VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

GENEROVÁNÍ PSEUDONÁHODNÝCH ČÍSEL CELULÁRNÍMI AUTOMATY

GENERATING PSEUDO-RANDOM NUMBERS BY MEANS OF CELLULAR AUTOMATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **LADA KROFINGEROVÁ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MICHAL BIDLO, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

148418

Ústav:	Ústav počítačových systémů (UPSY)
Studentka:	**Krofingerová Lada**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Generování pseudonáhodných čísel celulárními automaty**
Kategorie:	Umělá inteligence
Akademický rok:	2022/23

Zadání:

1. Seznamte se s principy a možnostmi návrhu celulárních automatů s důrazem na jejich použití pro generování pseudonáhodných čísel.
2. Zvolte vhodný způsob návrhu přechodových pravidel pro celulární automat, pomocí kterého bude možné generovat pseudonáhodná čísla s odpovídajícími statistickými vlastnostmi.
3. Navržený způsob implementujte a demonstrujte alespoň dva scénáře generování pseudonáhodných čísel pomocí celulárních automatů.
4. Vykonejte sadu experimentů se systémem z bodu 3 a proveďte statistické testy za účelem posouzení kvality generovaných posloupností.
5. Zhodnoťte dosažené výsledky a diskutujte možnosti pokračování projektu.

Literatura:
Dle pokynů vedoucího projektu.

Při obhajobě semestrální části projektu je požadováno:
Splnění bodů 1 a 2 zadání, demonstrace prototypu automatu z bodu 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Bidlo Michal, Ing., Ph.D.**
Vedoucí ústavu:	Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	31.10.2022

Abstrakt

Tato práce se zabývá generováním pseudonáhodných čísel celulárními automaty. Byly prozkoumány již používané metody, včetně postupů, které byly při návrzích celulárních automatů použity. Jako navázání na tyto metody byl navržen čtyřstavový celulární automat za účelem zlepšení kvality generovaných čísel. K návrhu tabulky pravidel byl použit genetický algoritmus. Pro porovnání s již používanými metodami bylo využito statistických testů. Ty ukázaly, že ačkoliv je čtyřstavový celulární automat dobrým generátorem pseudonáhodných čísel, ve stavu, ve kterém byl navržen, není lepší než již používané celulární automaty.

Abstract

This thesis deals with generating pseudo-random number by means of cellular automat. The methods which are already used were explored, including steps which were used to desing the cellular automata. As follow-up to these methods the four state celullular automaton was designed for the purpose of improving the quality of generated numbers. For design of the rule table was used the genetic algorithm. For comparision with already used methods were used the statistical tests. They showed that even though the four state cellular automaton is good generator of the pseudo-random numbers, in state which it was designed it isn't better in generating then already used cellular automata.

Klíčová slova

celulární automaty, pseudonáhodná čísla, generátory pseudonáhodných čísel, genetický algoritmus, čtyřstavový celulární automat

Keywords

cellular automata, pseudo-random numbers, pseudo-random number generators, genetic algorithm, four state cellular automaton

Citace

KROFINGEROVÁ, Lada. *Generování pseudonáhodných čísel celulárními automaty.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Bidlo, Ph.D.

Generování pseudonáhodných čísel celulárními automaty

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana Ing. Michala Bidla, Ph.D. Uvedla jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpala.

.....
Lada Krofingerová
8. května 2023

Poděkování

Chtěla bych velmi poděkovat Ing. Michalu Bidlovi, Ph.D. za trpělivost při vypracování a jakoukoliv pomoc při konzultacích, která byla potřeba. Dále velké díky patří mým úžasným holkám, které při mně stály celé tři roky bakalářského studia a poskytovaly mi převážně psychickou podporu. A v poslední řadě také přátelům, bez kterých by to také nebylo možné. Děkuji moc všem.

Obsah

Kapitola 1

Úvod

Pro správnou funkci mnnohých nástrojů a programů je třeba využít náhodných čísel. Nasimulovat ale náhodnost při výběru, byl výsledek opravdu náhodný, nemusí být vždy jednoduchý úkol. Jsou případy kdy samozřejmě nevadí, že generovaná náhodná čísla jsou na sobě nějakým způsobem závislá a je tedy možné snadno zjistit nadcházející číslo. Oproti tomu jsou ale případy, kdy toto není žádané a je třeba, aby se nedalo snadno uhodnout, jaké náhodné číslo bude jako další.

V dnešní době, kdy technika zabírá nemalou část našich životů, se generátory pseudonáhodných čísel, protože opravdu náhodná čísla nelze počítači generovat, využívají na mnoha místech. Využití najdou například v počítačových hrách, v matematických metodách jmenovitě při metodě Monte Carlo, simulací přírodních jevů, nebo například při šifrování pro zabezpečení posílaných zpráv například mimo jiné v dnes tolik rozšířeném internetovém bankovnictví.

Cílem této práce je prozkoumat už použité metody generátorů pseudonáhodných čísel a tři vybrané implementovat. Dalším cílem je navrhnout vícestavový celulární automat s využitím genetického algoritmu pro návrh přechodových pravidel, který by mohl posunout kvalitu generovaných pseudonáhodných čísel dopředu. Závěrečným cílem je poté porovnat implementované celulární automaty využité pro generování pseudonáhodných čísel mezi sebou a zjistit, zda nově navržený způsob přinese nějaké zlepšení do oblasti generování pseudonáhodných čísel.

V kapitole 2 jsou představeny celulární automaty, jimiž se práce primárně zabývá. V kapitole 3 jsou popsány metody generování náhodných čísel, které využívají jiné přístupy než celulární automaty. Kapitola 4 se zabývá generátory, které byly konkrétně v rámci práce implementovány. Následující kapitola 5 popisuje nové řešení generování pseudonáhodných čísel navržené dle nastudovaných článků. Poslední kapitola 6 poté popisuje testování a zhodnocení všech implementovaných generátorů.

Kapitola 2

Celulární automaty

V rámci této kapitoly budou představeny celulární automaty (dále pouze CA)[1], které jsou základním prvkem celé této práce, konkrétně obecné principy automatů, které jsou typově použité dále. Podkapitola 2.2 se týká v podstatě nejzákladnějších formy, jednorozměrného CA. Poslední podkapitola 2.3 popisuje poté trochu složitější typ, dvourozměrný CA.

2.1 Základní princip

Celulární automaty jsou diskrétní[2] abstraktní výpočetní systémy. Každý takový celulární automat se skládá z konečného počtu malých jednotek, označovaných jako buňky[4]. Odtud také pochází pojmenování celulární. Každá taková buňka má svůj vnitřní stav a po uběhnutí jedné časové jednotky přechází do jiného podle přechodové funkce, kterou má každá buňka určenou.

Celulární automat se skládá z několika složek. Jedná se konkrétně o tyto čtyři komponenty [15]:

- Pole buněk
- Konečná množina stavů
- Okolí buněk
- Přechodová pravidla

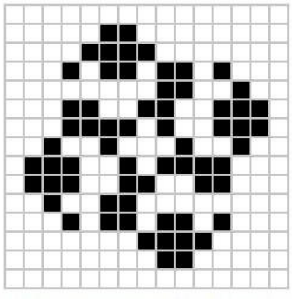
Pole buněk bývá obecně N-rozměrné, kde N je přirozené číslo. Typicky se však používají jednorozměrná, nebo dvourozměrná pole. Buňky v poli mohou mít různé tvary, pro jedno pole však platí, že všechny buňky jsou stejné. Tvar buňky může ovlivňovat i poté použité okolí. Příklady některých polí jsou na obrázcích 2.1. Důležitá je také velikost pole, která se může pro každý automat lišit.

Všechny buňky pracují s konečnou **množinou stavů**. V jednu chvíli může mít buňka pouze jeden stav z této množiny. Velmi často se využívají binární, nebo také dvoustavové celulární automaty, které mohou nabývat hodnot z množiny $\{0, 1\}$.

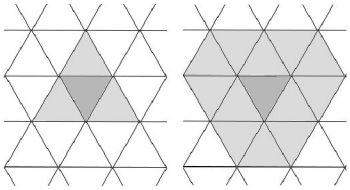
Pro přechod do dalšího stavu využívá buňka svého **okolí**. V rámci definice CA můžeme okolí chápat jako počet a pozice okolních buněk, které využívá k přechodu do dalšího stavu.

[1] Mimo zkratky CA bude v textu použito pro celulární automat označení i jen pouze „automat“. Pokud by byl myšlen automat (tedy konečný automat), bude to v textu zmíněno

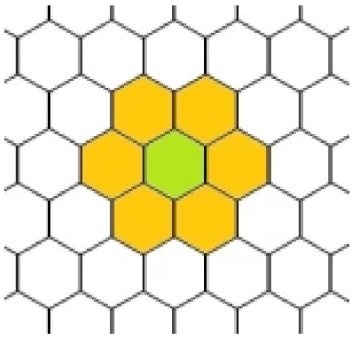
[2] Diskrétní systémy nepracují ve spojitém čase



(a) Pole s buňkami ve tvaru čtverce[3]



(b) Pole s buňkami ve tvaru trojúhelníku[4]



(c) Pole s buňkami ve tvaru šestiúhelníku[5]

Obrázek 2.1: Různé tvary polí CA

O různých typech okolí více pro jednorozměrné CA v podkapitole 2.2 a pro dvourozměrné v podkapitole 2.3.

Přechodová pravidla lze chápat jako funkci, která má na vstupu současné stavy okolí a na výstupu má nový stav řešené buňky. Nový stav lze poté získat následovně:

$$s(t+1) = f(s(t), N_s(t))$$

Označme si množinu stavů jako S. Poté platí: $s \in S$. $N_s(t)$ označuje stavy všech buněk okolí dané buňky v čase t, $s(t)$ je stav buňky v čase t a funkce $f(S, N_s$ je poté přechodová funkce, která mapuje stavy okolí na nový stav podle přechodových pravidel.

Pro účely práce je třeba zmínit jednu z vlastností celulárních automatů, uniformitu. Automat nazveme uniformním, pokud všechny buňky automatu využívají k přechodu do nového stavu stejnou přechodovou funkci. V případě neuniformního automatu má každá buňka svojí funkci [18]. V tomto případě se může stát, že některé buňky mají stejnou přechodovou funkci, ale neplatí to pro všechny buňky automatu.

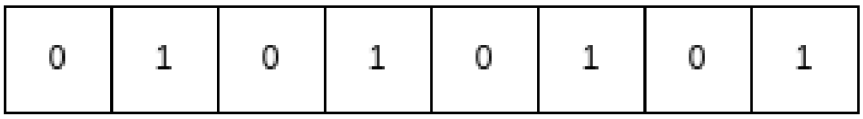
Celulární automaty obecně mají mnoho využití. Mimo generování pseudonáhodných čísel, čímž se zabývá tato práce, se využívají i pro simulace, například konkrétně epidemií, chemických reakcí, obecně biologických problémů nebo z oblasti fyziky, dále také v počítačové grafice v rámci procedurálního generování, nebo v oblasti zpracování obrazu, například i opravy obrazů poškozené šumem a další.

2.2 Jednorozměrné celulární automaty

Nejjednodnoduším typem celulárního automatu je jednorozměrný. Buňky automatu jsou umístěny v jednorozměrném poli, stejném jako využívá většina programovacích jazyků, viz obrázek 2.2.

Buňky jsou poté seřazeny za sebou a každá sousedí s jednou buňkou po pravici a levici. Přechodové funkce poté pracují s *radiem*, běžně značeným r. V práci jsou implementovány CA s okolím o velikost tři a pět buněk, ukázáno na obrázcích 2.3a a 2.3b. Radius označuje

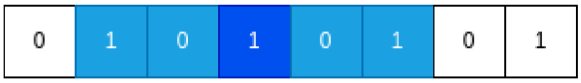
[5]Obrázek byl převzat z:[14]
[5]Obrázek byl převzat z:[20]
[5]Obrázek převzat z: https://geometricolor.wordpress.com/2013/01/11/hexagonal-cellular-automata/



Obrázek 2.2: Jednorozměrný celulární automat

celkový počet buněk, které se budou používat pro přechod. V případě radiu o velikosti tři se poté bere buňka, jejíž stav se počítá, a jedna buňka po pravici a jedna po levici.

(a) Ukázka trojokolí CA

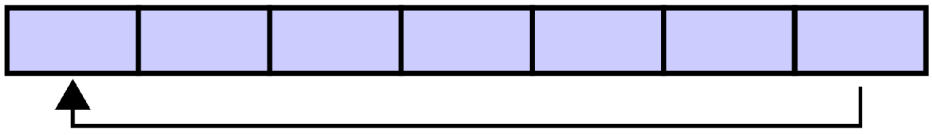


(b) Ukázka pětiokolí CA

Obrázek 2.3: Okolí buňky v jednorozměrném CA, buňka přecházející do nového stavu je zvýrazněna tmavším odstínem

Při přechodu do nového stavu je třeba znát buňky zprava i zleva. Problém nastává u krajních buněk, kde končí hranice pole. Budeme-li brát okolí o velikosti tři, tak poté konkrétně buňce na indexu nula a poslední buňce. V tomto případě jsou dva způsoby, jak tuto situaci řešit. Prvním je nastavení pevné hodnoty, kterou bude mít „virtuální“ buňka při počtech pokaždé. Většinou se v tomto případě používá hodnota nula.

Další možností je v podstatě spojení obou konců pole, čímž vznikne takzvané kruhové pole. Poté bude mít prvek na nultém indexu jako souseda vlevo prvek na posledním indexu a poslední prvek bude mít po své pravici buňku na indexu nula. Tento druh pole je na obrázku 2.4.



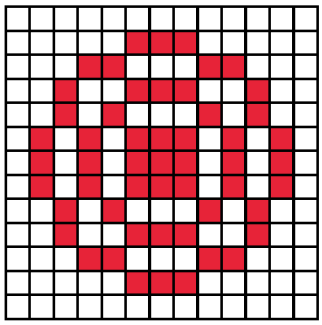
Obrázek 2.4: Ukázka kruhového pole[6], šipkou naznačena kruhová závislost

2.3 Dvourozměrné celulární automaty

Trochu složitější koncept představují dvourozměrné celulární automaty. Ty pracují s dvourozměrným polem buněk jako je na obrázku 2.5.

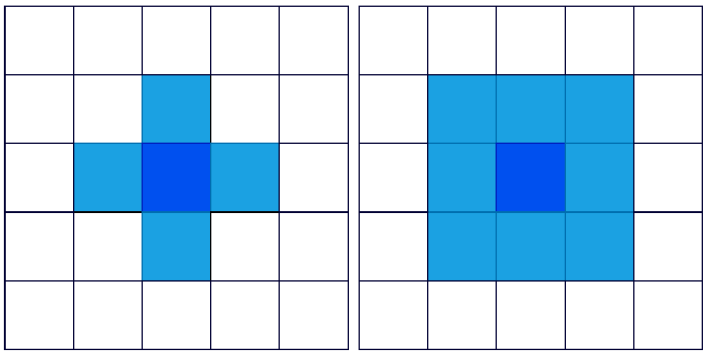
Podstatným rozdílem oproti jednorozměrným CA představuje nová dimenze. Díky té se v tomto případě použije jiný typ okolí, než tomu bylo u jednoho rozměru. Využívají se dva obecné typy okolí, konkrétně von Neumannovo a Moorovo. Von Neumannovo okolí,

[6]Převzato z: https://commons.wikimedia.org/wiki/File:Circular_buffer_-_empty.svg



Obrázek 2.5: Ukázka 2D CA[7]

které je na obrázku 2.6a, pracuje s pěti buňkami. V tomto případě bere centrální buňku a poté buňky, které s ní mají společné stěny. Naproti Moorovo okolí, které je na obrázku 2.6b, pracuje až s devíti buňkami. Doplňuje Neumannovo okolí i o buňky, které sousedí s centrální buňkou přes rohy.



