ZZLL Reference: Last Layer Stage of ZZ/b Method

Ilia Kopchinskii | ilya.kopchinski@gmail.com

14th October 2024

The magick of PLL-skip inspires you to enter ZZLL.
The bore of EPLL makes you not abort.

Introduction

This is a smartly organized reference of 160 ZZLL algorithms (Zbigniew Zborowski Last Layer) supplemented with 9 Phased PLL algorithms. All of these algorithms have been taken from miscellaneous freely accessible sources such as:

- [1]SpeedCubeDB (interactive library of popular algorithmic sets)
- [2]Web-pages by Michael Hordecki (general guide on ZZ speedcubing system, ZZ/b being just a variant)
- [3]Tao Yu's Alg Trainer (comprehensive collection of various algorithmic sets)

ZZLL completes the last layer in ∼14.86 moves[4] once phasing stage of ZZ/b is done. Phasing itself is incorporated into the last F2L pair and adds only ∼1.75 extra moves. Since 160 algorithms of ZZLL form several supersets over the 40 COLL algs (each COLL case is expanded into 4 ZZLL layouts, so $160 = 40 \times 4$), the optimal learning strategy reads as follows.

1. Phasing. Start doing the Phasing stage[5] (not covered by this reference).
2. Phased COLL. Learn 40 algs (29 unique), choosing ZZLL layouts highlighted in yellowish on figure 4 (page 3).
3. Halved ZZLL. Learn complementing 40 algs (12 unique), adding ZZLL layouts highlighted like this on figure 4.
4. ZZLL. Learn the other 80 algs (57 unique) covering the rest of layouts.

The table below compares the sub-optimal ZZLL levels. Column 'Unique' counts the reduced number of algs to learn if one enables[6] right-to-left mirroring[7]. Column 'PLL-skip' shows the probability to complete LL in a single algorithm. Column 'Fidelity' estimates the chance to strike at least 3 PLL-skips within any 5 solves[8]. Entries of 'Avg. moves' refer to their algosets only, whereas columns 'Phasing' and 'EPLL' list extra moves you need to complete a solve.

Algorithmic set	Algs	Unique	PLL-skip	Fidelity	Avg. moves	Phasing	EPLL	Learning	Recognizing
Plain COLL	40	29	1/12	0.005	11.98	none	+7.25	easy	rapid
Phased COLL	40	29	1/4	0.10	13.64	+1.75	+7.00	easy	rapid
Halved ZZLL	80	41	1/2	0.5	14.56	+1.75	+4.25	medium	fast
ZZLL	160	98	1	1	14.86	+1.75	none	difficult	normal
ZBLL	480	∼300	1	1	12.08	none	none	near impossible	slow

The last row of this table describes ZBLL superset with 480 algorithms. On the one hand, it scraps phasing stage and saves a tasty amount of moves. On the other hand, ZBLL is 3 times harder to learn, 3 times longer to practice and 3 times slower to recognize than ZZLL. Therefore, we generically discourage learning ZBLL unless one gets extremely bored with ZZLL. The figure 1 below visualizes some information from the table and the author's attitude to these LL variants.

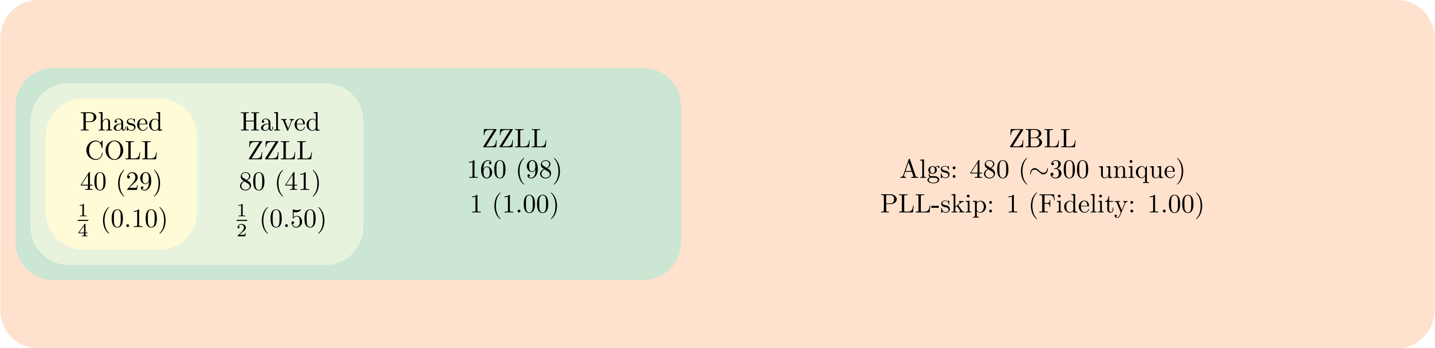


Figure 1. Advanced Last Layer algorithmic sets compared

[1] https://speedcubedb.com/
[2] https://web.archive.org/web/20090903194420/http://www.emsee.110mb.com/Speedcubing/ZZ%20speedcubing%20system.html
[3] https://drive.google.com/drive/u/0/folders/1bK3wUbRcqYZX8IGkp7LkOV8HLk-KSK6N
[4] Here and below the optimal HTM metric (slices included) is used
[5] 12 Phasing algorithms just extend solution of the last F2L slot and are rather intuitive to understand and easy to memorize.
[6] We highly recommend to grab on mirroring right now. Symmetry can be as functional as elegant!
[7] Furthermore, one might practice algorithms and their inverses in pairs. This speeds up learning as well.
[8] In a real-use case (AO5 on a WCA contest) this is the chance to make 2-3 PLL-skips dominate among the 3 median solves to be averaged.

Naming and recognition scheme of ZZLL algorithms

Let us look in detail at an exemplary entry from this reference and then remember the recognition of 7 non-trivial OLL classes, 40 COLL cases and 8 ZZLL layouts.

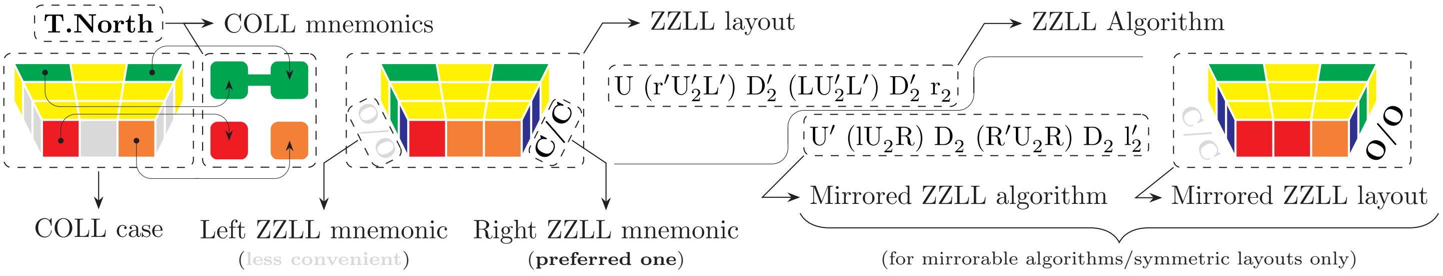


Figure 2. Reference entry explained

The complete naming of entry on figure 2 reads **T.North.C/C** (for the first algorithm) and **T.North.O/O** (mirrored one). Such naming scheme closely follows the recognition order **OLL.COLL.ZZLL**. In the family of ZZ speedsolving methods only the following 7 non-trivial OLL classes appear (the edges have been oriented during the EOLine stage of any ZZ solve).

