f*) S' M B2 D S2 D' R2 F2 D' M S' L2 F2 R2 U' (15s*)	u+ f+ r++ l++
U L U D L2 D2 B' F L R' U L R U2 R2 B F' D' (18f*) L E M D2 F2 D' U' F E' S U2 L2 D' U' (14s*)	u+ d++ r++ l+
U L R F2 B2 U2 L' R' D' F B' L2 F B U2 D2 L2 F2 (18f*) x' M B2 M S2 L2 D2 L2 F R2 E2 M' D2 L' R' U' (15s*)	u+ d+ f++ b++
U' L' U L U L U L' U' R' L' U2 B2 D2 L B2 U2 F2 R2 (19f*) z L' D L R D2 S M' S' L2 B E S' D2 R2 U' (15s*)	u+ d++ f++ r+
U2 B2 L B2 D' U R L' B F D U F' R' L D U B' F' (19f*) y' E' B F M S R2 U' R L U2 L2 S' M' F' U2 (15s*)	u+ d+ f- l+
U F B L2 U D' F' B' U' D L2 F' L R F2 L' R' B' D (19f*) z' x E S M' E' F2 U2 B' F' D' L R U2 L' R' U' (15s*)	u+ d+ f+ b-
U F B' L' R' U F2 U2 D2 B2 L' R' F' B U D L R2 D (19f*) y' S M E S' E' M' D' M E' M' U (11s*)	u+ f++ b++ l-
U R' L' F' B U D R D' F B' R L U' D' R2 U2 F U2 R2 (20f*) z2 x' F' R2 B2 M' S M S2 U' B' F' U2 B2 M S' R' U' (16s*)	u+ d- f+ r+
U L R B' L2 D2 R2 F2 U D' L U D F' U2 R2 D2 B2 L R' (20f*) z' x E' M E R' F E M S M' E' B' U (12s*)	u+ f+ b+ l+
U F U2 D2 F B' R L F2 R L B U' D' F2 D (16f*) z x2 L' B2 L R F' D' U' B2 D' U' S' E2 F' U' (14s*)	u++ d++ f++ r++ l++
U R L U2 L2 U' D R' L D2 F2 U' D' R L' D' (16f*) E' M E M' U2 S' M' S M U2 (10s*)	u+ d+ f++ r+ l-
U R U D' F' B R L' D' F B2 U' D R' L F B' (17f*) x E M E' R2 L S E' S' E R (10s*)	u+ d- f+ b++ r+
U R L' B' U D F' B R' L U' D F2 R F B' D' (17f*) z' M E M' E' L2 F M' S M B' U2 (11s*)	u+ d+ f++ b- r+
U D R2 U D' L2 U2 F2 L F2 U2 D2 B2 R' D2 L2 F2 (17f*) z2 x' M S' M' F2 M' S' L E M2 E' R' U2 (12s*)	u++ f++ b++ r+ l+
U R U D' F' B R L' D' F2 B2 U' D R' L F B' D' (18f*) x' S M S M S2 L' E M E' R (10s*)	u+ d++ f++ b++ r+
U R L2 U D F' B R' L U' D F2 D R L' F B' D' (18f*) y x S2 E' S' E F R2 S' M S L R U' (12s*)	u+ d++ f++ r+ l++
F B U2 R2 L2 F' B' U R2 F2 U2 R2 F2 R2 D2 R2 F2 D (18f*) x2 S' E2 S U' B' L' R' F2 L R B' D' U' F2 U' (15s*)	u+ d- f++ r++ l++
U R L U2 L2 F U D F' B' R L' U D' F2 R' F' B D (19f*) y' E' B F M' S L2 D' M E' L2 B2 L' R' F' U2 (15s*)	u+ d+ f+ r+ l++
U F2 R L' U' D F B R' L' F' U D' R L F B R2 F2 (19f*) F2 M S R L U' D' L' S' M' B2 U2 B' F' U' (15s*)	u+ f+ b++ r- l-
U2 F' B' R2 U D B L' R' U' D F B R U' D' F2 L R (19f*) z' x S' M S R' F M' E S E' M B' U (12s*)	u+ d+ f++ b+ r-
U R2 F R2 B R2 U D R2 L F' B' U' R2 D' F' B' L' D (19f*) z' x' E' S' D2 M' F2 U D M B' R2 S' M D2 L R U' (16s*)	u+ d+ f- b- r++
U R2 F2 B U2 L F2 U2 D2 B2 R D2 B' L2 D F2 R2 L2 B2 (19f*)	u+ d+ f++ r+ l+

y x2 S' U2 M E' S M' F2 L2 B2 D L R U2 L' R' U' (16s*)	u+ d++ f++ r++ l-
U R F2 R' L B2 U D F B R' L U' R' L' F' B L2 D (19f*) z2 x' F' R2 E' F B R' S M B2 D2 S M F2 R' U' (15s*)	u+ d- f++ r+ l-
U R U D R2 U' D' L U D' R L F2 L2 U' D R L' D' (19f*) y x E R L F2 R2 E' S M D' L R D2 L' R' U' (15s*)	u+ d- f++ b- r+
U R U' D' F' B R L D R2 U2 B U' D R L' B2 D2 F' B' (20f*) z' R' B' E S D2 R2 U' D' R' S' E' S U2 F2 U' (15s*)	u+ f++ b- r++ l++
U D' F B U F' B' U' D' B' U2 F2 B L2 D2 B U2 B R2 B (20f*) z' D' S' E S D' F2 U' D' B' M' S R2 U2 L' R' U' (16s*)	
U R L U2 D2 R' L' F2 B2 U' D' R2 L2 D (14f*) S2 E M S2 M S M2 S' E' (9s*)	u++ d++ f++ b++ r++ l++
U L R' U D' L R' U' D B2 F2 L' R D' (14f*) S2 M' S' E M S' E' (7s*)	u+ d- f++ b++ r- l+
U R L U' D F2 R L' U D R' L B2 D' (14f*) y' E' M U2 S U D S' D2 M' U2 (10s*)	u+ d+ f++ b++ r+ l+
U R L F2 B2 R' L' D F B R2 L2 F' B' D2 (15f*) y M E2 M' D S2 E' M S2 M' U' (10s*)	u+ d- f++ b++ r++ l++
U2 R L D' F' B' R2 L2 D2 F B U' R' L' F2 B2 (16f*) y2 F B M2 D2 B' F' U' R L S2 D2 L' R' U' (14s*)	u+ d+ f++ b++ r++ l++
R L' U D' L F B' U D' R L' F' U D' F B' U2 D2 (18f*) S' E' S M E R' S M' S' R (10s*)	u+ d- f+ b++ r++ l-
U F L R' U' D F' B L R2 B2 U D' L' R F B' D (18f*) y2 S E' S' D2 M' E' R S' M S R' U2 (12s*)	u+ d+ f+ b++ r++ l+
R2 U2 L' F2 D2 U2 L2 R2 F2 R' U2 R2 D' L2 B2 F2 R2 U' (18f*) E' S' E S U2 S' E M E' S M' U2 (12s*)	u+ d+ f++ b++ r- l-
F U' D' B R L F' U' D' B R L F U' D' B' R L (18f*) M' E R2 B2 L2 S' D' M E' S L2 D M2 D' B2 R2 U' (17s*)	u+ d+ f+ b+ r- l-
U R' F B' D R L' F' U D' L F B' U' R L' B D' (18f*) S E' M' S' E M U S2 M S2 M' U' (12s*)	u+ d- f+ b- r+ l-
U R F' B' L2 B2 U D' R' L B R2 U' D F' B R' L' D (19f*) y' E' R B' F' L2 B2 E S' D B2 M S L' R' U2 (15s*)	u+ d+ f++ b- r+ l++
F U R2 F B' U D F' B L2 U' D' F B D F' B' D2 B (19f*) z' x2 R' F M' E S D2 L2 U2 M' E F D U F2 U' (15s*)	u+ d- f+ b+ r++ l++
U R2 U2 D2 L2 F B R L' U' D' B U' D' R' L F B D2 (19f*) y M E' M' D M2 E' M' S' E' S M' U' (12s*)	u+ d++ f++ b- r++ l++
F' U D B2 D2 F B' R' L U D' R' F B' U' D L2 B2 R' L' (20f*) z L' B2 R2 S' M' S D' M E' R2 B2 L R F' U' (15s*)	u+ d- f+ b++ r+ l++
U L R U2 D' B2 U R2 D L R' U D L2 R' F2 L D2 R D' (20f*) z x' L' F2 E' M E' L2 U' B' F' D2 M' S M' F2 R' U' (16s*)	u+ d++ f++ b++ r++ l+
U F L2 F R2 F U D' R' D2 R2 F B' U' F B' L2 D2 L' D' (20f*) y x' M' S' E' M E B R' S' M' S L' R2 U' (13s*)	u+ d++ f+ b++ r- l-
L R D' F' B' L2 R B2 L' F' B' U' L R D2 F L2 F2 B' U2 (20f*) z2 R U2 M E2 R2 B' E S' M E' S L2 U2 L' R' U' R (17s*)	u+ d+ f+ b- r- l+
U F2 R2 B2 L F2 U2 D2 B2 R' D R2 F2 L2 B R2 U2 D2 L2 F' (20f*) z' x F E M E' R' L' U2 S' E B' M' S2 R2 U2 L' R' U' R (18s*)	u+ d+ f+ b+ r+ l+

F U F B' R2 L U' D F' B2 R L' D' R L' B U D' R' B' (20f*) x' E M' E' L' E' S E F M S' M' F' R (13s*)	u+ d- f+ b+ r- l-
U D R' L U D' B R L' U D R' L B D2 L2 B' U2 D2 F R2 (21f*) y M' E M D M' S M F' E S' E' F U' (13s*)	u+ d+ f+ b+ r+ l-

Download

Cube Explorer 5.01 needs 128 MB of RAM and runs on all Windows platforms from Windows 98 to Windows 7 (32 bit or 64 bit).