(a) Von Neumannovo okolí (b) Moorovo okolí

Obrázek 2.6: Okolí buňky ve dvourozměrném CA

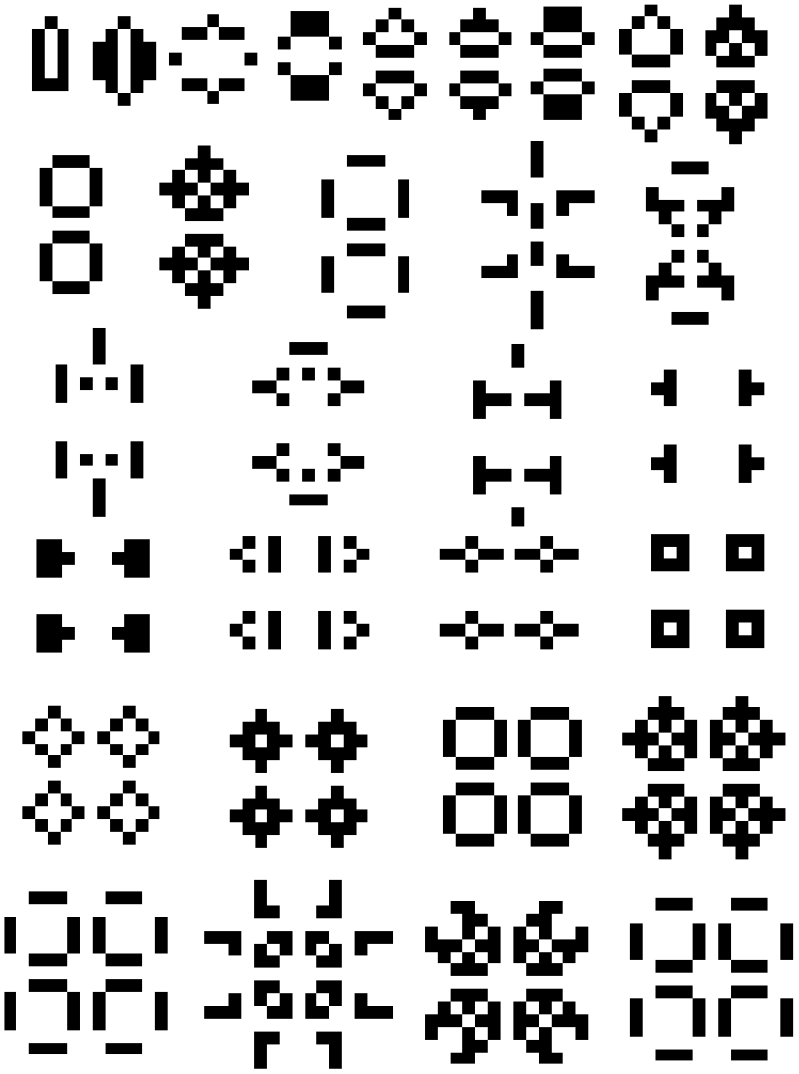
Stejně jako v případě jednorozměrných celulárních automatů nastává komplikace při hraničních buňkách. Tato situace se řeší stejně, buď je celý automat ohraničený hranicí buněk s hodnotou nula, nebo buňky na kraji navazují na druhý okraj.

Známým příkladem dvourozměrného celulárního automatu je Conwayova Hra života (v originále *Game of Life*). Jedná se o jednoduchý dvoustavový dvourozměrný celulární automat. Každá buňka může nabývat stavu nula, kde je považována za mrtvou, nebo jedna, kdy je naopak živá. Celý automat se řídí následujícími jednoduchými pravidly [12]:

- Pokud je buňka na živu, zůstane živá, pokud má dva nebo tři živé sousedy, jinak zemře
- Pokud je buňka mrtvá, ožije, pokud má právě tři živé sousedy, jinak zůstane mrtvá

I automat s těmito v podstatě primitivními pravidly dokáže generovat složité a „žijící“ obrazce. Některé z nic jsou na obrázku 2.7.

[7]Převzato z: https://tatasz.github.io/compound_ca/



Obrázek 2.7: Příklady obrazců ve Hře života[8]

[8] Převzato z: [1]

Kapitola 3

Generování pseudonáhodných čísel

Pro generování pseudonáhodných čísel se využívá několik různých přístupů. Využití celulárních automatů pro tento úkol je věnován zbytek této práce, v této kapitole budou zmíněny generátory pracující na jiném principu. Konkrétně se jedná o lineární kongruentní generátor v podkapitole 3.1, posuvný registr se zpětnou vazbou v podkapitole 3.2, generátor Blum Blum Shub v podkapitole 3.3 a jako poslední generátor ISAAC v podkapitole 3.4. Pro účely této práce bude zaveden pojem „konvenční generátory“. Tímto označením budou myšleny právě tyto generátory, které nejsou založeny na celulárních automatech.

3.1 Lineární kongruentní generátor

Lineární kongruentní generátor (anglicky *linear congruential generator*) je jedním z nejčastěji používaných generátorů. Pracuje na jednoduchém principu[9]:

$$x_{i+1} = (a \cdot x_i + b)\ mod\ m$$

kde:

- $\mathbf{x_{i+1}}$ je nové pseudonáhodné číslo
- $\mathbf{x_i}$ je poslední generované pseudonáhodné číslo
- **a**, **b** a **m** jsou vhodně zvolené konstanty
- **mod** je operace modulo, zbytek po celočíselném dělení.

Konstanty a, b a m poté určují kvalitu generátoru, mohou ovlivnit periodu, kdy se čísla začnou zase opakovat. Číslo x_0 je taky vybíráno uživatelem. Je to první číslo celé sekvence, od kterého se počítají další čísla. Obvykle bývá označováno jako „semínko“ (anglicky *seed*). Tuto první hodnotu je nutné určit pro jakýkoliv typ generátorů čísel, jedná se o inicializační nastavení generátoru, který na tuto hodnotu různými způsoby navazuje.

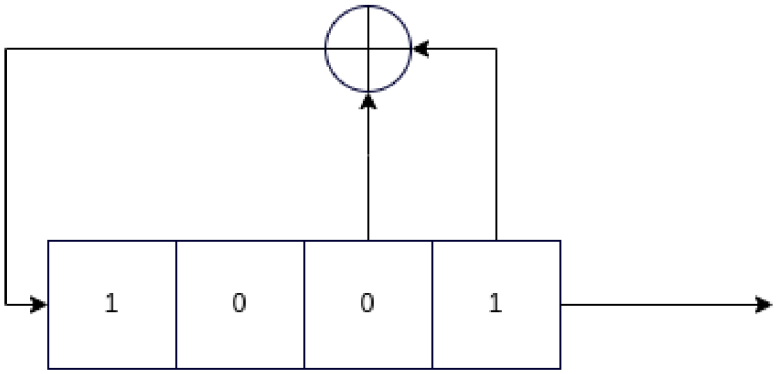
Takový generátor poté vytváří čísla v rozmezí: $0 \leq x < m$[15]. Nevýhodou tohoto generátoru je, že nové náhodné číslo je dost závislé na předchozím. Tato nevýhoda znemožňuje využít takový generátor v kryptografii nebo jiných službách, kde je třeba co největší náhodnost a nepředvídatelnost, protože při zjištění konstant je poté snadné zjistit další číslo, což v oblasti bezpečnosti není nejlepší.

Na princip lineárního kongruentního generátoru navazují i další. Často také bývá spojován s dalšími principy, například s posuvným registrem se zpětnou vazbou. Mezi takové

generátory patří: KISS (*Keep It Simple Stupid* - Zachovej to jednoduše hloupé)[17] nebo například Mersenne Twister.

3.2 Posuvný registr se zpětnou vazbou

Posuvný registr se zpětnou vazbou (anglicky *linear feedback shift register*, dále v textu pouze registr nebo LSFR) je další možností generování pseudonáhodných čísel. Lze ho implementovat jak softwarově, tak i hardwarově[8]. Používá dvě základní operace: bitový posun a operaci XOR. Základní struktura takového registru je na obrázku 3.1.



Obrázek 3.1: Čtyřbitový posuvný registr se zpětnou vazbou

Posuvný registr funguje následujícím způsobem. Nejdříve je proveden klasický bitový posun doprava. Bit, který „přeteče“ mimo registr se poté přidává do výstupního proudu bitů, představující náhodná čísla. Na uvolněné místo na pozici bitu s největším indexem přijde nový bit, který vznikne pomocí právě provedením operace XOR mezi příslušnými bity jak lze vidět na obrázku 3.1.

Při použití vícebitových registrů může být poté operace XOR několikrát zopakována, postupně s bity na různých pozicích.

Semínko tohoto generátoru představují hodnoty jednotlivých bitů v registru, například `1001` pro čtyřbitový registr. Pro každý registr, stejně jako jakýkoliv jiný generátor platí, že po určité době se začne sekvence opakovat, to je označováno jako perioda. Pro generátory pseudonáhodných čísel je žádané, aby tato perioda byla co největší možná.

Stejně jako lineární kongruentní generátor z předchozí podkapitoly je tento druh generátoru nepoužitelný pro oblast zabezpečení a kryptografii. Z několika vygenerovaných čísel lze totiž některými operacemi dostat počáteční konfiguraci a poté tedy moci i zjistit dopředu budoucí čísla reference internet, což není pro tento případ využití optimální.

3.3 Blum Blum Shub

Dalším představeným konvenčním generátorem je Blum Blum Shub (dále zkráceno pouze na BBS). Název vznikl spojením příjmení jeho autorů. Na rozdíl od předchozích dvou způsobů je tento využitelný v kryptografii. Jedná se tedy jednoduchý generátor pseudonáhodných čísel, patřící mezi kryptograficky bezpečné generátory (anglicky *cryptographically secury pseudorandom number generators*, zkracováno na CSPRNG).

Stejně jako lineární kongruentní generátor pracuje pomocí jednoduchého vzorce[7]:

$$x_{i+1} = x_i^2 \bmod M$$

kde:

- x_{i+1} je nově vygenerované číslo
- x_i je předchozí číslo
- *mod* je operace zbytek po celočíselném dělení
- M je konstanta

Pro konstantu M poté platí $M = p \cdot q$. V tomto případě představují p a q dvě dostatečně velká prvočísla, která splňují podmínku, aby byla takzvaná Blumova prvočísla (anglicky *Blum prime*). Pro Blumovo prvočíslo platí následující definice[10]:

Definice 1 *Prvočíslo* p *nazveme* Blumovo prvočíslo*, pokud:*

$$p \equiv 3 \bmod 4.$$

Tento generátor může být použit i při generování sekvence bitů. V tomto případě je ještě přidán druhý výpočetní krok využívající získané číslo:

$$b_i = x_i \bmod 2$$

V tomto případě vše zůstává, b_i potom označuje nový bit přidávaný do bitového proudu.

Velmi zajímavou vlastností tohoto generátoru je i možnost zjištění jakéhokoliv pseudonáhodného čísla sekvence pouze z iniciálních hodnot. K tomu lze využít *Eulerovu větu* (anglicky *Euler's theorem*) následovně:

$$x_i = x_0^{2^i \bmod \lambda(M)} \bmod M,$$

kde

- i je přirozené číslo označující pořadí pseudonáhodného čísla v sekvenci
- x_i je pseudonáhodné číslo, které chceme získat
- x_0 je počáteční hodnota - semínko generátoru
- M je stejná konstanta jako ve vzorci generátoru
- *mod* je operace zbytku po celočíselném dělení
- $\lambda(N)$ je takzvaná *Carmichaelova funkce.*

Konstanta M je použitá stejně jako v původním vzorci generátoru (viz výše). Ze zbytku vzorce stojí za zmínku ještě *Carmichaelova funkce*[1] je funkce, která využívá následujícího vztahu:

$$a^m \equiv 1 \bmod n$$

Vstupem Carmichaelovy funkce je přirozené číslo n a vrací poté m, takové, že platí výše uvedený vztah a zároveň je m co nejmenší možné. Dále ve vzorci a představuje všechna přirozená čísla menší než n a zároveň nesoudělná s n.

[1] Více vysvětleno v literatuře

3.4 ISAAC

Posledním generátorem představeným v této práci je ISAAC. Zkratka vznikla z anglických slov: Indirection, Shift, Accumulate, Add a Count (v překladu *Nepřímost*, *Posun*, *Seskupení*, *Sčítání* a *Počet*). Název tvoří operace, které algoritmus generátoru používá. [11]

Jedná se o generátor, který produkuje sekvence třiceti dvoubitových čísel. Perioda generátoru je garantována minimálně na 2^{40}, tedy pro představu 1 099 511 627 776 čísel, než se začne sekvence opakovat. Průměrně poté bývá perioda 2^{8295}[2] čísel. [11]

Stejně jako generátor Blum Blum Shub je i tento přístup využitelný v kryptografii. Výsledky generátoru jsou uniformně distribuované, nezkreslené a hlavně, což je pro generátory využitelné v kryptografii velmi důležité, nepředpověditelné, pokud neznáme počáteční hodnotu semínka.

Algoritmus 1 představuje algoritmus, který používá tento generátor. Mimo jiné využívá funkci `f(a, i)`, která je definována následovně:

$$f(a,i) = \begin{cases} a \ll 13 & \text{když } i \equiv 0 \bmod 4 \\ a \gg 6 & \text{když } i \equiv 1 \bmod 4 \\ a \ll 2 & \text{když } i \equiv 2 \bmod 4 \\ a \gg 16 & \text{když } i \equiv 3 \bmod 4 \end{cases}$$

Vstupem algoritmu jsou proměnné `a`, `b`, `c` a pole `s`. Hodnota `a` představuje třiceti dvoubitovou hodnotu použitou jako akumulátor entropie, proměnná `b` slouží pro uchování předchozího pseudonáhodného čísla a proměnná `c` je obyčejný osmibitový čítač, počítající počet průchodů algoritmem. Pole `s` představuje vnitřní stav, jedná se o pole 256 hodnot, každá o třiceti dvou bitech.

Jako výstupem je pole `r`, které obsahuje 256 třiceti dvoubitových čísel.

Samotný algoritmus pracuje se základními operacemi, jako je číselné sčítání (+), bitový posun ($\ll$ a $\gg$) a poté zbytek po celočíselném dělení (mod).

