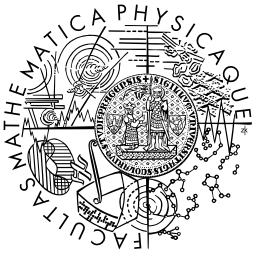


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Marek Polák

Využití celulárních automatů pro kompresi dat

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce:	Mgr. Otakar Trunda
Studijní program:	Informatika
Studijní obor:	Programování a softwarové systémy

Praha 2016

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Název práce: Využití celulárních automatů pro kompresi dat

Autor: Marek Polák

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Otakar Trunda, Katedra teoretické informatiky a matematické logiky

Abstrakt: V této práci se zabýváme možnostmi využití celulárních automatů pro bezeztrátovou kompresi dat. Popisujeme klasifikaci celulárních automatů a jejich dosavadní využití. Zkoumáme vlastnosti jednotlivých typů elementárních celulárních automatů (tzv. Wolframova pravidla), popisujeme jejich třídy ekvivalence, možnosti jak dopředné, tak i zpětné simulace, zkoumáme pravidla se zajímavým chováním. Stavy získané těmito pravidly hodnotíme z hlediska jejich uspořádanosti (např. poměr živých buněk či aproximace entropie). Implementujeme některé standardní kompresní algoritmy a porovnáváme je z hlediska využitelnosti pro nejlépe ohodnocené stavy. Aplikací získaných poznatků navrhujeme nový kompresní algoritmus, testujeme jej na textových a obrazových datech a výsledky srovnáváme s tradičními kompresními algoritmy.

Klíčová slova: celulární automat, komprese dat, transformace dat, Wolframova pravidla

Title: Using Cellular Automata for Data Compression

Author: Marek Polák

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Otakar Trunda, Department of Theoretical Computer Science and Mathematical Logic

Abstract: In this thesis we research the possibilities of using cellular automata for lossless data compression. We describe the classification of cellular automata and their current usage. We study the properties of various types of elementary cellular automata (i.e. Wolfram rules), describe their equivalence classes, the ways of forward as well as backward simulation, we examine the rules with interesting behavior. The states provided by these rules are evaluated in terms of their orderliness (e.g. the ratio of living cells or approximation of entropy). We implement some standard compression algorithms and compare them in terms of usability for best rated states. By application of acquired knowledge we propose a new compression algorithm, test it on text and image data and compare the results with traditional compression algorithms.

Keywords: cellular automaton, data compression, data transforms, Wolfram rules

Rád bych poděkoval vedoucímu mé bakalářské práce Mgr. Otakaru Trundovi za trpělivost a ochotu, se kterou mou práci vedl, jakož i za poskytnuté rady, díky kterým jsem práci mohl dokončit.

Obsah

1. Úvod

1.1 Motivace

Celulární automaty jsou v informatice stále nedostatečně probádanou oblastí, toto opomíjení je však neoprávněné. I přes jejich zdánlivě jednoduchou definici jsou například schopny z triviálních vstupních dat generovat vysoce náhodné binární posloupnosti či simulovat Turingův stroj a tedy řešit libovolný problém řešitelný pomocí jiného turingovsky úplného výpočetního modelu. Nabízí se tedy otázka, zda některé celulární automaty mohou transformovat vstupní data způsobem výhodným pro určité metody bezeztrátové komprese dat.

1.2 Cíl práce

Cílem této práce je prokázat či vyvrátit hypotézu, že v závislosti na parametrech mohou některé typy celulárních automatů transformovat vstupní data takovým způsobem, že tato se stanou snáze komprimovatelnými některým bezeztrátovým kompresním algoritmem. Tento algoritmus může být jak již zavedený, tak vyvinutý speciálně za tímto účelem.

Pokud se hypotéza potvrdí, bude výsledkem práce bezeztrátový kompresní algoritmus, využívající v některém svém kroku transformaci celulárním automatem a dosahující na reálných datech kladné úspory místa a lepšího kompresního poměru než s vynecháním této transformace. Rychlost algoritmu není prioritou, měl by však být bez větších obtíží spustitelný na běžném osobním počítači.

1.3 Struktura práce

V kapitole Úvod nastíníme danou problematiku a představíme cíle této práce. V kapitole Analýza shrneme základní poznatky týkající se problematiky komprese dat a celulárních automatů. Kapitola Návrh popisuje námi navrhované řešení problému a veškeré implementované algoritmy, které může toto řešení využívat. V kapitole Experimenty představíme testovací data, s jejich pomocí sadou experimentů určíme nejvýhodnější parametry námi navrženého řešení a zhodnotíme výsledky jeho aplikace. V kapitole Závěr shrneme výsledky této práce, zhodnotíme splnění cíle práce a navrhneme další bádání v této oblasti.

2. Analýza

V této části se nejprve seznámíme s problémem komprese dat. Následně představíme obecné principy celulárních automatů. Obzvláštní důraz budeme klást na jejich podmnožinu, elementární celulární automaty. Popíšeme jejich klasifikaci (takzvaná Wolframova pravidla), ekvivalence mezi nimi a některé jejich parametry. Krátce popíšeme jejich chování při dopředné a zpětné simulaci.

2.1 Komprese dat

Kompresí dat nazýváme proces, jehož cílem je zmenšit objem počítačových dat a tím snížit jejich nároky na diskovou, případně přenosovou kapacitu. Nutnou podmínkou pro chápání takového procesu jakožto komprese dat je jeho reverzibilita, tj. musí existovat proces opačný, nazývaný dekomprese dat, pomocí kterého lze opětovně získat původní data. Platí tedy, že komprimovaná data zachovávají informaci obsaženou v původních datech (v určitých případech pouze její aproximaci, viz níže).

Volba konkrétního kompresního algoritmu se odvíjí od typu komprimovaných dat a našich požadavků na ně. Na obecná data (tj. bez znalosti jejich struktury) a textová data je obvykle nutno aplikovat tzv. bezeztrátovou kompresi dat, při které je kompletně zachována obsažená informace a dekompresí komprimovaných dat získáme data identická s daty původními. Uvažujeme-li data jako řetězce bitů nenulové délky, pak si lze bezeztrátovou kompresi představit jako prosté zobrazení v množině všech těchto řetězců $c\colon \{0,1\}^* \to \{0,1\}^*$, dekompresi pak jako inverzní zobrazení $c^{-1}\colon \mathrm{H}(c) \to \{0,1\}^*$. Komprese dosáhneme, zobrazíme-li vstupní řetězec na řetězec menší délky.

Jednoduchým pozorováním lze ukázat, že zobrazí-li komprese některý vstupní řetězec na řetězec kratší, pak existuje jiný řetězec, který bude naopak zobrazen na řetězec delší. Předpokládejme kompresi c, pro kterou uvedený výrok neplatí. Volme nejmenší m takové, že řetězec s délky m je zobrazen na řetězec délky $n < m$. Z minimality m vyplývá, že všechny řetězce délky n se zobrazí též na řetězce délky n. Řetězců délky n je 2^n, včetně s se na ně tedy musí zobrazit $2^n + 1$ řetězců. Důsledkem Dirichletova principu pak je, že zobrazení c nemůže být prosté.

Z tohoto pozorování vyplývá, že pro každý bezeztrátový kompresní algoritmus existují vstupní data, která po kompresi tímto algoritmem zvětší svůj objem. Algoritmus tedy dokáže úspěšně komprimovat pouze určitá data. V tomto kontextu se často používá míra známá jako Kolmogorovova složitost, neformálně definovaná jako délka nejkratšího programu, který na výstup vypíše daná data. Pokud je Kolmogorovova složitost dat menší než jejich velikost, pak je zjevně lze bezeztrátově komprimovat. Naopak, Kolmogorovova složitost dat, která nelze bezeztrátově komprimovat, je rovna minimálně jejich velikosti (viz Sayood, 2012, Kapitola 2.5). Mezi nejznámější bezeztrátové kompresní algoritmy patří např. LZ77, LZ78 a DEFLATE.