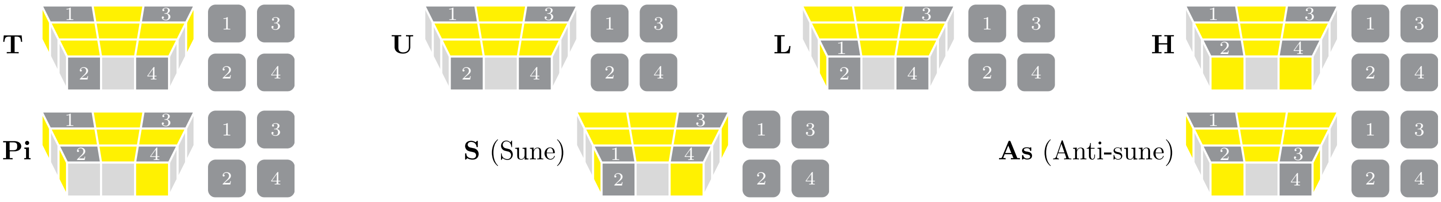
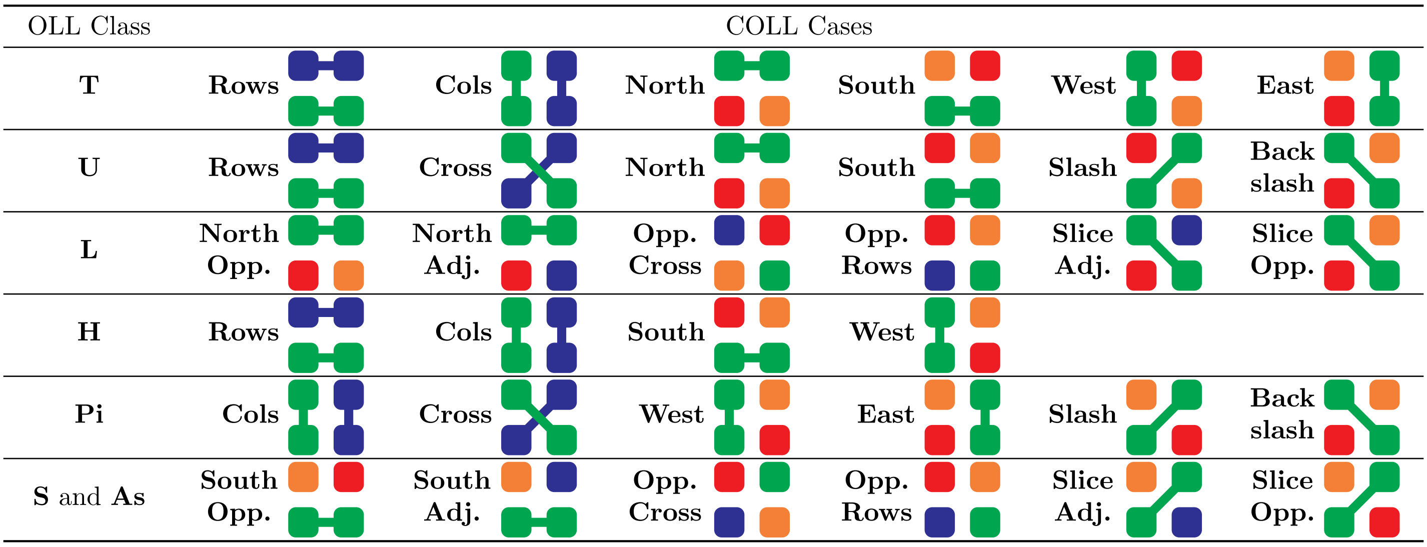


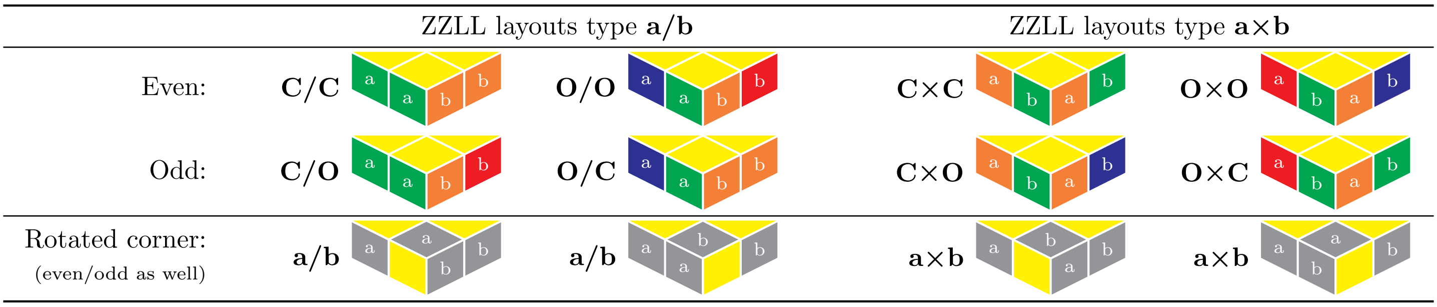
Figure 3. Recognition of OLL class and columnwise mapping of corner stickers to COLL patterns

After OLL class is detected one distinguishes COLL case by comparing dark-gray highlighted stickers (on figure 3) to those of COLL mnemonic pattern. As shown here, exact mapping of COLL-determining stickers depends on the OLL class. Actual patterns may vary in colors, only the *relative* color of any two stickers matters (are they same, opposite or adjacent).



OLL Class	COLL Cases					
T	**Rows**	**Cols**	**North**	**South**	**West**	**East**
U	**Rows**	**Cross**	**North**	**South**	**Slash**	**Back slash**
L	**North Opp.**	**North Adj.**	**Opp. Cross**	**Opp. Rows**	**Slice Adj.**	**Slice Opp.**
H	**Rows**	**Cols**	**South**	**West**		
Pi	**Cols**	**Cross**	**West**	**East**	**Slash**	**Back slash**
S and **As**	**South Opp.**	**South Adj.**	**Opp. Cross**	**Opp. Rows**	**Slice Adj.**	**Slice Opp.**

Finally, one classifies ZZLL layout of the COLL case met. Each of 40 COLL cases comprises 4 ZZLL layouts. However, there are 8 kinds of ZZLL layouts (not mentioning 3 possible corner rotations) since COLL cases may have either even or odd combination of their edge and corner parities. The next table shows principal ZZLL layouts and their mnemonics.



	ZZLL layouts type **a/b**		ZZLL layouts type **a×b**	
Even:	**C/C**	**O/O**	**C×C**	**O×O**
Odd:	**C/O**	**O/C**	**C×O**	**O×C**
Rotated corner: (even/odd as well)	**a/b**	**a/b**	**a×b**	**a×b**

Again, only the relations between colors matter: **C** = same/'correct', **O** = opposite. This reference provides two mnemonics for each ZZLL layout: left FL and right FR (see figure 2). It is a matter of habit to recognize particular ZZLL layout by its FL or FR mnemonic. Author's favourite mnemonics are typeset **in bold** whereas less convenient ones are paled.