[Download Cube Explorer \(938 kb\)](#)

I also have a special version **Cube Explorer 5.01s** available, which uses more than 2 GB of RAM for the huge optimal solver tables. It is about 15 times faster than the standard optimal solver and **optimally** solves a random cube in **less than two minutes** on average on a 3 GHz Pentium 4 machine.

With a 32 bit operating system, only Windows XP Professional supports a virtual address space of more than 2 GB for an application. Even with these versions you have to use the /3GB switch in the Boot.ini file to support more than 2 GB of RAM for an application (see [here](#) for details). You may download this special version [here](#).

If you run Windows Vista 64 bit, this version will run without any problems if you have about 4 GB of RAM installed. With an Intel Core i7 920 CPU I solved about 300 cubes optimally within one hour by doing the computations in parallel to keep all 8 cores (4 physical and 4 virtual cores) busy.

If you are interested in an **Optimal Cube Solver in the Quarter Turn Metric** which runs on the command line under **LINUX** and **WINDOWS**, you can download the documented **C source code** [here](#). An already compiled version for Windows is available [here](#). The program also accepts the file format of Cube Explorer, so you can generate your cubes in Cube Explorer and feed them to this program.

Bruce MacKenzie has ported this command line program to run on a **MAC** (nomen est omen). You can download the program [here](#).

A working version of the two-phase-algorithm is not too easy to program. For demonstration purposes I wrote a **Java package** which implements the two-phase-algorithm in its simplest form without any symmetry reductions.

The package org.kociemba.twophase, the sourcecode and the corresponding javadocs are included in the file [twohase.jar](#). The little Java program [GUI example.jar](#) (Version 2009.02.16).

which is an executable jar file shows an example how to use the package.

The tables in this implementation take only about 5 MB and are generated within seconds. Nevertheless the package routine solved about 26000 random cubes/hour if the maximum maneuver length was set to 21 moves and about 800 random cubes/hour if it was set to 20 moves maximum length.

You may use this package for free but you must include an appropriate credit line.

Last but not least I implemented the Two-Phase-Algorithm into a [Mathematica-package](#). The code runs very slow, but it is also very short. It might be interesting from a theoretical point of view.

The interactive editor function in the package needs at least Version 6.0 of Mathematica.

Cube Links

Here are a few of my favorite links concerning Rubik's Cube:

[Domain of the Cube Forum](#)

[Jaap's Puzzle Page](#)

[Michael Reid's Rubik's Cube Page](#)

[David Joyner's Homepage](#)

[Werner Randelshofer's Pretty Pattern Page](#)

[Speedsolving the Rubik's Cube & Other Puzzles](#)

[Jessica Fridrich's Speed Cubing Page](#)

[Speedcubing.com](#)

Introduction

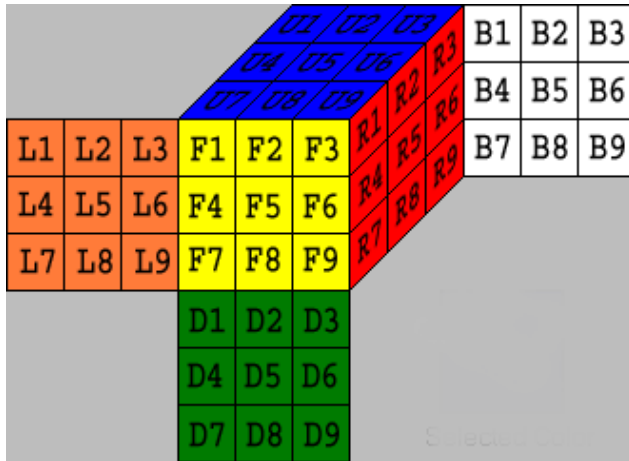
The following pages are an attempt to give some insight of the mathematical ideas and algorithms developed and used in Cube Explorer.

There are several problems I had to struggle with. First, English is not my native language and some of my explanations may be difficult to understand or incomprehensible at all. Second, I studied mathematics a long time ago and my terminology will surely be incorrect in some parts. Third, I only could sketch the main ideas of all that, what was necessary to write Cube Explorer.

But I hope that nevertheless it is a help for those who are interested in understanding the Two-Phase Algorithm or want to implement the algorithm in their own program.

Permutations and the Facelet Level

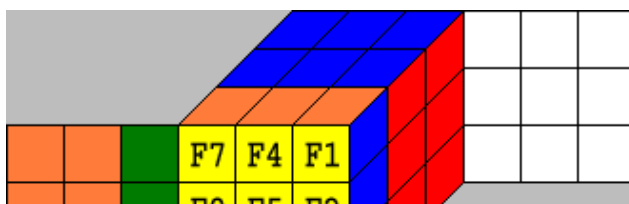
If we look at clean the cube, we see 6*9 facelets.

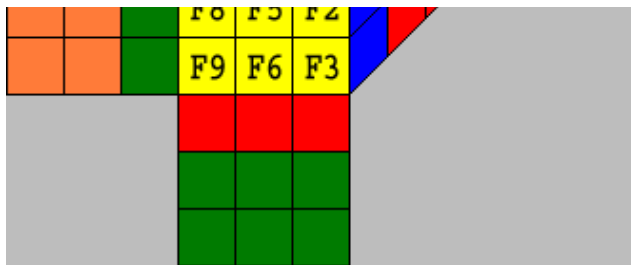


If we apply a move to the cube, the facelets are rearranged. Such a rearrangement is called a permutation.

We use the six letters U, R, F, D, B, L to describe the six 90° clockwise face movements. We use for example F2 to denote a 180° turn and F' to denote a 270° turn, i.e. a 90° turn anti-clockwise of the front face.

If we apply for example a F-move to the cube depicted above we get the following result:





To explain the representation of such permutations, we will only look at the yellow facelets for a moment. There are two possibilities for this representation in the example.

1. F1 is carried to F3 (F1→F3), F2→F6, F3→F9, F4→F2, F5→F5, F6→F8, F7→F1, F8→F4, F9→F7.
We can write

F1	F2	F3	F4	F5	F6	F7	F8	F9
F3	F6	F9	F2	F5	F8	F1	F4	F7

2. F1 is replaced by F7 (F1←F7), F2←F4, F3←F1, F4←F8, F5←F5, F6←F2, F7←F9, F8←F6, F9←F3. We can write

F1	F2	F3	F4	F5	F6	F7	F8	F9
F7	F4	F1	F8	F5	F2	F9	F6	F3

Because the first row of the tables is always the same, we can omit this row. So we can write just (F3,F6,F9,F2,F5,F8,F1,F4,F7) in the **is carried to** representation or (F7,F4,F1,F8,F5,F2,F9,F6,F3) in the **is replaced by** representation.

In most cases we will not use the short form without a table here for the sake of clearness.

We use the first representation on the facelet level, and the second on the cubie level.
In the rest of this chapter the **is carried to** representation is used.

We are able to define a **product of two permutations**.

For example

F1	F2	F3	F4	F5	F6	F7	F8	F9
F2	F1	F6	F3	F5	F4	F8	F7	F9

*

F1	F2	F3	F4	F5	F6	F7	F8	F9
F3	F6	F9	F2	F5	F8	F1	F4	F7

-

-

F1	F2	F3	F4	F5	F6	F7	F8	F9
F6	F3	F8	F9	F5	F2	F4	F1	F7

because for example $F1 \rightarrow F2$ by the first permutation and $F2 \rightarrow F6$ by the second, so we have $F1 \rightarrow F6$ in the product.

The multiplication of permutation has some similarities with the common multiplication with numbers, but there is one big difference: While for example $3 \cdot 5 = 5 \cdot 3$, you usually may not exchange the order of the two permutations.

But in the above example, we have

F1	F2	F3	F4	F5	F6	F7	F8	F9
F3	F6	F9	F2	F5	F8	F1	F4	F7

*

F1	F2	F3	F4	F5	F6	F7	F8	F9
F2	F1	F6	F3	F5	F4	F8	F7	F9

=

F1	F2	F3	F4	F5	F6	F7	F8	F9
F6	F4	F9	F1	F5	F7	F2	F3	F8

and this is something different. The multiplication of permutations [is not commutative](#).

Another important term is the [inverse permutation](#).

Consider the F-move

F1	F2	F3	F4	F5	F6	F7	F8	F9
F3	F6	F9	F2	F5	F8	F1	F4	F7

and the permutation

F1	F2	F3	F4	F5	F6	F7	F8	F9
F7	F4	F1	F8	F5	F2	F9	F6	F3

In this case

F1	F2	F3	F4	F5	F6	F7	F8	F9
----	----	----	----	----	----	----	----	----

F3	F6	F9	F2	F5	F8	F1	F4	F7
----	----	----	----	----	----	----	----	----

*

F1	F2	F3	F4	F5	F6	F7	F8	F9
F7	F4	F1	F8	F5	F2	F9	F6	F3

=

F1	F2	F3	F4	F5	F6	F7	F8	F9
F1	F2	F3	F4	F5	F6	F7	F8	F9

This last permutation does nothing at all, so when we multiply a permutation with its inverse, we get the [identity permutation I](#). In fact, in this example the second permutation is the representant of F' , so we have $F * F' = I$, which is quite obvious.

