Math Research Proof

Michael Moorman, Gabe Quijano, Hugh Williams

October 2022

Maximal Set Generating Algorithm

For all following algorithms, 0-indexing is standard unless explicitly specified, such as the indices of the digits of an orbit.

Input : The degree d, rotational number $\frac{p}{q}$, and an orbit $\mathcal{O}$ with it's digits in ascending order $\{\mathcal{O}_1, \mathcal{O}_2, ..., \mathcal{O}_q\}$
Output : A set of all maximal rotational sets containing $\mathcal{O}$, denoted M

let N = an array of the integers in $\mathcal{O}$;
while $N_1 \neq 0$ **do**
    foreach $i = 1 \ldots, q$ **do**
        $N_i = N_i - 1$;
let `gapSizes` = getGapSizes($\mathcal{O}$, p, q);
let `maximalSets` = {}, an empty set;
foreach $i = 0 \ldots N_q - 1$ **do**
    let `placements` = {}, an empty list;
    let S = the range $[i, d-3]$ of integers, inclusive;
    foreach *combo* $\in \binom{S}{d-1-N_q}$ [a] **do**
        let `placement` = {}, an empty list;
        foreach $j = 0 \ldots d-3$ **do**
            if $j \in$ *combo* **then** placement.append($q-1$);
            else placement.append($\emptyset$);
        placements.append(placement);
    let `placements` = `fillGap`(i, 0, gapSizes, $N_q - 1$, placements);
    foreach *placement* $\in$ *placements* **do**
        maximalSets = maximalSets $\cup$ `convertPlacementToSet`(placement,d, p, q);
return `maximalSets`

Algorithm 1: `MaxSetGeneration()`: let N_j represent the jth digit of N in ascending order

[a] Allow this to denote the set of all $d-1-N_q$ combinations of elements of S, similarly to the combination in `fillGap()`

Input : position, currentGap, gapSizes, numPreimagesLeft, placements
Output : A list of preimage placements that fulfill the conditions set out by given gap sizes required to make the maximal rotational set contain the orbit.

```
1  if numPreimagesLeft == 0 then return placements;
2  let newPlacements = {}, an empty list;
3  let S = the range [0, numPreimagesLeft−1] of integers, inclusive;
4  foreach i = 0 . . . |placements| − 1 do
5  |  foreach combo ∈ (S choose gapSizes[currentGap]) do
6  |  |  newPlacements.append(placements[i]);
7  |  |  let emptyGaps = 0;
8  |  |  foreach j = 0 . . . |newPlacements[−1][b]| − 1 do
9  |  |  |  if newPlacements[−1][j] = ∅ then
10 |  |  |  |  if emptyGaps ∈ combo then
11 |  |  |  |  |  if j < position then newPlacements[−1][j] = currentGap+1;
12 |  |  |  |  |  else newPlacements[−1][j] = currentGap;
13 |  |  |  |  emptyGaps = emptyGaps +1;
14 return fillGap(position, currentGap + 1, gapSizes, numPreimagesLeft -
     gapSizes[currentGap], newPlacements)
```

Algorithm 2: Auxiliary function: `fillGap()`

[b] Allow this to represent the index of the last element in this list.

Input : an orbit $\mathcal{O}$ with it's digits in ascending order $\{\mathcal{O}_1, \mathcal{O}_2, ..., \mathcal{O}_q\}$, the corresponding rotational number $\frac{p}{q}$
Output :

```
1  let sizes = {}, an empty list;
2  foreach i = 1 . . . q − 1 do
3  |  sizes.append(O_{i+1} − O_i);
4  sizes[q − p − 1] = sizes[q − p − 1] − 1;
5  return sizes;
```

Algorithm 3: Auxiliary function: `getGapSizes()`

Correctness

Proof. ■

Runtime

The runtime of this algorithm is _

Proof. ■