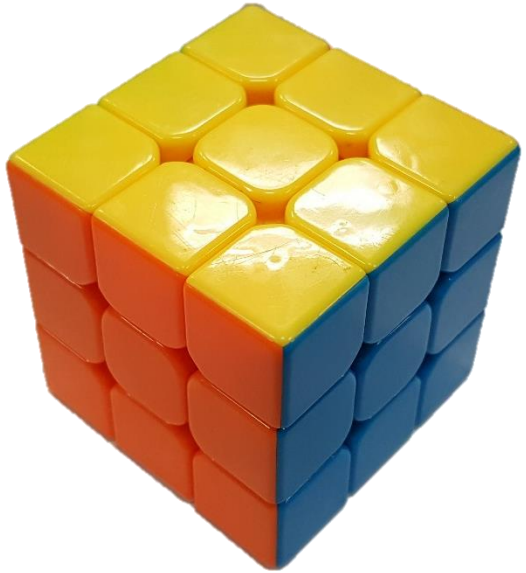
Optimally Solving Rubik's Cube with Pattern Databases

Adam Hayse

What is a Rubik's Cube?

- 3-dimensional combination puzzle with 43,252,003,274,489,856,000 solvable states.
- The objective is to turn the faces in order to make each face the same color.
- Pealing off the stickers and putting them back on is not allowed.

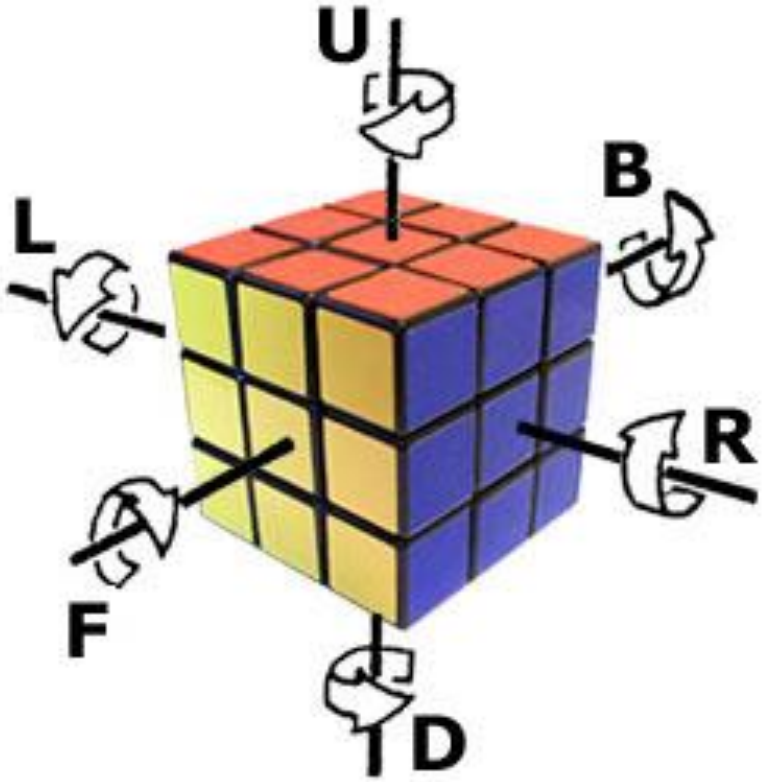


Turn Metric

- There are many metrics for defining what counts as a move.
  - Half Turn Metric
  - Quarter Turn Metric
  - Slice Turn Metric
- For this application, I use the half turn metric.
- In 2010, God's number was shown to be 20 using the half turn metric.