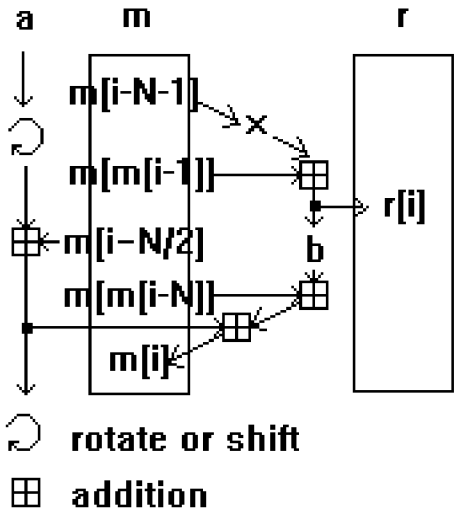
Vstup: a, b, c a pole 256 třiceti dvoubitových čísel s představující vnitřní stav
Výstup: pole r 256 třiceti dvoubitových čísel

1: c = c + 1
2: b = b + c
3: **for** *i = 0, ..., 255* **do**
4: x = s_i
5: a = f(a, i) + $s_{i+128 \bmod 256}$
6: s_i = a + b + $s_{x \gg 2 \bmod 256}$
7: r_i = x + $s_{s_i \gg 10 \bmod 256}$
8: b = r_i
9: **end for**
10: **return** r

Algoritmus 1: ISAAC

Fungování generátoru lze také zobrazit i diagramem na obrázku 3.2. Stejný diagram by poté bylo možné využít i pro generátor IBAA od stejného autora. ISAAC poté na tento generátor navazuje a zlepšuje jeho vlastnosti.

[2] Z důvodu velikosti výsledku nebude uvedeno přesné číslo, ale je jisté, že se v tomto případě jedná o opravdu velký výsledek



Obrázek 3.2: Diagram generátoru ISAAC[3]

Autor generátoru, Robert J. Jenkins, o něm prohlásil:

> „Neexistují žádné špatné počáteční stavy ani ty, kde jsou samé nuly.“

Článek [2] poté ukazuje, že to není úplně pravda a naopak představuje ty, které jsou zranitelné, a navrhuje i vylepšení původního algoritmu (v článku označeno jako ISAAC+) spočívající v přidání operace XOR do dvou kroků algoritmu. Toto vylepšení je ukázáno na algoritmu 2.

Vstup: a, b, c a pole 256 třiceti dvoubitových čísel s představující vnitřní stav
Výstup: pole r 256 třiceti dvoubitových čísel

1: $\mathrm{c} = \mathrm{c} + 1$
2: $\mathrm{b} = \mathrm{b} + \mathrm{c}$
3: **for** $i = 0, \ldots, 255$ **do**
4: $\quad \mathrm{x} = s_i$
5: $\quad \mathrm{a} = \mathrm{f}(\mathrm{a}, \mathrm{i}) + s_{i+128 \bmod 256}$
6: $\quad s_i = \mathrm{a} \oplus \mathrm{b} + s_{x \gg 2 \bmod 256}$
7: $\quad r_i = \mathrm{x} + \mathrm{a} \oplus s_{s_i \gg 10 \bmod 256}$
8: $\quad \mathrm{b} = r_i$
9: **end for**
10: **return** r

Algoritmus 2: ISAAC+ - vylepšení původního algoritmu

Co se bezpečnosti týče, nakonec není o tolik lepší než původní algoritmus ISAAC, jak je ukázáno v článku reference. I tak ale oba přístupy tvoří velmi dobré kryptograficky bezpečné generátory pseudonáhodných čísel.

[3] Diagram převzat z dokumentace [11]

Kapitola 4

Generování pseudonáhodných čísel celulárními automaty

V této kapitole budou představeny už zkoumané a otestované generátory pseudonáhodných čísel využívající celulární automaty. Konkrétně budou představeny automaty: jednorozměrný uniformní dvoustavový (podkapitola 4.1, jednorozměrný uniformní třístavový (podkapitola 4.2 a dvourozměrný neuniformní dvoustavový celulární automat (podkapitola 4.3. Tyto automaty byly převzaty a implementovány z článků, které se jimi více podrobně zabývaly[1].

4.1 Jednorozměrný dvoustavový automat

Prvním z implementovaných automatů je jednorozměrný dvoustavový[2] celulární automat. Jedná se automat fungující na jednoduchém principu, ale i tak se jedná o velmi kvalitní generátor.

Při implementaci byl použit kód z článku [19]. Pozměněno bylo pouze tisknutí čísel na výstup, jinak byl kód zachován.

Tento celulární automat pracuje s jednorozměrným polem buněk, kdy každá může nabývat jednoho ze dvou stavů odpovídajícím hodnotám bitů, tedy 0 nebo 1. Automat je uniformní, tedy pracuje pouze s jednou přechodovou funkcí.

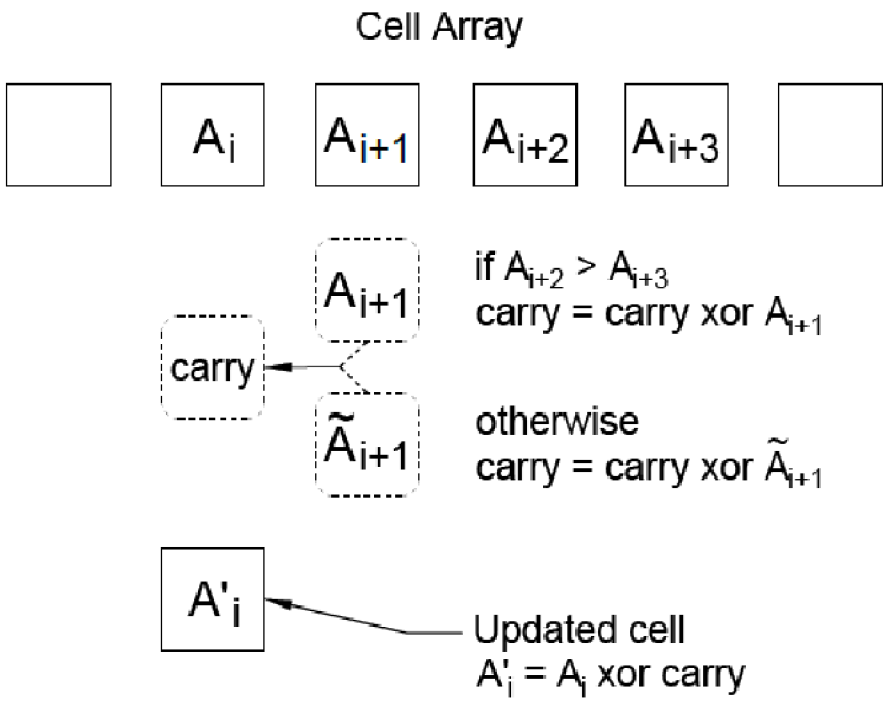
Pro fungování tohoto celulárního automatu je potřeba nastavit pět parametrů. Jsou to konkrétně tyto:

- a - velikost pole buněk, v implementaci konkrétně hodnota 128
- b - velikost buňky, v implementaci nastavena na třicet dva bitů
- c - hodnota carry (přenosu), v implementaci hodnota 987 654 321 v desítkové soustavě
- d - konstanta, v implementaci nastavena na hodnotu 010101... vyjádřenou binárně na třiceti dvou bitech
- e - hodnota kroků pro inicializační „zamíchání“ hodnot, než začne generátor produkovat opravdu pseudonáhodná čísla

Celá přechodová funkce je zobrazena na obrázku 4.1 níže.

[1] U každého celulárního automatu je poté příslušný článek zmíněn

[2] Dále v textu označován také jako binární



Obrázek 4.1: Přechodové pravidlo celulárního automatu[3]

Samotná přechodová funkce je rozdělena na tři následující kroky:

1. Aktualizace stavu hodnoty carry pro aktuální buňku
2. Aktualizace stavu buňky
3. Aktualizace stávající hodnoty carry pro další buňku

V prvním kroku automat aktualizuje hodnotu carry pro současnou buňku. K tomu se použije okolí aktuální buňky, které je v tomto případě definováno trochu jinak, než v obecném případu představeného v kapitole 2.2. V tomto případě okolí buňky představují tři buňky po prvé straně aktuální, jak je zobrazeno na obrázku 4.1. K aktualizace hodnoty carry je využita operace XOR a využívá následující funkci, kde c' je nová hodnota carry, c je současná hodnota carry a A_x představují buňky automatu na indexu x:

$$c' = \begin{cases} c \oplus A_{i+1} & \text{když } A_{i+2} > A_{i+3} \\ c \oplus \overline{A}_{i+1} & \text{jinak} \end{cases}$$

Dalším krokem je samotná aktualizace stavu aktuální buňky zase pomocí operace XOR pomocí vzorce:

$$A'_i = A_i \oplus c'$$

Nakonec posledním krokem je aktualizace hodnoty carry pro další buňku přičtením konstanty d:

$$c' = c' + d$$

[3]Obrázek převzat z původního článku [19]

Funkce generátoru je poté následující. Generátor po spuštění provede e počet iterací pro dostatečné „zamíchání“ hodnot ze semínka pro generování pseudonáhodných čísel. Poté už začne generovat náhodná čísla na výstup, jejich počet je závislý na zadaném vstupním parametru.

Implementovaný generátor potřebuje ke spuštění dva parametry a spouští se:

`./gen COUNT SEED`

Parametr `COUNT` představuje požadovaný počet vygenerovaných pseudonáhodných čísel a parametr `SEED` představuje počáteční semínko generátoru. U tohoto generátoru může být semínkem řetězec i číslo, v obou případech je použita binární hodnota z paměti, ne doslovná hodnota semínka.

4.2 Jednorozměrný třístavový automat

Druhým implementovaným celulárním automatem je třístavový uniformní jednorozměrný automat. Implementace byla převzata z článku [6]. Uvedená implementace v jazyce Java byla poté převedena na odpovídající implementaci v jazyce C.

Oproti předchozímu automatu byl přidán jeden stav. Množina stavů pro tento automat je tedy: $\{0, 1, 2\}$.

K vyjádření přechodové funkce je použita tabulka pravidel z obrázku 4.2

P.S.	**222**	**221**	**220**	**212**	**211**	**210**	**202**	**201**	**200**	**122**	**121**	**120**	**112**	**111**	**110**	**102**	**101**	**100**	**022**	**021**	**020**	**012**	**011**	**010**	**002**	**001**	**000**
RMT	**(26)**	**(25)**	**(24)**	**(23)**	**(22)**	**(21)**	**(20)**	**(19)**	**(18)**	**(17)**	**(16)**	**(15)**	**(14)**	**(13)**	**(12)**	**(11)**	**(10)**	**(9)**	**(8)**	**(7)**	**(6)**	**(5)**	**(4)**	**(3)**	**(2)**	**(1)**	**(0)**
N.S.	1	2	0	0	2	1	1	2	0	0	2	1	0	2	1	1	2	0	0	2	1	0	2	1	2	1	0

Obrázek 4.2: Tabulka přechodových pravidel pro třístavový celulární automat[4]

Automat využívá okolí o velikosti tří buněk, proto je v tabulce přechodových pravidel nutné mít dvacet sedm možností. Při přechodu do nového stavu se převede číslo, které vznikne z hodnot stavů buněk za sebou, a převede se do desítkové soustavy. Toto číslo poté představuje index do tabulky pravidel.

Při spuštění generátor provede inicializaci současné konfigurace stavů podle zadané semínka. Po převedení do trojkové soustavy si z něj vezme několik prvních hodnot, které nastaví do pole aktuálního stavu. Počet hodnot, které bude automat používat závisí na velikosti čísel, které má generovat, v bitech. Tato velikost je označena jako „okno“. Experimenty v původním článku bylo stanoveno, že je potřeba: $10 \cdot \text{pocet\_bitu}/16$ buněk, kde `pocet_bitu` vyjadřuje velikost generovaných čísel v bitech.

Než začne generátor produkovat pseudonáhodná čísla, musí projít několika iteracemi „na prázdno“, tedy bez vygenerování čísel. Konkrétně je třeba projít počtem cyklů, který odpovídá velikosti automatu n. Podle článku vychází velikost automatu: $n = \text{window} \cdot 2,5$. Po projdutí n iterací začne automat generovat validní výsledky. Číslo je poté z automatu získáváno zase pomocí okna. To určuje počet buněk, které se vezmou ze začátku automatu a převedou se na dekadickou hodnotu.

V rámci práce bylo implementována ještě jedna forma tohoto typu automatu. Původní okolí o třech buňkách bylo rozšířeno na buněk pět jako součást experimentu, jestli větší okolí nějak znatelně pozmění výsledky generátoru ve statistických testech.

[4] Převzato z původního dokumentu: [6]

Při okolí o velikosti tři buňky byla třeba tabulka pravidel o velikosti 3^3. Při rozšíření okolí o dvě buňky se toto číslo zvětšilo na 3^5, tedy 243 možností.

Pro implementaci bylo třeba nalézt odpovídající nejlepší možnou sadu pravidel, aby měl výsledný generátor co nejlepší výsledky. Stejný problém je řešen i v kapitole 5.2, kde je využit genetický algoritmus. V tomto případě bylo využito jiného přístupu a to „hrubou silou".

Postup při hledání nejlepší možné tabulky pravidel pro pětiokolí byl následující:

- Pomocí skriptu je vygenerována náhodná sekvence 243, která obsahuje pouze znaky z množiny stavů. Tato sekvence je použitá jako tabulka pravidel pro celulární automat.
- Pomocí této tabulky je vygenerováno 1 024 náhodných čísel.
- Na vygenerovaných číslech jsou spuštěny statistické testy.

Tento postup byl několikrát zopakován a nakonec z výsledků byla vybrána tabulka pravidel, který vygenerovala čísla procházející co nejvíce testy s co nejlepšími výsledky. Samotné testování je popsáno v kapitole 6. Pro otestování bylo vybráno 1 024 čísel, protože se jedná už o poměrně dost čísel a zároveň je to sympatická mocnina dvou.

Z důvodu velikosti výsledné tabulky (243 znaků) není v textu uvedena, lze ji najít ve zdrojovém kódu `three.c` v globální proměnné `rule5`.

Generátor je poté implementován v rámci jednoho programu. Lze ho spustit následovně:
`./three SIZE SEED COUNT RADIUS`

Program potřebuje následující parametry:

- `SIZE` - velikost generovaných čísel v bitech
- `SEED` - použité semínko, v tomto případě zadáváno jako číslo
- `COUNT` - počet generovaných čísel
- `RADIUS` - použité okolí, možno zadat pouze hodnotu 3 nebo 5

4.3 Dvourozměrný binární neuniformní automat

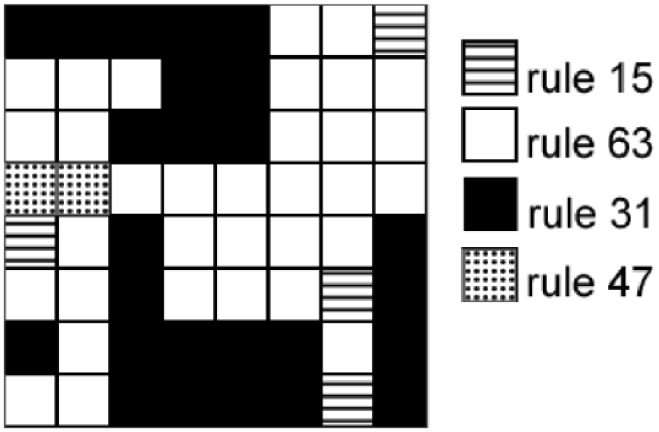
Posledním automatem implementovaným dle článku je dvourozměrný binární neuniformní celulární automat. Myšlenka automatu včetně přechodových pravidel byla převzata z článku [16]. Od předchozích dvou automatů se liší druhou dimenzí a ještě neuniformitou.