Internal ZZLL structure

As already mentioned, ZZLL algorithmic set can be factored as 40 COLL cases expanded into 4 ZZLL layouts each. Fortunately, there are only 36 self-mirroring algorithms, i.e. $160 = 36 + 2 \cdot 62$ and one has to learn $36 + 62$ unique algs. Furthemore, just 32 algorithms are self-inverting, whereas the rest $160 - 32 = 2 \cdot 64$ algs split into 64 'forward/inverse' pairs. Such internal relationships between 160 ZZLL algorithms are illustrated on figure 4 as follows: mutually mirroring algs' mnemonics are connected with solid lines, mutually inverting ones — with dashed lines. Self-mirroring/self-inverting algs miss their solid/dashed lines respectively. The 20 disconnected algorithms are self-inverting and self-mirroring at the same time and cannot be learned in pairs. In contrast, $64 = 4 \cdot 16$ algs can be practiced not only in pairs, but already in quartets.

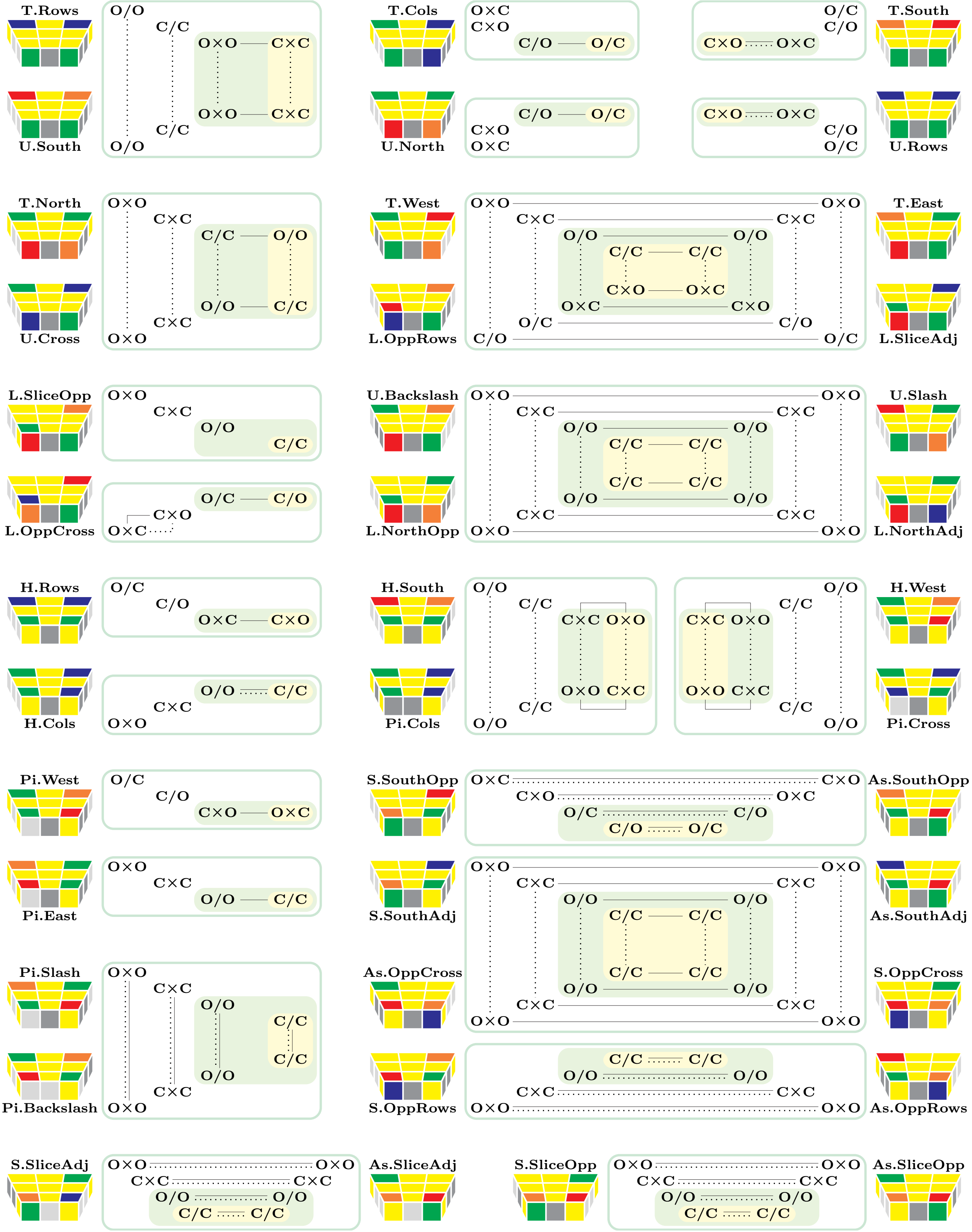
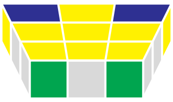
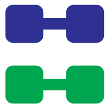


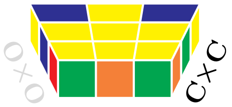
Figure 4. Internal structure of ZZLL algorithmic set

T



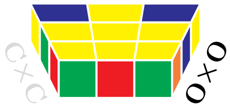
T.Rows

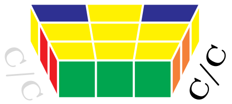




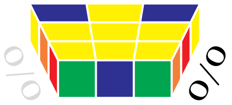
$RU'_2\ R'U'\ RU'\ R'_2U_2\ (RUR'UR)$

$L'U_2\ LU\ L'U\ L_2U'_2\ (L'U'LU'L')$

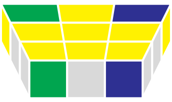




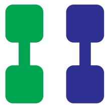
$U'\ (RUR'URU'_2R')\ (L'U'LU'L'U_2L)$

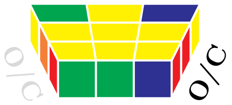


$xD'\ (R'UR)\ D\ R'_2D_2\ (RU'R')\ D_2\ (lR)$



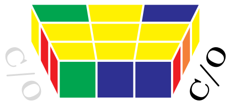
T.Cols

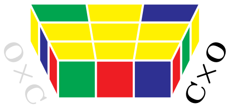




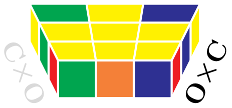
$R'U\ R_2D\ (r'U_2r)\ D'\ (R'_2U'R)$

$LU'\ L'_2D'\ (lU'_2l')\ D\ (L_2UL')$

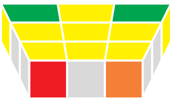




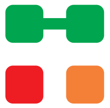
$(RU'_2R'U)\ LU'\ RU\ L'_2\ (UR'U'L)$

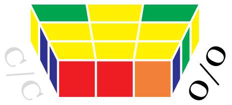


$f'U'F\ R'D'\ (RU_2R')\ DRU'\ (fR'F'R)$



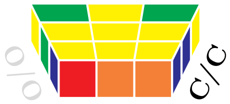
T.North

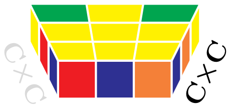




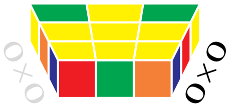
$U'\ (lU_2R)\ D_2\ (R'U_2R)\ D_2\ l'_2$

$U\ (r'U'_2L')\ D'_2\ (LU'_2L')\ D'_2\ r_2$

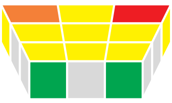




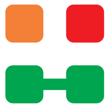
$rU'_2\ (R'_2FRF')\ U'_2\ (r'F)\ (RURU'R')\ F'$