Notation

- U – Upper
- F – Front
- L – Left
- B – Back
- R – Right
- D - Down

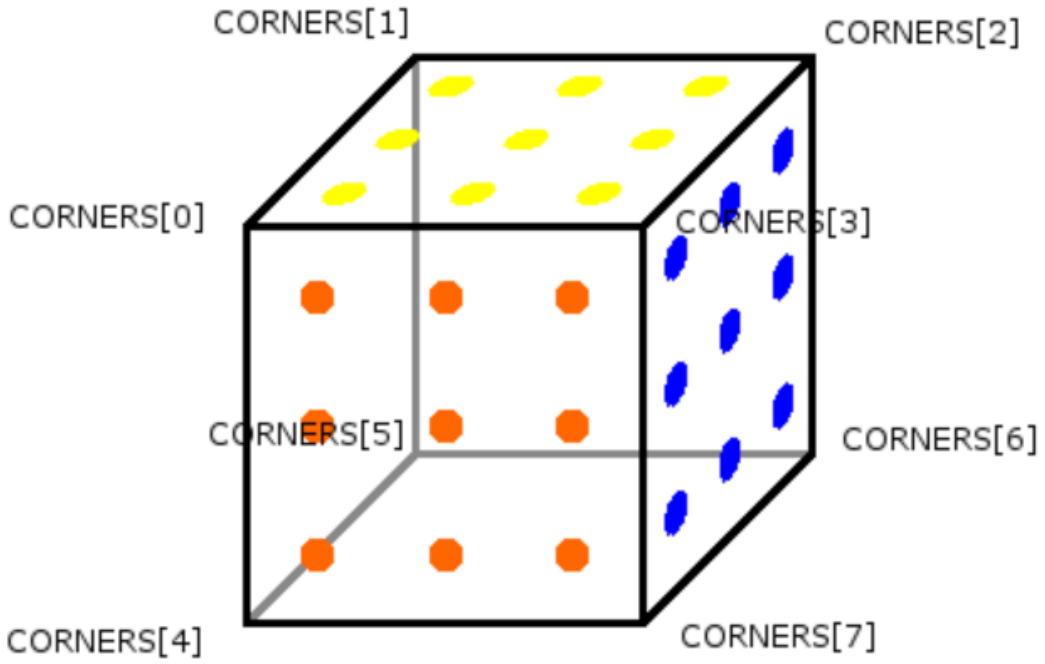


- U – Turn Upper Face 90° Clockwise
- U2 – Turn Upper Face 180°
- U’ – Turn Upper Face 90° Counter-clockwise

- Example set of moves:

  R U R’ U’

Representing the cube in memory




```
Cubie 1    YOG  0    GYO  1    OGY  2
Cubie 2    YGR  3    RYG  4    GRY  5
Cubie 3    YRB  6    BYR  7    RBY  8
Cubie 4    YBO  9    OYB 10    BOY 11
Cubie 5    WGO 12    OWG 13    GOW 14
Cubie 6    WRG 15    GWR 16    RGW 17
Cubie 7    WBR 18    RWB 19    BRW 20
Cubie 8    WOB 21    BWO 22    OBW 23
```

Yellow face is Up, Orange face is Front

Solved corners:

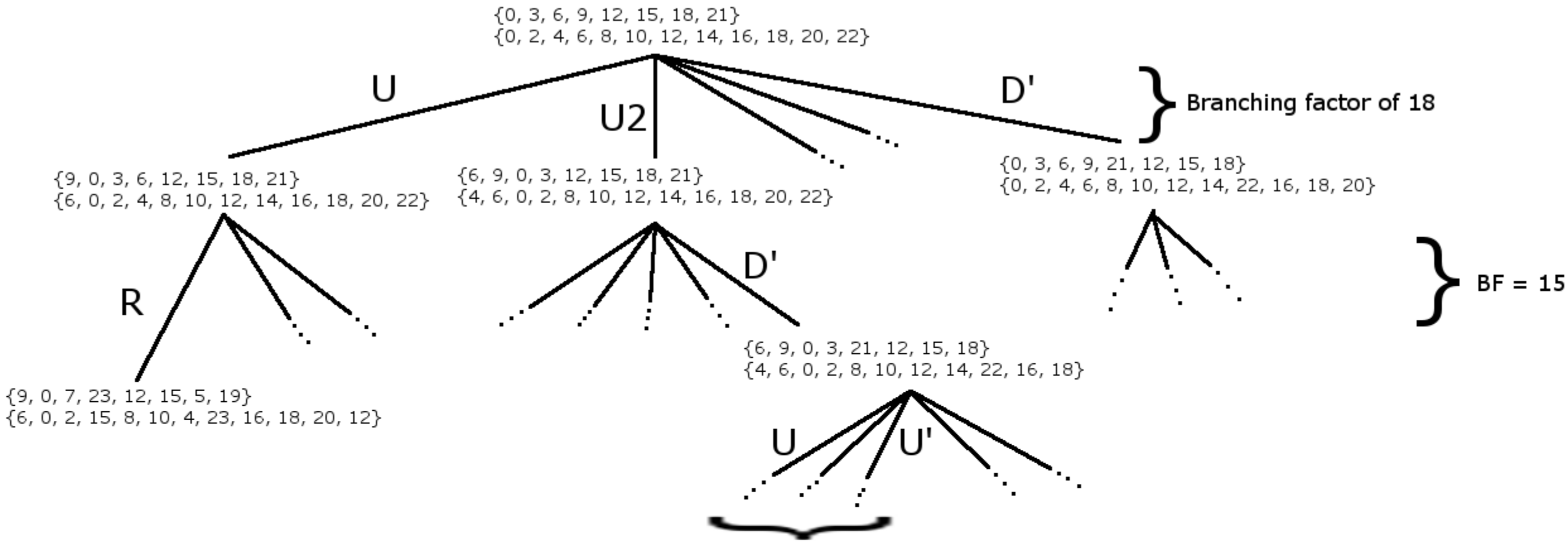
CORNERS = {0, 3, 6, 9, 12, 15, 18, 21}

Heuristics

- 3D Manhattan distance
  - Sum up the number of moves needed to move every piece into its solved position with correct orientation.
  - Divide this value by 8 because each move affects 4 edges and 4 corners.
- Pattern database
  - Find the minimum number of moves needed to solve a set of pieces from any given position.
  - Sets of pieces:
    - All corners: $8! \times 3^7$ = 88,179,840 solvable combinations
    - 6 of the 12 edges: $12!/6! \times 2^6$ = 42,577,920 solvable combinations
    - Remaining 6 edges: $12!/6! \times 2^6$ = 42,577,920 solvable combinations

Making the Pattern Database

- Perform a breadth-first search from the solved state.
- Only expand states that haven’t been visited.
- All moves have the same cost, so breadth-first search finds the shortest path from the solution state to any reachable combination.
- This also means that the shortest path is also found from any valid combination to the solved state.


{0, 3, 6, 9, 12, 15, 18, 21}
{0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22}
U
U2
D'
Branching factor of 18
{9, 0, 3, 6, 12, 15, 18, 21}
{6, 0, 2, 4, 8, 10, 12, 14, 16, 18, 20, 22}
{6, 9, 0, 3, 12, 15, 18, 21}
{4, 6, 0, 2, 8, 10, 12, 14, 16, 18, 20, 22}
{0, 3, 6, 9, 21, 12, 15, 18}
{0, 2, 4, 6, 8, 10, 12, 14, 22, 16, 18, 20}
R
D'
BF = 15
{9, 0, 7, 23, 12, 15, 5, 19}
{6, 0, 2, 15, 8, 10, 4, 23, 16, 18, 20, 12}
{6, 9, 0, 3, 21, 12, 15, 18}
{4, 6, 0, 2, 8, 10, 12, 14, 22, 16, 18}
U
U'
These states were visited before.
Don't expand.

Encode function

- Need a function that turns a set of pieces into an index in a database.

Solved state:

C_get_index({0, 3, 6, 9, 12, 15, 18, 21}) = 0

From solved state, turn U:

C_get_index({9, 0, 3, 6, 12, 15, 18, 21}) = 33067440

From solved state, turn U2:

C_get_index({6, 9, 0, 3, 12, 15, 18, 21}) = 25194240

```
// Get index of stored combination.
unsigned C_get_index(uint8_t *comb) {
  unsigned i, add = 0;

  // Calculate which permutation number.
  unsigned long long state = 0xFEDCBA9876543210;
  for (i=0; i<7; i++) {
    int p4 = comb[i]/3 * 4;
    add += fact[7-i] * (state >> p4 & 15);
    state -= 0x1111111111111110ULL << p4;
  }

  // Scale for permutation offset.
  add *= 2187;  // power(NUM_CFACES, NUM_CORNERS-1)

  // Calculate which orientation number for orientation offset.
  for (int i=0; i<7; i++)
    add += comb[i] % NUM_CFACES * three_to_the[i];

  return add;
}
```