Autoři článku použili k nalezení optimální sady pravidel pro celulární automat velikosti 8x8 buněk genetický algoritmus. Výsledek pro tento automat je na obrázku 4.3. Postup hledání pravidel nebyl v rámci této práce zopakován, pouze byl převzat výsledek.

Implementovaný automat využívá přechodovou funkci:

$$s_{i,j}(t+1) = X \oplus (C \cdot s_{i,j}(t)) \oplus (N \cdot s_{i-1,j}(t)) \oplus (W \cdot s_{i,j-1}(t)) \oplus (S \cdot s_{i+1,j}(t)) \oplus (E \cdot s_{i,j+1}(t))$$

Každá buňka má poté přiřazeno pravidlo, které určuje hodnoty konstant X, C, N, W, S, E. Přiřazení pravidel je v tabulce pravidel na obrázku 4.3. Konstanta X vyjadřuje, zda je pravidlo lineární ($X = 0$) nebo ne ($X = 1$). Konstanty C (*center*, střed), N (*north*, sever), W (*west*, západ), S (*south*, jih) a E (*east*, východ) poté vyjadřují, zda se příslušná buňka bude používat. Při hodnotě konstanty 1 je buňka zahrnuta do výpočtu, v opačném případě



Obrázek 4.3: Tabulka pravidel pro 2D CA[5]

nikoliv. Nový stav buňky na řádku i a sloupci j je poté označen: $s_{i,j}(t+1)$ a současné stavy poté $s(t)$ s příslušným indexem podle polohy buňky.

Jak lze vidět na obrázku 4.3, automat používá čtyři různá pravidla. Jsou to:

- Pravidlo 15 – $s_{i,j} = s_{i-1,j}(t) \oplus s_{i,j-1}(t) \oplus s_{i+1,j}(t) \oplus s_{i,j+1}(t)$
- Pravidlo 31 – $s_{i,j} = s_{i-1,j}(t) \oplus s_{i,j-1}(t) \oplus s_{i+1,j}(t) \oplus s_{i,j+1}(t) \oplus s_{i,j}(t)$
- Pravidlo 47 – $s_{i,j} = 1 \oplus s_{i-1,j}(t) \oplus s_{i,j-1}(t) \oplus s_{i+1,j}(t) \oplus s_{i,j+1}(t)$
- Pravidlo 63 – $s_{i,j} = 1 \oplus s_{i-1,j}(t) \oplus s_{i,j-1}(t) \oplus s_{i+1,j}(t) \oplus s_{i,j+1}(t) \oplus s_{i,j}(t)$

Po provedení experimentů bylo zjištěno, že inicializační „zamíchání“ hodnot automatu v tomto případě neznamená pro výsledky nějaké vylepšení, proto nebylo využito. Pro vygenerování sady N-bitových čísel musí generátor projít N kroků. Při každém kroku vygeneruje jeden bit výsledné sady čísel. Pro osmibitová čísla je tedy potřeba projít osmi kroky. Výsledkem je poté šedesát čtyři čísel.

V rámci experimentů bylo zjišťováno, jak moc se budou lišit výsledky při použití jiného počtu pravidel. Byly vybrány tyto automaty:

- Uniformní dvourozměrný automat
- Dvourozměrný neuniformní automat se dvěma různými pravidly
- Dvourozměrný neuniformní automat se třemi různými pravidly

K výběru, která pravidla použít byla využita podobně jako v kapitole 4.2 takzvaná hrubá síla. Na výběr byla již výše čtyři zmiňovaná pravidla, v rámci pokusů se pouze hledala nejlepší možná kombinace.

Postup byl následující:

- Pomocí pravidel bylo vygenerováno 1 024 čísel
- Otestování výstupu pomocí statistický testů

[5]Obrázek převzat z původního článku [16]

Z výsledků statistický testů byla poté vybrána pravidla, která produkovala nejlepší výsledky. Výsledky jsou následující. Pro uniformní dvoustavový automat bylo vybráno pravidlo 63. Pro neuniformní se dvěma pravidly pro první čtyři řádky automatu pravidlo 15 a pro zbylé čtyři pravidlo 63. A neuniformní automat se třemi pravidly má pro první třetinu řádků pravidlo 15, pro druhou pravidlo 31 a pro poslední pravidlo 63.

Generátor je poté spouštěn následovně:

```
./two_dim COUNT  SEED MODE
```

Parametry generátoru jsou následující:

- `COUNT` - počet iteracích, které automat provede, vygeneruje poté osmkrát tolik čísel
- `SEED` - je semínko generátoru zadáváno jako binární sekvence 64 čísel
- `MODE` - určuje, který z automatu se spustí, možnosti jsou: 1, 2, 3 a 4 a určují počet využitých pravidel

Kapitola 5

Nový postup generování pseudonáhodných čísel celulárními automaty

Tato kapitola se věnuje hlavnímu cíli celé této práce. Tím je pokus o vytvoření čtyřstavového celulárního automatu, který by měl lepší výsledky, než doposud prozkoumané metody. Pro hledání nejlepších možných pravidel je použit genetický algoritmus, kterému se věnuje podkapitola 5.1. Samotný celulární automat je poté popisován v podkapitole 5.2.

5.1 Použití genetického algoritmu pro návrh pravidel celulárního automatu

K nalezení přechodové funkce celulárních automatů se využívá mnoho metod. Jednou z nich je genetický algoritmus, který byl využit i v této práci. Jedná se o jednu z technik inspirovaných reálnou přírodou spolu s například optimalizací mravenčí kolonií nebo optimalizací hejnem částic, kterými se tato práce nezabývá.

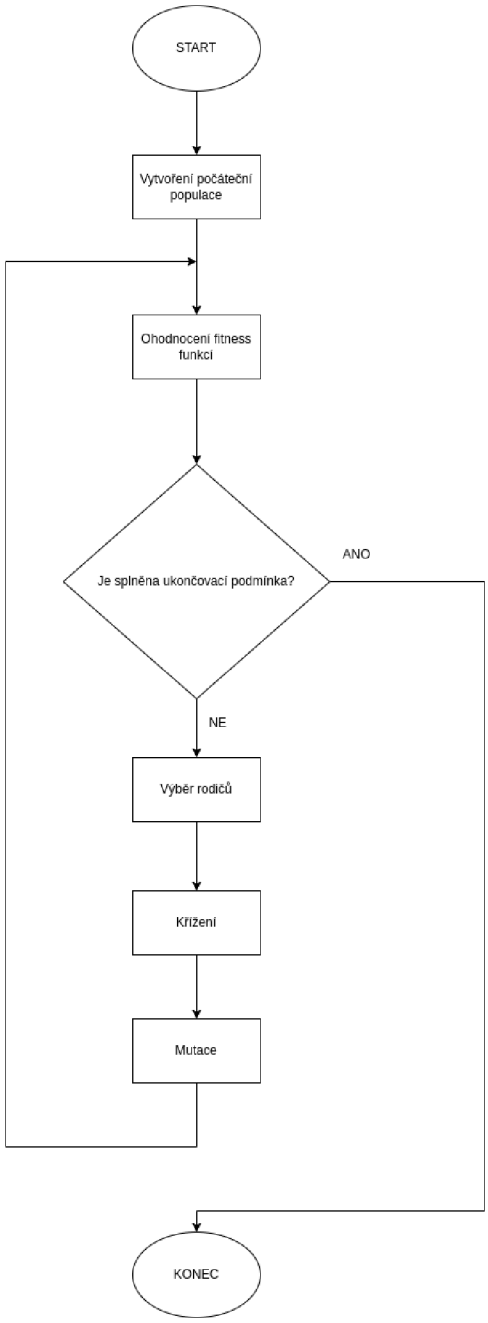
Genetický algoritmus je metoda inspirovaná Darwinovou teorií o přírodní evoluci [13]. Z populace jedinců jsou vybíráni ti nejsilnější, kteří mají největší šanci přežít. Jedinec představuje v algoritmu možné hledané řešení. Nejsilnější jedinec poté představuje nejoptimálnější řešení problému.

Genetický algoritmus pracuje s populací, kterou představuje pole jedinců. Každý jedinec představuje nějaké řešení, které může a nemusí být hledaným řešením. Při hledání je každý z jedinců ohodnocen fitness funkcí, který určuje, jak moc dobré řešení představuje. Fitness funkce je volena pro každý problém individuálně, nemá pevně daný tvar.

Genetický algoritmus se skládá z těchto kroků:

1. Výpočet fitness funkce každého jedince
2. Vybrání jedinců ke křížení
3. Křížení
4. Mutace

Genetický algoritmus je ukázán na diagramu na obrázku 5.1.



Obrázek 5.1: Diagram genetického algoritmu

Na počátku algoritmu je vygenerována počáteční populace jedinců. V tomto konkrétním případě je jedinec reprezentován řetězcem představujícím výsledný stav při dané kombinaci stavů okolí. Každý z jedinců je následně ohodnocen fitness funkcí. V dalším kroku jsou vybráni jedinci ke křížení pro vznik další generace. To lze provést více způsoby. Jedním z nich je výběr náhodných dvou jedinců a jejich následné zkřížení. Další z možností je využití „ranků“. Tento způsob byl využit i v rámci implementace.

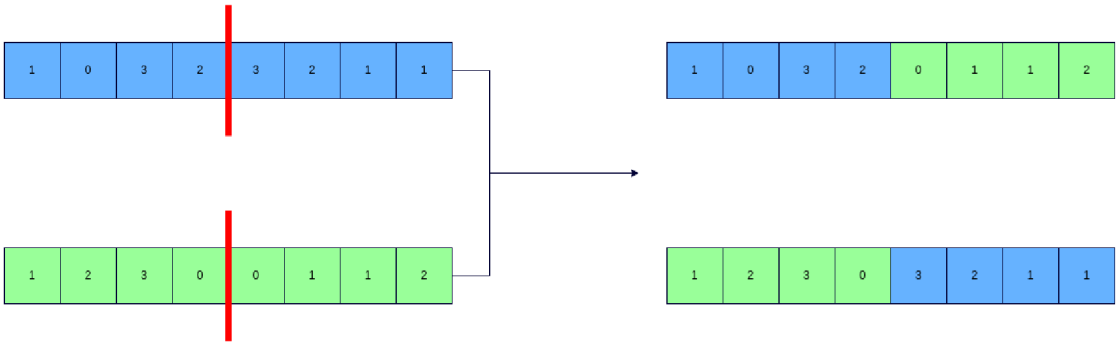
Při využití ranků se při výběru bere v potah ohodnocení fitness funkcí. Jedinci se seřadí podle této hodnoty a každému je přeřazen krom fitness funkce i rank, který vyjadřuje

relativní kvalitu vůči ostatním jedinců populace. V tomto případě je rank přiřazován podle počtu jedinců, kteří mají horší hodnotu fitness než konkrétní jedinec. Dále je každému přiřazena pravděpodobnost výběru daného jedince. Ta je vypočítána následovně:

$$P_{sel}(k) = \frac{1-s}{\mu} + \frac{2 \cdot k \cdot s}{\mu \cdot (\mu - 1)}$$

Kde k je rank jedince, s je uživatelem zvolený parametr - konkrétně byla použita hodnota 0,7 náhodně vybrána bez nějakého většího smyslu - a μ je velikost populace - konkrétně sto.

Po výběru jedinců dochází ke křížení. Křížení je ilustrováno na obrázku 5.2. Obrázek je přizpůsoben řešenému způsobu v práci, pouze chrozomy byly pro účel obrázky zmenšeny na osm znaků. Při křížení je vybráno náhodně jedno místo a nové vznikají spojením částí před tímto místem z prvního rodiče a částí po tomto místo druhého rodiče. Místo křížení může být i pevně dáno, jak tomu bylo v této práci, konkrétně na polovinu řetězce.



Obrázek 5.2: Křížení chromozomů - Vlevo rodiče, vpravo potomci

Z jednoho křížení vzniknou dva nové chromozomy do další generace. Při křížení může dojít k mutaci. Pravděpodobnost mutace byla nastavena na 0,01. Mutace je poté provedena na náhodném znaku řetězce zvýšením hodnoty o jedna.

Po vytvoření celé nové generace algoritmus znovu ohodnocuje jedince hodnotou fitness funkce a znovu tvoří novou generaci. Postup je opakován, dokud není nalezeno ideální řešení nebo je „násilně“ ukončen po uběhnutí předem daného počtu generací, v tomto případě 1 024.

V rámci experimentů s generátorem byla plánována implementace dvou čtyřstavových automatů. Stejně jako v případu třístavového tedy jeden s okolím o velikosti tři a druhý s okolím o velikosti pět buněk. V prvním případě případě je tedy jedinec vyjádřen jako řetězec 4^3, tedy 64 znaků. V druhém případě je hledaný výsledek vyjádřený řetězcem o 4^5 (1 024) znaků.

Fitness funkce pro hledání nejlepší tabulky přechodových pravidel byla převzata z článku [16]. Nejdříve bylo vygenerováno 1 024 čísel pomocí pravidla jedince. Ze sekvence byla poté vypočítána míra entropie, která poté sloužila pro ohodnocení jedince. Pro míru entropie byl využit vzorec:

$$E_h = -\sum_{j=1}^{k^h} p_{h_j} \log_2 p_{h_j}$$

Ve vzorci představuje k počet všech možných hodnot, tedy v tomto případě počet všech možných stavů a h potom délku řetězce představující číslo. Pravděpodobnost výskytu čísla ve vygenerované sekvenci je poté označena p_{h_j}.

Výsledné automaty jsou prozatím navrženy pro osmibitová čísla. Pro osmibitové číslo jsou ve čtyřkové soustavě potřeba čtyři znaky. Množina stavů obsahuje čtyři prvky, jako k je tedy dosazeno číslo čtyři a za h také číslo čtyři. V případě vícebitových čísel by bylo pro vyjádření třeba více buněk.

Optimální řešení má poté co největší míru entropie. Nejlepší řešení je potom takové, kdy mají všechna čísla, která mohou být vygenerována stejnou pravděpodobnost výskytu a to rovnu: $1/k^h$. Míra entropie poté vychází při tomto ideálním stavu rovna pro čtyřstavový automat: $E_h = 2 \cdot h$. Lze říci, že tato hodnota se bude vždy rovnat počtu bitů, které jsou použity pro vyjádření zkoumaných čísel. V tomto konkrétním případě byla hledaná optimální míra entropie rovna hodnotě osm. Pokud by nějaký jedinec dosáhl této hodnoty, bylo by nalezeno optimální řešení.

Velikost jedné populace byla stanovena na 100 jedinců a běh algoritmu byl omezen na 1 024 generací. V rámci implementace byl algoritmus trochu modifikován. Při běhu algoritmu je zapamatován nejlepší jedinec, který je výstupem programu.

