Automat komórkowy Von Neumana

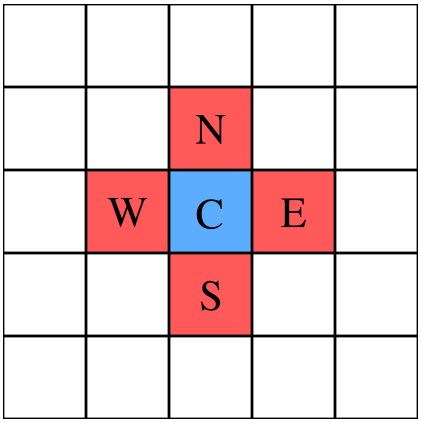
Paweł Buglewicz, Krzysztof Cieśla, Justyna Olczak

31 października 2016

1 Wstęp

Automatem komórkowym nazywamy układ, którego stan w danej chwili jest zależny tylko i wyłącznie od swojego stanu w chwili poprzedniej. Taki układ opisywany jest na siatce dyskretnej. Stan każdego punktu na takiej siatce określa się w danym kroku za pomocą pewnego zbioru zasad.

Zasady te określają jak zmieniają się właściwości danego punktu w zależności od jego stanu i stanu jego sąsiadów. Sąsiadami danego punktu, inaczej nazywanego też komórką, są punkty mu najbliższe. Dla sieci kwadratowej najczęściej używaną definicją sąsiedztwa jest 5-sąsiedztwo, nazywane też sąsiedztwem von Neumanna, przedstawione na rys. 1. Jest to najprostszy do zdefiniowania rodzaj sąsiedztwa. Z tego powodu jest on również najczęściej używany.

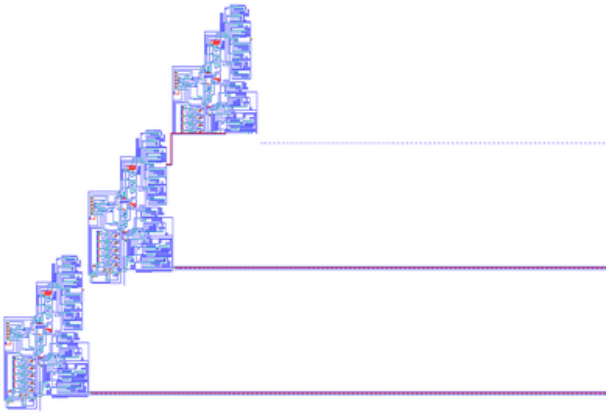


Rysunek 1: Sąsiedztwo von Neumana.

Dana komórka w automacie może przyjmować dowolną, całkowitą ilość stanów. Bardzo prostym i popularnym przykładem automatu komórkowego jest „gra w życie Conwaya". Gdzie każda komórka może znajdować się jedynie w 2 stanach, a całość jest określana przez 6 zasad. W ogólności dla n-sąsiedztwa i k stanów układ jest określony przez n^k zasad. Liczba zasad powiększa się niezwykle szybko, co jest ogromną przeszkodą w poszukiwaniach automatów komórkowych o pożądanych właściwościach, gdzie zasady muszą być odpowiednio dobrane.

Uniwersalnym automatem komórkowym von Neumanna nazywamy automat komórkowy posiadający zdolność samo-replikacji. Oznacza to, że dana konfiguracja komórek korzystając z pewnego zestawu zasad po skończonej liczbie kroków wytwarza dokładną kopię samej siebie, całkowicie niezależnej.

Przez niezależność rozumiemy to, że te dwie konfiguracje w dalszych krokach nie oddziałują na siebie, a także nie znajdują się w swoim bezpośrednim sąsiedztwie. Przykład takiego automatu znajduje się na rys. 2



Rysunek 2: Przykład automatu von Neumana - implementacja Nobili-Pesavento dla 32 różnych stanów.

Potencjalnym zastosowaniem takiego automatu komórkowego jest kolonizacja odległych planet. Posiadający możliwość samoreplikacji automat komórkowy mógłby niezwykle szybko rozprzestrzeniać się po galaktyce. Odbywałoby się to według następujących zasad:

- automat dodarłszy na planetę sprawdza, czy nadaje się ona do zamieszkania przez ludzi,
- jeśli nie, używa zasobów planety, aby wytworzyć jak największą liczbę kopii samego siebie,
- następnie kopie te wysyłane są w kierunku najbliższych planet, gdzie procedura jest powtarzana.

Jeśli planeta jest zdatna do zamieszkania przez ludzi, to taki automat, posiadając ludzkie DNA, kopiuje je, zaczynając istnienie nowej kolonii.

Oprócz dylematów etycznych dotyczących niszczenia planet (nie ma takich - przyp. KC) pojawiają się także problemy czysto techniczne.

Automat taki będzie musiał mieć bardzo dużą złożoność, prawdopodobnie złożoną tak samo, lub bardziej niż ludzkie ciało. Tymczasem jak dotąd

udało się znaleźć tylko kilkadziesiąt automatów komórkowych spełniających założenia uniwersalnego automatu von Neumanna. Poszukiwanie takich maszyn jest bardzo skomplikowanym procesem. Znalezione jak dotąd automaty są bardzo proste, o małej liczbie stanów i rozmiarach.

2 Projekt

Naszym pomysłem było wyszukiwanie takich automatów za pomocą losowo generowanych zasad. Podobne badania odbywają się na MIT.

Losowanie zasad w języku Python odpowiada stworzeniu słownika, gdzie kluczem są wszystkie możliwe krotki w formacie C, N, E, S, W, wartością zaś liczba wylosowana wśród wszystkich możliwych stanów.

