High dimensional game of life with low dimensional initial state

Michal Maršálek, reddit.com/u/mstksg/, reddit.com/u/p_tseng

January 6, 2021

Abstract

This document summarises results and ideas from a reddit thread about a generalised version of a problem from Advent of code. This problem is about a Game of life, a 0-player game invented by John Conway. In this case we are in a setting of high dimensions but with special initial conditions that give rise to many interesting patterns and structures.

1 Description of the problem

1.1 Game of life

Let $d \in \mathbb{N}$. Consider the space $\mathbb{Z}^d$. Each element of this space is called a *cell* and each subset of this space is called a *state*. If cell $x \in S \subset \mathbb{Z}^d$, we say that cell x is *alive*, otherwise, we say that it's *dead*.

Let $a, b \in \mathbb{Z}^d$. If $\max |a_i - b_i| \leq 1$, we say that cells a and b are neighbours. By $\texttt{neigh}(a)$ we denote all neigbours of b.

Note: in this notation we consider a cell to be a neighbour of itself. That is each cell has exactly 3^d neighbours.

Game of life is a mapping from states to states,

$$\texttt{nxt}_{\mathcal{A},\mathcal{D}} : \mathcal{P}(\mathbb{Z}^d) \to \mathcal{P}(\mathbb{Z}^d)$$

such that

$$a \in \texttt{nxt}_{\mathcal{A},\mathcal{D}}(S) \iff \begin{cases} a \in S \wedge |\texttt{neigh}(a) \cap S| \in \mathcal{A} \\ \text{or} \\ a \notin S \wedge |\texttt{neigh}(a) \cap S| \in \mathcal{D} \end{cases}$$

where $\mathcal{A}, \mathcal{D} \subset \mathbb{N}$.

Note: we usually omit the indeces $\mathcal{A}, \mathcal{D}$.

State $\texttt{nxt}(S)$ is called the *next (time step)* state of the state S. Function $\texttt{nxt}$ is also called the *step* function, since it determines the next (time) step of the game of life.

In another words:

- if a cell is alive and has $n \in \mathcal{A}$ alive neighbours it survives, otherwise it dies,
- if a cell is dead and has $n \in \mathcal{D}$ alive neighbours it comes to life, otherwise it remains dead.

By *Game of life with initial state* we mean a pair $(S_0, \texttt{nxt}_{\mathcal{A},\mathcal{D}})$ uniquely generating a sequence

$$S_{i+1} = \texttt{nxt}_{\mathcal{A},\mathcal{D}}(S_i).$$

1.2 Low dimensional initial state and problem statement

Let $\ell, k \in \mathbb{N}^+, \ell + k = d$.

Let $\widehat{S_0} \subset \mathbb{Z}^\ell$, $S_0 = \pi_d(\widehat{S_0}) = \widehat{S_0} \times \{0\}^k = \{x||0^k; x \in \widehat{S_0}\} \subset \mathbb{Z}^d$.

We say S_0 is a *ℓ-dimensional state* in d-dimensional Game of life.

The problem we are trying to solve is

$$\text{Given } k, \ell \in \mathbb{N}^+, \ell \ll k, d = k + \ell, \texttt{nxt}, \widehat{S_0} \subset \mathbb{Z}^\ell, t \in \mathbb{N}^+, \text{ determine}$$
$$|S_t| = |\texttt{nxt}^t(S_0)| = |\texttt{nxt}^t(\pi_d(\widehat{S_0}))|.$$

1.3 Generalisation/specialisation

This problem is a generalisation of Advent of code - year 2020 - day 17.

In this state of the research/experimenting we focus on the cases of $\ell = 2, t = 6$, $\widehat{S_0} \subset \{6, \ldots, 13\}^2$.

2 Solution methods

2.1 Brute force

In this approach we don't use the fact that the initial state is low dimensional. Each alive cell tells all it's neigbours that it's there. Then, we go trough all cells that have at least one alive neighbour and set it alive or dead base on $\mathcal{A}$ and $\mathcal{D}$.