Pro multimediální data (např. obrazové či zvukové soubory) se ve většině případů používají různé metody tzv. ztrátové komprese dat. S přihlédnutím k nedokonalosti lidských smyslů je totiž možné část původní informace v multimediálních datech vypustit a dosáhnout tak lepší komprese. Dekompresí komprimovaných dat pak vzniká pouhá aproximace dat původních, při odpovídající multimediální interpretaci by však rozdíly měly být lidskými smysly v ideálním případě téměř nerozpoznatelné. Mezi v současné době široce užívané ztrátové kompresní algoritmy patří JPEG pro obrazové soubory či MP3 pro zvukové soubory.

Komprese dat obecně funguje na principu vyhledávání a vypouštění redundantní informace z komprimovaných dat, a tedy zvyšování informační entropie dat (viz Shannon, 2001, Kapitola 6). Kvalita komprese dat je obvykle udávána veličinou zvanou *kompresní poměr*, definovanou jako

$$R = \frac{U}{C},$$

kde U je objem původních nekomprimovaných dat a C objem týchž dat komprimovaných. Kompresní poměr tedy v případě úspěšné komprese nabývá hodnot z intervalu $[1, \infty)$, přičemž platí, že čím větší kompresní poměr je, tím kvalitnější komprese bylo dosaženo. Hodnoty kompresního poměru menší než jedna značí, že data po kompresi mají větší objem než před ní a tedy ke kompresi nedošlo. Někdy se též udává veličina zvaná *úspora místa* a definovaná jako

$$S = 1 - \frac{C}{U} = 1 - \frac{1}{R}.$$

Úspora místa v případě úspěšné komprese může nabývat hodnot z intervalu [0,1), přičemž vyšší hodnoty značí kvalitnější kompresi. Lze též hovořit o záporné úspoře místa, pokud je objem dat po kompresi větší než před ní.

2.2 Celulární automaty

Celulární automat (zkráceně CA) je diskrétní matematický model simulující vývoj konfigurace buněk v závislosti na čase. Obvykle je chápán jako nekonečná n-rozměrná matice sestávající z polí či buněk, kdy každá buňka může nabývat m různých stavů. Na počátku simulace musí být automatu dána konkrétní konfigurace všech buněk automatu, tzv. počáteční stav. Stav automatu se následně mění v čase po diskrétních krocích — generacích. Stav jedné konkrétní buňky v generaci t je určen přechodovou funkcí z jejího stavu a stavu všech jejích sousedních buněk v generaci $t - 1$. Celkový stav automatu tedy v každém okamžiku závisí pouze na počátečním stavu v generaci 0 a počtu odsimulovaných generací.

Formální definice obecného celulárního automatu je vzhledem k jeho různorodým variantám obtížná. Za cenu jistých nepřesností můžeme definovat celulární automat jako čtveřici $\mathrm{CA} = (\Sigma, C, \rho, \sigma)$, kde

- Σ je konečná neprázdná množina možných stavů jednotlivých buněk,
- C je neprázdná množina souřadnic jednotlivých buněk,
- $\rho\colon C \to C^k, k \in \mathbb{N}$ je funkce pro výběr souřadnic sousedních buněk, a
- $\sigma\colon \Sigma^k \to \Sigma$ je přechodová funkce pro jednu buňku.

Stav celého automatu pak můžeme chápat jako zobrazení množiny souřadnic do množiny stavů buněk $s\colon C \to \Sigma$ a množinu všech stavů automatu jako množinu $S = \{s \mid s\colon C \to \Sigma\}$. Nevýhodou této definice je například ztráta informace o dimenzionalitě automatu (je skryta uvnitř množiny souřadnic) či nevyslovený požadavek na „lokalitu“ funkce sousednosti (není zohledněna blízkost sousedních buněk). Předností definice je naopak její univerzalita vůči různým exotičtějším formám celulárních automatů (např. hexagonálním či s asymetrickou sousedností). Simulaci jedné generace automatu můžeme zadefinovat pomocí zobrazení v množině stavů automatu jako

$$\Phi\colon S \to S, \quad \Phi(s_1) = s_2 \iff \forall c \in C\colon \rho(c) = (c_1, \dots c_k) \wedge \\ \wedge\, s_2(c) = \sigma(s_1(c_1), \dots s_1(c_k)),$$

simulaci n generací (nazývejme ji *dopředným během* automatu) například rekurzivně jako

$$\Phi^n\colon S \to S, \quad \Phi^n(s) = \begin{cases} s, & \text{je-li } n = 0, \\ \Phi(\Phi^{n-1}(s)), & \text{je-li } n \in \mathbb{N}. \end{cases}$$

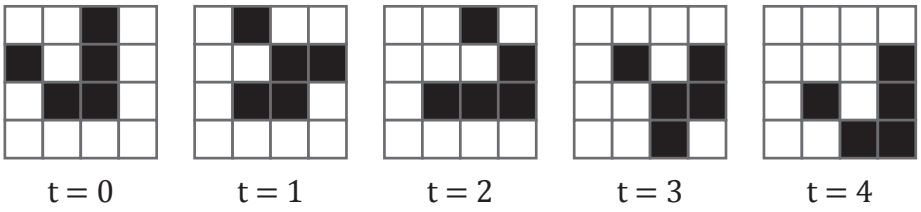
Obdobně lze zadefinovat i *zpětný běh*[1] automatu jako

$$\Psi\colon S \to \mathcal{P}(S), \quad \Psi(s) = \{t \in S \mid \Phi(t) = s\}, \text{ a} \\ \Psi^n\colon S \to \mathcal{P}(S), \quad \Psi^n(s) = \{t \in S \mid \Phi^n(t) = s\}.$$

Populárním příkladem celulárního automatu je *Life* (též známý jako *Game of Life*, česky *Hra života*), vytvořený Johnem Conwayem. Jedná se o dvojrozměrný dvoustavový celulární automat, kde možné stavy buňky jsou živá (1) či mrtvá (0). Přechodová funkce byla dle Gardnera (Gardner, 1970) navržena tak, aby výsledné chování automatu bylo pokud možno nepředvídatelné, avšak připomínalo vývoj společenství mikroorganismů: buňka je v generaci t naživu, pokud v generaci $t-1$ naživu nebyla, avšak měla tři sousední živé buňky (zrození nového mikroorganismu), nebo v generaci $t-1$ naživu byla a měla dvě či tři sousední živé buňky (přežití mikroorganismu díky vyhovující hustotě populace); jinak je buňka v generaci t mrtvá (smrt mikroorganismu v důsledku lokálního přemnožení či vymírání populace). Toto pravidlo je někdy zkráceně označováno jako B3/S23.

Přes zdánlivou jednoduchost vykazuje *Life* mnohé zajímavé vlastnosti, např. je turingovsky úplný (viz Rendell, 2002), byly v něm objeveny sebereplikující se

[1] Narozdíl od deterministického dopředného běhu se v tomto případě výpočet chová nedeterministicky, zobrazujeme tedy stav na množinu stavů.



Obrázek 2.1: Pět po sobě jdoucích generací *Game of Life*. Tato konfigurace je známá jako glider.

konfigurace (viz Aron, 2013) apod. Vznikly i práce, jejichž cílem bylo využít *Life* či jiný dvojrozměrný celulární automat k bezeztrátové kompresi dat, setkávaly se však se střídavými úspěchy (jmenujme například Lafe, 1996; Khan a kol., 1999; Haukeli, 2012). Z důvodu tohoto pokrytí se jimi tedy práce dále nezabývá.