0 F F F F F F F F ... F F F F F F F F F F ... F F F F F F F F F F ... F F F F F F F F F F F F F F F F

{0, 3, 6, 9, 12, 15, 18, 21}

{0, 3, 6, 9, 12, 15, 18, 21}

Perform all moves on faces except for the last turned face, and add to the queue and record depth in database if it is a new state:

{0, 3, 6, 9, 12, 15, 18, 21}

Perform all moves on faces except for the last turned face, and add to the queue and record depth in database if it is a new state:

U: {9, 0, 3, 6, 12, 15, 18, 21}
U2: {6, 9, 0, 3, 12, 15, 18, 21}
.
.
.
D': {0, 3, 6, 9, 21, 12, 15, 18}

{0, 3, 6, 9, 12, 15, 18, 21}

Perform all moves on faces except for the last turned face, and add to the queue and record depth in database if it is a new state:

		Encode:	
U:	{9, 0, 3, 6, 12, 15, 18, 21}	====>	33,067,440
U2:	{6, 9, 0, 3, 12, 15, 18, 21}	====>	25,194,240
	.		
	.		
	.		
D':	{0, 3, 6, 9, 21, 12, 15, 18}	====>	39,366

{0, 3, 6, 9, 12, 15, 18, 21}

Perform all moves on faces except for the last turned face, and add to the queue and record depth in database if it is a new state:

		Encode:	
U:	{9, 0, 3, 6, 12, 15, 18, 21}	====>	33,067,440
U2:	{6, 9, 0, 3, 12, 15, 18, 21}	====>	25,194,240
	.		
	.		
	.		
D':	{0, 3, 6, 9, 21, 12, 15, 18}	====>	39,366

{0, 3, 6, 9, 12, 15, 18, 21}

Perform all moves on faces except for the last turned face, and add to the queue and record depth in database if it is a new state:

		Encode:	
U:	{9, 0, 3, 6, 12, 15, 18, 21}	====>	33,067,440
U2:	{6, 9, 0, 3, 12, 15, 18, 21}	====>	25,194,240
	.		
	.		
	.		
D':	{0, 3, 6, 9, 21, 12, 15, 18}	====>	39,366

{0, 3, 6, 9, 12, 15, 18, 21}

Perform all moves on faces except for the last turned face, and add to the queue and record depth in database if it is a new state:

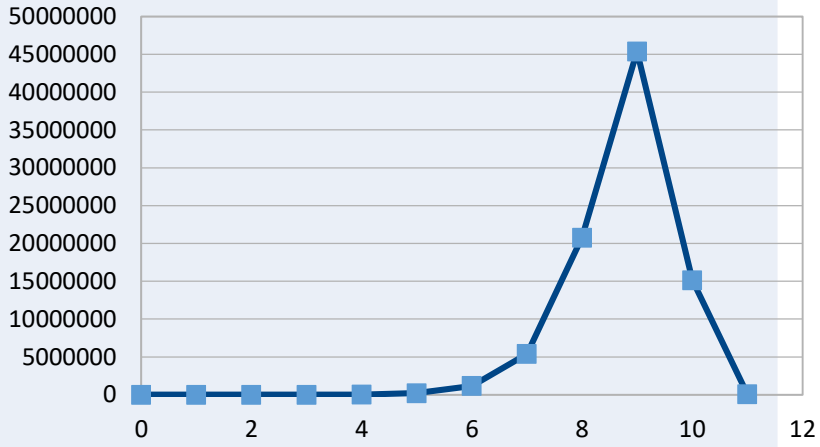
		Encode:		Decode:	
U:	{9, 0, 3, 6, 12, 15, 18, 21}	====>	33,067,440	====>	{9, 0, 3, 6, 12, 15, 18, 21}
U2:	{6, 9, 0, 3, 12, 15, 18, 21}	====>	25,194,240	====>	{6, 9, 0, 3, 12, 15, 18, 21}
	.				.
	.				.
	.				.
D':	{0, 3, 6, 9, 21, 12, 15, 18}	====>	39,366	====>	{0, 3, 6, 9, 21, 12, 15, 18}

