Algorithm Filtering - Algorithm similarity measure

10 June 2024 22:47

Goal:
In algorithm generator, prevent generating many, very similar algorithms.

Definitions:
Algorithms are sequences of base moves. Their effects on the puzzle can be described by a single permutation, just like each base move. For more information, we can decompose the puzzle into pieces (collections of points that can never be separated. These points would correspond to the same physical piece of the puzzle).

Cycles in a permutation are the sets obtained by following each element to where it gets mapped by repeatedly applying the same permutation until this element reaches its initial position again. (see cycle notation for permutations or here)

Now we can define several properties of algorithms that can indicate or disproof similarity between algorithms.
- Number of pieces affected
- For each piece type (number of points contained), find how many cycles of any given length move this piece type.

\Number of points in piece

	1	2	3	4
1				
2				
3				
4				

Cycle signature:
For each piece type, how many of that type are affected by the cycle

Each cycle in a move/ algorithm can only affect one type of piece:
- Either the cycle only affects points on the same piece, or
- The cycle moves points from one piece to another piece. Since the pieces are physical objects that can't be broken and we consider non-bandaging puzzles, pieces can only be moved to the locations of other pieces of the same "type". (this is enforced by the representation in the simulator)

For each cycle, calculate a **cycle signature** from:
- Number of pieces affected
- Types of pieces affected
- Length (=order) of the cycle

Algorithm signature contains:
- For each piece type, how many are affected by the algorithm
- Number of cycles with each cycle signature

Implicit information:
- Algorithm length = lowest common multiple of cycle lengths

Problems:
- This algorithm signature is not yet enough to reliably tell two algorithms apart. Some actually different algorithms may share the same signature.
  => we can't securely reject algorithms if we already found another one with the same signature.

Solution:
- Tell apart two algorithms with the same signature by checking how all base moves interact with the pieces changed by the algorithm.
- Check if algorithms are identical up to symmetry:
  For each algorithm, store all symmetric permutations of it and its inverse (alg composed with each symmetry)
  If a new algorithm is equal to any of these, it's identical. Otherwise, it's new.

Difficulties:
- There can be different move sequences resulting in the same permutations
  => telling apart algorithms based on their moves is not reliable

- Identical move sequence => identical algorithm effect
  But NOT:
  identical algorithm effect => similar move sequence
  Unknown:
  Similar move sequence => similar algorithm (most likely not)

Example 1:
algorithm turning two edges of a rubik's 3x3x3 by 180°
Cycles: [
    (1, 2),
    (3, 4),
]

Cycle signatures: [
    (length=2, n_pieces=1, piece_type=2),
    (length=2, n_pieces=1, piece_type=2),
]

Algorithm signature: [
    (piece_type=2, num=2),
    ((length=2, n_pieces=1, piece_type=2), num=2),
]

Example 2:
algorithm swapping three edges of a rubik's 3x3x3 cyclicly
Cycles: [
    (1, 3, 5),
    (2, 4, 6),
]

Cycle signatures: [
    (length=3, n_pieces=3, piece_type=2),
    (length=3, n_pieces=3, piece_type=2),
]

Algorithm signature: [
    (piece_type=2, num=3),
    ((length=3, n_pieces=3, piece_type=2), num=2),
]

Example 3:
algorithm turning four edges of a rubik's 3x3x3 by 180°
Cycles: [
    (1, 2),
    (3, 4),
    (5, 6),
    (7, 8),
]

Cycle signatures: [
    (length=2, n_pieces=1, piece_type=2),
    (length=2, n_pieces=1, piece_type=2),
    (length=2, n_pieces=1, piece_type=2),
    (length=2, n_pieces=1, piece_type=2),
]

Algorithm signature: [
    (piece_type=2, num=2),
    ((length=2, n_pieces=1, piece_type=2), num=4),
]

Example 4:
algorithm turning two corners of a rubik's 3x3x3 by 120° and swapping two edges.
Cycles: [
    (a, b, c),
    (d, e, f),
    (1, 3),
    (2, 4),
]

Cycle signatures: [
    (length=2, n_pieces=2, piece_type=2),
    (length=2, n_pieces=2, piece_type=2),
    (length=3, n_pieces=1, piece_type=3),
    (length=3, n_pieces=1, piece_type=3),
]

Algorithm signature: [
    (piece_type=2, num=2),
    (piece_type=3, num=2),
    ((length=2, n_pieces=2, piece_type=2), num=2),
    ((length=3, n_pieces=1, piece_type=3), num=2),

]