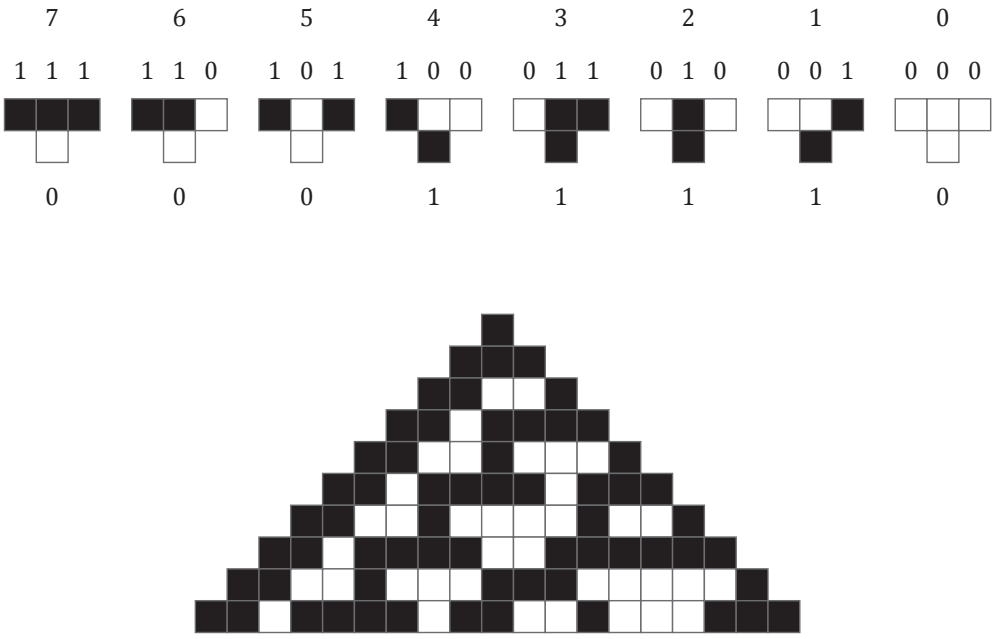
2.3 Elementární celulární automaty

Jednorozměrný dvoustavový celulární automat, nazývaný též elementární celulární automat, je nejjednodušší varianta celulárního automatu. Může být chápán jako lineární pole buněk nabývajících dvou možných stavů, opět vykládaných jako živá (1) či mrtvá (0). Sousedícími buňkami se v tomto případě rozumí dvě buňky obklopující danou buňku zleva, resp. zprava.[2] Elementární celulární automaty jsou dnes používány mimo jiné v oblasti generování náhodných čísel, bylo též navrhováno jejich užití v oblasti kryptografie (viz Wolfram, 2002, Kapitoly 7 a 10).

Stav buňky v generaci t závisí na stavu tří buněk v generaci $t-1$. Pro každou možnou trojici platí, že výsledkem může být živá či mrtvá buňka. Existuje tedy $2^{2^3} = 256$ různých přechodových funkcí. Pro klasifikaci elementárních celulárních automatů se obvykle používá systém, který navrhl Wolfram (2002, Kapitola 3): všem možným trojicím buněk jsou přidělena čísla 0 až 7 podle jejich binární reprezentace (tedy např. konfigurace „mrtvá, živá, živá“ má přiděleno číslo $011_2 = 3_{10}$), tato čísla jsou poté chápána jako pozice bitů zprava v osmibitovém čísle a bit na uvedené pozici je nastaven na 1 právě tehdy, pokud výsledkem dané trojice buněk v následující generaci je živá buňka. Každé přechodové funkci elementárního CA je tak vzájemně jednoznačně přiřazeno číslo z množiny $\{0 \dots 255\}$, daná funkce se pak nazývá pomocí tohoto $r \in \{0 \dots 255\}$ *Wolframovo pravidlo* r.

Pokud není specifikováno jinak, probíhá simulace elementárního celulárního automatu na nekonečném poli buněk, a to i za předpokladu, že vstupní konfigurace, předaná automatu v generaci 0, je konečná. V tom případě ovšem paměťové nároky na reprezentaci stavu automatu mohou růst až lineárně v závislosti na počtu

[2] Dle výše uvedené formální definice CA je tedy množina stavů $\Sigma = \{0, 1\}$, množina souřadnic například $\mathbb{Z}$ a funkce sousednosti ρ: $z \mapsto (z-1, z, z+1)$, kde $z \in \mathbb{Z}$.



Obrázek 2.2: Grafické znázornění Wolframova pravidla 30 a prvních deseti generací jeho simulace z počátečního stavu s jedinou živou buňkou.

simulovaných generací. V této práci tedy uvažujeme pouze automaty operující na konečném poli buněk s pevnou, předem určenou velikostí. Podle způsobu nakládání s krajními buňkami rozlišujeme mezi dvěma typy konečných elementárních celulárních automatů:

- cyklické (*Periodic boundary*) — za levého souseda první buňky je považována buňka poslední a za pravého souseda poslední buňky buňka první, a
- acyklické (*Null boundary*) — za levého souseda první buňky a pravého souseda buňky poslední je považována buňka s konstantní hodnotou 0.

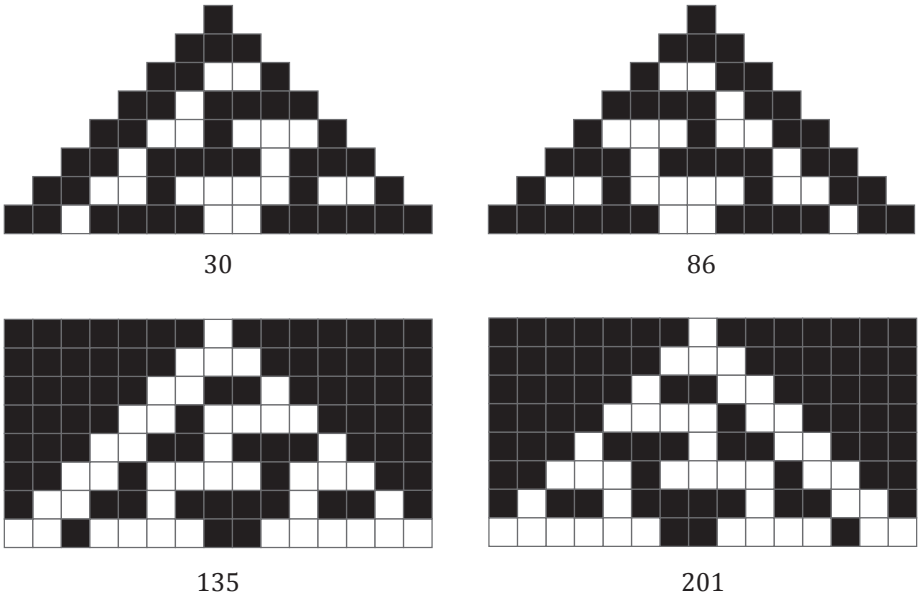
Pro zjednodušení si zaveďme následující značení:

- $S_n = \{0,1\}^n, n \in \mathbb{N}$ je množina všech stavů automatu délky n,
- $s = s_{[1]}s_{[2]}\ldots s_{[n]} \in S_n$ je stav automatu (binární řetězec),
- $\mathrm{Rev}\colon S_n \to S_n, \mathrm{Rev}(s) = s_{[n]}s_{[n-1]}\ldots s_{[1]}$ je stranově převrácený stav s,
- $\mathrm{Inv}\colon S_n \to S_n, \mathrm{Inv}(s) = \neg s_{[1]}\neg s_{[2]}\ldots\neg s_{[n]}$ je bitově inverzní stav s,
- $R = \{0\ldots255\} \times \{a, c\}$ je množina všech pravidel s oběma variantami zakončení (tj. acyklická či cyklická),
- $\mathrm{W}_r\colon S_n \to S_n, r \in R$ je zobrazení stavu na jeho následníka v automatu s pravidlem r, a
- $\mathrm{W}_r^d\colon S_n \to S_n, r \in R, d \in \mathbb{N}_0$ je rekurzivní aplikace zobrazení W_r hloubky d (pro $d = 0$ můžeme zobrazení dodefinovat jako identitu).