Decode function

```
// Transform an index of the database to a combination.
void E1_decode_index(uint64_t index, uint8_t *comb) {
  uint64_t temp = index/two_to_the[TRACKED_EDGES];  // Get permutation number
  for (int i=0; i<NUM_EDGES; i++) {
    comb[i] = TRACKED_EDGES*2;
  }
  unsigned long long state = 0xfedcba9876543210ULL;
  for (int i = 0; i < TRACKED_EDGES; i++) {
    int p4 = temp/(fact[11-i]/fact[NUM_EDGES-TRACKED_EDGES]) * 4;
    temp %= (fact[11-i]/fact[NUM_EDGES-TRACKED_EDGES]);
    comb[(state >> p4) & 15] = 2*i + ((index % two_to_the[TRACKED_EDGES] >> i) % 2);
    unsigned long long mask = ((unsigned long long)1 << p4) - 1;
    state = (state & mask) | ((state >> 4) & ~mask);
  }
}
```

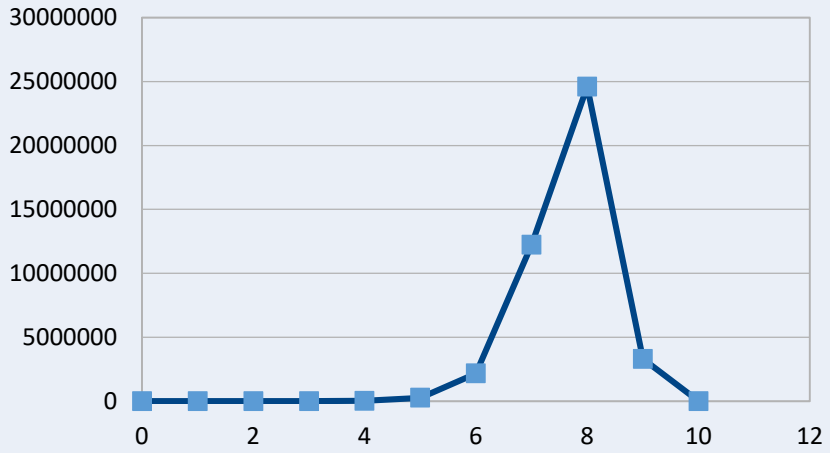
Pattern Database Results

	Corner DB	42.05MiB			
		Number of moves from solved state	Number of states		
	0.0000011%	0	1	0	
	0.0000204%	1	18	18	
	0.0002756%	2	243	486	
	0.0032592%	3	2874	8622	
	0.0317533%	4	28000	112000	
	0.2329512%	5	205416	1027080	
	1.3251510%	6	1168516	7011096	
	6.1268290%	7	5402628	37818396	
	23.5611405%	8	20776176	166209408	
	51.4761832%	9	45391616	408524544	
	17.1690219%	10	15139616	151396160	
	0.0734136%	11	64736	712096	
			88179840	772819906	
				8.76413368	average heuristic



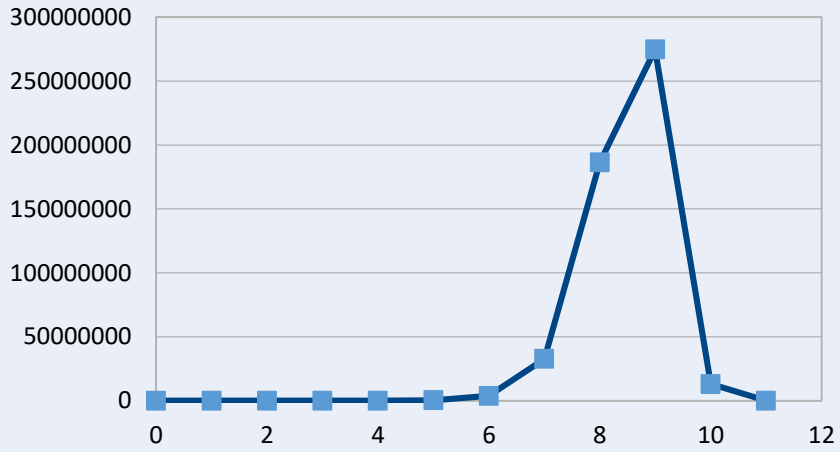
Pattern Database Results

6 Edge DB		20.30MiB			
		Number of moves from solved state	Number of states		
	0.0000023%	0	1	0	
	0.0000352%	1	15	15	
	0.0004275%	2	182	364	
	0.0051858%	3	2208	6624	
	0.0594886%	4	25329	101316	
	0.6078902%	5	258827	1294135	
	5.0861104%	6	2165560	12993360	
	28.7066818%	7	12222708	85558956	
	57.7687966%	8	24596752	196774016	
	7.7645244%	9	3305973	29753757	
	0.0008573%	10	365	3650	
			42577920	326486193	
				7.667969525	average heuristic