5.2 Čtyřstavový jednorozměrný celulární automat

Hlavní přínosem této práce je jednorozměrný čtyřstavový uniformní celulární automat. Při návrhu a implementaci bylo vycházeno z již nastudovaných předchozích článků. Hlavní inspirací byl třístavový automat z článků [6] a [5], který také využíval více stavů.

Jednou z nejdůležitější částí návrhu bylo navržení vhodných přechodových pravidel, pro nejlepší možné výsledky. Jednou z možností je použití metody „hrubou silou“, jak je představeno v kapitolách 4.2 a 4.3, ovšem toto řešení se už v předchozích případech neprojevilo jako ideální.

K návrhu přechodových pravidel pro tento čtyřstavový automat byl využit genetický algoritmus, které je vysvětlen v kapitole 5.1. Tento způsob byl převzat z článku o dvourozměrném celulárním automatu [16].

Byly navrženy a implementovány dvě možnosti automatu, stejně jako v případě třístavového v kapitole 4.2. První používá okolí o velikosti tři buňky a druhá větší okolí o velikosti pět buněk. Výsledné tabulky přechodových pravidel budou obsahovat tedy 64 a 1 024 položek. Druhá možnost slouží k experimentu, zda rozšířené okolí ovlivní nějak víc kvalitu generovaných pseudonáhodných čísel.

Tento způsob však nezaručí, že generovaná čísla budou procházet i statistickými testy, proto vždy byla ještě vygenerovaná tabulka pravidel otestována pomocí sady testů. Ta je popsána v kapitole 6.1. V případě, že by výsledky nebyly optimální, neprocházely by tedy statistické testy, bylo by třeba postup opakovat a vygenerovat novou tabulku pravidel.

Po několika pokusech byla vybrána následující tabulka přechodových pravidel[1] jako nejlépe vyhovující:

`3221012133302310322032320203213021331220030130011312320102033031`

Při přechodu do nového stavu se poté vezme okolí buňky, kdy v tomto případě vznikne číslo o třech číslicích ve čtyřkové soustavě, a to se převede to desítkové soustavy. Výsledné číslo v desítkové soustavě poté udává index do výše uvedené tabulky pravidel.

[1] Tabulka je vyjádřená jako řetězec z důvodu délky, kterou zabírá. Stavy jsou za sebou seřazeny jako prvky pole v jazyce C

Genetický algoritmus byl použit i při hledání pravidel pro automat pracující s pětiokolím. I zde bylo ale třeba přetestovat nalezený řetězec a případně postup opakovat. Výsledná tabulka pravidel v tomto případě obsahuje 1 024 možností, což je pro zobrazení v dokumentu příliš, proto zde není uvedena. Lze ji najít poté ve zdrojovém kódu souboru `four.c` jako globální proměnnou `rules5`.

Při přechodu do nového stavu nevznikne číslo o třech číslice, ale pěti. I to je převedeno do desítkové soustavy a představuje index do nalezené tabulky přechodových pravidel.

Při spuštění generátor se nejdříve inicializuje automat dle zadaného semínka, které v tomto případě musí být ve tvaru čísla ve čtyřkové soustavě. Zbývající buňky automatu jsou poté inicializovány na hodnotu nula, stejně jako tomu je v případě třístavového automatu.

Byly provedeny experimenty, kdy se automat spustil a před generováním bylo provedeno „zamíchání“ stavů jako u předchozích jednorozměrných automatů. V tomto případě nebyl zaznamenán velký rozdíl ve výsledcích, proto nakonec nebylo využito. Po inicializaci počátečních hodnot buněk začne automat okamžitě generovat pseudonáhodný čísla.

Získávání čísla z automatu bylo v podstatě převzato od třístavového automatu. Tento automat má napevno nastaveno generování osmibitových hodnot, vícebitové hodnoty nebyly prozatím implementovány. Osmibitovou hodnotu lze vyjádřit na čtyřech buňkách čtyřstavového automatu. Při výpisu čísla se vezmou první čtyři buňky a ty se převedou do desítkové soustavy.

Výhodou čtyř stavů je vztah mezi číslem dva a čtyři. Číslo čtyři je druhou mocninou čísla dvě, proto pro získání N-bitového čísla není třeba složitě přepočítávat počet nutných buněk, jako je to třeba u třístavového celulárního automatu, stačí požadovaný počet bitů pouze vydělit dvěma.

Generátor je potřeba spouštěn následovně:

```
./four RADIUS COUNT SEED
```

Parametry jsou následující:

- `RADIUS` - rozměr okolí, může být hodnota 3 nebo 5
- `COUNT` - počet chtěných pseudonáhodných čísel
- `SEED` - semínko generátoru vyjádřené jako číslo ve čtyřkové soustavě

Kapitola 6

Experimentální výsledky

V poslední kapitole budou porovnány jednotlivé generátory pseudonáhodných čísel využívající celulární automaty, které byly v textu představeny. V podkapitole 6.1 je představena použitá testovací sada a způsob provedení experimentů pro porovnání. V podkapitole 6.2 jsou otestovány a porovnány generátory využívající jednorozměrné celulární automaty. Generátory s dvourozměrnými celulárními jsou porovnány v podkapitole 6.3. Následně jsou vybrány dle výsledků čtyři nejlepší generátory z obou typů a porovnány v podkapitole 6.4.

6.1 Testovací sada

Pro testování generátorů pseudonáhodných čísel je možné využít několik různých způsobů testování. V jazyce Python například lze využít jeden z balíčků, jmenovitě `randtest`, který obsahuje funkci pro otestování sekvence čísel.

Jistějším způsobem je využití statistických testů, které bývají seskupeny do testovacích sad. Jednou z nejčastěji používaných testovacích sad je poté sada Diehard nebo testovací sada institutu NIST, která se zaměřuje i na kvalitu pro kryptografické účely. Kromě jazyka Python jsou testy implementované i v jazyce C jako knihovna `TestU01`.

V rámci této práce byla vybrána právě testovací sada NIST [3]. Nebyla využita celá, ale pouze sedm testů z ní. Konkrétně se jedná o tyto testy:

1. Frekvenční test v rámci bloku (anglicky *Frequency Test within a Block*)
2. Test kumulativních součtů (anglicky *Cumulative Sums Test*)
3. Test diskrétní Fourierovy transformace (anglicky *Discrete Fourier Transform (Spectral) Test*)
4. Frekvenční (Monobit) test (anglicky *Frequency (Monobit) Test*)
5. Test na nejdelší běh jedniček v bloku (anglicky *Test for the Longest Run of Ones in a Block*)
6. Test stupně binární matice (anglicky *Binary Matrix Rank Test*)
7. Sériový test (anglicky *Serial Test*)

Všechny statistické testy pracují na podobném principu. Na začátku je vyslovena nulová hypotéza. V případě testování pseudonáhodných čísel je nulová hypotéza, že se jedná o sekvenci náhodných čísel. Každý z testů poté počítá nějakým způsobem takzvanou P-hodnotu

(*P-value*). V rámci těchto testů P-hodnota vyjadřuje pravděpodobnost, že dokonalý generátor náhodných čísel vygeneruje méně náhodnou sekvenci čísel, než je právě ta testovaná. Kdyby vyšla P-hodnota rovna jedné, byla by brána tato sekvence čísel za dokonale náhodnou. V opačném případě, kdy vyjde P-hodnota rovna nule, je sekvence brána za kompletně nenáhodnou.

K vyhodnocení testů je třeba ještě určit hladinu významnosti α. Pro přijmutí nulové hypotézy je poté třeba porovnat P-hodnotu s touto hodnotou α. V případě, že P-hodnota $\geq \alpha$, nulová hypotéza je přijata, sekvence je tedy náhodnou. V opačném případě je sekvence vyhodnocena jako nenáhodná. Pro testy je nastavena na hodnotu 0,01, což je jedna z nejčastějších možností.

Jednotlivé testy a způsoby počítání P-hodnoty jsou popsány v dokumentaci [3]. Tématem práce není testování generátorů pseudonáhodných čísel, nebudou tedy detailně popsány.

Frekvenční test v rámci bloku

Tento test je v podstatě rozšířením frekvenčního testu. Zkoumá poměr jedničkových a nulových bitů ne v rámci celé sekvence, ale pouze v blocích o velikosti M. Při náhodné sekvenci by byl počet jedničkových bitů, stejně tak nulových, roven polovině velikosti bloku M pro každý ze všech bloků. Pokud se M rovná 1, tak potom je test roven frekvenčnímu (monobit) testu.

Test kumulativních součtů

Při počítání v tomto testu jsou nuly nahrazeny hodnotou -1. Test poté zkoumá odchylku od nuly pro náhodné podsekvence definovanou kumulativním součtem[1]. Pro náhodnou sekvenci by odchylka od nuly měla být blízko nule. Test probíhá ve dvou částech: od začátku a od konce.

Test diskrétní Fourierovy transformace

V tomto testu je využita diskrétní Fourierova transformace (DFT) používaná při práci se signály pro získávání spektra. Toho využívá i tento test. Zkoumá výšky vrcholů v provedené DFT na zkoumané sekvenci. Účelem je hledání pravidelnosti v sekvenci.

Frevenční (Monobit) test

V původní sadě testů NIST je uveden tento test jako první, v této práci je v seznamu testů až čtvrtý, ale na výsledcích to nic nemění. Podle dokumentace na projdutí tohoto testu závisí i projdutí všech ostatních. Hlavním cílem tohoto testu je zkoumat poměr jedniček a nul v bitové sekvenci. Pravá náhodná sekvence by měla počet nulových a jedničkových bitů přibližně stejně.

Test na nejdelší běh jedniček v bloku

Tento test pracuje s bloky o velikosti M bitů. V každém bloku zkoumá nejdelší sekvenci jedniček jdoucích bezprostředně za sebou. Sekvence stejných bitů za sebou se nazývá běh (anglicky *run*). Velikost takového nejdelšího běhu by pro každý blok měla být stejná při sekvenci náhodných čísel.

[1] Kumulativní součet je součet všech předchozích hodnot

Test stupně binární matice

V tomto testu se pracuje s disjunktními maticemi vzniklé ze původní zkoumané sekvence rozdělené na menší. Účelem tohoto testu je hledat lineární závislost mezi podsekvencemi. Mimo sady NIST se tento test také vyskytuje v Diehard testovací sadě.

Sériový test

Tento test se zaměřuje na frekvenci všech možných překrývajících se podsekvencí původní sekvence bitů. Každá podsekvence má velikost právě M bitů, což je jeden z parametrů funkce vyhodnocující tento test. V případě, že hodnota M je rovna jedné, tak je sériový test ekvivalentní frekvenčnímu testu výše. Čísla v náhodné sekvenci by měla být rozdělena rovnoměrně. Proto by měla mít každá sekvence o M bitech stejnou pravděpodobnost, že se objeví v původní sekvenci. Tento test počítá dvě P-hodnoty.

Testy nejsou součástí implementace, byla použita oficiální implementace z oficiální stránky institutu[2].

6.2 Testování jednorozměrných celulárních automatů

V rámci testování jednorozměrných celulárních automatů byly otestovány po implementaci konkrétně tyto celulární automaty jako generátory pseudonáhodných čísel:

- Dvoustavový automat
- Třístavový automat s okolím o velikosti tři buňky
- Třístavový automat s okolím o velikosti pět buněk
- Čtyřstavový automat s okolím o velikosti tři buňky
- Čtyřstavový automat s okolím o velikosti pět buněk

Na každém z těchto automatů byly provedeny již zmíněně testy ze sekce 6. V první části bylo vygenerováno 1 024 osmibitových čísel a tato sekvence byla následně otestována.

Výsledky testů jsou v tabulkách 6.1 a 6.2 a grafu 6.1. Tabulka byla pro přehlednost rozdělena do dvou tabulek – zvlášť automaty přebrané z článků a zvlášť nový čtyřstavový. Tabulky poté zobrazují výslednou P-hodnotou zaokrouhlenou na tři desetinná místa pro přehlednost.

Graf má na svislé ose vyznačeny získané P-hodnoty, na vodorovné názvy testů. Čárkovaně je poté vyznačená hladina významnosti 0,01 pro lepší orientaci.

V tabulkách ani grafu nejsou pro přehlednost vypsány celá jména testů, pouze zkrácené verze podle názvů složek, kam poté program poté ukládá výsledky. Názvy jsou nahrazeny následovně:

- Frekvenční test v rámci bloku – BlockFrequency
- Test kumulativních součtů od začátku – CumulativeSumsF

[2]Odkaz ke stažení testovací sady: https://csrc.nist.gov/Projects/Random-Bit-Generation/Documentation-and-Software

- Test kumulativních součtů od konce – CumulativeSumsB
- Test diskrétní Fourierovy transformace – DFT
- Frekvenční (Monobit) test – Frequency
- Test na nejdelší běh jedniček v bloku – LongestRun
- Test stupně binární matice – Rank
- Sériový test – Serial1 a Serial2

Test	Dvoustavový	Třístavový, r = 3	Třístavový, r = 5
BlockFrequency	0,853	0,132	0,84
CumulativeSumsF	0,829	0,109	0,604
CumulativeSumsB	0,866	0,104	0,565
DFT	0,301	0,089	0,011
Frequency	0,965	0,057	0,426
LongestRun	0,115	0,028	0,024
Rank	0,626	0,017	0,962
Serial1	0,986	0	0
Serial2	0,978	0	0

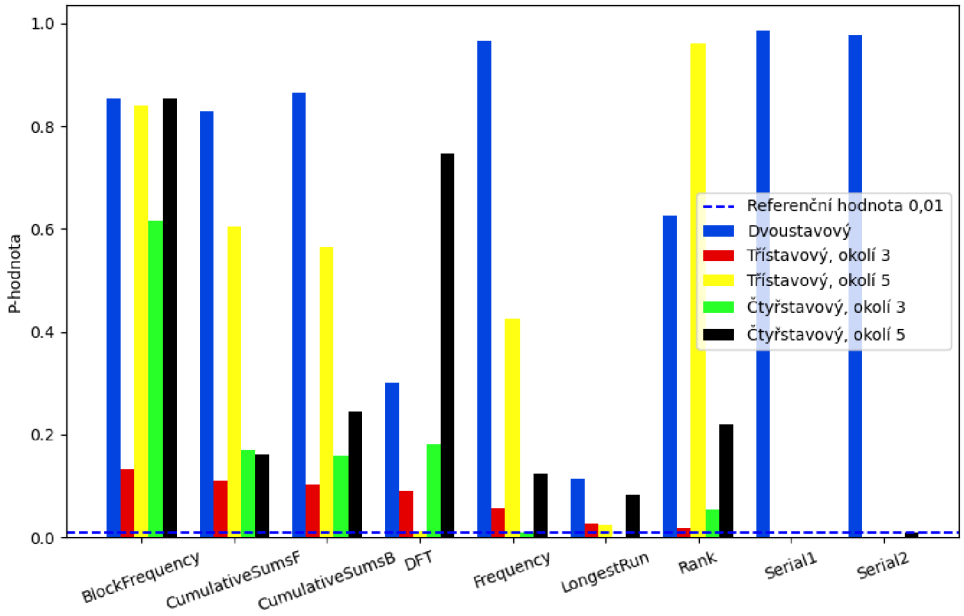
Tabulka 6.1: Výsledky statistických testů jednorozměrných CA s 1 024 čísly, část první

Test	Čtyřstavový, r = 3	Čtyřstavový, r = 5
BlockFrequency	0,616	0,854
CumulativeSumsF	0,17	0,162
CumulativeSumsB	0,16	0,244
DFT	0,181	0,746
Frequency	0,01	0,124
LongestRun	0	0,083
Rank	0,054	0,219
Serial1	0	0
Serial2	0	0,008

Tabulka 6.2: Výsledky statistických testů jednorozměrných CA s 1 024, část druhá

Dle výsledků pro testování s 1 024 čísly lze usoudit, že nejlepších z těchto pěti generátorů je první zmiňovaný a to generátor využívající binární celulární automat. Nejhůře si naopak vedl jeden z hlavních automatů této práce – čtyřstavový celulární automat s okolím o velikosti tři buňky, který neprošel dvěma z testů, ostatní pouze jediným.