Přestože všech 256 Wolframových pravidel je navzájem odlišných, můžeme mezi nimi zavést třídy ekvivalence. Dvě pravidla r_1 a r_2 považujeme za ekvivalentní, pokud platí alespoň jedna z následujících podmínek:

- $\forall s \in S_n : \mathrm{W}_{r_1}(s) = \mathrm{W}_{r_2}(s)$,
- $\forall s \in S_n : \mathrm{Rev}(\mathrm{W}_{r_1}(s)) = \mathrm{W}_{r_2}(\mathrm{Rev}(s))$,
- $\forall s \in S_n : \mathrm{Inv}(\mathrm{W}_{r_1}(s)) = \mathrm{W}_{r_2}(\mathrm{Inv}(s))$, nebo
- $\forall s \in S_n : \mathrm{Rev} \circ \mathrm{Inv}(\mathrm{W}_{r_1}(s)) = \mathrm{W}_{r_2}(\mathrm{Rev} \circ \mathrm{Inv}(s))$.[3]

Zjednodušeně, dvě ekvivalentní pravidla mohou být identická (tj. jedná se o totožné pravidlo), stranově převrácená, bitově inverzní, či vzniklá libovolným složením uvedených vlastností. Pravidla v rámci jedné třídy ekvivalence vykazují obdobné chování. Třídu ekvivalence určenou Wolframovým pravidlem x nazývejme *rodinou pravidla* x.



Obrázek 2.3: Grafické znázornění prvních osmi generací pravidel z rodiny pravidla 30 s ekvivalentními počátečními stavy.

Mezi zajímavá pravidla patří např. Wolframovo pravidlo 30. Bylo dokázáno, že z počátečního stavu s jedinou živou buňkou splňuje posloupnost, vzniklá zřetězením hodnot této buňky v generacích $0, 1, 2, \ldots$, standardní testy na náhodnost (na obrázku 2.2 se jedná o posloupnost stavů v prostředním sloupci). Pravidlo je tedy používáno jako generátor náhodných čísel v programu *Mathematica* (viz Wolfram, 2002, Kapitola 7).

Dalším významným pravidlem je Wolframovo pravidlo 110, u kterého Cook (2004) prokázal, že je (podobně jako Life) turingovsky úplné a tedy schopné simulovat libovolný algoritmus.

[3]Složení funkcí Rev a Inv je zjevně komutativní, nemusíme tedy zohledňovat opačnou variantu.

Každý stav libovolného elementárního CA má právě jednoho přímého následníka. Pro každý konečný elementární CA délky n existuje celkem 2^n různých stavů, na rozdíl od nekonečných elementárních CA tedy nejpozději po 2^n generacích automat nutně narazí na cyklus — navštívením stavu, který byl již navštíven v některé z předchozích generací.

Při zpětném běhu automatu je ovšem situace jiná. Stav může obecně mít 0 až 2^n přímých předchůdců, každý stav je však předchůdcem právě jednoho stavu, v průměru má tedy každý stav též jednoho přímého předchůdce.[4] Z nejednoznačnosti předchůdce vyplývá, že výpočet automatu se při zpětném běhu může větvit, každá větev však nakonec skončí stavem bez předchůdce či nalezením cyklu. Jednoduchou úvahou lze navíc ověřit, že nalezením cyklu může skončit nejvýše jedna větev výpočtu.[5]

[4]Například pro Wolframovo pravidlo 0 je předchůdcem stavu bez živé buňky libovolný z 2^n stavů, naopak žádný stav s alespoň jednou živou buňkou předchůdce nemá. Stavy, které nemají žádné předchůdce, jsou v anglické literatuře zabývající se celulárními automaty často označovány jako *Gardens of Eden*.

[5]Sporem: cyklus je při zpětném běhu vždy poprvé detekován nalezením výchozího stavu. Pokud by byl poprvé znovunavštíven jiný než výchozí stav, pak by při dopředném běhu z tohoto stavu nikdy nebylo dosaženo stavu výchozího. Pokud by nalezením cyklu končilo více větví výpočtu, pak by tedy byl výchozí stav poprvé znovunavštíven ve dvou různých větvích a tím pádem by byl součástí dvou různých cyklů. Při dopředném běhu se ovšem výpočet nevětví, takže každý stav může být součástí nejvýše jednoho cyklu.

3. Návrh

Tato kapitola představí námi navržený způsob, jakým lze využít vlastnosti elementárních celulárních automatů k bezeztrátové kompresi dat. Následuje výčet implementovaných kompresních či pomocných algoritmů, které byly využity v procesu určení optimálních parametrů tohoto návrhu, v jeho implementaci či pro srovnání jeho výsledků s obvyklými kompresními algoritmy.

Elementární celulární automat konečné velikosti sám o sobě nemá žádné předpoklady ke kompresi dat. V každé další generaci se mění pouze samotná data reprezentovaná stavem automatu, nikoliv jejich délka. Rozumným použitím elementárního CA namísto pokusu o samotnou kompresi je tedy snaha transformovat data do podoby, v níž se stanou lépe komprimovatelnými jinými kompresními algoritmy. Samotný postup použití by měl být následovný:

1. převést vstupní data do binární reprezentace,
2. získanou bitovou sekvenci použít jako počáteční stav elementárního CA,
3. simulovat běh CA do nalezení stavu s lepšími předpoklady pro kompresi, a
4. použít zvolený kompresní algoritmus na bitovou sekvenci získanou z tohoto stavu.

K implementaci tohoto postupu je však třeba vyřešit několik obtíží. Za prvé, spolu se získanou bitovou sekvencí je třeba do výstupních dat zaznamenat i generaci, v jaké byl odpovídající stav získán. Bez této informace by totiž nebylo možné při dekompresi zpětně rekonstruovat počáteční stav. Je nutné počítat s tím, že kvůli zápisu generace jsou data po transformaci větší než data původní.

Za druhé, pro celulární automaty platí, že každý stav má právě jednoho následníka. Neplatí ovšem, že každý stav má právě jednoho předchůdce. Existují takové konfigurace, které mají přímých předchůdců několik, stejně tak ovšem existují konfigurace, které žádného předchůdce nemají, a lze je tedy získat pouze jako počáteční stav. Pokud tedy budeme při kompresi simulovat dopředný běh automatu, nemusíme být při dekompresi schopni jednoznačně určit původní stav automatu ani se znalostí počtu simulovaných generací.

Jedním možným řešením tohoto problému je spolu se samotnými daty a počtem simulovaných generací zapsat i informaci o tom, který ze stavů byl původní. To lze provést buď setříděním potenciálních kandidátů pomocí pevně definovaného uspořádání, jejich očíslováním a zápisem čísla správného výchozího stavu, nebo zápisem speciální řídící sekvence, která umožní při každém větvení výpočtu rovnou zvolit správnou větev. Oba dva přístupy však sdílí stejné nevýhody: nutnost simulovat při kompresi jak dopředný, tak i zpětný běh automatu (kvůli detekci větvení výpočtu), a nové režijní náklady vznikající zápisem dodatečných dat (a tedy pravděpodobně i zhoršení kompresního poměru).

Elegantnějším řešením pak je provést simulaci opačně. Při kompresi budeme simulovat zpětný běh automatu, v kterémžto případě nepůsobí problém ani více předchůdců jednoho stavu a tedy více možných větví výpočtu, ani stav bez

předchůdce a tedy předčasné ukončení výpočtu. Při dekompresi budeme naopak simulovat běh dopředný, při kterém se výpočet nevětví a prostý počet generací stačí k získání počátečního stavu. Jak je vidno z obrázku 2.2, aplikací tohoto postupu například na zdánlivě nahodilou devatenáctibitovou sekvenci `1101111011001000111` bychom mohli pomocí Wolframova pravidla 30 získat po simulaci devíti generací zpět sekvenci `0000000001000000000`. Pro úspěšnou kompresi pak již stačí kódovat tuto sekvenci spolu s číslem 9 pomocí méně než devatenácti bitů. Existuje ovšem riziko, že pro počáteční stavy z reálných dat nebude k dispozici dostatečný počet předchozích stavů či tyto nebudou snáze komprimovatelné než původní data. Toto riziko je nutné minimalizovat vhodnou volbou parametrů celulárního automatu.