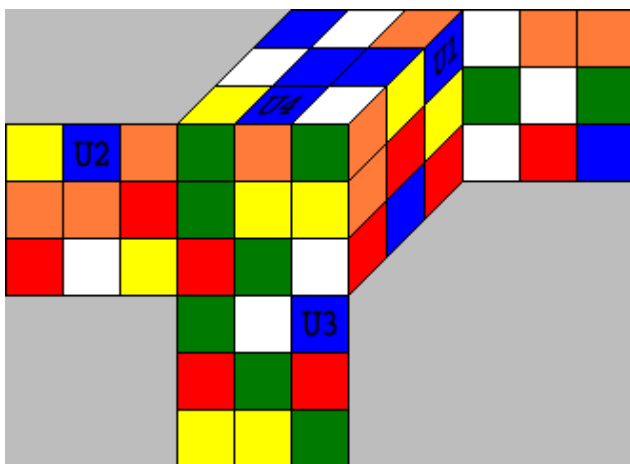
You can check, that also $F' * F = I$, so in this special case, the multiplication is commutative.

In the file [CubeDefs.htm](#) you can see the full definition of the basic moves. For example

$F = (U1, U2, U3, U4, U5, U6, R1, R4, R7, D3, R2, R3, D2, R5, R6, D1, R8, R9, F3, F6, F9, F2, F5, F8, F1, F4, F7, L3, L6, L9, D4, D5, D6, D7, D8, D9, L1, L2, U9, L4, L5, U8, L7, L8, U7, B1, B2, B3, B4, B5, B6, B7, B8, B9)$,

written in the short form without a table.

Not only the moves can be viewed as a permutations, every scrambled cubed can be written as a permutation.



Because $U1 \rightarrow R3$, $U2 \rightarrow L2$, $U3 \rightarrow D3$, $U4 \rightarrow U8, \dots$ this cube has the representation $(R3, L2, D3, U8, \dots)$.

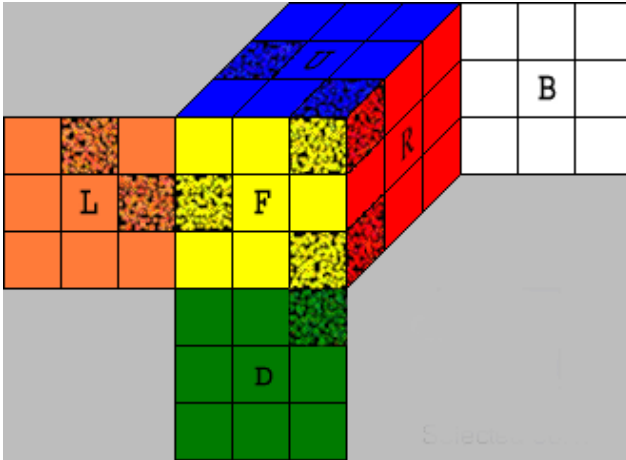
If you solve this cube, you in fact will try to find the inverse permutation of this cube composed as a product of the permutations corresponding to the elementary moves $U, U2, U', R, R2, R', \dots$. Recall that the product of a permutation with the inverse gives the identity permutation, and this is the clean cube. The solving algorithm of Cube Explorer tries to find short products for this inverse. For the example in the picture above it finds in a few seconds

$R^2 * L * U^2 * L^2 * D * R^2 * U^2 * L' * D^2 * R' * U * B * R' * F^2 * L^2 * B^2 * L^2 * B^2$

But representing permutations on the facelet level is not effective for a fast computation. There are two more levels to cope.

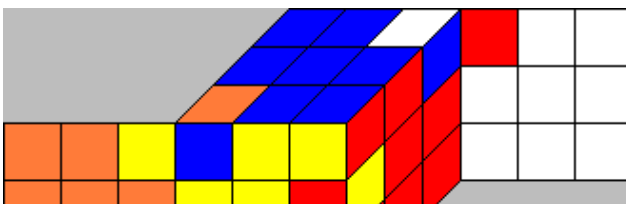
The Cubie Level

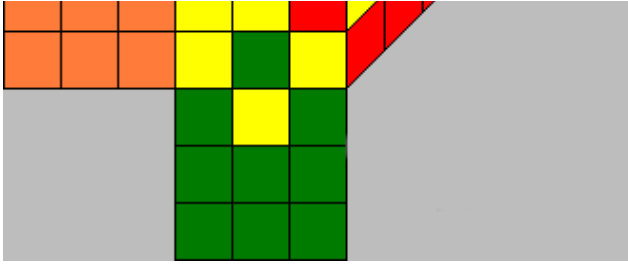
On the cubie level, the objects we permute are not the facelets, but the 12 edges and the 8 corners.



In the picture above, the URF-corner, the DFR-corner, the FL-edge and the UL-edge are marked. The corners are named URF, UFL, ULB, UBR, DFR, DLF, DBL and DRB. The edges are named UR, UF, UL, UB, DR, DF, DL, DB, FR, FL, BL and BR.

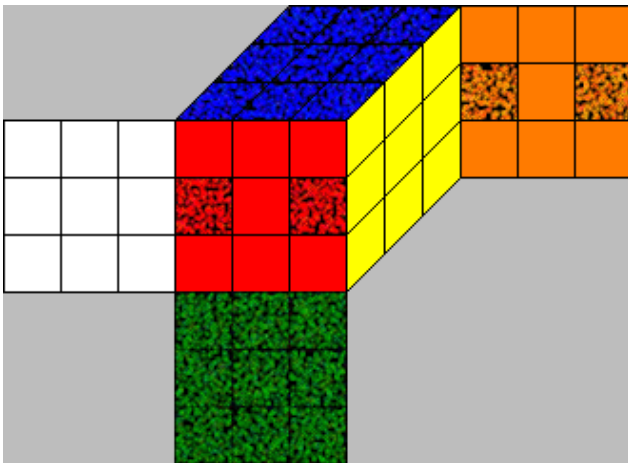
On the cubie level it is not possible to represent a move or a scrambled cube by a simple permutation, because the corners can be twisted and the edges can be flipped.



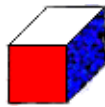


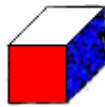
Here the cubies **are in their home-positions**, but the orientations have changed. The UFL-corner ist twisted clockwise, the UBR-corner is twisted anti-clockwise and the DF-edge and the FR-edge are flipped.

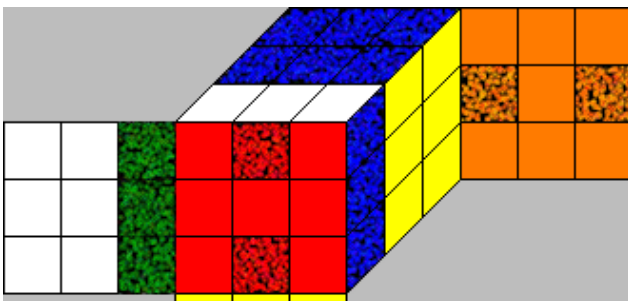
If the corners or edges **are not in their home positions**, there are many ways to define the orientations of the cubies. But for the Two-Phase-Algorithm, the following definiton is necessary.

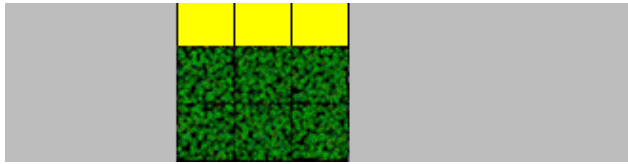


The **marked facelets** on the clean cube are the **reference** for the orientation.



In the picture, the corner  at the place URF is twisted clockwise **relative to the reference** facelet on the clean cube, also the corner at the place DLF. The corners at the places UFL and DFR are are twisted anti-clockwise. The edges sitting in the UF, DF, FL and FR positions are flipped.





The F-move

We use the "is replaced by" representation to write the permutations on the Cubie Level. In the above example: URF is replaced by UFL(URF<-UFL), UFL<-DLF, ULB <-ULB, UBR<-UBR, DFR<-URF, DLF<-DFR, DBL <-DBL, DRB<-DRB. We write

URF	UFL	ULB	UBR	DFR	DLF	DBL	DRB
UFL	DLF	ULB	UBR	URF	DFR	DBL	DRB

in this case for the permutation of the corners.

But we have also to keep track of the changes of the orientations, and so we write

F =

URF	UFL	ULB	UBR	DFR	DLF	DBL	DRB
c:UFL;o:1	c:DLF;o:2	c:ULB;o:0	c:UBR;o:0	c:URF;o:2	c:DFR;o:1	c:DBL;o:0	c:DRB;o:0

We use "0" if the twist does not change, "1" for a clockwise twist and "2" for a anti-clockwise twist. In this way we can add orientations in a simple way. If we do for example two anti-clockwise twist, the resulting twist is 2+2=4, and because 4 = 1 mod 3 the result is a clockwise twist. Take a look at [CubeDefs.htm](#) for the definition of the basic moves. The permutation of the edges is defined similar, with "1" for the orientation of a flipped edge and "0" for an unflipped edge.

We also need a notation to describe a permutation without using a table.

For the F-move above we write for example

$$F(URF).c = UFL$$

$$F(URF).o = 1$$

$$F(UFL).c = DLF$$

$$F(UFL).o = 2$$

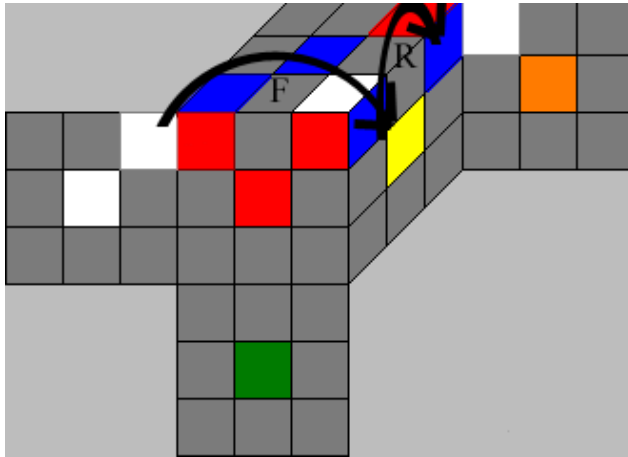
etc.