Jedním z cílů experimentů bylo zjistit, jestli bude mít větší okolí vliv na kvalitu generovaných čísel. Při pohledu na výsledky třístavového celulárního automatu lze usoudit, že celkově generuje o trochu lepší náhodná čísla než původní automat. Vzhledem k tomu, že jediným rozdílem jsou hodnoty P-hodnot, které jsou ve čtyřech ze sedmi testů lepší, nejedná se o příliš velké vylepšení.



Obrázek 6.1: Výsledky testování jednorozměrných CA na 1 024 číslech

U čtyřstavového automatu je vylepšení kvality generovaných čísel znatelně lepší. U celulárního automatu s trojokolím nepřesahují výsledné hodnoty, krom blokového frekvenčního testu, hodnoty 0,2. Oproti tomu automat s pětiokolím má poměrně vysoké hodnoty, prochází o test více a jedna z hodnot sériového testu nevychází jako čistá nula.

Třístavový CA s čtyřstavovým, oba s pětiokolím, jsou prozatím takřka srovnatelné. Oproti původnímu třístavovému automatu z článku představuje tento čtyřstavový CA jisté vylepšení, stejně jako třístavový automat s pětiokolím.

Další částí testování bylo zvětšení počtu čísel, aby se ukázalo, jak moc budou kvalitní výsledky i při větším počtu čísel. Konkrétně bylo vygenerováno 10 048 osmibitových čísel. Toto číslo bylo zvoleno jako nejbližší násobek čísla 64 větší než 10 000 kvůli pozdějšímu srovnání s dvourozměrnými automaty.

Nad vygenerovanou sekvencí byly spuštěny stejné testy jako v předchozím případě. Výsledky pro tuto sadu testů jsou v tabulkách 6.3 a 6.4 a poté shrnuty v grafu 6.2. Hodnoty v tabulkách jsou zase vypočtené P-hodnoty zaokrouhlené na tři desetinná místa. Graf má stejný popis jako v předchozím případě.

Krom binárního celulárního automatu lze pozorovat znatelné zhoršení. Zbývající automaty v tomto případě neobstály a pro sekvenci 10 048 negenerují kvalitní náhodná čísla. Lze usoudit, že binární dvoustavový automat je z této pětice tedy nejlepším generátorem pseudonáhodných čísel.

Zbývající čtyři celulární automaty jsou na tom v podstatě stejně. Ani jeden z těchto automatů neprošel dle sady hlavním testem – frekvenčním testem. Krom třístavového CA s pětiokolím vychází P-hodnota tohoto testu přesně nula, tedy sekvence je brána jako úplně nenáhodná.

U třístavových automatů nelze říci, že by zvětšení okolí přineslo znatelné výsledky. Oba generátory prošly dvěma ze sedmi testů. Automat s pětiokolím má oproti automatu

Test	Dvoustavový	Třístavový, r = 3	Třístavový, r = 5
BlockFrequency	0,523	0,001	1
CumulativeSumsF	0,505	0	0,001
CumulativeSumsB	0,888	0	0,001
DFT	0,856	0,018	0
Frequency	0,667	0	0,001
LongestRun	0,466	0	0
Rank	0,379	0,903	0,895
Serial1	0,777	0	0
Serial2	0,705	0	0

Tabulka 6.3: Výsledky statistických testů jednorozměrných CA s 10 048 čísly, část první

Test	Čtyřstavový, r = 3	Čtyřstavový, r = 5
BlockFrequency	1	0,078
CumulativeSumsF	0	0
CumulativeSumsb	0	0
DFT	0	0,484
Frequency	0	0
LongestRun	0	0
Rank	0,478	0,389
Serial1	0	0
Serial2	0	0

Tabulka 6.4: Výsledky statistických testů jednorozměrných CA s 10 048, část druhá

s trojokolím o dvě z hodnot, které nejsou nulové, více. Vzhledem k tomu, že tyto hodnoty nedosahují minima pro to, aby byla sekvence uznána za náhodnou, nelze je brát jako důkaz vylepšení výsledků.

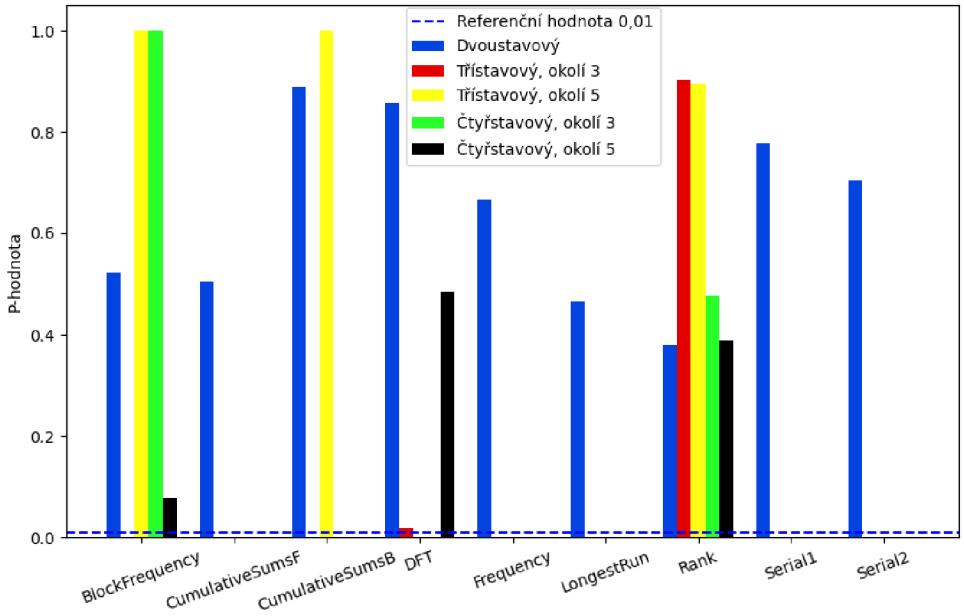
U čtyřstavových automatů automatů mírně zlepšení s větším okolím přichází. Celulární automat s pětiokolím prochází o jedním testem navíc oproti automatu s trojokolím. Nejedná se ale o velké vylepšení, proto toto zlepšení nestojí za velkou zmínku.

Je zajímavé si všimnout výsledků P-hodnot frekvenčního blokového testu (v tabulce *BlockFrequency*) u třístavového CA s pětiokolím a čtyřstavového CA s trojokolím. V těchto dvou případech vychází P-hodnota čistá jedna. Bohužel u ostatních testů není vygenerovaná sekvence tak úspěšná, proto tato hodnota nepředstavuje velký úspěch, spíše bude sekvence splňovat vlastnost, kterou tento test zkoumá, nejlépe.

Z těchto čtyř automatů si nejlépe vedl čtyřstavový CA s okolím o velikosti pět buněk. Na rozdíl od zbylých tří automatů prochází třemi testy ze sedmi. Oproti binárnímu celulárnímu automatu je to ale pořád špatný výsledek.

Jako třetí část testování je využit i vizuální test. Ten není tak spolehlivý jako statistické testy, neměl by být brán tedy jako směrodatný, ale pořád ho lze použít.

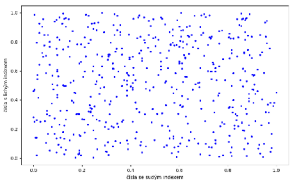
Pro sestavení grafu na obrázcích je potřeba využít vygenerované sekvence čísel. Postupně se berou dvojice čísel. Každá dvojice potom představuje souřadnice grafu. Před zanesením do grafu je třeba ještě hodnoty normalizovat, což je v tomto případě vydělením hodnotou 255, která představuje největší osmitbitové číslo, které můžeme dostat.



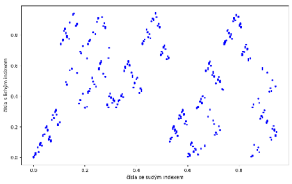
Obrázek 6.2: Výsledky testování jednorozměrných CA na 10 048 číslech

Ve vzniklých grafech lze pozorovat na první pohled, zda je sekvence náhodná, nebo ne. Náhodná sekvence by měla rovnoměrné rozložení bodů a nevznikaly by žádné pravidelné tvary, které by naznačovaly závislost mezi vygenerovanými čísly.

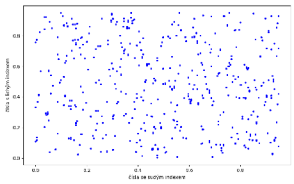
Výsledky tohoto testu jsou na obrázcích 6.3 pro sekvenci 1 024 čísel a 6.4 pro sekvenci 10 048 čísel.



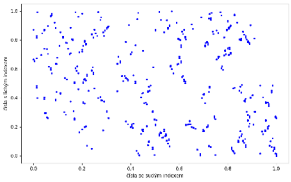
(a) binární



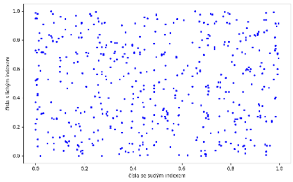
(b) třístavový, okolí 3



(c) třístavový, okolí 5



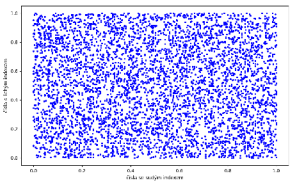
(d) čtyřstavový, okolí 3



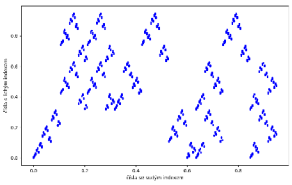
(e) čtyřstavový, okolí 5

Obrázek 6.3: Rozložení čísel pro 1 024 čísel 1D CA

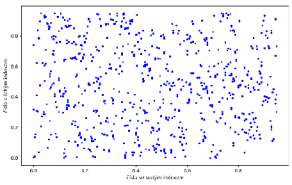
V obrázku 6.3b lze pozorovat jistou pravidelnost tvarů. Tomu by i odpovídaly výsledky statistických testů. Stejně tak u obrázku 6.3d lze vidět, že rozložení není tak rovnoměrné



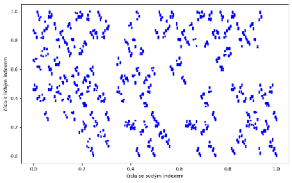
(a) binární



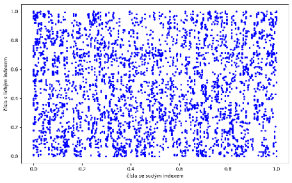
(b) třístavový, okolí 3



(c) třístavový, okolí 5



(d) čtyřstavový, okolí 3



(e) čtyřstavový, okolí 5

Obrázek 6.4: Rozložení čísel pro 10 048 čísel 1D CA

jako u ostatních. I tento automat měl slabší výsledky, takže i v tomto případě výsledky sedí s výsledky statistických testů.

Při zkoumání většího počtu vygenerovaných čísel se u třístavového CA s trojokolím pravidelnost na obrázku 6.4b pouze prohlubuje. Stejně tak v případě čtyřstavového automatu na obrázku 6.4d lze vidět opakující se tvary více znatelně. Naopak binární automat (obrázek 6.4a a čtyřstavový automat s pětiokolím (obrázek 6.4e) vykazují nejvíce rovnoměrné rozložení, což zase sedí s výsledky statistických testů, protože právě tyto dva automaty měly nejlepší výsledky.

6.3 Testování dvourozměrných celulárních automatů

Pro testování byly implementovány čtyři dvourozměrné celulární automaty. Konkrétně se jedná o tyto:

- Uniformní CA
- CA se dvěma různými pravidly
- CA se třemi různými pravidly
- CA se čtyřmi různými pravidly

K otestování těchto generátorů byly použity statistické testy popsané v kapitole 6.1. V první části bylo vygenerováno 1 024 čísel a nad vzniklou sekvencí zase spuštěny testy. Výsledky jsou poté shrnuty v tabulce 6.5 a grafu 6.5.

Testy v tabulce a grafech jsou pojmenovány stejně jako v případě jednorozměrných automatů, což je následovně:

- Frekvenční test v rámci bloku - BlockFrequency
- Test kumulativních součtů od začátku - CumulativeSumsF
- Test kumulativních součtů od konce - CumulativeSumsB

- Test diskrétní Fourierovy transformace - DFT
- Frekvenční (Monobit) test - Frequency
- Test na nejdelší běh jedniček v bloku - LongestRun
- Test stupně binární matice - Rank
- Sériový test - Serial1 a Serial2

V tabulce jsou výsledné P-hodnoty statistických testů stejně jako u jednorozměrných celulárních automatů zaokrouhleny na tři desetinná místa. Graf má na svislé ose zaneseny tyto hodnoty a na vodorovné poté statistické testy. Pro představu, zda byla v daném testu sekvence vyhodnocena jako náhodná, je v grafu čárkově vyznačena hranice 0,01.

Test	1 pravidlo	2 pravidla	3 pravidla	4 pravidla
BlockFrequency	0,817	0,583	0,918	0,673
CumulativeSumsF	0,604	0,991	0,204	0,407
CumulativeSumsB	0,829	0,965	0,315	0,423
DFT	0,935	0,855	0,181	0,935
Frequency	0,808	0,93	0,178	0,251
LongestRun	0,321	0,613	0,885	0,78
Rank	0,626	0,52	0,219	0,17
Serial1	0,645	0,911	0,056	0,773
Serial2	0,973	0,403	0,082	0,78

Tabulka 6.5: Výsledky statistických testů 2D CA s 1 024 čísly

Z výsledků lze pozorovat, že všechny tyto generátory pseudonáhodných čísel dopadly celkem podobně. Všechny úspěšně prošly celou použitou sadou statistických testů, liší se pouze v hodnotách. Nejslabším podle získaných hodnot je poté automat s využitím tří pravidel, který má až na dva případy podstatně slabší výsledky než zbývající.