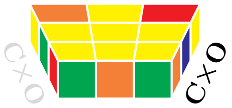


$U'_2\ F\ (RUR'U')\ (RU'R')\ (U'RUR')\ F'$



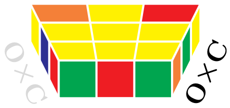
T.South

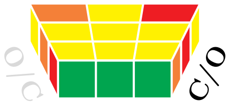




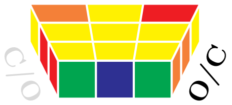
$R'U'_2\ (RUR'UR)\ FU\ (RU'_2R'U)\ (RUR'F')$

$LU_2\ (L'U'LU'L')\ F'U'\ (L'U_2LU')\ (L'U'LF)$

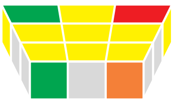




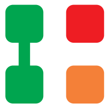
$(R'U'_2RU)\ f\ (R'F')\ (R'U'RU)\ S'\ (R'UR)$

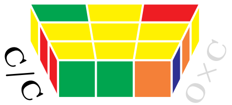


$U_2\ rU'r\ U'_2\ (R'F)\ RU'_2\ (r'_2F)$



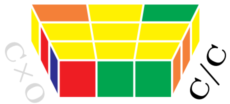
T.West

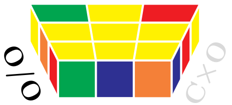




$U'\ (rUR'U')\ (r'F)\ (RF')$

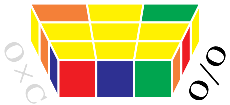
$U\ (l'U'LU)\ (lF')\ (L'F)$

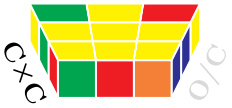




$U'\ F'\ (L'U'LU)_2\ F\ (L'U'LU)\ (LF'L'F)$

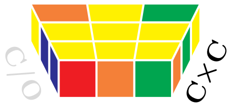
$U\ F\ (RUR'U')_2\ F'\ (RUR'U')\ (R'FRF')$

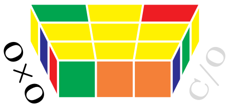




$(LU'_2L')\ F'U\ (LU'L')\ U'F\ (LU'L')$

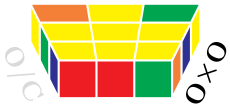
$(R'U_2R)\ FU'\ (R'UR)\ UF'\ (R'UR)$

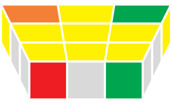




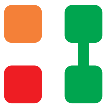
$U'\ (RUR'U')\ RU'\ (R'L)\ U'\ (RUR'L')$

$U\ (L'U'LU)\ L'U\ (LR')\ U\ (L'U'LR)$

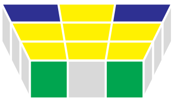




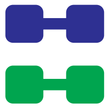
T.East

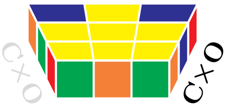


U



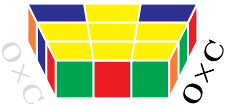
U.Rows

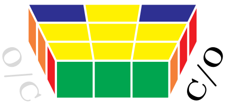




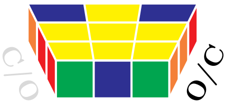
U′ (R′U′RF) R_2' D′ (RUR′) D R_2U′F′

U (LUL′F′) L_2 D (L′U′L) D′ L_2'UF

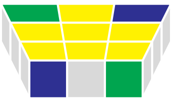




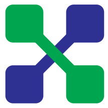
f R′F_2 (RS′R′) U′FU (rU_2) (RU′r′)

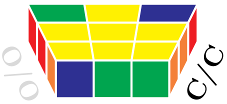


F (URU′R′S) Uf′ (R′f) (RU′R′f′R)



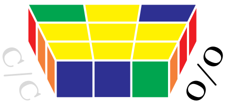
U.Cross

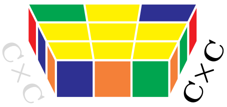




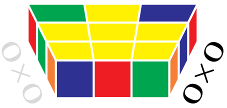
(lR) D_2 (R′U_2R) D_2 (R′U_2l′)

(r′L′) D_2' (LU_2'L′) D_2' (LU_2'r)

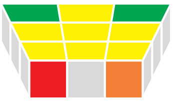




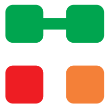
U_2RU′ (R_2'F) (RURU′) (R_2'F′) (RU′F′) U F

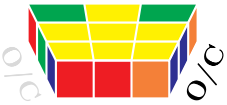


F (RU′R′U) (RUR′URU′R′) F′



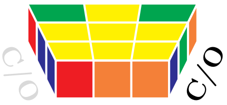
U.North

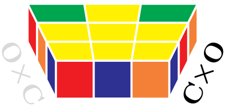




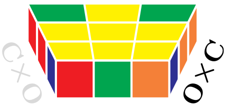
U (RUR′L′) U_2 (RUR′L) UL′UL

U′ (L′U′LR) U_2' (L′U′LR′) U′RU′R′

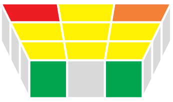




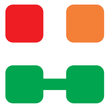
F $[(RU′R′U′) (RU_2R′U′)]_2$ F′

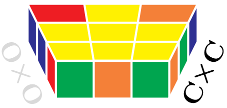


(R′F′) MU′M′ F M Ur



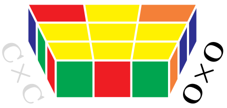
U.South

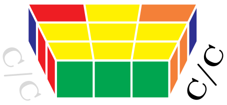




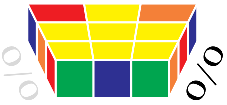
R′U′RU′ R′U_2' (R_2UR′URU_2'R′)

LUL′U LU_2 (L_2'U′LU′L′U_2L)

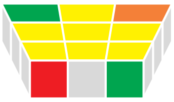




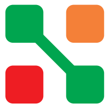
U (R′U_2'RUR′UR) (LU_2L′U′LU′L′)

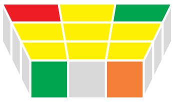


U_2 (l′R′) D_2 (RUR′) D_2R_2 D′ (R′U′R) D

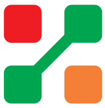


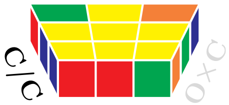
U.Backslash





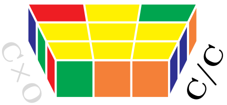
U.Slash

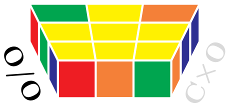




R_2'D′ (RU_2R′) D (RU_2R)