Za třetí, s rostoucí velikostí automatu roste jak paměťová, tak především časová složitost algoritmu. Z tohoto důvodu bychom měli volit takové parametry prohledávání stavového prostoru, aby složitost byla stále únosná. Nejjednodušším způsobem zachování přijatelné složitosti je provádět transformaci dat po částech, tj. nejprve vstupní data rozdělit do bitových sekvencí pevné délky, a poté transformovat každou zvlášť. Stinnou stránkou tohoto přístupu je nutnost zapisovat generaci pro každou sekvenci a z toho plynoucí zvyšování paměťové režie při zmenšování těchto sekvencí. Je však pravděpodobné, že reprezentace zapisované generace bude v průměru též kratší (např. vlivem předčasného ukončení výpočtu či dřívějším nalezením cyklu) a paměťová režie zůstane zachována.

Za čtvrté, musíme být schopni určit, který stav má lepší předpoklady pro kompresi. Předpokládejme, že pro každý kompresní algoritmus mohou být požadavky na optimální formu značně odlišné, je tedy žádoucí implementovat ohodnocovací funkci, jejíž kritéria budou přímo souviset s konkrétními kompresními algoritmy. Ohodnocovací funkce tedy podle zvoleného kritéria vrátí ohodnocení neboli *fitness* daného stavu.

3.1 Pomocné algoritmy

Následující algoritmy byly v našem výzkumu implementovány jako podpůrné algoritmy pro využití v simulaci celulárních automatů či v některém z implementovaných kompresních algoritmů.

3.1.1 Gama kód

Gama kód, též známý jako Eliasův Gama kód podle svého autora, Petera Eliase, je univerzální kód kódující přirozená čísla do jednoznačných bitových posloupností variabilní délky. Kódujeme-li čísla standardním způsobem pomocí fixního počtu bitů, musíme předem znát rozsah, z jakého kódovaná čísla pocházejí. Gama kód je navržen tak, že menším číslům přiřazuje kratší bitové posloupnosti a větším číslům delší, sekvenčním čtením lze navíc vždy určit, kdy končí zápis jednoho čísla a začíná zápis dalšího.

Formálně se jedná o zobrazení $\Gamma\colon \mathbb{N} \to \{0,1\}^*$. Princip kódu spočívá v zápisu

čísla minimálním možným počtem bitů (tj. zápis začíná nejlevější jedničkou v binárním zápisu čísla) a následném předřazení tolika nulových bitů, jaký je počet bitů čísla následujících po první jedničce.

Gama kód je obzvlášť výhodný, pokud zapisujeme častěji malé hodnoty než velké. Přičtením jedničky před samotným kódováním a naopak odečtením jedničky po dekódování lze navíc kód rozšířit o zápis nuly na $\Gamma_0 \colon \mathbb{N}_0 \to \{0,1\}^*$.

$$\Gamma_0(0) = \Gamma(1) = 1$$
$$\Gamma_0(12) = \Gamma(13) = \overbrace{000}^{n}\,1\,\overbrace{101}^{n}$$
$$\Gamma_0(136) = \Gamma(137) = \overbrace{0000000}^{n}\,1\,\overbrace{0001001}^{n}$$

Obrázek 3.1: Příklad zápisu nezáporných celých čísel pomocí Gama kódu.

3.1.2 Omega kód

Podobně jako Gama kód kóduje Eliasův Omega kód přirozená čísla do jednoznačných bitových posloupností variabilní délky. Na rozdíl od něj však funguje rekurzivně.

Formálně se jedná o zobrazení $\Omega \colon \mathbb{N} \to \{0,1\}^*$. Implicitní výchozí hodnota n Omega kódu (tj. před samotnou interpretací sekvence bitů) je jedna. Pokud v zápisu čísla následuje nulový bit, je za výslednou hodnotu považováno n. V opačném případě je přečteno následujících $n+1$ bitů a interpretováno jako nové n. Tato procedura se opakuje do prvního přečtení nulového bitu následujícího po přečtené sekvenci bitů.

Oproti Gama kódu prodlužuje Omega kód zápis menších čísel, avšak výrazně zkracuje zápis čísel větších. Stejnou modifikací jako v případě Gama kódu ho též lze rozšířit z přirozených na nezáporná celá čísla, tedy $\Omega_0 \colon \mathbb{N}_0 \to \{0,1\}^*$.

$$\Omega_0(0) = \Omega(1) = 0$$
$$\Omega_0(12) = \Omega(13) = \overbrace{11}^{3}\,\overbrace{1\,\underbrace{101}_{3}}^{13}\,0$$
$$\Omega_0(136) = \Omega(137) = \overbrace{10}^{2}\,\overbrace{1\,\underbrace{11}_{2}}^{7}\,\overbrace{1\,\underbrace{0001001}_{7}}^{137}\,0$$

Obrázek 3.2: Příklad zápisu nezáporných celých čísel pomocí Omega kódu.

3.1.3 Burrows-Wheelerova transformace

Burrows-Wheelerova transformace je oblíbený algoritmus pro zlepšení komprimovatelnosti textových dat. Vychází z pozorování, že v přirozených jazycích se některé sekvence znaků vyskytují výrazně častěji než jiné. Pomocí lexikografického uspořádání lze pak znaky vstupního řetězce permutovat tak, že:

1. vznikají časté skupiny po sobě jdoucích stejných znaků, a
2. využitím stejného lexikografického uspořádání lze rekonstruovat původní řetězec.

Produktem Burrows-Wheelerovy transformace řetězce je tedy řetězec o stejné délce (resp. větší, je totiž nutno uložit i speciální ukazatel symbolizující začátek původního řetězce), který lze efektivně komprimovat např. pomocí Move-to-front či Run-length kódování(viz Burrows a Wheeler, 1994).

3.1.4 Trace and backtrack

Pro vytvoření co nejefektivnějšího algoritmu založeného na simulaci elementárního celulárního automatu musíme být schopni efektivně simulovat jak dopředný, tak i zpětný běh automatu. Dopředný běh lze implementovat lineárním procházením trojic sousedících buněk a zápisem výsledných hodnot získaných zvolenou přechodovou funkcí. Toto řešení je sice naivní, pro naše potřeby však dostačující.

Hledání stavů předchozích nelze implementovat takto přímočaře. Některé stavy mohou mít předchůdců více, jiné nemusí mít žádného. Naším požadavkem je nalézt všechny přímé předchůdce. Naivní řešení hrubou silou, tj. generováním všech možných stavů a následným filtrováním podle toho, zda je jejich následujícím stavem stav výchozí, je nepřijatelné — časová složitost je totiž vzhledem k velikosti automatu exponenciální.

Pro hledání předchozích stavů (v anglické literatuře *preimages*) byl tedy implementován algoritmus Trace and backtrack (viz Jeras a Dobnikar, 2007), využívající de Bruijnův graf a jeho rozšíření, tzv. *preimage network*. Tento algoritmus nalezne a vrátí všechny přímé předchůdce výchozího stavu v daném elementárním celulárním automatu.

Průměrná časová složitost algoritmu je lineární v závislosti na velikosti automatu a počtu předchůdců výchozího stavu. Jak jsme však již výše naznačili, v průměru má každý stav právě jednoho předchůdce, závislost na počtu předchůdců výchozího stavu můžeme tedy v případě, že algoritmu dodáváme náhodné stavy, vynechat.

3.1.5 HashLife

Algoritmus HashLife vyvinul Gosper (1984) pro zrychlenou simulaci dvojrozměrných celulárních automatů, avšak lehkou modifikací lze docílit jeho využití i pro elementární celulární automaty.