We use another move

R =

URF	UFL	ULB	UBR	DFR	DLF	DBL	DRB
c:DFR;o:2	c:UFL;o:0	c:ULB;o:0	c:URF;o:1	c:DRB;o:1	c:DLF;o:0	c:DBL;o:0	c:UBR;o:2

to show how to define the product F*R of these two permutations including the orientations.



F*R applied to the UFL-corner

$F(\text{URF}).c = \text{UFL}$ and $F(\text{URF}).o = 1$ in the table above tells us that the corner at position URF is replaced by the corner at position UFL and that the orientation of the corner which moves to the position URF is increased by 1 when performing a F -move.

$R(\text{UBR}).c = \text{URF}$ and $R(\text{UBR}).o = 1$ tells us, that the corner at position UBR is replaced by the corner at position URF and the orientation increases by 1 when performing a R -move.

So when performing F^*R we have $\text{URF} \leftarrow \text{UFL}$ by the F -move and then $\text{UBR} \leftarrow \text{URF}$ by the R -move, which in the result is $\text{UBR} \leftarrow \text{UFL}$. So as result we have the $(F^*R)(\text{UBR}).c = \text{UFL}$. This means that $(F^*R)(\text{UBR}).c = F(R(\text{UBR}).c).c$.

The behavior of the orientation is more difficult to understand. F tells us that $F(\text{URF}).o = 1$ when $\text{URF} \leftarrow \text{UFL}$. This means, the orientation of the corner which moves from position UFL to position URF increases by 1. This orientation adds to the change of the orientation of the corner which moves from the URF to the UBR position ($\text{UBR} \leftarrow \text{URF}$) by the following R -move. So we have $F(\text{URF}).o + R(\text{UBR}).o$ for the resulting orientation change at position UBR, and because $\text{URF} = R(\text{UBR}).c$ we have $(F^*R)(\text{UBR}).o = F(R(\text{UBR}).c).o + R(\text{UBR}).o$.

In general, for two permutations A and B and for any corner position x we have

$$(A^*B)(x).c = A(B(x).c).c$$

and

$$(A^*B)(x).o = A(B(x).c).o + B(x).o$$

The same principle holds for the edge permutations. See [CubeDefs.htm](#) for the implementation of the multiplication routines.

If we want also want to include the case of reflections, which we need if we apply symmetries of the cube, things are a bit more complicated with the orientations of the corners. Instead of adding modulo 3 in the second equation above, which can be interpreted as a group operation in the cyclic group C_3 , we then work in the dihedral group D_3 . We describe the three extra elements in this group with the numbers 3, 4, and 5.

The Coordinate Level

On the coordinate level we describe the permutations and the orientations of the corners and edges by natural numbers. This level of abstraction is especially suited to implement a fast algorithm to solve the cube.

The definition of the corner orientation coordinate

If we apply for example the move R to a clean cube we get

URF	UFL	ULB	UBR	DFR	DLF	DBL	DRB
c:DFR;o:2	c:UFL;o:0	c:ULB;o:0	c:URF;o:1	c:DRB;o:1	c:DLF;o:0	c:DBL;o:0	c:UBR;o:2

The orientation of the 8 corners are described by a number from 0 to 2186 ($3^7 - 1$).

In cubicube.pas you find the following definition

```
function CubieCube.CornOriCoord:Word;
var co: Corner; s: Word;
begin
  s:=0;
  for co:= URF to Pred(DRB) do s:= 3*s + PCorn^[co].o;
  Result:= s;
end;
```

In the example above this functions computes

$$2*3^6 + 0*3^5 + 0*3^4 + 1*3^3 + 1*3^2 + 0*3^1 + 0*3^0 = 1494$$

This is just the number **2001100** in a ternary number system.

To make this easy method work we must write the permutation in the in the **is replaced by** representation. It will not work in the **is carried to** representation!

We ignore the orientation of the DRB-corner, because this orientation is determined by the orientations of the other seven corners: The sum of all orientations must be divisible by three.

The definition of the edge orientation coordinate

The edge orientation coordinate is defined in an analogous way:

The edge orientation coordinate is defined in an analogous way.

The orientation of the 12 edges is described by a number from 0 to 2047 ($2^{11} - 1$).

In `cubicube.pas` you find the following definition

```
function CubieCube.EdgeOriCoord: Word;
var ed: Edge; s: Word;
begin
  s:=0;
  for ed:= UR to Pred(BR) do s:= 2*s + PEdge^[ed].o;
  Result:= s;
end;
```

We use the binary number system instead of the ternary number system here. We ignore the orientation of the BR-edge because it is determined by the orientations of the other 11 edges. The sum of all orientations must be even.

The definition of the corner permutation coordinate

The corner permutation coordinate is given by a number from 0 to 40319 ($8! - 1$).

In this example, we use the permutation of the R-move again, but we ignore the orientations now.

URF	UFL	ULB	UBR	DFR	DLF	DBL	DRB
c:DFR	c:UFL	c:ULB	c:URF	c:DRB	c:DLF	c:DBL	c:UBR
	1	1	3	0	1	1	4

We define a natural order on the corners by $URF < UFL < ULB < UBR < DFR < DLF < DBL < DRB$.

The number in the third row - below a corner XXX in the second row - gives the number of all corners left of XXX, whose orders are higher than the order of XXX.

Above the entry **4** we have for example the corner UBR.

From the 7 corners left of UBR, 4 corners have a higher order - DFR, DLF, DBL, DRB.

Above the entry **1** we have for example the corner DLF.

From the 5 corners left of DLF, only 1 corner has a higher order - DRB.

We build the permutation coordinate with the numbers of the third row.

$$1*1! + 1*2! + 3*3! + 0*4! + 1*5! + 1*6! + 4*7! = 21021$$

So we use a system with variable base here.

The following function from `cubicube.pas` does the job.

```
function CubieCube.CornPermCoord: Word;
var i,j: Corner; x,s: Integer;
begin
  x:= 0;
```

```

for i:= DRB downto Succ(URF) do
begin
  s:=0;
  for j:= Pred(i) downto URF do
  begin
    if PCorn^[j].c>PCorn^[i].c then Inc(s);
  end;
  x:= (x+s)*Ord(i);
end;
Result:=x;
end;

```

The definition of the edge permutation coordinate

The edge permutation coordinate is described in an analogous way by a number from 0 to $12! - 1$.

We use the following function from cubicube.pas:

```

function CubieCube.EdgePermCoord: Integer;
var i,j: Edge; x,s: Integer;
begin
  x:= 0;
  for i:= BR downto Succ(UR) do
  begin
    s:=0;
    for j:= Pred(i) downto UR do
    begin
      if PEdge^[j].e>PEdge^[i].e then Inc(s);
    end;
    x:= (x+s)*Ord(i);
  end;
  Result:=x;
end

```

Now we are able to describe each cube with a tuple (x_1, x_2, x_3, x_4) and

$$0 \leq x_1 < 3^7, \quad 0 \leq x_2 < 2^{11}, \quad 0 \leq x_3 < 8!, \quad 0 \leq x_4 < 12!$$

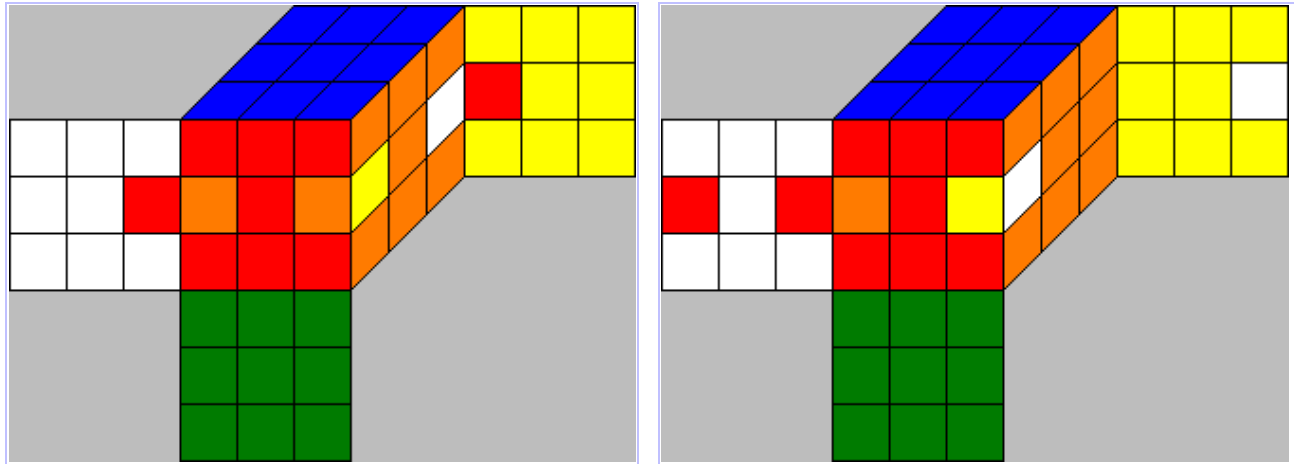
Only half of these cubes are really possible to generate because all achievable permutations are even and so we have only $12! * 8! / 2$ permutations (ignoring the orientations).

The number of different cubes is therefore given by

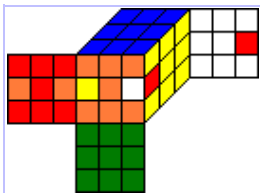
$$3^7 * 2^{11} * 8! * 12! / 2 = 43.252.003.274.489.856.000$$

There are some other coordinates we use for the Two-Phase-Algorithm which we introduce later.