Przeszukiwanie słownika w pythonie jest bardzo szybkie. Stosowane są listy hashowane, które są obecnie najszybszym dostępnym sposobem przeszukiwania dużych zbiorów danych.

Korzystamy z 5-sąsiedztwa i ustalonej liczby stanów (5). Stan układu podlega ewolucji przez odpowiednią liczbę kroków, a następnie jest sprawdzane, czy wytworzyły się tam odpowiednie struktury. Strukturę rozumiemy jako grupę komórek, która nie ma innych struktur w najbliższym sąsiedztwie (w rozumieniu von Neumanna).

Struktury są zliczane i sprawdzane pod względem liczby powtórzeń. Te, które występują wielokrotnie mogą być kandydatami na maszynę von Neumanna i powinny być sprawdzone oddzielnie.

Jednak prawie wszystkie zestawy zasad (przy dużej ilości dostępnych stanów) prowadzą do chaosu, gdzie nie sposób odnaleźć nawet pojedynczej struktury, nie mówiąc już o wydzielaniu interesujących przypadków.

3 Podsumowanie

Nasz sposób działa, jednak znalezienie poszukiwanej maszyny jest praktycznie niemożliwe przy dostępnej mocy obliczeniowej. Sama ilość zestawów zasad jest przytłaczająca. Do tego należy doliczyć ilość możliwych stanów początkowych. Przeliczenie takich ogromnych zbiorów danych jest niemożliwe przy użyciu tak prostych algorytmów i przy dostępnej nam mocy obliczeniowej.

Oczywiście podobne badania są przeprowadzane na całym świecie, jednak przy użyciu superkomputerów, jak i przy użyciu znacznie bardziej wyrafinowanych metod.

4 Kod programu

```
#!/bin/env python
from itertools import product
import pygame
import sys
import numpy as np
import scipy.ndimage as spndimage
import cv2
from cv2 import matchTemplate as cv2m
import pickle
from datetime import datetime

listaWszystkichZasad = {}

def rysuj(plansza, rozmiar):
  for x in range(rozmiar):
    for y in range(rozmiar):
      pole(kolor[plansza[y][x]], x*piksel, y*piksel)

def pole(p, x, y):
  pygame.draw.rect(okno, p, (x, y, x+piksel, y+piksel))

def cross2list(m3x3):
  l = []
  l.append(m3x3[1][1])
  l.append(m3x3[0][1])
  l.append(m3x3[1][2])
  l.append(m3x3[2][1])
  l.append(m3x3[1][0])
  return tuple(l)

def porownaj(plansza, subPlansza, x, y):
  lista = cross2list(subPlansza)
  if lista != (0, 0, 0, 0, 0):
    var = listaWszystkichZasad.get(lista)
    if var is not None:
      plansza[x][y] = var

def sasiady(arr, x, y, n=3):
```

```
    arr=np.roll(np.roll(arr, shift=-x+1, axis=0), shift=-y+1, axis=1)
    return arr[:n, :n]

def szukajNaPlanszy(plansza):
    staraPlansza = np.copy(plansza)
    xRoz = plansza.shape[0]
    yRoz = plansza.shape[1]
    for x in range(xRoz):
        for y in range(yRoz):
            subPlansza = sasiady(staraPlansza, x, y)
            porownaj(plansza, subPlansza, x, y)

def struktury(m1):
    print("Szukam ksztaltow")
    ksztalt, numlabels = spndimage.measurements.label(m1)

    sk=[]
    ssk=[]
    lsk=[]

    for i in range(1, numlabels):
        objekt = spndimage.measurements.find_objects(ksztalt==i)[0]
        if objekt in sk:
            sk.append(0)
            lsk.append(0)
        if objekt not in sk:
            sk.append(objekt)
            M = cv2m(m1.astype('uint8'), m1[sk[i]].astype('uint8'), c
            ssk.append(m1[sk[i]])
            lsk.append(len(np.where(M == 0)[0]))
    # ssk, lsk = sz_obr(ssk,lsk)
    return ssk, lsk

# Main

kolor = []
kolor.append((255, 255, 255))
kolor.append((255, 0, 0))
kolor.append((0, 255, 0))
kolor.append((0, 0, 255))
kolor.append((255, 255, 0))
```

```
kolor.append((255, 0 , 255))
kolor.append((0, 255, 255))
kolor.append((255, 102, 109))
kolor.append((0, 17, 100))

nStan = 3
sasiedztwo = 5
nKrok = 100
rozmiar = 100
ekran = 600
piksel = ekran/rozmiar

wariacje = list(product(range(0, nStan), repeat=sasiedztwo))
del wariacje[0]
print(len(wariacje))
wartosci = np.random.randint(0, nStan, len(wariacje))
listaWszystkichZasad.update(dict(zip(wariacje, wartosci)))
pickle.dump( listaWszystkichZasad, open( str(datetime.now()) + ".p",

pygame.init()
pygame.display.set_caption('cellular automata')
okno = pygame.display.set_mode((ekran, ekran))
okno.fill((255, 255, 255))

plansza = np.zeros((rozmiar, rozmiar), dtype=int)
plansza[round(rozmiar/2)][round(rozmiar/2)] = 1

rysuj(plansza, rozmiar)
pygame.display.update()
# for krok in range(nKrok):
while True:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      pygame.image.save(okno, "zrzut.png")
      pygame.quit()
      s, k = struktury(plansza)
      for i in k:
        print(s[i])
      sys.exit()

  szukajNaPlanszy(plansza)
```

```
    rysuj(plansza, rozmiar)
    pygame.display.update()
```