Algoritmus funguje na principu cachování částí již navštívených stavů automatu a získávání následujících stavů jejich kombinací. Zvláštností algoritmu je, že díky tomuto cachování je schopen provádět simulaci celulárního automatu po 2^n generacích zároveň namísto jedné, navíc se s pokračující simulací velikost kroku zvětšuje. Nevýhodou algoritmu naopak jsou jeho paměťové nároky. Zatímco u dvojrozměrných automatů typu *Life* existuje velké množství konfigurací, které jsou prostorově omezené a/nebo obsahují velké plochy mrtvých buněk, které mohou být ignorovány, elementární celulární automaty (zvláště pak ty, kterými se zabýváme především) naopak často rostou nade všechny meze, jejich konfigurace bývá chaotická a paměťové nároky algoritmu záhy začínají být neúnosné. Pro náš přístup je navíc stěžejní schopnost simulace automatu v obou směrech a tedy získávání jak následujících, tak předchozích stavů, pro tuto funkcionalitu však HashLife nemá předpoklady. Algoritmus je tedy využit pouze ve vizualizaci automatu pro srovnání s naivním přístupem.

3.2 Kompresní algoritmy

Pro potřeby našeho výzkumu nyní představíme či navrhneme dále použité kompresní algoritmy. Většina z nich je navržena či upravena tak, aby operovala nad jednotlivými bity místo obvyklejších bajtů. Důvodem je, že nad bity operují i elementární celulární automaty, zatímco bajtovou strukturu dat zachovávat nemusejí.

3.2.1 RLE Bit

Algoritmus RLE Bit, neboli bitová RLE komprese, vychází z jednoduchého, avšak účinného algoritmu známého jako *Run-length encoding* (RLE). Ten funguje na principu zkracování dlouhých sekvencí opakujících se znaků jediným zápisem tohoto znaku spolu s počtem jeho opakování. To bývá užitečné například pro bitmapovou grafiku, kde se podobné sekvence vyskytují často. Naopak pro textová data ve výchozí podobě algoritmus příliš účinný není. Existují však transformace (například již popisovaná Burrows-Wheelerova transformace), které jsou schopny text pro potřeby RLE upravit.

Algoritmus RLE Bit zapíše první bit komprimovaných dat a následně pro každou souvislou sekvenci stejných bitů zapíše její délku pomocí Gama kódu. Oproti klasické RLE kompresi, která na výstup zapisuje dvojice (znak, počet), bitové RLE kompresi ke stejným účelům postačí zapsat počet opakování — je totiž zřejmé, že po sekvenci nulových bitů musí následovat sekvence jedničkových bitů a naopak.

$$\overbrace{111}^{3}\overbrace{000000}^{6}\overbrace{1111111}^{7} = 1\overbrace{011}^{\Gamma(3)}\overbrace{00110}^{\Gamma(6)}\overbrace{00111}^{\Gamma(7)}$$

Obrázek 3.3: Příklad bitové RLE komprese na šestnáctibitovém řetězci.

3.2.2 Sparse Bit

Algoritmus Sparse Bit, neboli bitová Sparse komprese, byl navržen pro kompresi tzv. řídkých dat, tedy takových, která obsahují velké množství nevyužitého prostoru. To se v praxi projevuje tak, že v nich výrazně převažují nulové bajty nad nenulovými, či v našem případě nulové bity nad jedničkovými.

Komprese Sparse Bit prochází komprimovaná data po jednotlivých bitech a pomocí Gama kódu zapisuje počty nulových bitů mezi dvěma nejbližšími jedničkovými bity či jedničkovým bitem a „okrajem“ (tedy začátkem či koncem) dat (včetně nulového počtu nul v konfiguraci `11`). Data neobsahující žádné jedničkové bity jsou tedy zapsána jako prostý počet bitů těchto dat v Gama kódu.

$$1\overbrace{000000}^{6}11\overbrace{00000}^{5}1\overbrace{0}^{1} = \overbrace{1}^{\Gamma_0(0)}\overbrace{00111}^{\Gamma_0(6)}\overbrace{1}^{\Gamma_0(0)}\overbrace{00110}^{\Gamma_0(5)}\overbrace{010}^{\Gamma_0(1)}$$

Obrázek 3.4: Příklad bitové Sparse komprese na šestnáctibitovém řetězci.

3.2.3 LZ77

Kompresní algoritmus LZ77 je jeden ze dvou nejznámějších slovníkových algoritmů pro bezeztrátovou kompresi dat. I přes jeho stáří (navržen byl v roce 1977) se používá dodnes, např. jako komponenta široce užívaného algoritmu DEFLATE (viz např. Sayood, 2012, Kapitola 5.4.1).

Algoritmus funguje na principu posuvného okna („sliding window“), rozděleného na dvě části: prohlížecí okno („search buffer“, „back buffer“) a aktuální okno („look-ahead buffer“, „front buffer“). Na začátku běhu algoritmu je ze vstupu naplněno aktuální okno, prohlížecí okno zůstává prázdné. Při každé iteraci se algoritmus pokusí nalézt v prohlížecím okně co nejdelší prefix řetězce v aktuálním okně. Poté na výstup zapíše trojici (p, l, b), kde p značí pozici začátku řetězce v prohlížecím okně, l značí délku nalezeného řetězce (potenciálně nulovou) a b představuje bajt následující v aktuálním okně za nalezeným řetězcem. Na konci iterace algoritmus přesune zapsaný řetězec s přidaným bajtem do prohlížecího okna a doplní aktuální okno ze vstupu. Po vyprázdnění aktuálního okna algoritmus končí. Velikost prohlížecího a aktuálního okna je variabilní, může být dána parametrem. Platí však, že prohlížecí okno musí být ostře větší než okno aktuální.

V naší implementaci se pro zápis pozice nalezeného řetězce fixně využívá x bitů (x je parametrem algoritmu), maximální délka prohlížecího okna je tedy $2^x - 1$, větší rozsah by pomocí x bitů nebylo možno adresovat. Obdobně, délka nalezeného řetězce se zapisuje fixně pomocí y bitů (y je též parametrem), délka aktuálního okna je však 2^y, což zaručuje, že kromě hledaného prefixu maximální délky $2^y - 1$ bude okno obsahovat i následující bit pro zapsání. Celková délka posuvného okna je tedy v závislosti na parametrech $2^x + 2^y - 1$.

3.2.4 LZ77 Bit

Algoritmus LZ77 Bit je pouhou modifikací výše popsaného LZ77 algoritmu. Data jsou zde chápána jako posloupnost bitů, nikoliv bajtů či znaků. Z důvodu efektivnější implementace bylo též zavedeno omezení parametru y, který nyní nesmí přesáhnout hodnotu 6, maximální rozsah aktuálního okna je tedy $2^6 = 64$ bitů. To nám umožňuje vyhledávat řetězec z aktuálního okna pomocí bitových operací exkluzivní disjunkce (XOR) a posuvu (shift).

3.2.5 LZ78

Spolu s algoritmem LZ77 je LZ78 dalším populárním slovníkovým kompresním algoritmem. Na rozdíl od něj nedisponuje LZ78 posuvným oknem, namísto toho konstruuje explicitní slovník (viz Sayood, 2012, Kapitola 5.4.2).

Na začátku běhu algoritmu je slovník prázdný, resp. obsahuje implicitní záznam pro prázdné slovo. Následně algoritmus po jednotlivých bajtech načítá data ze vstupu do bufferu, dokud se načtený řetězec nachází ve slovníku. Při načtení znaku, po jehož přidání do bufferu by se daný řetězec již ve slovníku nenacházel, je na výstup zapsána dvojice (i, b), kde i značí index řetězce ve slovníku zapsaný pomocí Gama kódu a b je poslední načtený bajt. Na konci iterace je řetězec prodloužený o nový bajt přidán do slovníku a buffer vyprázdněn. Algoritmus končí ve chvíli, kdy na vstupu nezbývají další znaky a buffer je prázdný. Pokud buffer po dosažení konce vstupních dat prázdný není, je na výstup zapsán jako poslední index řetězce v bufferu (již bez následujícího znaku).