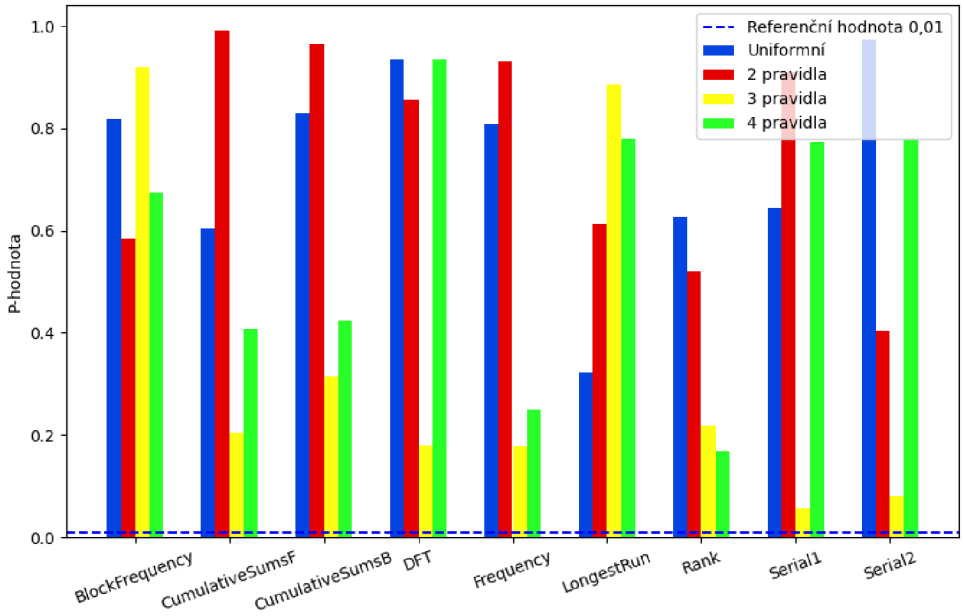
Trochu může být překvapivé, že automat využívající čtyři pravidla, který byl převzat z článku jako automat s nejlepší kombinací použitých pravidel, není ve všech testech nejlepší. To může být způsobeno implementací, protože byla použita pomyslná hranice buněk s hodnotou nula na okrajích, ne vazba na buňky na druhé straně. Toto by však mohlo být předmětem dalších experimentů.

Po první části testování zatím nelze říci, že by změna počtu pravidel přivedla nějakou extrémní změnu. Sice se změnily hodnoty, ale celkově ne o tolik, aby to stálo za zmínku.

Další části bylo i tentokrát porovnání generátorů na základě vygenerované sekvence čísel většího počtu. Byla vygenerována sekvence 10 048 náhodných čísel, která byla dále zkoumaná. Výsledky statistických testů nad těmito sekvencemi jsou zobrazeny v tabulce 6.6 a shrnuty do grafu 6.6. Značení a významy zůstávají stejné jako v předchozích případech.

Rozdíly oproti výsledkům 1 024 čísel nejsou u dvourozměrných celulárních automatů tak obrovské jako v případě jednorozměrných. Všechny čtyři zase prošly všemi testy úspěšně, proto i v tomto případě nelze říci, že by změna počtu pravidel změnila výsledky příliš výrazně.

Nejhůře se pořád vede celulární automat se třemi pravidly. Naopak velmi dobře si vede uniformní automat, který má pro skoro všechny testy velmi vysoké výsledky P-hodnoty. Nejlépe si vedl právě uniformní celulární automat a původní automat se čtyřmi pravidly.



Obrázek 6.5: Výsledky testování dvourozměrných CA na 1 024 číslech

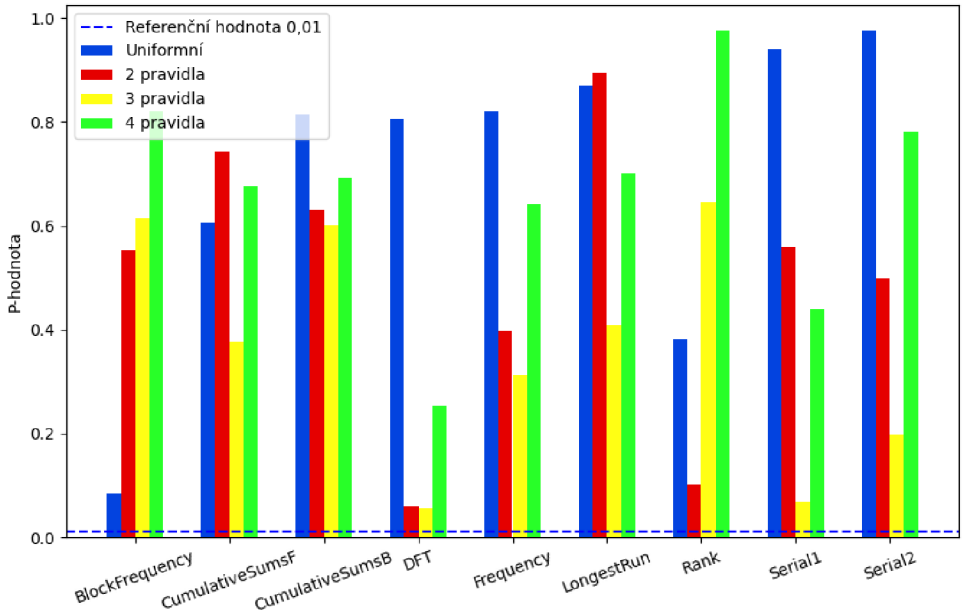
Test	1 pravidlo	2 pravidla	3 pravidla	4 pravidla
BlockFrequency	0,085	0,552	0,615	0,821
CmulativeSumsF	0,605	0,743	0,376	0,677
CumulativeSumsB	0,814	0,631	0,602	0,693
DFT	0,806	0,059	0,055	0,252
Frequency	0,821	0,397	0,313	0,642
LongestRun	0,87	0,894	0,408	0,7
Rank	0,381	0,103	0,646	0,976
Serial1	0,94	0,56	0,068	0,44
Serial2	0,976	0,499	0,198	0,781

Tabulka 6.6: Výsledky statistických testů 2D CA s 10 048 čísly

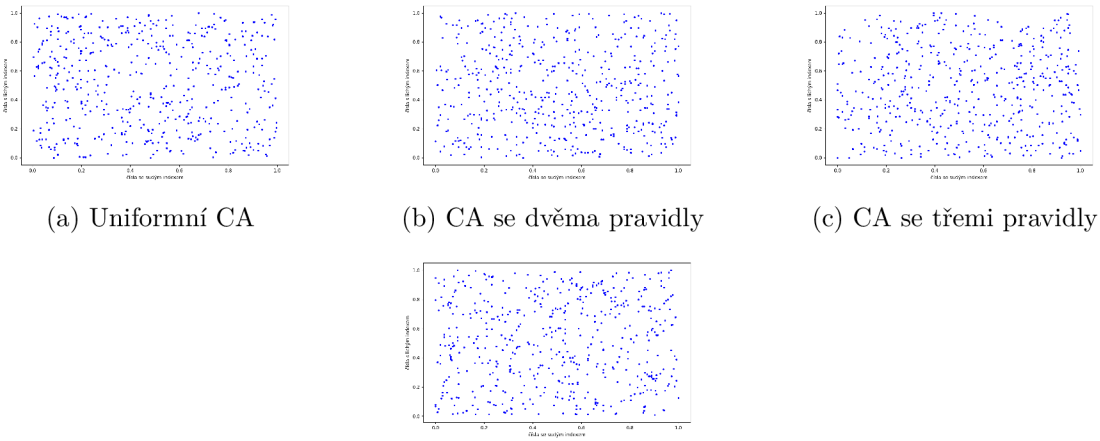
V poslední části testování byl proveden vizuální test. Vygenerovaná čísla byla brána po dvojicích, které sloužily jako souřadnice bodu do grafu. Byla použita normalizovaná čísla, tedy v případě použití osmibitových čísel bylo třeba před zanesením do grafu vydělit hodnotou 255.

Výsledky tohoto testu jsou na obrázcích 6.7 pro sekvenci 1 024 čísel a 6.8 pro sekvenci 10 048 čísel.

Stejně jako u statistických testů vychází i tato část testování pro všechny generátory pseudonáhodných čísel s dvourozměrnými celulárními automaty podobně. U všech lze pozorovat v podstatě rovnoměrné rozložení, které by odpovídalo tomu, že všechny generují velmi kvalitní náhodná čísla.



Obrázek 6.6: Výsledky testování dvourozměrných CA na 10 048 číslech



(a) Uniformní CA

(b) CA se dvěma pravidly

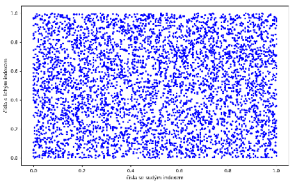
(c) CA se třemi pravidly

(d) CA se čtyřmi pravidly

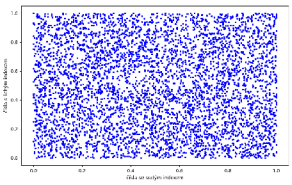
Obrázek 6.7: Rozložení čísel pro 1 024 čísel 2D CA

6.4 Celkové srovnání

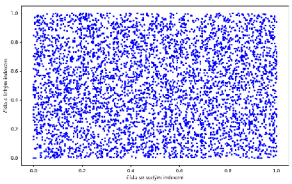
V poslední podkapitole budou srovnány čtyři nejlepší generátory z obou předchozích částí. Pro celkové srovnání byly vybrány dva nejlepší jednorozměrné a dva dvourozměrné celulární automaty. Pokud by měly být vybrány celkově čtyři nejlepší generátory, byly by pro svoji podobnost vybrány pouze ty dvourozměrné. Vzhledem k tomu, že jejich výsledky ale nejsou tolik odlišné, budou vybrány pouze dva a další dva z jednorozměrných celulárních automatů, ačkoliv nebudou představovat nejlepší řešení.



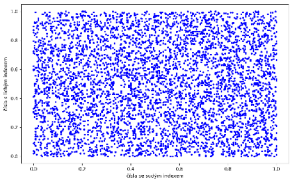
(a) Uniformní CA



(b) CA se dvěma pravidly



(c) CA se třemi pravidly



(d) CA se čtyřmi pravidly

Obrázek 6.8: Rozložení čísel pro 10 048 čísel 2D CA

Podle výsledků ze statistických testů byly vybrány pro celkové zhodnocení tyto jednorozměrné celulární automaty:

- Dvoustavový jednorozměrný celulární automat
- Čtyřstavový jednorozměrný celulární automat s okolím o velikosti pět buněk

Z dvourozměrných celulárních automatů budou v celkovém srovnání porovnávány tyto:

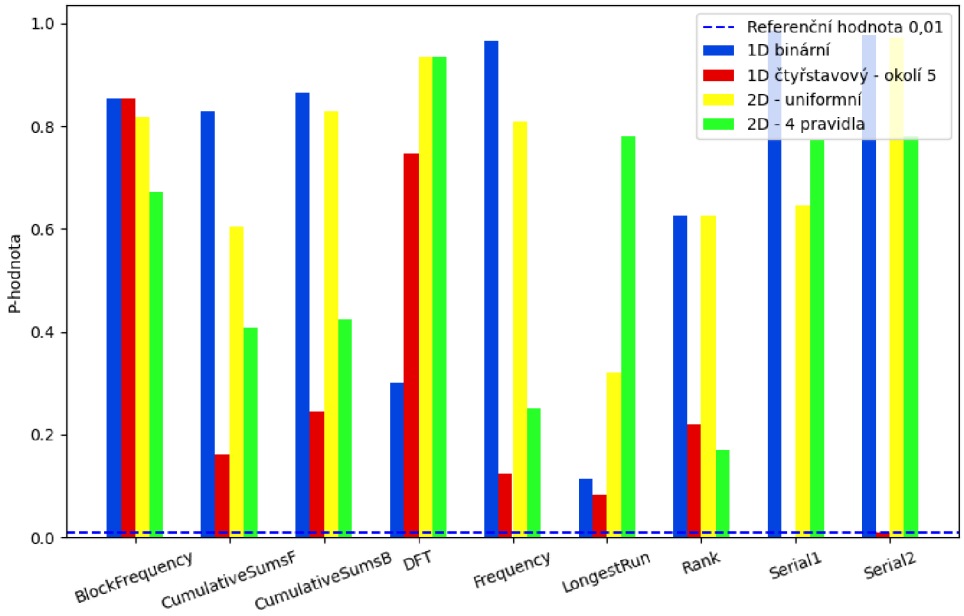
- Dvourozměrný uniformní celulární automat
- Dvourozměrný neuniformní celulární automat se čtyřmi různými pravidly

Generátory nebyly znovu přetestovány, byly využity už předtím získané hodnoty, protože obě skupiny generátorů byly testovány na 1 024 číslech i sekvenci 10 048 čísel za použití stejné sady testů.

Výsledky těchto automatů se sekvencí 1 024 čísel ukazuje dohromady tabulka 6.7 a graf 6.9. Sekvence 10 048 čísel dále tabulka 6.8 a graf 6.10. Značení testů v tabulkách, zaokrouhlení a popisy grafů jsou stejné jako u všech předchozích případů.

Test	1D dvoustavový	1D čtyřstavový	2D - uniformní	2D - čtyři pravidla
BlockFrequency	0,853	0,854	0,817	0,673
CumulativeSumsF	0,829	0,162	0,604	0,407
CumulativeSumsB	0,866	0,244	0,829	0,423
DFT	0,301	0,746	0,935	0,935
Frequency	0,965	0,124	0,808	0,251
LongestRun	0,115	0,083	0,321	0,78
Rank	0,626	0,219	0,626	0,17
Serial1	0,986	0	0,645	0,773
Serial2	0,978	0,008	0,973	0,78

Tabulka 6.7: Výsledky statistických testů pro 1 024 čísel



Obrázek 6.9: Výsledky testování CA na 1 024 číslech

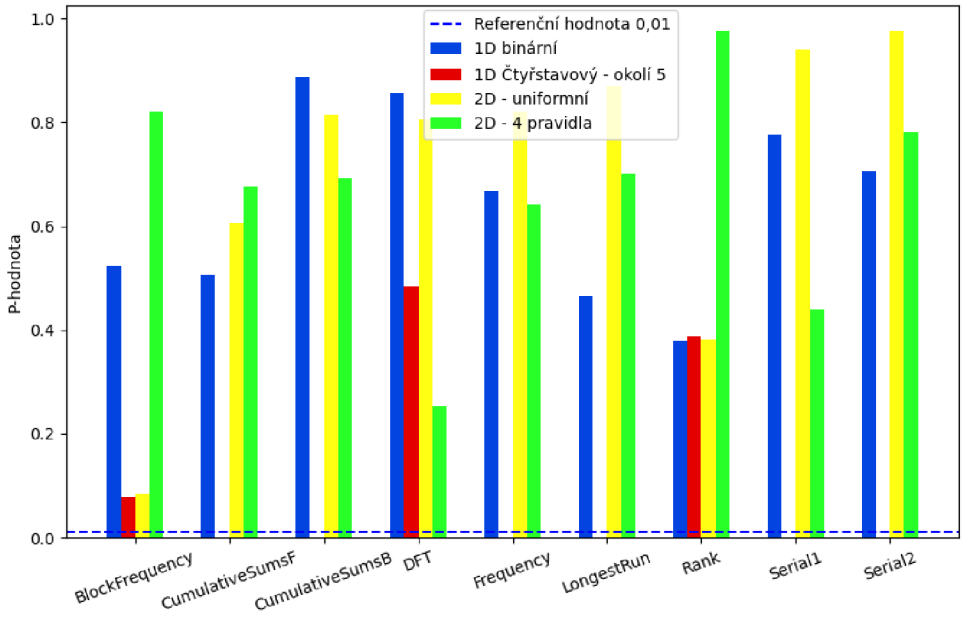
Test	1D dvoustavový	1D čtyřstavový	2D - uniformní	2D - čtyři pravidla
BlockFrequency	0,523	0,078	0,085	0,821
CumulativeSumsF	0,505	0	0,605	0,667
CumulativeSumsB	0,888	0	0,814	0,693
DFT	0,856	0484	0,806	0,252
Frequency	0,667	0	0,821	0,642
LongestRun	0,466	0	0,87	0,7
Rank	0,379	0,389	0,381	0,976
Serial1	0,777	0	0,94	0,44
Serial2	0,705	0	0,976	0,781

Tabulka 6.8: Výsledky statistických testů pro 10 048 čísel

Už při pohledu na výsledky testování vygenerované sekvence 1 024 čísel lze poznat, že nejslabším z těchto automatů je jednorozměrný čtyřstavový. Zatímco všechny v tomto případě prochází všemi statistickými testy, čtyřstavový pouze šesti z nich. Důvodem může být nenalezení úplně nejlepšího pravidla, což vzhledem k počtu možných řešení je pravděpodobné, nebo zvýšení počtu stavů nepředstavuje lepší řešení než doposud známá.