Equivalent Cubes and Symmetry



Look at the two cubes above. They look different, but basically they are the same. If you turn the whole cube in the left picture 90 degrees around an axis through the U-center and D-center cubies, you get



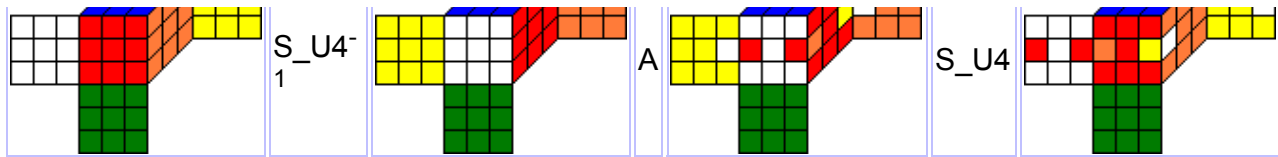
this cube, and if you recolor the facelets again so that the color of the F-face is red again etc. you get the cube in the right picture. We call these two cubes equivalent.

Because equivalent cubes have the same structure, the number of moves necessary to solve them is the same.

Defining equivalent cubes with the aid of recoloring of facelets is not really what we want, because we move back to the facelet level. We prefer to define the equivalence on the cubie level with permutations. With S_{U4} we denote the 90 degrees turn through the U-center and D-center cubies of the whole cube. Lets denote the permutation which defines the left cube above with A and the permutation for the right cube above with B.

Then we have





So we have $B = S_U4^{-1} * A * S_U4$ in this example. In general, two cube permutations A and B are equivalent, if there is a symmetry S of the cube with

$$B = S^{-1} * A * S$$

For each cube there are up to 48 equivalent cubes, because the cube has 48 symmetries including reflections. In Cube Explorer, these 48 symmetries are generated by four "basic" symmetries:

S_URF3 , a 120 degree turn of the cube around an axis through the URF-corner and DBL-corner,
 S_F2 , a 180 degree turn of the cube around an axis through the F-center and B-center,
 S_U4 , a 90 degree turn of the cube around an axis through the U-center and the D-center
 S_LR2 , a reflection at the RL-slice plane.

These basic symmetries are [permutations of the corners](#) and [permutations of the edges](#) and are described in cubedefs.htm.

Any of the 48 symmetries is uniquely generated by the product

$$(S_URF3)^{x1} * (S_F2)^{x2} * (S_U4)^{x3} * (S_LR2)^{x4}$$

with $x1$ from 0..2, $x2$ from 0..1, $x3$ from 0..3 and $x4$ from 0..1. This tuple $(x1, x2, x3, x4)$ is mapped to a natural number from 0..47 by

$$16 * x1 + 8 * x2 + 2 * x3 + x4$$

In this way each of the symmetries has an associated index from 0..47. With $S(i)$ we denote the symmetry which belongs to the index i .

Two cubes with the permutations A and B are equivalent if and only if there is an i with

$$S(i)^{-1} * A * S(i) = B$$

All cubes which are equivalent, belong to the same equivalence class.

In Cube Explorer the $S(i)$ are implemented in the arrays CornSym and EdgeSym in the unit symmetries.pas

A mathematical view of Coordinates: Cosets

If you have a group G and a subgroup H , then for each g from G the set $\{a * g \mid a \text{ from } H\}$ is called a right coset of H .

Each scrambled cube can be seen as a permutation with attached orientations. All these permutations define the group G .

Every coordinate-value used in Cube Explorer can be mapped onto a right coset, where the subgroup H mentioned above is defined by the type of the coordinate.

The following table describes the subgroups H , which correspond to the various types of coordinates used in Cube Explorer.

subgroup of G	coset coordinate	used in
all permutations with corner orientations = 0	corner orientations coordinate range 0..2186	phase 1, optimal solvers
all permutations with edge orientations = 0	edge orientation coordinate range 0..2047	phase 1, optimal solvers
all permutations which leave the four UD-slice edges in their slice	UDSlice coordinate range 0..493	phase 1, optimal solvers
all permutations with edge orientations = 0 and which leave the four UD-slice edges in their slice.	FlipUDSlice coordinate range 0..494*2048 - 1	phase 1, standard optimal solver
all permutations which leave the corners in their place with arbitrary twist.	corner permutation coordinate range 0..40319	phase 2, optimal solvers
all permutations which leave the 8 edges of the U-face and D-face in their place with arbitrary flip	phase 2 edge permutation coordinate range 0..40319	phase 2
all permutations which leave the four UD-slice edges in their place with arbitrary flip	UDSliceSorted coordinate range 0..11879	phase 2, optimal solvers

Some of the above coordinates are "reduced by symmetries" in a second step before actually being used. We will discuss this in the next chapter.

Take for example the subgroup C_0 which defines the corner orientation coordinate

$$C_0 = \{ g \text{ from } G \text{ with } g(x).o = 0 \text{ for all corners } x \}$$

In this case the right cosets are defined by

$$C_0 * g = \{a * g \mid a \text{ from } C_0\}$$

For each element a from C_0 and an element g from G and any corner x we have regarding the [definition of the multiplication](#)

$$(a * g)(x).o = a(g(x).c).o + g(x).o = 0 + g(x).o = g(x).o$$

So all elements of the coset $C_0 * g$ have the same corner orientations (defined by the permutation g) and all elements of $C_0 * g$ have the same [corner orientation coordinate](#). And if on the other hand two permutations have the same corner orientation coordinate, they are in the same coset (we omit the proof here). So there is a one to one mapping between the corner orientation coordinate and the cosets defined by C_0 .

Coordinates and Symmetry

Coordinates represent cosets, each coset usually consists of many permutations.

If you move the whole cube and recolor it or you do a conjugation with a symmetry $S(i)$ as [explained two chapters ago](#), the coordinates usually change.

You must be careful if you want to map a coordinate by a symmetry conjugation to another coordinate. If you have two different permutations P and Q in a coset, $S(i)^{-1}P*S(i)$ and $S(i)^{-1}Q*S(i)$ always have to be in the same coset, else you cannot do this mapping nor can you define equivalent cosets.

This restricts the symmetries which are applicable here. It is not difficult to show that exactly those symmetries $S(i)$ are applicable, for which the subgroup H which defines the cosets has the property $S(i)^{-1}H*S(i) = H$.

The following table shows, which symmetries are applicable to which coordinates and where this is used.

coordinate	full symmetry group with 48 elements	subgroup generated by S_F2, S_U4 and S_LR2 with 16 elements	used in
corner orientation (twist)	no	yes	phase 1, optimal solvers
edge orientation (flip)	no*	no*	-----
UDSlice	no	yes	-----
FlipUDSlice	no	yes	phase 1, optimal solver, 64430 equivalence classes
corner permutation	yes**	yes	phase 2, 2768 equivalence classes
phase 2 edge permutation	no	yes	phase 2
UDSliceSorted coordinate	no	yes	huge optimal solver, 788 equivalence classes

*It is possible to give another definition for the edge orientation, so that the full symmetry group

It is possible to give another definition for the edge-orientations, so that the full symmetry group can be used with the edge orientation coordinate. But we prefer the usual definition which is better suited for the two-phase algorithm.

**Not used in Cube Explorer

For the FlipUDSlice coordinate, the corner permutation coordinate and the UDSliceSorted coordinate, the number of equivalence classes is given in the table.

Take for example the FlipUDSlice coordinate. The range of this coordinate x is $0..495*2048-1$. But by symmetry conjugation these 1.013.760 coordinates are "reduced" to 64430 equivalence classes.

Up to 16 coordinates belong to each equivalence class - 16 and not 48 because we only use symmetries which preserve the UD-axis. [For each equivalence class we store the smallest coordinate as the representant of this class in an integer array](#) of size 64430. In general we call this array the [ClassIndexToRepresentantArray](#).

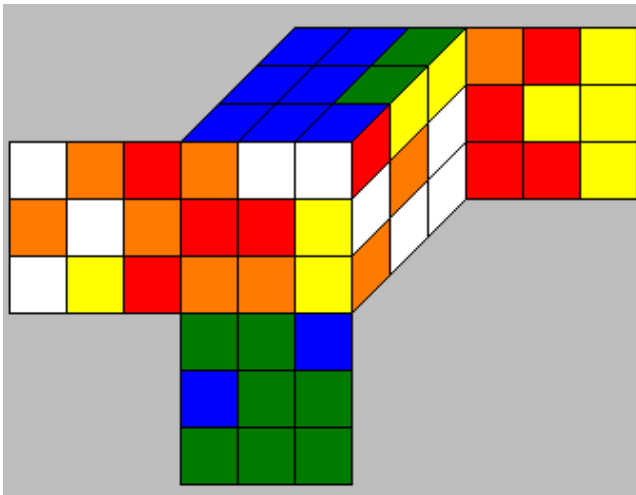
The old integer coordinate x is then substituted by a new coordinate $16*y + i$, where y is the index of the equivalence class it belongs to ($0..64429$) and i is the index of a symmetry ($0..15$) which transforms the coordinate of the representant to x . To distinguish the old and new coordinate in the text, we will call the old coordinate a [raw-coordinate](#) and the new coordinate a [sym-coordinate](#) in the following text.

This sym-coordinate has $64430*16 = 1.030.880$ different values, which is more than $495*2048=1.013.760$. The reason is, that for some cubes with symmetries there belong less than 16 original coordinates to one equivalence class and the replacement by the sym-coordinate is not unique because $16*y + i1$ and $16*y + i2$ describe the same raw-coordinate x for some $i1$ and $i2$.