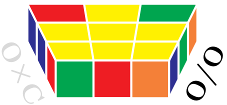
L_2D (L′U_2'L) D′ (L′U_2'L′)

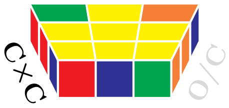




U′R′U_2' R_2D (R′U_2R) D′ R_2'U (RU_2R′U_2'R)

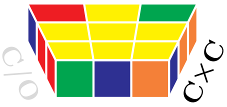
ULU_2 L_2'D′ (LU_2'L′) D L_2U′ (L′U_2'LU_2L′)

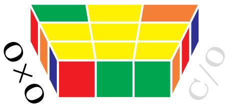




RD (r′U_2r) D′ (R′U_2R′U) (RUR′UR)

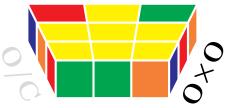
L′D′ (lU_2'l′) D (LU_2'LU′) (L′U′LU′L′)



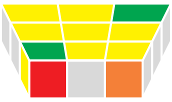


(RU′R′) D (R′U′R) D′ (R_2UR′) U′ (R′U_2R′)

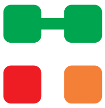
(L′UL) D′ (LUL′) D (L_2'U′L) U (LU_2'L)

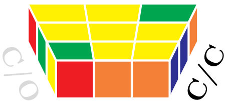


L



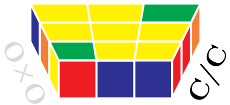
L.NorthOpp

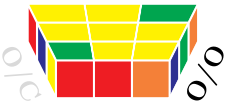




$(UR'U_2R')\ D'\ (RU_2R')\ D\ R_2$

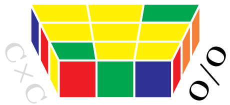
$(LU_2'L)\ D\ (L'U_2'L)\ D'\ L_2'$

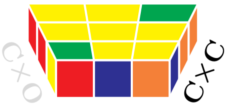




$UR'U_2'\ (RU_2R'U')\ R_2D\ (R'U_2R)\ D'\ R_2'U_2'R$

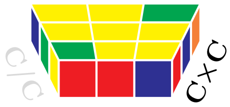
$LU_2\ (L'U_2'LU)\ L_2'D'\ (LU_2'L')\ D\ L_2U_2L'$

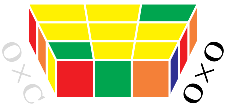




$R'U'RU'R'U'\ (RU_2R)\ D\ (r'U_2r)\ D'R'$

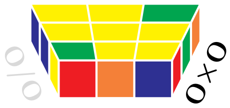
$ULUL'ULU\ (L'U_2'L')\ D'\ (lU_2'l')\ DL$

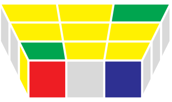




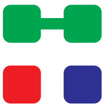
$U_2RU_2'\ (RURU'R_2')\ D\ (R'UR)\ D'\ (RUR')$

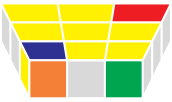
$U'L'U_2\ (L'U'L'UL_2)\ D'\ (LU'L')\ D\ (L'U'L)$



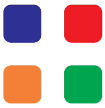


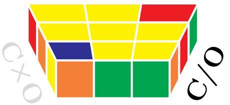
L.NorthAdj





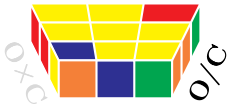
L.OppCross

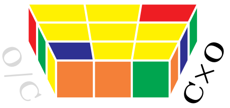




$F\ (RUR'U')\ (R'F)\ (R_2U'R'U')\ (RUR')\ F_2$

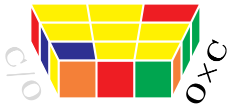
$UF'\ (L'U'LU)\ (LF')\ (L_2'ULU)\ (L'U'L)\ F_2'$

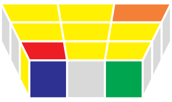




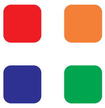
$U\ (rUr')\ (RUR'U')\ (rU'r')\ F\ (URU'R')\ F'$

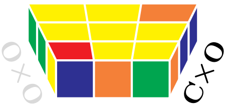
$(l'U'l)\ (L'U'LU)\ (l'Ul)\ F'\ (U'L'UL)\ F$





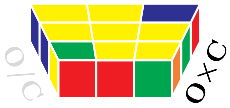
L.OppRows

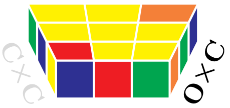




$U'x\ (LU'L')\ D\ (LUL')\ D'$

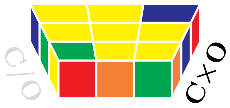
$U_2x\ (R'UR)\ D'\ (R'U'R)\ D$

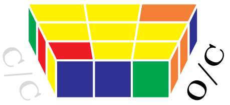




$U\ (L'U'lF')\ (LFl'U)\ (L'U'LU_2')\ (L'U_2L)$

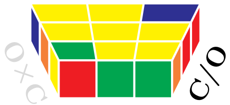
$(RUr'F)\ (R'F'rU')\ (RUR'U_2)\ (RU_2'R')$

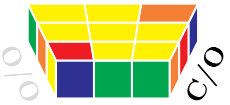




$U'\ (LUL'F')\ ULUL'U'\ F\ (LU_2L')$

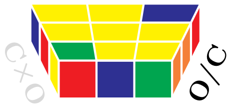
$U_2\ (R'U'RF)\ U'R'U'RU\ F'\ (R'U_2'R)$

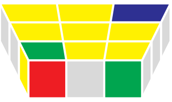




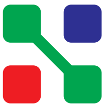
$U'\ LRU'\ r'FM'\ U\ (L'ULU'L')$

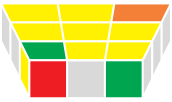
$U_2\ R'L'U\ lF'M'\ U'\ (RU'R'UR)$



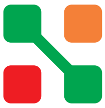


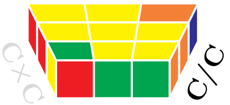
L.SliceAdj



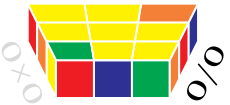


L.SliceOpp

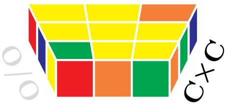




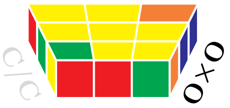
$(URU'R')\ L'U_2LUL'UL\ (RU_2'R')$



$RU\ (R'URU')_3\ U'R'$

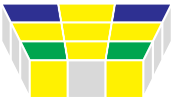


$UR'U_2'\ (RUR'UR)\ (L'U'LU'L'U_2L)$

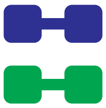


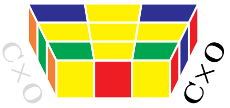
$U_2\ (RUR'URU_2'R')\ (LU_2L'U'LU'L')$

H



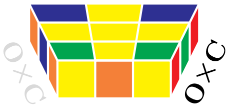
H.Rows

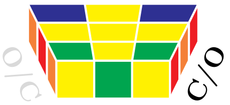




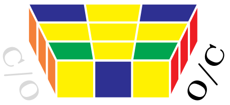
$\mathrm{UR'U'_2\ (RUR'U)\ (RUR'U')\ (RUR'F')\ (RUR'U')\ (R'F)\ R_2}$