Naopak nejlepší výsledky testů má nejjednodušší z implementovaných celulárních automatů a to jednorozměrný dvoustavový uniformní automat. Jako jediný prochází všemi statistickými testy v případě 1 024 čísel i 10 048 čísel.

Ve výsledném porovnání má nejlepší výsledky jednorozměrný dvoustavový celulární automat, jako další poté dvourozměrné celulární automaty a nejhorší řešení z těchto čtyř nakonec představuje experimentální čtyřstavový. Na základě testů lze říci, že původní dvoustavové automaty jsou velmi kvalitní a naopak čtyřstavový celulární automat nepřináší oproti nim žádné vylepšení.



Obrázek 6.10: Výsledky testování CA na 10 048 číslech

Kapitola 7

Závěr

Práce měla za cíl prozkoumat metody využité ke generování pseudonáhodných čísel využívající celulární automaty, porovnat je a pokusit se nějakým vhodným způsobem vylepšit.

K porovnání byly vybrány tři již prozkoumané celulární automaty a to konkrétně: jednorozměrný uniformní dvoustavový, jednorozměrný uniformní třístavový a dvourozměrný neuniformní dvoustavový. Po nastudování současných metod byl přidán ještě nový způsob, který by mohl současná řešení vylepšit, a to jednorozměrný uniformní čtyřstavový automat.

Po implementaci všech automatů, včetně verzí se zvětšeným okolím, nebo jiným počtem celkových pravidel pro dvourozměrný automat, bylo provedeno testování pomocí sady statistických testů od institutu NIST a následně generátory porovnány. Provedené experimenty ukázaly, že nejlepší z implementovaných automatů jsou binární, konkrétně potom jednorozměrný, avšak dvourozměrné nejsou o tolik horší.

Navržený čtyřstavový automat sice není lepší než tyto dva typy automatů, ale dle výsledků testování přináší vylepšení oproti třístavovému automatu, který ve srovnání dopadl nejhůře.

Na práci by bylo možné navázat pokusem o zdokonalení čtyřstavového automatu, protože nebyla prozkoumána všechna možná řešení. Jednou z možností by bylo použít jiný způsob přechodu do nového stavu nebo by se dalo zaměřit i na způsob získávání pseudonáhodných čísel. Další jistě zajímavou možností by mohl být tento celulární automat v neuniformní podobě.

Dalším celulárním automatem, který by mohl generovat zajímavé výsledky, je dvourozměrný třístavový celulární automat. Tento způsob by určitě přinesl zajímavé výsledky, protože spojuje výhody dvourozměrného, který se v této práci ukázal jako velmi kvalitní, a třístavového, který sice neměl tak dobré výsledky, ale přidání nového rozměru by mohlo k nějakému zlepšení dojít.

Literatura

[1] Alfaro, H. Generating Interesting Patterns in Conway's Game of Life Through a Genetic Algorithm. 2009.

[2] Aumasson, J.-P. On the pseudo-random generator ISAAC. *IACR Cryptology ePrint Archive*. leden 2006, sv. 2006. Dostupné z: https://eprint.iacr.org/2006/438.pdf.

[3] Bassham, L. E., Rukhin, A., Soto, J., Nechvatal, J., Smid, M. et al. *A statistical test suite for random and pseudorandom number generators for cryptographic applications*. 2010. Dostupné z: https://doi.org/10.6028/nist.sp.800-22r1a.

[4] Berto, F. a Tagliabue, J. Cellular Automata. In: Zalta, E. N., ed. *The Stanford Encyclopedia of Philosophy* [online]. 5. vyd. Metaphysics Research Lab, Stanford University, Březen 2012, editováno 26.12.2022. Dostupné z: https://plato.stanford.edu/archives/spr2022/entries/cellular-automata/.

[5] Bhattacharjee, K. a Das, S. A list of tri-state cellular automata which are potential pseudo-random number generators. *International Journal of Modern Physics C*. World Scientific Pub Co Pte Lt. září 2018, sv. 29, č. 09. DOI: 10.1142/s0129183118500882. Dostupné z: https://doi.org/10.1142/s0129183118500882.

[6] Bhattacharjee, K., Paul, D. a Das, S. Pseudo-random number generation using a 3-state cellular automaton. *International Journal of Modern Physics C*. World Scientific Pub Co Pte Lt. duben 2017, sv. 28, č. 06. DOI: 10.1142/s0129183117500784. Dostupné z: https://doi.org/10.1142/s0129183117500784.

[7] Blum, L., Blum, M. a Shub, M. A Simple Unpredictable Pseudo-Random Number Generator. *SIAM Journal on Computing*. 1986, sv. 15, č. 2, s. 364–383. DOI: 10.1137/0215025.

[8] Canteaut, A. Linear Feedback Shift Register. In: Tilborg, H. C. A. van a Jajodia, S., ed. *Encyclopedia of Cryptography and Security*. Boston, MA: Springer US, 2011, s. 726–729. DOI: 10.1007/978-1-4419-5906-5_357. ISBN 978-1-4419-5906-5. Dostupné z: https://doi.org/10.1007/978-1-4419-5906-5_357.

[9] Elveny, M., Syah, R., Jaya, I. a Affandi, I. Implementation of Linear Congruential Generator (LCG) Algorithm, Most Significant Bit (MSB) and Fibonacci Code in Compression and Security Messages Using Images. *Journal of Physics: Conference Series*. IOP Publishing. červen 2020, sv. 1566, č. 1. DOI: 10.1088/1742-6596/1566/1/012015. Dostupné z: https://dx.doi.org/10.1088/1742-6596/1566/1/012015.

[10] Geisler, M., Krøigård, M. a Danielsen, A. About Random Bits. Prosinec 2004.

[11] Jenkins, R. ISAAC: a fast cryptographic random number generator. *Bob Jenkins' web page*. Viděno 28.4.2023. Dostupné z: https://burtleburtle.net/bob/index.html. Path: Home page; cryptographic pseudorandom number generator ISAAC,.

[12] Johnston, N. a Greene, D. *Conway's Game of Life*. Self-published, 2022. ISBN 978-1-794-81696-1. Dostupné z: https://conwaylife.com/book/.

[13] Mallawaarachchi, V. *Introduction to Genetic Algorithms — Including Example Code* [online]. 2017 [cit. 2023-04-17]. Dostupné z: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3.

[14] M'hamdi, A. a Nemiche, M. Bottom-Up and Top-Down Approaches to Simulate Complex Social Phenomena. *International Journal of Applied Evolutionary Computation*. Duben 2018, sv. 9, s. 1–16. DOI: 10.4018/IJAEC.2018040101.

[15] Peringer, P. *Modelování a simulace - studijní opora*. 2021 [cit. 2023-04-19].

[16] Perrenoud, M., Sipper, M. a Tomassini, M. On the generation of high-quality random numbers by two-dimensional cellular automata. *IEEE Transactions on Computers*. Institute of Electrical and Electronics Engineers (IEEE). 2000, sv. 49, č. 10, s. 1146–1151. DOI: 10.1109/12.888056. Dostupné z: https://doi.org/10.1109/12.888056.

[17] Rose, G. KISS: A bit too simple. *IACR Cryptology ePrint Archive*. leden 2018, sv. 2011. DOI: 10.1007/s12095-017-0225-x.

[18] Sipper, M. *Evolution of Parallel Cellular Machines - The Cellular Programming Approach*. 2004 [cit. 2023-04-19]. Dostupné z: http://csis.pace.edu/~marchese/CS396x/Computing/epcm.pdf.

[19] Vuckovac, R. Secure and Computationally Efficient Cryptographic Primitive Based on Cellular Automaton. *Complex Systems*. Wolfram Research, Inc. prosinec 2019, sv. 28, č. 4, s. 457–474. Dostupné z: https://doi.org/10.25088/complexsystems.28.4.457.

[20] Warne, D. a Hayward, R. The Dynamics of Cellular Automata on 2-Manifolds is Affected by Topology. *Journal of cellular automata*. jan 2015, sv. 10, s. 319–339.

Příloha A

Obsah SD karty

```
/
├── src/
│   ├── Makefile
│   ├── 1d_bin/
│   ├── 1d_three_state/
│   ├── 2d_bin/
│   ├── 1d_four_state/
│   └── scripts/
├── src_latex/
├── pdf/
└── doc/
```

Složka **src** obsahuje zdrojové soubory implementovaných programů v jazyce C. Ve složce **1d_bin** se nachází zdrojové kódy pro jednorozměrný dvoustavový automat, ve složce **1d_three_state** se nachází zdrojové soubory třístavového automatu, složka **2d_bin** obsahuje zdrojové kódy dvourozměrného automatu a ve složce **1d_four_state** jsou poté zdrojové kódy čtyřstavového automatu a implementovaného genetického algoritmu. Ve složce **scripts** se nachází pomocné skripty v jazyce Python. Složka **src_latex** obsahuje zdrojové soubory písemné zprávy v LaTeXu. Ve složce **pdf** je tato písemná zpráva ve formátu pdf. Složka **doc** obsahuje soubor readme.md popisující spuštění a návod k použití.

Příloha B

Návod k použití programů

Pro překlad programů je třeba použít překladač pro jazyk C. V rámci práce byly programy překládány pomocí překladače **gcc 11.3.1**.

Všechny programy je možné přeložit po použití příkazu `make` pomocí přiloženého `Makefile`. Dále je možné použít překládat programy jednotlivě pomocí následujících příkazů:

- `make gen` - přeloží program pracující s jednorozměrných dvoustavovým celulárním automatem
- `make three` - přeloží program pracující s jednorozměrným třístavovým celulárním automatem
- `make two_dim` - přeloží program pracující s dvourozměrným dvoustavovým celulárním automatem
- `make four` - přeloží program pracující s jednorozměrným čtyřstavovým celulárním automatem
- `make evolution` - přeloží program s genetickým algoritmem hledající pravidla čtyřstavového celulárního automatu

Dále lze použít příkaz `make` spolu s příkazem `clean`, který smaže přeložené programy.

1D dvoustavový automat

Program s jednorozměrným dvoustavovým celulárním automatem se spouští s následujícími parametry:

```
./gen COUNT SEED
```

- `COUNT` - požadovaný počet vygenerovaných čísel
- `SEED` - semínko generátoru ve formátu řetězce

Příklad spuštění: `./gen 1024 entropy`.
Po spuštění generátor vypisuje vygenerovaná osmibitová čísla na standardní výstup.

1D třístavový automat

Program s jednorozměrným třístavovým celulárním automatem se spouští s následujícími parametry:

```
./three SIZE SEED COUNT RADIUS
```

- `SIZE` - požadovaná velikost generovaných číslech v bitech
- `SEED` - semínko generátoru zadáváno jako číslo
- `COUNT` - požadovaný počet vygenerovaných čísel
- `RADIUS` - velikost okolí, se kterým má CA pracovat; lze zadat pouze hodnotu 3 nebo 5

Příklad spuštění: `./three 8 123456 1024 3`
Po spuštění generátor vypisuje vygenerovaná čísla na standardní výstup.

2D dvoustavový automat

Program s dvourozměrným dvoustavovým celulárním automatem se spouští s následujícími parametry:

```
./two_dim COUNT SEED MODE
```

- `COUNT` - požadovaný počet vygenerovaných čísel
- `SEED` - semínko generátoru zadávané jako binární sekvence maximálně 64 čísel, při použití méně jsou zbývající stavy nastaveny na hodnotu 0
- `MODE` - počet pravidel, se kterými je automat spuštěn, validní jsou hodnoty: 1, 2, 3 a 4

Příklad spuštění: `./two_dim 1024 1010101010 2`
Po spuštění generátor vypisuje vygenerovaná osmibitová čísla na standardní výstup.

1D čtyřstavový automat

Program s jednorozměrným čtyřstavovým celulárním automatem se spouští s následujícími parametry:

```
./four RADIUS COUNT SEED
```

- `RADIUS` - velikost okolí, se kterým CA pracuje; lze zadat pouze hodnotu 3 nebo 5
- `COUNT` - požadovaný počet vygenerovaných čísel
- `SEED` - semínko generátoru zadané jako číslo ve čtyřkové soustavě

Příklad spuštění: `./four 3 1024 1032100`
Po spuštění generátor vypisuje vygenerovaná osmibitový čísla na standardní výstup.

Program s genetickým algoritmem

Program využívající genetický algoritmus pro generování tabulky pravidel čtyřstavového automatu lze spustit následovně:

`./evolution RADIUS`

- `RADIUS` - určuje velikost okolí, pro které má generovat pravidla; lze použít pouze hodnoty 3 a 5

Příklad spuštění: `./evolution 3`

Pro spuštění program začne hledat nejlepší tabulku pravidel, kterou poté vypíše na standardní výstup. Běh programu může trvat několik minut, je třeba s tím počítat. Prozatím nepřekročil nikdy více jak 30 minut.

Pomocné skripty

Pro potřeby práce byly sestaveny následující pomocné skripty v jazyce Python 3.10.6:

- `generate3stateRules.py`
- `tobin.py`

K jejich běhu není třeba zvláštních knihoven ani jiných závislostí.

Skript `generate3stateRules.py` slouží k vygenerování náhodné tabulky pravidel pro třístavový automat s pětiokolím.

Spuštění: `python3 generate3stateRules.py`.

Skript po spuštění vypíše na standardní výstup řetězec představující vygenerovanou tabulku pravidel.

Skript `tobin.py` slouží k převedení vygenerované sekvence čísel do binární podoby pro testování. Spuštění `python3 tobin.py FILENAME NUMBER_OF_BITS`..

- `FILENAME` - název souboru obsahující převáděnou sekvenci čísel
- `NUMBER_OF_BITS` - počet bitů, na kolik má čísla převést, bylo použito pouze pro osmibitová čísla

Po spuštění skript vypíše na standardní výstup sekvenci čísel ze zadaného souboru v binární soustavě na požadovaném počtu bitů.