The Two-Phase Algorithm Coordinates

If you turn the faces of a solved cube and do not use the moves R, R', L, L', F, F', B and B' you will only generate a subset of all possible cubes. This subset is denoted by $G1 = \langle U, D, R2, L2, F2, B2 \rangle$.

A typical representant of $G1$ looks like this:



In this subset, the orientations of all corners and all edges are 0. And the four edges in the UD-slice (between the U-face and D-face) stay isolated in that slice.

On the other hand, if the orientations of all corners and all edges is 0, and the four edges of the UD-slice are in their slice, we have an element of $G1$.

The Two-Phase Algorithm solves the Cube in to steps.

In **phase 1**, the algorithm looks for maneuvers which will transform a scrambled cube to $G1$. That is, the orientations of corners and edges have to be constrained and the edges of the UD-slice have to be transferred into that slice. In **phase 2** we restore the cube.

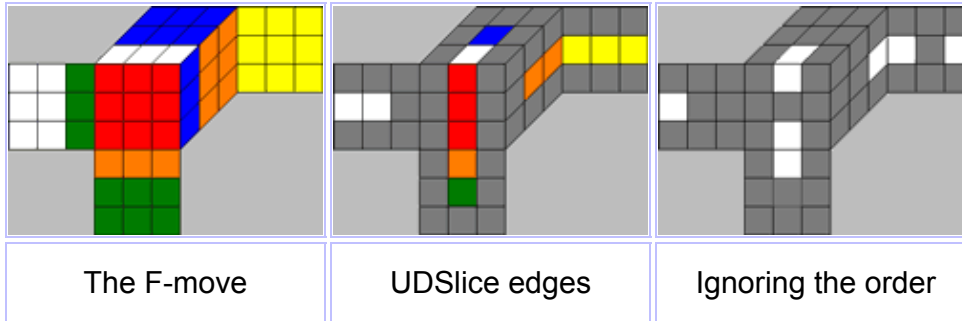
There are many different possibilites for maneuvers in phase 1. The algorithm tries different phase 1 maneuvers to find a most possible short overall solution.

In **phase 1**, any cube is described with three coordinates:

In the [corner orientation coordinate \(0..2186\)](#), the [edge orientation coordinate \(0..2047\)](#) and [UDSlice coordinate](#)

The **UDSlice coordinate** is number from 0 to 494 ($12 \cdot 11 \cdot 10 \cdot 9 / 4! - 1$) which is determined by the positions of the 4 UDSlice edges. The order of the 4 UDSlice edges within the positions is ignored.

We take the F-move as an example:



The following function from `cubicube.pas` implements the computation of this coordinate. The explanation how this works is not obvious and is explained in more detail [here](#). $C(n,k)$ is the binomial coefficient (n choose k).

```
function CubieCube.UDSliceCoord;
var s: Word; k,n: Integer; occupied: array[0..11] of boolean; ed: Edge;
begin
  for n:= 0 to 11 do occupied[n]:=false;
  for ed:=UR to BR do if PEdge^[ed].e >= FR then occupied[Word(ed)]:=true;
  s:=0; k:=3; n:=11;
  while k>= 0 do
  begin
    if occupied[n] then Dec(k)
    else s:= s + C(n,k);
    Dec(n);
  end;
  Result:= s;
end;
```

So each cube relevant for phase 1 is described by a coordinate triple (x_1, x_2, x_3) , and the triple is $(0,0,0)$ if and only if we have a cube from G_1 . The problem space of phase 1 has

$$2187 \cdot 2048 \cdot 495 = 2.217.093.120$$

different states.

In [phase 2](#), any cube is also described with three coordinates:

The [corner permutation coordinate \(0..40319\)](#), the [phase 2 edge permutation coordinate \(0..40319\)](#), and the [phase2 UDSlice coordinate \(0..23\)](#).

The phase 2 triple (0,0,0) belongs to a pristine cube.

The **phase 2 edge permutation coordinate** is similar to the [edge coordinate](#) given in the description of the coordinate level. It is valid only in phase 2.

We have $8! = 40320$ possibilities to permute the 8 edges of the U and D face (remember that we only allow 180 degree turns for all faces R, L, F and B).

```
function CubieCube.Phase2EdgePermCoord: Word;
var i,j: Edge; x,s: Integer;
begin
  x:= 0;
  for i:= DB downto Succ(UR) do
  begin
    s:=0;
    for j:= Pred(i) downto UR do
    begin
      if PEdge^[j].e>PEdge^[i].e then Inc(s);
    end;
    x:= (x+s)*Ord(i);
  end;
  Result:=x;
end;
```

The **phase 2 UDSlice coordinate** should have a range from 0..23 because it represents the 4! permutations of the UDSlice edges in their slice. But we use an extension of the [UDSlice coordinate](#) instead, which is used in the huge optimal solver anyway and where we also regard the order of the four edges. This **"sorted" coordinate** has a range from 0 to $11879=12*11*10*9-1$. But in phase 2 this coordinate indeed only takes values from 0 to 23.

This is the implementation from cubicube.pas:

```
function CubieCube.UDSliceSortedCoord: Word;
var j,k,s,x: Integer; i,e: Edge; arr: array[0..3] of Edge;
begin
  j:=0;
  for i:= UR to BR do
  begin
    e:=PEdge^[i].e;
    if (e=FR) or (e=FL) or (e=BL) or (e=BR) then begin arr[j]:= e; Inc(j); end;
  end;

  x:= 0;
  for j:= 3 downto 1 do
  begin
    s:=0;
    for k:= j-1 downto 0 do
    begin
      if arr[k]>arr[j] then Inc(s);
    end;
    x:= (x+s)*j;
  end;
```

```
end;  
Result:= UDSliceCoord*24 + x;  
end;
```

The problem space of phase 2 has
 $40320 \cdot 40320 \cdot 24 / 2 = 19.508.428.800$
different states.

The Move Tables

If you apply one of the 18 possible faceturns (a "move") to the cube, the permutation of the corners and edges change. On the coordinate level, a move maps a coordinate to another coordinate.

This mapping is possible, because we can show that if we apply a move M onto two different permutations a and b with the same coordinate x , both results have the same coordinate x' . If a and b have the same coordinate x , and H is the subgroup defining the cosets for this coordinate, there exist a permutation g from the cube group G so that a and b are elements of $H * g$ (remember that we use right cosets). Then $a * M$ and $b * M$ are of course elements from $[H * g] * M = H * [g * M]$ and hence are in the same coset and have the same coordinate x' .

Move tables are twodimensional arrays which describe how this mapping is done. We distinguish between move tables for "simple" raw-coordinates and movetables for sym-coordinates, which are reduced by symmetries.

Move tables for raw-coordinates

All move tables for raw-coordinates have the same structure. Let us take for example the move table for the corner orientation coordinate:

TwistMove: array[0..2187-1,Ux1..Bx3] of Word;

If you apply for example the move R2, TwistMove[oldCoordinate,Rx2] gives the new coordinate. This is done pretty fast compared with doing a permutation on the cubie level or the on the facelet level.

Here is the documented code from CordCube.pas to generate this move table:

```

procedure CreateTwistMoveTable;
var c: CubieCube; i,k: Integer; j: TurnAxis;
begin
  c:= CubieCube.Create;//create a cube c on the cubie level
  for i:=0 to 2187-1 do
  begin
    c.InvCornOriCoord(i);//generate a permutation with corner orientation i
    for j:= U to B do
    begin
      for k:= 0 to 3 do //k=3 restores the original state
      begin
        c.Move(j);//apply all 18 face turns on c
        if k<>3 then TwistMove[i.Move(3*Ord(i)+k)]:=c.CornOriCoord://save result in the array
      end
    end
  end
end

```

```

end;
end;
end;
c.Free;
end;

```

Move tables for sym-coordinates

If we reduce a coordinate by symmetries, we only generate a move table for the representants of the equivalence classes. Let $R(j)$ be a permutation belonging to the representant of the equivalence class with index j .

When we apply a move M on this representant, the result will be in another equivalence class k , so that there is a symmetry $S(i)$ with $R(j)*M = S(i)^{-1}*R(k)*S(i)$. Then the resulting movetable entry is the corresponding sym-coordinate, that is $\text{MoveTable}[j,M] := 16*k + i$.

Here is an example for the FlipUDSlice move table from `cordcube.pas` (all unimportant parts removed):

```

procedure CreateFlipUDSliceMoveTable;
var c: CubieCube; i,k,n: Integer; j: TurnAxis;
begin
  SetLength(FlipSliceMove,64430,18); //18 different faceturns
  c:= CubieCube.Create;
  for i:=0 to 64430-1 do //iterate over all equivalence classes
  begin
    n:= FlipUDSliceToRawFlipUDSlice[i]; //get the raw-coordinate of the representant
    c.InvUDSliceCoord(n div 2048); //and generate a permutation which has this FlipUDSlice
coordinate
    c.InvEdgeOriCoord(n mod 2048);
    for j:= U to B do
    begin
      for k:= 0 to 3 do
      begin
        c.Move(j); //apply all 18 faceturns
        if k<>3 then FlipSliceMove[i,3*Ord(j)+k]:= c.FlipUDSliceCoord; //the sym-coordinate
      end;
    end;
  end;
end;
end;
end;

```

The procedure to find the sym-coordinate for a given permutation P is not really difficult but a bit more complicated than the computation of the raw-coordinates. We only need this procedure in the initialization phase where we have to calculate the coordinates of the cube we want to solve

the initialization phase where we have to calculate the coordinates of the cube we want to solve.

For $0 \leq i < 16$ we apply $S(i) * P * S(i)^{-1}$ and compute the raw-coordinate until we find the raw-coordinate in the `ClassIndexToRepresentantArray` at some position k . Let us denote this coordinate with $R(k)$.