$\mathrm{U'LU_2\ (L'U'LU')\ (L'U'LU)\ (L'U'LF)\ (L'U'LU)\ (LF')\ L'_2}$

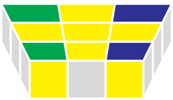




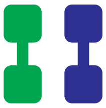
$\mathrm{F\ (RUR'U')_3\ F'}$

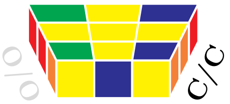


$\mathrm{U\ x'\ U'RU'\ (R'UR'F_2)\ (RU'RUR'U)\ x}$



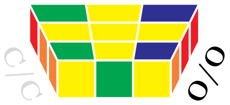
H.Cols

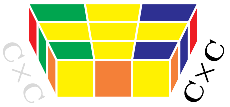




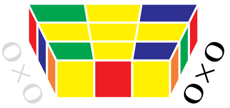
$\mathrm{(R'U'_2RUR'UR)\ U\ (RUR'URU'_2R')}$

$\mathrm{(LU_2L'U'LU'L')\ U'\ (L'U'LU'L'U_2L)}$

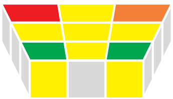




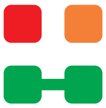
$\mathrm{U'\ F\ \left[U\ (R'FRF')\ (RU'R')\right]_2\ F'}$

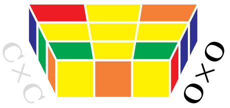


$\mathrm{U'\ (RUR'URU'_2R')\ U'\ (R'U'_2RUR'UR)}$



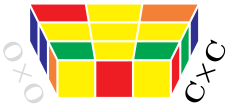
H.South

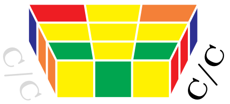




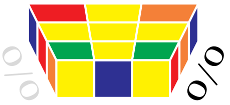
$\mathrm{UR'U'\ (RU'R'U)\ F'\ (RUR'U')\ (R'F)\ R_2U'\ (R'UR)}$

$\mathrm{U'LU\ (L'ULU')\ F\ (L'U'LU)\ (LF')\ L'_2U\ (LU'L')}$

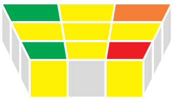




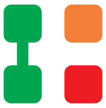
$\mathrm{(RUR'URU_2R')\ F\ RU'R'U'\ (RU_2R'U')\ F'}$

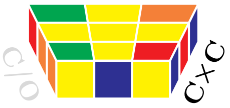


$\mathrm{(RUR'U')\ (R'U'_2)\ R_2D\ (R'UR)\ D'\ (R'_2U)\ (R_2U'_2R')}$



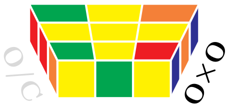
H.West

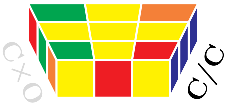




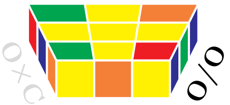
$\mathrm{(RU_2R'U')\ (RUR'U')\ F'\ (RUR'U')\ (R'F)\ R_2U'R'}$

$\mathrm{U_2\ (L'U'_2LU)\ (L'U'LU)\ F\ (L'U'LU)\ (LF')\ L'_2UL}$



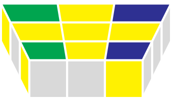


$\mathrm{U'\ F\ (RU'R'U'RU'_2R'U')\ F'\ (RU'_2R'U'RU'R')}$

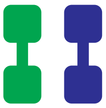


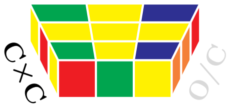
$\mathrm{U'\ F\ (RU'R'U)\ (RU'_2R'U')\ (RUR'U')\ F'}$

Pi

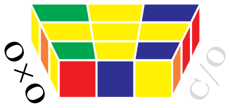


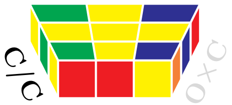
Pi.Cols



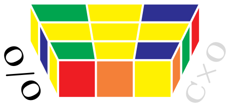


R′U′RU R_2 (F′RURU′R′F) (U′RUR′UR)
U'_2 LUL′U′ L'_2 (FL′U′L′ULF′) (UL′U′LU′L′)

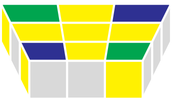




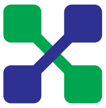
U′ F U (RU'_2R′U) (RUR′) F′ RU'_2R′U′RU′R′

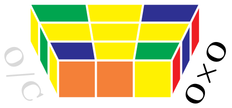


R$U'_2$$R'_2$U′ R_2D (R′U′R) D′ $R'_2$$U'_2$ (RURU′R′)

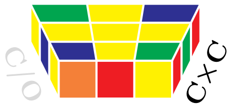


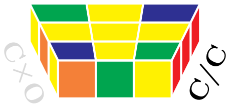
Pi.Cross



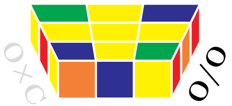


URU (R'_2F′) (RURU′R′F) (URU′R′URU'_2R′)
UL′U′ (L_2F) (L′U′L′ULF′) (U′L′ULU′L′U_2L)

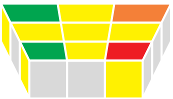




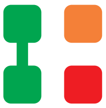
(U′RUR′URU'_2R′) F (URU'_2R′URUR′) F′

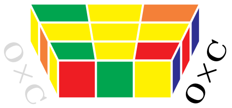


U F (URU′R′U) (RU'_2R′U′RUR′) F′

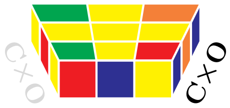


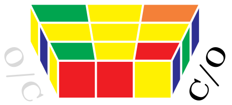
Pi.West



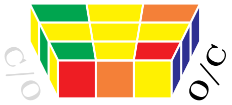


R'_2D′ (RUR′) DRU (RU′R′U) (RUR′UR)
$U_2$$L_2$D (L′U′L) D′L′U′ (L′ULU′) (L′U′LU′L′)

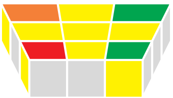




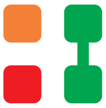
F UR′U′ R_2U′$R'_2$$U'_2$ (RU_2) (RUR′F′)

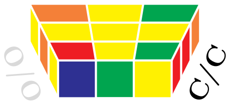


UR′U′R′ D′ (RUR′) D (RU_2R′) D′ (RU_2R′) DR_2

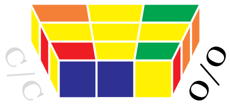


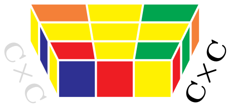
Pi.East



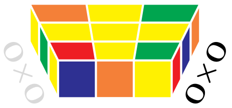


(RUR′URU'_2R′) U′ (RUR′URU'_2R′)
U_2 (L′U′LU′L′U_2L) U (L′U′LU′L′U_2L)

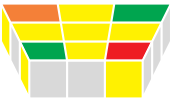




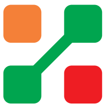
F (RUR′U′)$_2$ F′ (RUR′U′) M′ (URU′r′)