3.2.6 LZ78 Bit

Opět se jedná o modifikaci již uvedeného algoritmu LZ78 operující nad bity místo bajtů. Jediný rozdíl pak spočívá v ukončení běhu algoritmu. Pokud v bajtové variantě po přečtení celého vstupu zbývají v bufferu znaky, je na výstup zapsán pouze index tohoto řetězce ve slovníku. Dekompresní algoritmus tuto situaci rozpozná tak, že po zapsaném indexu nenásleduje ve zkomprimovaném souboru celý bajt. Pokud však operujeme s jednotlivými bity, může nastat varianta, kdy po posledním zapsaném indexu zbývají ve výstupním souboru nulové bity bez významu (kvůli zaokrouhlení velikosti souboru na celé bajty) a dekompresní algoritmus není schopen určit, zda se nejedná o další bity původního vstupu. Tuto eventualitu můžeme řešit například zapsáním indexu řetězce o jeden bit kratšího a následně jeho posledního bitu. Poté již algoritmus spolehlivě pozná konec zkomprimovaného souboru podle nepřítomnosti dalšího indexu.

3.3 Fitness kritéria

V této podkapitole představíme takzvaná *fitness kritéria* pro hodnocení stavů celulárních automatů.[1] Pomocí těchto kritérií budeme hodnotit stavy získané simulací celulárního automatu z počátečního stavu na základě jejich vhodnosti k bezeztrátové kompresi. Nejlépe ohodnocený stav poté budeme považovat za výsledek transformace celulárním automatem. Kritéria pro výpočet vhodnosti daného stavu musí být dostatečně komplexní, aby dokázala přijatelně aproximovat komprimovatelnost dat v závislosti na zvoleném kompresním algoritmu.

Vzhledem k zamýšlenému využití pro hodnocení všech simulovaných stavů je výpočet fitness jednou z nejfrekventovaněji volaných procedur. Při transformaci souboru o velikosti $V = 1\,\mathrm{MB}$ po blocích o velikosti $v = 1024\,\mathrm{b}$ s prohledáváním $d = 10\,000$ generací zpět bude počet volání výpočtu fitness průměrně přes všechny vstupní soubory dané velikosti $\frac{V}{v} \times d = 81\,920\,000$. Kromě efektivního výpočtu fitness je tedy žádoucí volit takové parametry, abychom počet jeho volání pokud možno minimalizovali (například volbou pravidla, které brzy dosahuje dobře ohodnocených stavů, a omezením hloubky prohledávání), či implementovat vhodnou heuristiku místo úplného prohledávání.

Většina dat, se kterými se setkáváme, je bajtově orientovaná, a pro tuto jejich strukturu je tedy navržena i většina kompresních algoritmů. Elementární celulární automaty však operují nad jednotlivými bity a bajtovou strukturu vstupních dat nezachovávají. Implementovaná fitness kritéria vycházející z konvenčních kompresních algoritmů by tedy zpravidla měla zohledňovat spíše jejich bitově orientované modifikace; využívání jejich původní, bajtové specializace pro data transformovaná pomocí elementárních CA postrádá smysl. Jiným možným přístupem je namísto elementárního CA využít jednorozměrný CA o 256 stavech a tedy operující nad bajty namísto bitů, tuto variantu však ponecháváme pro budoucí výzkum.

Všechna popsaná kritéria jsou navržena tak, aby pro libovolný stav elementárního CA vracela hodnotu v rozmezí $[0, 1]$, kde vyšší hodnota značí lepší uspořádanost stavu a tedy větší potenciál pro kompresi.

3.3.1 Obecná kritéria

Fitness kritéria z této skupiny jsou navržena tak, aby jednoduše aproximovala uspořádanost libovolného stavu bez ohledu na generaci, v jaké byl stav získán, či účinnost reálného kompresního algoritmu aplikovaného na tento stav. Tato kritéria jsou implementačně jednodušší, negarantují však, že jimi nalezené stavy skutečně bude možné komprimovat lépe než jiné. Jejich použití je tedy navrhováno převážně pro klasifikaci pravidel celulárních automatů.

[1] Termínu *fitness* se v informatice užívá zpravidla v souvislosti s genetickými algoritmy. V této práci je však užit pouze ve významu „vhodnost“.

Homogeneity

Kritérium Homogeneity hodnotí stav na základě homogenity bitů v jeho binárním zápisu. Stav sestávající ze stejného počtu buněk s hodnotou 0 a 1 dostane nejnižší možné ohodnocení, naopak stav složený z buněk pouze jedné hodnoty dostane ohodnocení nejvyšší. Toto kritérium je vhodné pro kompresní algoritmy pro řídká data, například tedy pro námi navrženou Sparse Bit kompresi.

RunCount

Kritérium RunCount hodnotí stav na základě počtu sledů buněk se stejnou hodnotou, či ekvivalentně na základě počtu konfigurací `01` či `10`. Nejvyšší ohodnocení získá stav složený z buněk pouze jednoho typu a tedy z jediného sledu, nejnižší ohodnocení získá stav, kde se buňky pravidelně střídají a počet sledů je tedy roven počtu buněk. Toto kritérium je výhodné pro kompresi typu Run-length encoding aplikovanou na úrovni bitů, například výše uvedenou RLE Bit kompresi.

Deflate

Kritérium Deflate ohodnocuje stav na základě úspory místa při kompresi daného stavu pomocí stejnojmenného kompresního algoritmu. Využijeme knihovní třídy .NET Frameworku `System.IO.Compression.DeflateStream` s nastavením optimální komprese. Kompresní algoritmus této třídy je bajtově orientovaný, pro jeho aplikaci na bitové sekvence různých délek je tedy žádoucí tyto nejprve převést na odpovídající sekvence bajtů např. hodnot 0 a 1. Zjevnou nevýhodou tohoto postupu je osminásobné zvětšení paměťových nároků stavu ještě před jeho kompresí. Výsledná úspora místa tedy vypovídá o uspořádanosti stavu, avšak nikoliv o jeho skutečných předpokladech ke kompresi pomocí DeflateStream. Výhodou je naopak možnost použití i pro stavy relativně malých délek. Ty by totiž vzhledem k paměťové režii kompresního algoritmu v původním formátu nebyly komprimovatelné a tedy ani ohodnotitelné. Časová složitost kritéria závisí na implementaci DeflateStream.

BiEntropy

Kritérium BiEntropy implementuje algoritmus BiEntropy (viz Croll, 2013), jehož úkolem je aproximovat entropii konečného binárního řetězce. Algoritmus počítá vážený průměr informačních entropií řetězce a jeho binárních derivací.[2]

Vzhledem ke vztahu entropie a komprese dat (řetězce s nižší entropií obsahují více redundantní informace, která může být vypuštěna) má toto kritérium potenciál nalézt stavy vhodné pro většinu standardních kompresních algoritmů.

[2]Poznamenejme, že výpočet využívá téměř totožný mechanismus jako acyklické Wolframovo pravidlo 102.

Nevýhodou kritéria je, že samotný algoritmus, založený na vyhledávání a hodnocení periodicity ve vstupním řetězci, mnohem silněji zohledňuje periody délky $P = 2^n, n \in \mathbb{N}$. Tabulka 3.1 obsahuje fitness stavů o délce 32 buněk, jejichž hodnoty byly vytvořeny periodickým opakováním prefixu délky $n \in \{1 \dots 16\}$ zvoleného řetězce.