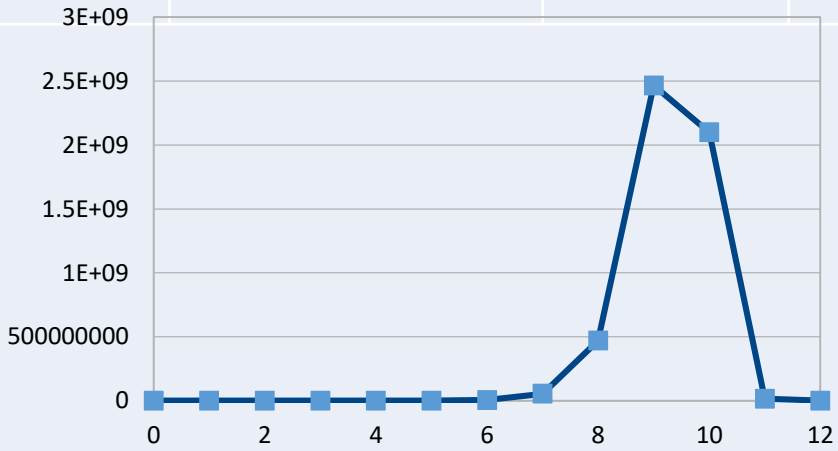
Pattern Database Results

7 Edge DB		243.63MiB			
		Number of moves from solved state	Number of states		
	0.0000002%	0	1	0	
	0.0000029%	1	15	15	
	0.0000374%	2	191	382	
	0.0004805%	3	2455	7365	
	0.0059732%	4	30519	122076	
	0.0697666%	5	356462	1782310	
	0.7372170%	6	3766700	22600200	
	6.4038409%	7	32719467	229036269	
	36.4619755%	8	186297009	1490376072	
	53.7680158%	9	274719633	2472476697	
	2.5526742%	10	13042507	130425070	
	0.0000159%	11	81	891	
			510935040	4346827347	
				8.507592955	average heuristic