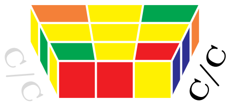


(R′U'_2RUR′U) (RUR′U'_2) (RUR′UR)

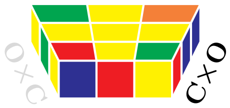


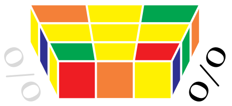
Pi.Slash



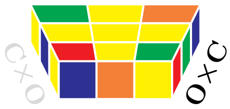


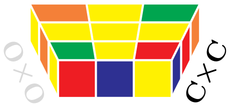
r′F′ (rUrU'_2) (r′F_2) U′ (RUR′U′) RU′R′
U_2 lF (l′U′l′U_2) (lF'_2) U (L′U′LU) L′UL



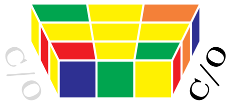


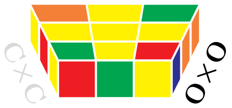
(RU'_2R′U) R_2D (R′U_2R) D′R_2U′ (RU_2R′U′) (RUR′)
U'_2 (L′U_2LU′) L'_2D′ (LU'_2L′) DL'_2U (L′U'_2LU) (L′U′L)



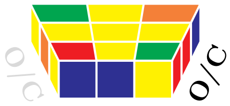


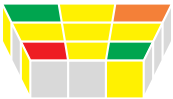
(RUR′U′) (R′F) (R_2UR′U′) (RUR′U′) F′
U_2 (L′U′LU) (LF′) (L'_2U′LU) (L′U′LU) F



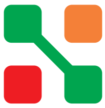


U_2 (RU'_2R′U_2) (R′F) (R_2l) U′R′U R′D′
(L′U_2LU'_2) (LF′) (L'_2r′) ULU′ LD

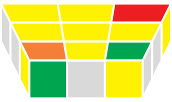
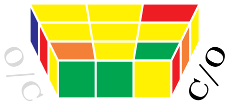
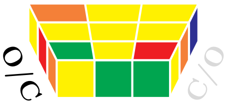
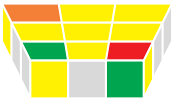
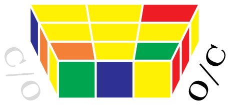
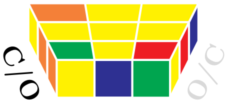
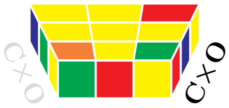
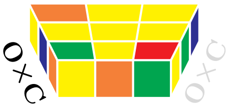
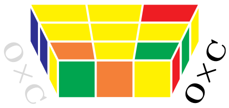
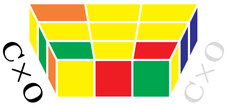


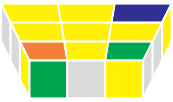
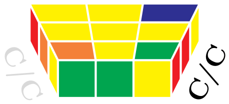
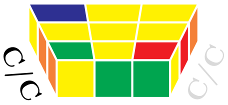
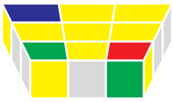
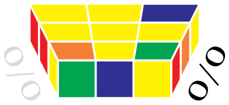
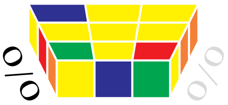
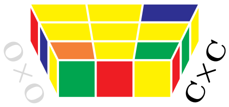
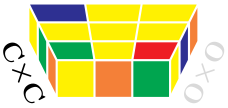
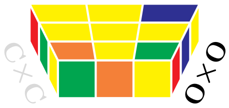
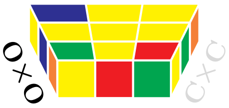


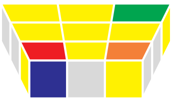
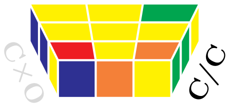
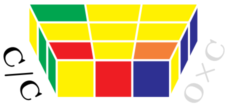
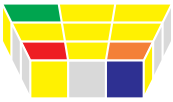
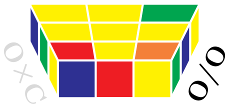
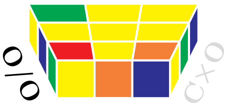
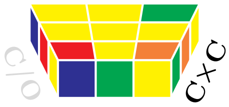
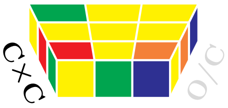
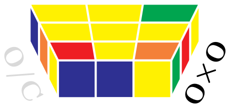
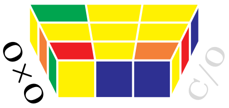
Pi.Backslash

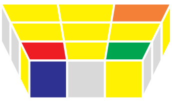
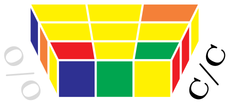
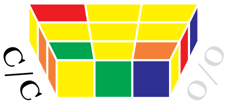
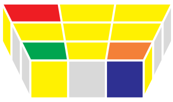
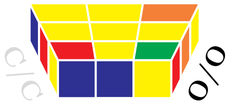
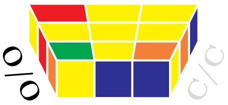
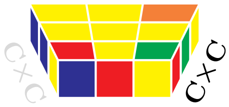
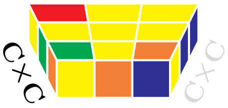
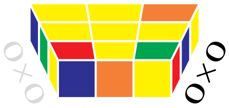
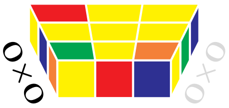


S and As

 **S.SouthOpp** 🟧 🟥 🟩–🟩		U_2 R′U′FU (RU′R′F′) (R_2UR′U′) (R′FRF′) U'_2 LUF′U′ (L′ULF) (L'_2U′LU) (LF′L′F)		 **As.SouthOpp** 🟧 🟥 🟩–🟩
		U_2 f (R′F′) (Rf′) R′D (RUR′) D′ RU_2 F U'_2 f′ (LF) (L′f) LD′ (L′U′L) D $L'U'_2$ F′		
		U′ $R'U'_2$ (RUR′U′) RF U′R′U′RU F′ U LU_2 (L′U′LU) L′F′ ULUL′U′ F		
		U′ (FR′F′R) U (RU′R′F) $URU'_2R'U'$ (RUR′F′) U (F′LFL′) U′ (L′ULF′) $U'L'U_2LU$ (L′U′LF)		