Algorithm 1 Bruteforce

Input: $d, \mathcal{A}, \mathcal{D}, \widehat{S_0}, t$
Output: solution

```
function NXT(S ⊂ ℤ^d)
    counter ← empty table(ℤ^d) → ℕ with default value = 0
    result ← {}
    for all a ∈ S do
        for all b ∈ NEIGH(a) do
            counter[b] ← counter[b] + 1
    for all (a, neigh_count) ∈ counter do
        if (a ∈ S ∧ neigh_count ∈ 𝒜) ∨ (a ∉ S ∧ neigh_count ∈ 𝒟) then
            result ← result ∪ {a}
    return result
S ← π_d(Ŝ_0)
for all i = 1..t do
    S ← NXT(S)
return |S|
```

The fast exponential growth of the number of neigbours: 3^d as well as the alive cells makes the time complexity of this approach grow very fast with growing d.

2.1.1 Technical details

If we know that each coordinate will fit into b bits (we know that the maximum coordinate can only grow by 1 for each time step) we can pack the whole cell into a single $d \times b$-bit integer.

This version is implemented in nd_gol.nim.

For our input state (8×8) and $t = 6$, this naive approach only works for low dimensions: $d = 6$ takes 8 seconds, $d = 7$ takes 4 minutes.

2.2 Symmetries

The low dimension of the initial state gives us tremendous advantage. The game of life evolves in very symmetric ways and we can use its structure to speed up the computation.

Each cell now breaks into ℓ dimensional general component *gen* and a k dimensional symmetric component *sym*.
Let $\varphi : \mathbb{Z}^k \to \mathbb{N}_0^{\mathbb{N}_0}, \varphi(\text{sym}) = \{* \ |\text{sym}_i|; i = 1..k \ *\}$. (We interpret $m \in \mathbb{N}_0^{\mathbb{N}_0}$ as multiset of values in $\mathbb{N}_0$, employ notation $m(i)$ to mean number of occurences of i in m and $\{**\}$ also denotes multiset). Consider the following equivalence

$$a \simeq b \iff a_{1..\ell} = b_{1..\ell} \quad \wedge \quad \varphi(a_{\ell+1..d}) = \varphi(b_{\ell+1..d})$$

The key observation is that *equivalent cells are always either all alive or all dead.*

Verification of this fact as well as the fact that $\simeq$ is equivalence is left as an exercise for the reader.

With this knowledge we can devise a much more efficient algorithm by only tracking which cosets $[A] \in \mathcal{P}[\mathbb{Z}^d]/\simeq$ are alive.

First, we need to solve two problems:

1. How to get the final number of alive cells from a list of alive cosets?
2. How to count alive neigbours?

2.2.1 Determining the size of a coset

Consider the function $\texttt{final\_w}(\text{gen}, \text{sym}) = |[(\text{gen}, \text{sym})]|$. A quick thought tells us, that

$$\texttt{final\_w}(\text{sym}) = 2^{k-\varphi(\text{sym})(0)} \frac{k!}{\prod_{i=0}^{\infty} \varphi(\text{sym})(i)!}.$$

In another words it is the number of distinct permutations multiplied by 2 for each nonzero coordinate.

2.2.2 Counting neighbours

The nontrivial part is handling the neighbours in the symmetric component. Neighbours of $\text{sym} \in \mathbb{Z}^k$ are exactly such points that we get by decrementing some coordinates by one and incrementing some coordinates by one. In the φ representation, it manifests as some amount of i's changing to $i-1$'s and some amount changing to $i+1$'s. Note that in this representation 0's can only change to 1's but this action can be a manifestation of two different changes in $\mathbb{Z}^k$ ($0 \to -1$ and $0 \to +1$).

Furthermore in the φ representation, some changes can cancel out, yielding the same element. For example $(1,2)$ neighbours $(2,1,1)$ but $\varphi((1,2,1)) = \{*1,1,2*\} = \varphi((2,1,1))$. In a similar way a set of different changes to a point can all produce the same point in the φ representation that nevertheless differs from the original point (like $(1,1) \to (1,2)$ and $(1,1) \to (2,1)$ where $\varphi((1,2)) = \varphi((2,1)) \neq \varphi((1,1))$.

Consider $s \in \varphi(\mathbb{Z}^k)$ representing a symmetrical component of a state after t time steps.
Then

$$s \in \mathbb{N}_0^{\{0..t\}}, \sum_{i=0}^{t} s(i) = k$$

.