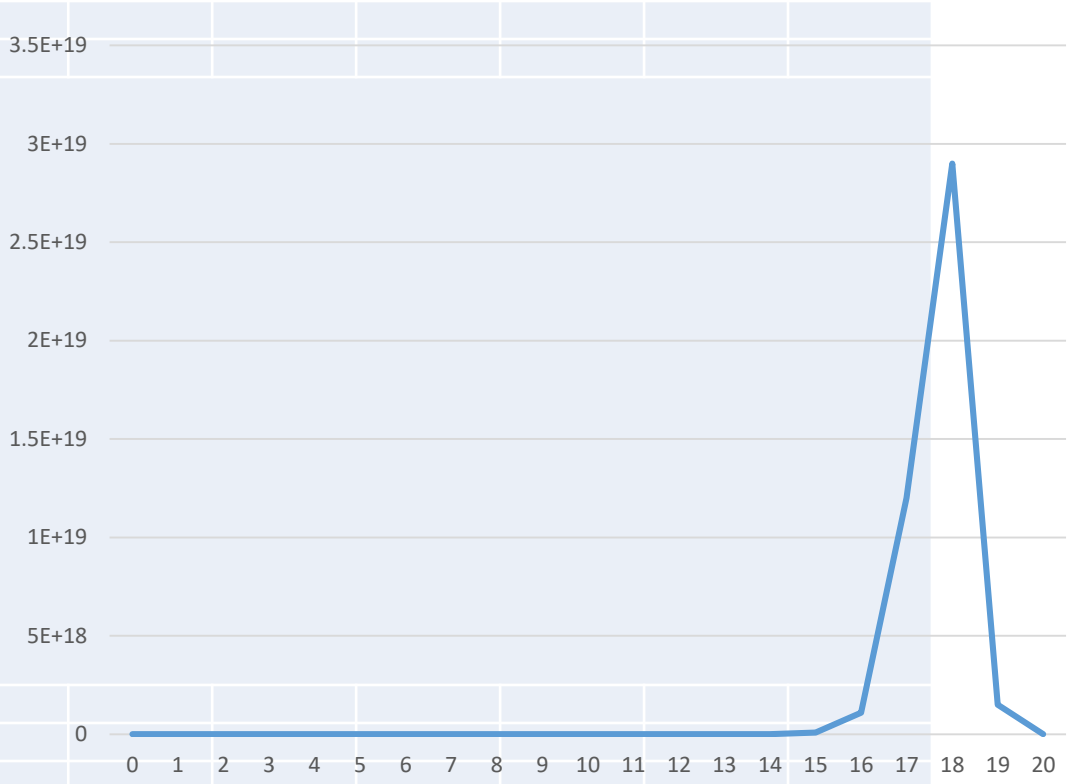
Pattern Database Results

8 Edge DB	2436.33MiB			
	Number of moves from solved state	Number of states		
0.000000020%	0	1	0	
0.000000294%	1	15	15	
0.000003875%	2	198	396	
0.000051474%	3	2630	7890	
0.000666093%	4	34033	136132	
0.008290154%	5	423573	2117865	
0.097858546%	6	4999936	29999616	
1.051052870%	7	53701974	375913818	
9.202244399%	8	470174911	3761399288	
48.255789699%	9	2465557384	22190016456	
41.073059992%	10	2098566555	20985665550	
0.310982253%	11	15889173	174780903	
0.000000333%	12	17	204	
		5109350400	47520038133	
			9.300602701	average heuristic



Database for the entire cube

2.28839E-20	0	1	0
4.11909E-19	1	18	18
5.56078E-18	2	243	486
7.41437E-17	3	3240	9720
9.89475E-16	4	43239	172956
1.31561E-14	5	574908	2874540
1.74339E-13	6	7618438	45710628
2.30676E-12	7	100803036	705621252
3.04892E-11	8	1332343288	1.0659E+10
4.02675E-10	9	17596479795	1.5837E+11
5.31473E-09	10	2.32248E+11	2.3225E+12
7.00999E-08	11	3.06329E+12	3.3696E+13
9.23923E-07	12	4.03744E+13	4.8449E+14
1.21663E-05	13	5.31653E+14	6.9115E+15
0.000159943	14	6.98932E+15	9.785E+16
0.002090787	15	9.13651E+16	1.3705E+18
0.025172241	16	1.1E+18	1.76E+19
0.274606267	17	1.2E+19	2.04E+20
0.663631813	18	2.9E+19	5.22E+20
0.034325783	19	1.5E+18	2.85E+19
1.12131E-11	20	490000000	9800000000
		4.36989E+19	7.7358E+20
			17.702396 average heuristic



Heuristic Search Algorithm

- A*
  - Time efficiency – exponential
  - Space efficiency – exponential
- Iterative deepening A*
  - Time efficiency – exponential
  - Space efficiency – O(bd)

Demo

- 18 move scramble: B2 F' L2 B D U' L' R' U2 L D' L' U R' F2 B U2 R’
- 18 move solution: U R' B' U L F U' B' L' B2 R2 F' U' L F R2 U L

- 17 move scramble: L' B' R' B2 U2 L2 U2 L F' R F2 U' L' F' R' B D’
- 17 move solution: B' R F L U' F U2 D' F2 L2 R' F B R U' B2 R2

References

- Rokicki, Tomas, et al. “God's Number Is 20.” *God's Number Is 20*, www.cube20.org/.
- Korf, Richard E. “Finding Optimal Solutions to Rubik's Cube Using Pattern Databases.” *Https://Www.cs.princeton.edu*, www.cs.princeton.edu/courses/archive/fall06/cos402/papers/korfrubik.pdf.
- Clausecker, Robert. “Notes on the Construction of Pattern Databases.” *OPUS 4*, Zuse Institute Berlin, Nov. 2017, opus4.kobv.de/opus4-zib/frontdoor/index/index/docId/6558.