$S(i) * P * S(i)^{-1} = R(k)$ is equivalent to $S(i)^{-1} * R(k) * S(i) = P$, and this means P has the sym-coordinate $16 * k + i$.

Look at the function `CubieCube.FlipUDSliceCoord` in `cubicube.pas` for an example.

Applying a move also is more complicated for sym-coordinates compared to raw-coordinates, because we only have built a movetable for the representants of the equivalence classes. But the advantage of using sym-coordinates - reducing the big tables by a factor of about 16 - is much higher than the disadvantage due to the increased complexity.

If we have the sym-coordinate x , we can extract from this coordinate the index j of the equivalence class and the index i of the symmetry. For a move M we have, using the associativity of the permutation group and denoting the representant of the equivalence class with $R(j)$:

$$[S(i)^{-1} * R(j) * S(i)] * M = [S(i)^{-1} * R(j) * S(i)] * M * [S(i)^{-1} * S(i)] = [S(i)^{-1} * R(j)] * [S(i) * M * S(i)^{-1}] * S(i)$$

$[S(i) * M * S(i)^{-1}]$ is the conjugation of a move by a symmetry which is another move. In `symmetry.pas` the array `SymMove[SymIdx, Move]` is initialized, so that `SymMove[i, M]` gives the desired result. Let us denote the result by M_1 .

So we have to compute

$$[S(i)^{-1} * R(j)] * M_1 * S(i) = S(i)^{-1} * [R(j) * M_1] * S(i)$$

The sym-coordinate y for $[R(j) * M_1]$ can be read off from `MoveTable[j, M_1]`. From y we then extract the class index j_1 and the symmetry index i_1 . That means $[R(j) * M_1] = S(i_1)^{-1} * R(j_1) * S(i_1)$.

So we have

$$S(i)^{-1} * [R(j) * M_1] * S(i) = S(i)^{-1} * [S(i_1)^{-1} * R(j_1) * S(i_1)] * S(i) = [S(i)^{-1} * S(i_1)^{-1}] * R(j_1) * [S(i_1) * S(i)]$$

and because $[S(i)^{-1} * S(i_1)^{-1}] = [S(i_1) * S(i)]^{-1}$ we can write this as

$$[S(i_1) * S(i)]^{-1} * R(j_1) * [S(i_1) * S(i)].$$

$[S(i_1) * S(i)]$ is the product of two symmetries, which is another symmetry $S(i_2)$. The array `SymMult[SymIdx, SymIdx]` - created in `symmetries.pas` - does this computation. Let us denote `SymMult[i_1, i]` with i_2 . So our result is

$$S(i_2)^{-1} * R(j_1) * S(i_2) \text{ and the corresponding sym-coordinate is } 16 * j_1 + i_2.$$

So in comparison with the movetables for raw-coordinates where we only need one table-lookup we now need three table-lookups in the tables `SymMove`, `MoveTable` and `SymMult`.



Pruning Tables

The speed of the Two-Phase-Algorithm and the Optimals Solvers depend on the ability to give a lower bound for the number of moves it takes to bring the cube back to a goal state from a given state because it allows tree pruning during the search. The goal state is a certain subgroup [G1](#) in case of the first phase of the Two-Phase-Algorithm. The goal states for the Optimal Solvers are described on the page [Optimal Solvers](#).

We base the pruning tables on coordinates. Remember that a coordinate or also a tuple of several coordinates represent cosets corresponding to some subgroup H (if we use a tuple of coordinates, the corresponding subgroup H is the intersection of the subgroups defining the single coordinates). A coordinate itself or an index computed from two or three coordinates define the position in the pruning table. In this position we store the number of moves which are necessary to bring the cube back to the subgroup H.

Because the goal state is always included in H (phase 2, optimal solvers) or is identical with H (phase 1), the number of moves stored in the pruning table is always a lower bound for the number of moves to bring the cube back to the goal state. This is essential to make the algorithm work.

We need pruning tables for phase 1 and phase 2 of the Two-Phase Algorithm and for the huge optimal solver. The pruning table for the standard optimal solver is identical to the phase1 pruning table.

In all three cases, the position in the pruning table is computed from a [sym-coordinate](#) and one or two raw-coordinates.

Pruning Table	Sym-coordinate	Raw-coordinate(s)	Number of Table-Entries	Maximal pruning depth
Phase 1	FlipUDSlice (64430 equivalence classes)	Corner Twist x = UDTwist (2187 cases)	140,908,410	12
Phase 2	Corner Permutation (2768 equivalence cases)	Edge Permutation x (40320 cases)	111,605,760	18
Huge Optimal Solver	UDSliceSorted (788 equivalence cases)	Edge Flip x1 (2048 cases) Corner Twist x2 (2187 cases)	3,529,433,088	13

If you are interested in the exact distribution of the pruning values have a look at [this table](#)

Let the FlipUDSlice sym-coordinate be for example correspond to the pair (y,i) , where y is the index of the equivalence class and i is the index of the corresponding symmetry and let the corner twist be x . Let P be a permutation of the cube belonging to these indices. Then by conjugation $S(i)*P*S(i)^{-1}$ we get a cube which has the same distance from the goal state and which has the FlipUDSlice sym-coordinate $(y,0)$ and the corner twist x' . Then the position in the pruning table is computed by $2187*y + x'$.

This principle holds for the computation of the indices in all pruning tables: Extract the index i ($0 \leq i < 16$) of the symmetry out of the sym-coordinate and transform the cube by conjugation with $S(i)$ to an equivalent cube with the same equivalence class index y but the symmetry index 0. Transform the raw-coordinate x (or x_1 and x_2 in case of the Huge Optimal Solver) to x' (x'_1, x'_2).

The index in the pruning table in phase 1 is then computed by $2187*y + x'$, in phase 2 by $2768*x' + y$ (hmmm, why did I take not $40320*y + x'$?) and in the huge solver $(2048*y+x'_1)*2187+x'_2$.

The transformation of the raw indices is done by tables (see sourcecode, CordCube.pas)

```
TwistConjugate: array[0..2187-1,0..15] of Word;
FlipConjugate: array of array of array {[0..2048-1,0..15,0..788-1]}of Word;
Edge8PosConjugate: array[0..40320-1,0..15] of Word;
```

As you can see, the conjugation for the edge flip which is used in the huge solver pruning table is a bit more complicated. The reason is, that as [mentioned here](#) the subgroup which defines the flip coordinate cosets is not compatible with the 16 symmetries. But if you add the information of the UDSliceSorted equivalence class, the transformation is possible.

To reduce memory size, we actually do not store the number of moves but only the number of moves modulo 3. This is possible because each faceturn changes the number of moves only by -1, 0 or 1. So if you apply a faceturn you are able to keep track of the number of moves, if you know the number of moves to solve the cube for the initial state.

This number for the initial state also can be reconstructed with the table mod 3: From the initial state try which one of the 18 faceturns decreases the number modulo 3 (there must be a faceturn with this property, or you already are in the goal state). Repeat this until you have reached the goal state and count the number of moves you needed to do so (procedures GetPrun, GetPrunBig and function GetPrunPhase2 in CordCube.pas)

During table generation we use 2 bits for each entry because we need a fourth state to mark an entry as empty. Afterwards we compress the table storing 5 entries in one byte, using only 1.6 bits for each entry. We do the compression not linear like $(0,1,2,3,4), (5,6,7,8,9), (10,11,12,13,14)$... but in the way $(0,1,2,3,x), (4,5,6,7,x+1), (8,9,10,11,x+2), \dots$ where x is about $4/5$ of the total number of entries. In this way we do not need a $(\text{div } 5)$ and $(\text{mod } 5)$ arithmetic but a much faster $(\text{div } 4)$ and $(\text{mod } 4)$ arithmetic.

We generate the table in a breadth-first "forward-search" manner. We store depth 0 at the position of the goal state and apply all 18 moves to this state. At the corresponding positions we store depth 1. In the next pass we apply the 18 moves to all states corresponding to those positions in the pruning table which have an entry 1. We write 2 at the resulting position if it is marked as empty etc...

If you take a look for example at `CreateFlipUDSlicePruningTable` in `CordCube.pas` you see, that the code is not as straightforward as described above. Because we use a [sym-coordinate](#) (`FlipUDSlice`) together with a raw-coordinate (`UDTwist`) to compute the index in the pruning table, we will built an incorrect table if we do not proceed very carefully.

The problem is a permutations A , where the sym-coordinate is not unique because the permutation has itself some symmetries. Let (y, i_1) and (y, i_2) correspond to two classindex-symmetryindex pairs of the `FlipUDSlice` coordinate which belong to the same `FlipUDSlice` raw-coordinate. Let the UD-Twist coordinate of the permutation A be x . The index of the pruning table is computed by $2187*y + x'$, where x' is the `UDTwist` coordinate of the permutation $S(i_1)*A*S(i_1)^{-1}$ respective $S(i_2)*A*S(i_2)^{-1}$. Because these 2 permutations usually have different `UDTwist` coordinates, there is more than one position in the pruning table we have to fill in this case. So we must carefully analyze the symmetries of the `FlipUDSlice` coordinate.

If there are not many empty entries left in the pruning table, we flip to "backward search". We apply the 18 moves to all permutations which belong to empty entries and look if the result is a permutation which has a entry corresponding to the depth d of the last pass. In this case we fill the entry with $d+1$. In this way we save a considerable amount of time when generating the tables.

Two-Phase Algorithm Details

I developed the Two-Phase Algorithm in 1991 and 1992. It was inspired by the the Thistlethwaite algorithm to solve the cube. His method involves working through the following sequence of subgroups:

$H_0 = \langle L, R, F, B, U_2, D_2 \rangle$, $H_1 = \langle L, R, F_2, B_2 U_2, D_2 \rangle$, $H_2 = \langle L_2, R_2, F_2, B_2, U_2, D_2 \rangle$ to find a solution. He used static tables for the maneuvers and the algorithm requires at most 52 moves.