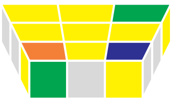
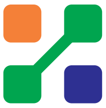
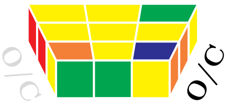
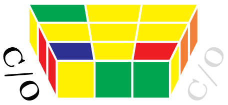
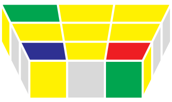
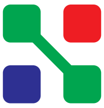
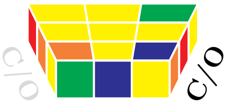
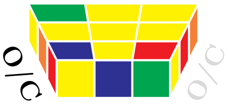
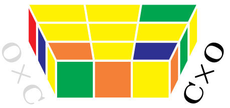
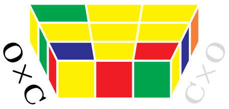
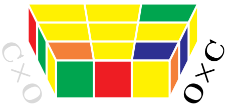
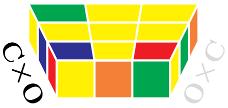
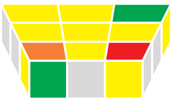
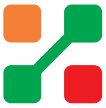
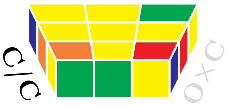
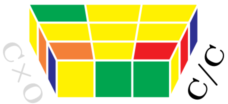
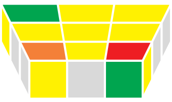
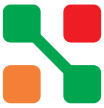
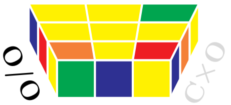
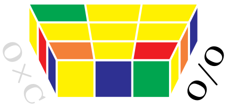
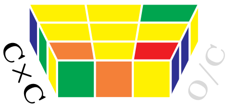
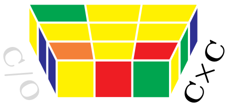
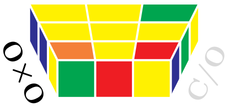
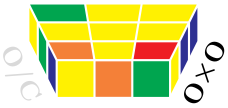
 **S.SouthAdj** 🟧 🟦 🟩–🟩		U′F (URU′R′) S (R′F′) (RURU′R′) S′ UF′ (U′L′UL) S′ (LF) (L′U′L′UL) S		 **As.SouthAdj** 🟦 🟥 🟩–🟩
		UR'_2 D′ (rU'_2r') D R_2 (UR′UR) $U'L_2$ D ($l'U_2l$) D′ L'_2 (U′LU′L′)		
		$U'R'U'_2$ (RUR′UR′) D′ rU'_2r' D (RU'_2R) ULU_2 (L′U′LU′L) D $l'U_2l$ D′ ($L'U_2L'$)		
		U_2 RU′ (L′UR′) UL′D′ (LU'_2L') D L_2 U'_2 L′U (RU′L) U′RD ($R'U_2R$) D′ R'_2		

 **S.OppCross** 🟥 🟩 🟦 🟧		U′ S′ (r′F′) (rUrU′r′) S (L′U′LU) F U S (lF) (l′U′l′Ul) S′ (RUR′U′) F′		 **As.OppCross** 🟩 🟧 🟥 🟦
		U (RUR′U) R_2D ($r'U_2r$) D′ R'_2 U′ (L′U′LU′) L'_2D (lU'_2l') D L_2		
		U′F′ ($RUR'URU'_2R'$) FU ($RU'R'U_2$) (RU′R′) UF ($L'U'LU'L'U_2L$) F′U′ ($L'ULU'_2$) (L′UL)		
		(R′F) (RUR′U′) (R′F′) D′ (RUR′) D R_2 (LF′) (L′U′LU) (LF) D (L′U′L) D′ L'_2		

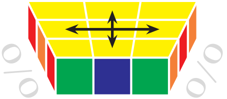
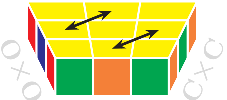
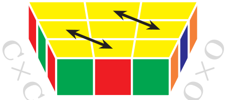
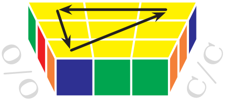
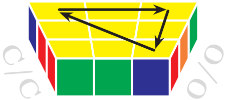
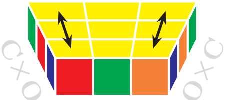
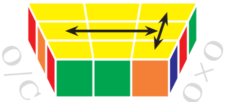
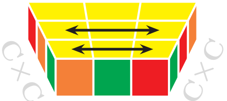
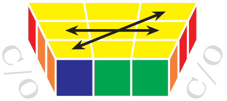
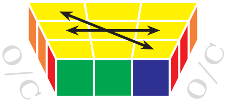
 **S.OppRows** 🟥 🟧 🟦 🟩		U'_2 (RUR′U′) $R'U'_2$ RURU′ (R′UR′UR) U_2 (L′U′LU) LU_2 L′U′L′U (LU′LU′L′)		 **As.OppRows** 🟥 🟧 🟩 🟦
		U′ (R′F′) U′FU (RF) (URU′R′) F′ U (LF) UF′U′ (L′F′) (U′L′UL) F		
		U′ (RUR′U′) (RUR′U) $RURU'_2$ R′U′RU′ (R′UR′) U (L′U′LU) (L′U′LU′) $L'U'L'U_2$ LUL′U (LU′L)		
		U_2 ($R'U'_2$) R_2U (R'_2U) RU′RU′R′ U'_2 (LU_2) L'_2U' (L_2U') L′UL′UL		

See **S/As.SliceAdj** and **S/As.SliceOpp** on the next page

See **S/As.SouthOpp**, **S/As.SouthAdj**, **S/As.OppCross**, **S/As.OppRows** on the previous page

 **S.SliceAdj** 		U′RU′ (L′UR′) U′L UL′U (RU′L) UR′		 **As.SliceAdj** 
		U′F R_2D (R′UR) D′R'_2 U′$(RUR'U')_2$ F′ UF′ L'_2D′ (LU′L′) DL_2 U$(L'U'LU)_2$ F		
		U (RUR′F′) (RUR′U) RU′R′U′ (R′F) (R_2U′R′) U′ (L′U′LF) (L′U′LU′) L′ULU (LF′) (L'_2UL)		
		U'_2 (RUR′U) (R$U'_2R'_2$U) (RU_2R′L′) U (RU′L) U_2 (L′U′LU′) (L′U_2L_2U′) (L′U'_2LR) U′ (L′UR′)		
 **S.SliceOpp** 		U'_2 (RUR′U) RU′RD (R′UR) (r′U_2r) D′R'_2 U_2 (L′U′LU′) L′UL′D′ (LU′L′) (lU'_2l′) DL_2		 **As.SliceOpp** 
		U_2F (RUR′U′) (R′F′) U'_2 RU (RU′R′UR′UR) U'_2F′ (L′U′LU) (LF) U_2 L′U′ (L′ULU′LU′L′)		
		RU'_2 (R′L′) U_2 (RUR′U_2) (LR) U'_2R′ L′U_2 (LR) U'_2 (L′U′LU'_2) (R′L′) U_2L		
		U'_2 (FR′) U'_2 (RF′R′F) U'_2 (F′R) U_2 (F′L) U_2 (L′FLF′) U_2 (FL′)		

Phased PLL

H-perm		M'_2 U M'_2 U_2 M'_2 U M'_2		
Z-perm		M'_2 U M'_2 U M U_2 M'_2 U_2 M M'_2 U′ M'_2 U′ M U'_2 M'_2 U'_2 M		Z̄-perm
A-perm		rU′L D'_2 (L′UL) D'_2 (r′L′) l′UR′ D_2 (RU′R′) D_2 (lR)		Ā-perm
E-perm		x′ (RU′R′) D (RUR′) D′ (RUR′) D (RU′R′) D′		
T-perm		(RUR′U′) (R′F) R_2U′ R′U′ (RUR′F′)		
F-perm		(R′U_2R′) d′ (R′F′) R_2U′ (R′UR′F) RU′ F		
N-perm		LU′ (LF′L′F) (L′UL) FU′F′ (L′U′LU) L′ R′U (R′FRF′) (RU′R′) F′UF (RUR′U′) R		N̄-perm