Let $L_i \in \mathbb{N}_0, i = 1..t-1$ denote the number i's in s that change to $(i-1)$'s.
Let $R_i \in \mathbb{N}_0, i = 0..t-1$ denote the number i's in s that change to $(i+1)$'s.
Of course, we require $L_i + R_i \leq s(i)$.
These vectors L, R identify all possible neighbours of $s \in \varphi(\mathbb{Z}^k)$ but they don't identify them *uniquely*.
Let us put

$$F_i = R_i - L_{i+1}, i = 0..t-1.$$

We refer to the vector F as a *flow* between the coordinates of s. The flow is what uniquely identifies each and every possible neighbour s_2 of s as

$$s_2(i) = s(i) - F_i + F_{i-1}, i = 0, ..., t,$$

where F_{-1} and F_t are understood as 0.

To summarize, to enumerate all neighbours (neigbouring cosets), we need to enumerate all possible flows. But to count the neighbours (we need the count of actual neigbours, not just cosets), we need to further go trough all pairs of (L_i, R_i) that yield the corresponding F_i as there's many of such pairs that yield a different actual point but the same coset.

While enumerating all possible ways that s neigbours s_2 we can exit early once we find that the multiplicity of the neighbouringness is greater than $\max \mathcal{A} \cup \mathcal{D}$ as at that point, we already know that the point (coset) will be dead next in the next step no matter what.

Notation: Let $a \in \mathbb{R}$. We denote $a^- = \min(a, 0), a^+ = \max(a, 0)$.

Let `getNeigbours` be a function that maps a symmetric component of a coset $s \in \varphi(\mathbb{Z}^k)$ to a list neighbours along with the multiplicities.

Algorithm 2 Using symmetries

Input: $d, \mathcal{A}, \mathcal{D}, \widehat{S_0}, t$
Output: solution

```
function GETNEIGHBOURS(s ∈ φ(Z^k))
    for all F_0 = −s(1), ..., s(0) do
        for all F_1 = −s(2), ..., s(1) + F_0^− do
            ⋱
            for all F_{t−2} = −s(t − 1), ..., s(t − 2) + F_{t−3}^− do
                for all F_{t−1} = 0, ..., s(t − 1) + F_{t−2}^− do
                    s_2(i) ← s(i) − F_i + F_{i−1}, i = 0, ..., t
                    w ← 0
                    for all R_0 = F_0^+, ..., s(0) do
                        L_1 ← R_0 − F_0
                        m ← 2^{L_1}
                        for all R_1 = F_1^+, ..., s(1) − F_1 do
                            L_2 ← R_1 − F_1
                            m ← m · binom(s_2(1), R_0) · binom(s_2(1)−R_0, L_2)
                            ⋱
                            for all R_{t−2} = F_{t−2}^+, ..., s(t − 2) − F_{t−2} do
                                L_{t−1} ← R_{t−2} − F_{t−2}
                                m ← m · binom(s_2(t−2), R_{t−3}) · binom(s_2(t−2)−R_{t−3}, L_{t−1})
                                m ← m · binom(s_2(t−1), R_{t−2})
                                w ← w + m
                                if w > max A ∪ D then
                                    go to enough
                    label enough
                    yield (s_2, w)
function NXT(S ⊂ Z^ℓ × φ(Z^k))
    counter ← empty table(Z^ℓ × φ(Z^k) → N) with default value = 0
    result ← {}
    for all (gen_1, sym_1) ∈ S do
        for all gen_2 ∈ NEIGH(gen_1) do
            for all (sym_2, w) ∈ GETNEIGHBOURS(sym_1) do
                counter[(gen_2, sym_2)] ← counter[(gen_2, sym_2)] + w
    for all (a, neigh_count) ∈ counter do
        if (a ∈ S ∧ neigh_count ∈ A) ∨ (a ∉ S ∧ neigh_count ∈ D) then
            result ← result ∪ {a}
    return result
S ← Ŝ_0 × {∗0^k∗}
for all i = 1..t do
    S ← NXT(S)
return Σ_{(gen,sym)∈S} FINAL_W(sym)
```