Reducing the number of intermediate subgroups would give shorter solutions and I decided to use only one subgroup $G_1 = \langle U, D, R_2, L_2, F_2, B_2 \rangle$ which is equivalent to Thistlethwaite's H_1 . But it was clear, that in this case static tables for the maneuvers were impossible because of the size of the subgroup. So these maneuvers had to be computed dynamically during the solving procedure. With the hardware I used (8 MHZ Atari ST with 1 MB of RAM) this was far from trivial because there are about 2217 million different positions in phase 1 (getting into G_1) and about 19508 million positions in phase 2 (getting the cube solved in G_1).

After a long struggle I finally found the ingredients which made the maneuver search work:

- Mapping permutations and orientations to natural numbers and implementing moves as table-lookups for these numbers.
- Computing from these numbers some indices for tables which hold information about the distance to the goal state.

Phase 1 needs at most 12 moves (see [distribution](#)) and phase 2 needs at most 18 moves (Michael Reid showed this in 1995, do **not** see [distribution](#) because the phase 2 pruning table only holds 1/24 of all possible phase 2 positions). So the first solution generated by the Two-Phase Algorithm will always have at most 30 moves. The idea to combine [suboptimal](#) solutions of phase 1 with optimal solutions of phase 2 to get shorter overall solutions was quite obvious then, but I was surprised how short the overall solutions are - usually within seconds 22 moves or less on the Atari ST and 20 moves or less in the current implementation and a year 2000 PC.

I did not use symmetry reductions in this first version of the Two-Phase Algorithm. The idea for symmetry reduction came from Mike Reid who used it in 1997 to hold a complete phase 1 pruning table in memory then in his one-phase optimal solver.

In the current implementation (Cube Explorer 2) symmetry reduction also is used.

In phase 1 we use two coordinates: The FlipUDSlice coordinate (a [sym-coordinate](#) with 64430 different classes which combines the [edge orientation coordinate](#) and the [UDSlice coordinate](#)) and the [corner orientation coordinate](#).

When computing the index for the pruning table, both coordinates are used. This means, that we have an entry for each possible phase 1 position.

In phase 2 we have the problem of initializing the three phase 2 coordinates [corner permutation](#), [phase 2 UDSlice](#) and [phase 2 edge permutation](#).

Because the phase 2 UDSlice and phase 2 edge permutation coordinates are not defined in phase 1, we would have to go back to the cubie level to apply our phase 1 solution to the cube before computing the phase 2 coordinates.

So we use three helper-coordinates which are [also](#) defined in phase 1 (UDSliceSorted, RLSliceSorted and FBSliceSorted), each describing the exact positions of the 4 edges of a slice. Helper-coordinate div 24 gives the positional part, helper-coordinate mod 24 describes the possible permutations of the 4 edges within the position.

In phase 2 the [phase 2 UDSlice coordinate](#) is identical to the UDSliceSorted coordinate, so we do not need to do any computation at all.

The [phase 2 edge permutation coordinate](#) can be extracted from the RLSliceSorted and FBSliceSorted coordinates with help of the table GetEdge8Perm of size 11880*24.

GetEdge8Perm[RLSliceSorted, FBSliceSorted mod 24] gives the coordinate. We may use FBSliceSorted mod 24 here, because the positional part information of the FB slice cubies is redundant.

The [corner permutation coordinate](#) is already defined in phase 1. We use a raw-coordinate (0.40329) in the movetable. Before building the index in the pruning table (together with the phase 2 edge permutation coordinate), it is mapped to a sym-coordinate with 2768 classes.

As already mentioned above, the algorithm does not stop when a first solution is found but continues to search for shorter solutions by carrying out phase 2 from suboptimal solutions of phase 1.

For example, if the first solution has 10 moves in phase 1 followed by 12 moves in phase 2, the second solution could have 11 moves in phase 1 and only 5 moves in phase 2. The length of the phase 1 maneuvers increases and the length of the phase 2 maneuvers decreases.

Usually the phase 2 length drops very soon (typically below 9). The performance of the algorithm increases considerably if we do not initialize all three coordinates when entering phase 2 but only the corner permutation coordinate. A small pruning table only for this coordinate shows in most cases, that even the corner permutation coordinate cannot be restored within this small number of moves.

So we can jump back immediately to find the next suboptimal phase 1 solution .

Another way to considerably increase the performance is to throw away certain phase 1 suboptimal solutions. If the maneuver M defines a phase 1 solution, then for example M R2 or M U F2 of course are also suboptimal phase 1 solutions, because R2, F2 and U are phase 2 moves. But these solutions are irrelevant, because phase 2 applied to M R2 will never give a shorter overall solution than phase 2 applied to M.

In the current implementation we throw away any phase 1 suboptimal solution maneuver, if some submaneuver beginning with the first move already is a phase 1 solution. We might lose some solutions doing like this, but in practice this is irrelevant except for the fact, that the algorithm now is not suited any more to prove a certain maneuver to be optimal. But this is done better by using

the optimal solver anyway.

Take for example the cube C generated by $R L U^2 R L . F$ (6 moves). The algorithm will not find the solution $F' . L' R' U^2 L' R'$ because applying F' to C brings the cube into the subgroup G_1 and is therefore is a phase 1 solution. Any suboptimal phase 1 solution starting with F' will be discarded.

The Optimal Solvers

An optimal solver never needs more moves to restore a scrambled cube than the number of moves used to scramble the cube.

The [standard optimal solver](#) implemented in Cube Explorer uses Mike Reid's method from 1997. We do a [triple](#) phase 1 search [in parallel](#) in three different directions. That means that our goal state is the intersection of the groups $\langle U, D, R^2, L^2, F^2, B^2 \rangle$, $\langle U^2, D^2, R, L, F^2, B^2 \rangle$ and $\langle U^2, D^2, R^2, L^2, F, B \rangle$. By the way, this intersection is not the group $\langle U^2, D^2, R^2, L^2, F^2, B^2 \rangle$ but a group six times larger.

Because the phase 1 pruning table has an entry for each possible position, phase 1 solutions are generated very fast. So we just produce triple phase 1 suboptimal solutions and throw them away until the cube is solved (the solved cube is a phase 1 solution).

Using the pruning table in parallel in three different directions is a nice thing because it substantially improves the tree-pruning quality. If p_1 , p_2 and p_3 are the pruning values in the three different directions, we can use $\max(p_1, p_2, p_3)$ as the effective pruning value in our search.

A look at the [distribution](#) of the pruning values in phase 1 shows that the probability to have the pruning value 10 in each direction is relatively high. The following idea (suggested by Michiel de Bondt) improves the performance of the algorithm by about 35%:

If we apply an arbitrary move to a cube from the goal state, the resulting cube stays at least in one of the three subgroups mentioned above. This implicates, that at least one of the three pruning values stays 0. So if we do for example 10 moves from the goal state, at least one of the pruning values is 9 or less. If on the other hand all three pruning values are 10, we know that we can use 11 as the effective pruning value. In general: if all three pruning values are n , we can use $n+1$ as the effective pruning value.

My [huge optimal solver](#) works the same way as the standard optimal solver does. The only difference is that it uses the UDSliceSorted coordinate instead of the UDSlice coordinate to build the pruning table. The goal state is a subgroup of the group which defines the goal state for the standard optimal solver, because all 12 edges are in place now. The pruning table is about 24 times bigger and the average pruning value is higher, as documented in the [distribution](#) of the pruning table. It runs about 5 times faster than the standard optimal solver.

Symmetric Patterns



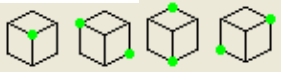

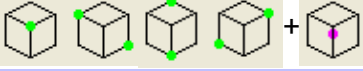
With the Symmetry Editor of Cube Explorer you can search for symmetric cube patterns. We will give some explanation concerning the mathematics of such symmetries here.





A cube has 48 symmetries which build the [symmetry group M](#) with 48 elements. A cube symmetry is a geometric transformation, which maps the cube onto itself. If the cube has a pattern, this pattern usually will not map onto itself too.

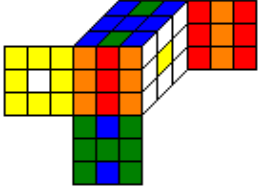
	But take for example this cube. If you do 1/4 rotation about the UD-axis, the result is...
	this cube. And if you recolor the facelets again by exchanging yellow with red, red with white, white with orange and orange with yellow you get...
	the first cube again. We call this pattern symmetric with respect to the 1/4 rotation about the UD-axis.

Here is a table of the possible 48 symmetries of the cube

	1/2 rotation around an edge	6 elements
	Reflection through a plane	6 elements
	1/2 rotation around a face	3 elements
	Reflection through a plane	3 elements
	1/4 rotation around a face	2 x 3 elements

	1/4 rotation around a face	2 x 3 elements
	1/4 rotation + reflection through the center	2 x 3 elements
	1/3 rotation around an edge	2 x 4 elements
	Reflection through the center	1 element
	1/3 rotation + reflection through the center	2 x 4 elements
	Identity: do nothing	1 element

There are patterns which only have one of the above symmetries (except the identity), but there also are patterns which have several symmetries. A pattern could be for example symmetric with respect to all three reflections through a plane . This automatically implies the symmetries  and . The resulting symmetry type in this example is . An

example for a pattern, which has this symmetry is . Altogether there are 33

basically different symmetry types which correspond to certain subgroups of the symmetry group M.

[Look at this table](http://kociemba.org/math/symmetric.htm) to get more information about the 33 symmetry types and cube patterns with these symmetries.