2.2.3 Technical details

The expression

$$f(s_2(i), R_{i-1}, L_{i+1}) = \binom{s_2(i)}{R_{i-1}}\binom{s_2(i) - R_{i-1}}{L_{i+1}}$$

amounts to the number of different ways that quantity $s_2(i)$ can be partitioned into 3 parts of sizes R_{i-1}, L_{i+1} and the rest. In another words it counts how many ways we can realise the resulting amount of i's in s_2. We can therefore replace this expression with any function g such that

$$g(s_2(i), R_{i-1}, L_{i+1}) > \max(\mathcal{A} \cup \mathcal{D}) \text{ if } f(s_2(i), R_{i-1}, L_{i+1}) > \max(\mathcal{A} \cup \mathcal{D})$$

and

$$g(s_2(i), R_{i-1}, L_{i+1}) = f(s_2(i), R_{i-1}, L_{i+1}) \text{ otherwise}$$

without changing the result, as at that point, we know the cell (coset) will be dead no matter what.

We can pack the symmetric part $s \in \varphi(\mathbb{Z}^k)$ into a single $(t+1)\log_2(k)$ bit integer. Furthermore we pack it together with the asymmetric part.

Evaluation of `getNeighbours` can be precomputed (or memoized) and then reused across multiple time steps as well as across multiple cosets, that differ in the asymmetric component, but share the symmetric one. For our case the precomputation was only beneficial for lower dimensions as with higher d's states become sparse.

This version is implemented in nd_gol3.nim (with precomputation) and nd_gol3_single.nim (without precomputation).

This version solves $d = 10$ in under 1 second, $d = 20$ in under 1 minute and $d = 30$ in under 16 minutes.

2.3 Further structure

While playing around with the problem we noticed several interesting facts about the structure of the states in the Game of life with low dimensional initial state.

1. For higher dimensions, final number of active cosets after given number of time steps t follows a exact pattern. For one of the inputs:
   (a) At $t = 1$, a linear progression $21d - 15$ starting from $d = 2$,
   (b) At $t = 2$, a constant number 48 starting from $d = 4$,
   (c) At $t = 3$, a quadratic progression $15.5d^2 + 29.5d - 166$ starting from $d = 4$,
   (d) At $t = 4$, a constant number 147 starting from $d = 7$,
   (e) At $t = 5$, a quadratic progression $51d^2 - 62d - 173$ starting from $d = 7$,
   (f) At $t = 6$, a quadratic progression $d^2 + 109d + 70$ starting from $d = 7$.

   What is more the 147 cosets after are the same points (modulo some transformation for making the dimensions work). This point enables us to:
   (a) Predict the number of active cosets for any d in virtually no time.
   (b) Calculate the final answer after $t = 4$ in virtually no time.
   (c) Slightly speed up the calculation to $t > 4$ by skipping the first 4 time steps.
2. For higher dimensions, the number of unique sets of sym's found in the state becomes constant. For one of the inputs, there is only 49 different sets of symmetric parts found. For another one, it was 102. This point allows us to introduce another speed up to our algorithm, by memoizing the cummulative effect of each such set to their neighbours.
3. Multiplicities follow sequences in OESIS (TODO).

Remark: The above points are only true for our inputs and $\ell = 2, t = 6 (t \leq 6)$. We do not yet understand where they come from or how much they can be generalised.

2.4 Using cummulative effects of sets of cosets

TODO

2.4.1 Technical details

This version is implemented in nd_gol4.nim and solves $d = 10$ in 0.1 seconds, $d = 20$ in 5 seconds, $d = 30$ in 1 minute, $d = 40$ in 8 minutes and $d = 50$ in half an hour.

3 Open questions

1. Proof of quadratic progressions of coset counts.
2. Extrapolate the multiplicities under each gen and explain the behaviour.
3. Proof the low (and constant!) number of unique sets of sym's.
4. Explore the treshold for d for the regularities to occur and understand the structure that triggers it (like relationship of d and t).
5. Experiment with different inputs, different t's and different ℓ's.

4 Sources

TODO replace this with proper LaTeX sources.

1. Advent of code, `https://adventofcode.com/2020/day/17`
2. [Day 17] Getting to t=6 at for higher <spoilers>'s `https://www.reddit.com/r/adventofcode/comments/kfb6zx/day_17_getting_to_t6_at_for_higher_spoilerss/`
3. github.com/MichalMarsalek, `https://github.com/MichalMarsalek/Advent-of-code/tree/master/2020/misc/day17-highdims`