Délka prefixu	`1000000000000000`	`1011001110001111`
1	1,000 000	1,000 000
2	0,991 501	0,991 501
3	0,082 441	0,082 441
4	0,962 647	0,962 647
5	0,124 928	0,107 727
6	0,162 429	0,113 279
7	0,100 568	0,092 921
8	0,884 941	0,878 370
9	0,277 265	0,054 870
10	0,355 977	0,051 252
11	0,094 500	0,243 957
12	0,165 687	0,059 335
13	0,099 284	0,069 344
14	0,150 978	0,040 664
15	0,182 199	0,056 524
16	0,713 723	0,736 754

Pozn: Vzhledem k originálnímu návrhu algoritmu BiEntropy (kde vyšší hodnota značí vyšší neuspořádanost) a využití algoritmu jako fitness funkce (kde vyšší hodnota naopak indikuje vhodnější konfiguraci) jsou uvedené hodnoty převráceny pomocí funkce $f(x) = 1 - x$.

Tabulka 3.1: Ohodnocení stavů vzniklých periodickým opakováním prefixu dané délky zvoleného řetězce kritériem BiEntropy.

Tento nedostatek neznemožňuje využití kritéria pro hledání stavů výhodných pro kompresi, naznačuje ovšem, že velké množství stavů s periodou „nevhodné“ délky bude kritériem ignorováno i přes jejich extrémní vhodnost. Časová složitost kritéria je navíc kvadratická, možnost jeho využití pro delší stavy ve větší míře je tedy značně omezená.

3.3.2 Kritéria implementující kompresi

Oproti obecným kritériím sestává tato skupina z kritérií, která při hodnocení stavu CA provádí skutečnou kompresi některým z navržených kompresních algoritmů, včetně zápisu generace, v jaké byl stav získán. Ohodnocení stavu libovolným z těchto kritérií odpovídá úspoře místa, které by bylo dosaženo aplikací zvoleného kompresního algoritmu na daný stav spolu se zápisem generace pomocí Omega kódu. V případě, že k reálné úspoře místa nedojde (neboť je výsledný zápis stavu stejně velký jako stav samotný, či dokonce větší), vrací kritéria shodně nulovou hodnotu bez ohledu na míru zvětšení zápisu. Tato kritéria již zaručují,

že jimi nejlépe ohodnocený stav je ze všech hodnocených stavů nejlépe komprimovatelný zvolenou metodou.

SparseCompression

Kritérium SparseCompression zpřesňuje kritérium Homogeneity, oproti němu zohledňuje skutečnou komprimovatelnost stavu kompresním algoritmem Sparse Bit včetně zápisu generace.

Na rozdíl od kritéria Homogeneity není uvažována varianta s převažujícími živými buňkami a tedy se zápisem počtu jedničkových bitů mezi dvěma nulovými. Pokud by tato varianta v praxi převažovala, můžeme předpokládat, že volbou bitově inverzního pravidla ze stejné rodiny dosáhneme lepších výsledků pro námi zvolenou variantu. Navíc bychom bez tohoto omezení museli komprimovaný zápis prodloužit o bit signalizující, která varianta byla zvolena.

RLECompression

Obdobně jako SparseCompression, kritérium RLECompression zpřesňuje kritérium RunCount aplikací algoritmu RLE Bit na hodnocený stav automatu a zápisem generace stavu.

LZB7Compression

Stejně jako u dvou předchozích kritérií, LZB7Compression implementuje výše uvedený kompresní algoritmus LZ77 Bit, kterým zkomprimuje stav automatu spolu se zápisem jeho generace. Jako jediné z kritérií závisí LZB7Compression na parametrech. Ty určují vlastnosti obsaženého LZ77 Bit algoritmu (viz výše).

4. Experimenty

V této části představíme sadu testovacích dat, navrhneme experimenty, pomocí kterých klasifikujeme elementární celulární automaty podle jejich využitelnosti ke kompresi dat, a uvedeme výsledky těchto experimentů. Na jejich základě pak vybereme nejvhodnější automaty a navrheme další parametry, jejichž užitím vytvoříme kompresní algoritmus založený na transformaci pomocí těchto automatů. Výsledky námi navrženého algoritmu nakonec porovnáme s ostatními kompresními algoritmy.

4.1 Testovací data

Zvolená data určená k testování by měla vhodným způsobem reprezentovat skutečná data, se kterými se můžeme setkat, potýkáme-li se s problémem komprese dat. První skupinou dat, která bychom měli do testování zahrnout, jsou náhodná binární data. Pomocí těch bychom měli být schopni určit některé vlastnosti elementárních CA, přestože skutečné komprese pravděpodobně nedosáhneme (a pokud ano, pak námi zvolená data nebyla dostatečně náhodná). Vzhledem k faktu, že se zabýváme bezeztrátovou kompresí, jsou další přirozenou volbou textová data, pro která byla většina původních bezeztrátových kompresních algoritmů navržena. Nakonec můžeme zahrnout též nekomprimovaná obrazová data, jejichž bezeztrátová komprese je stále žádanější vzhledem k jejich výraznému zastoupení například na webových stránkách.

4.1.1 Náhodná data

Tato práce se zabývá celulárními automaty, pro generátor náhodných dat tedy nemusíme chodit daleko. Využijeme elementární celulární automat s Wolframovým pravidlem 30, jehož prostřední sloupec na cyklickém, avšak dostatečně velkém poli z počátečního stavu s jedinou živou buňkou generuje, jak již bylo zmíněno, náhodnou posloupnost bitů. Z respektu k nejrozšířenější, bajtové reprezentaci dat budeme testovat po sobě následující sekvence délky $8n$ z této posloupnosti.

```
11011100 11000101 10010011 10101110
01110101 01100001 10010101 10101011
11110000 11110001 01011100 00010010
1100...
```

Obrázek 4.1: Prvních 100 bitů posloupnosti generované prostředním sloupcem Wolframova pravidla 30.

4.1.2 Textová data

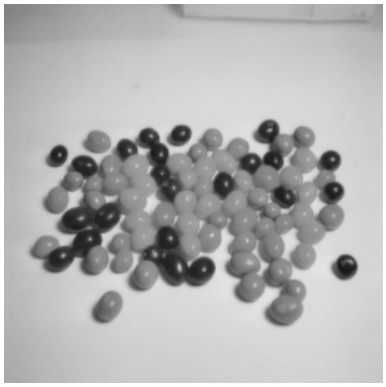
Jako vzorek textových dat použijme standardní pseudolatinský text „Lorem ipsum“, často užívaný pro obdobné účely, převedený do bitové posloupnosti odpovídající kódování UTF-8. Celý text délky 72 391 bajtů (cca 70,7 kB) se nachází v příloze pod názvem `ipsum.txt`. Bude-li zapotřebí kratších úseků, využijeme po sobě následující sekvence o délce 2^n znaků ze začátku tohoto textu.

```
Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Cras sed fermentum arcu.
Nunc euismod justo sodales laoreet blandit...
```

Obrázek 4.2: Počátek testovacího textu „Lorem ipsum“.

4.1.3 Obrazová data

Pro testování komprese na obrazových datech volme takový formát, který žádnou kompresi neimplementuje. Nabízí se jeden z nejjednodušších plnohodnotných formátů, BMP s barevnou hloubkou 24 bitů na pixel. Díky fixnímu počtu bitů nutných pro uložení každého pixelu je velmi snadné spočítat jeho přesnou velikost v závislosti na rozlišení. Zvolme tedy BMP soubor s rozlišením 256×256 pixelů a velikostí 196 662 bajtů (tedy téměř přesně 192 kB — 54 bajtů využívá hlavička souboru). Tento soubor se nachází v příloze pod názvem `jelly.bmp`.[1]



Obrázek 4.3: Testovací bitmapa pro kompresi.

4.2 Délka cyklu

Účelem prvního testu je určit obvyklou délku cyklu daného automatu, tj. počet různých stavů, které automat může navštívit při dopředném běhu z určitého výchozího stavu, než opětovně navštíví již jednou navštívený stav.[2] Vycházíme tak

[1] Zdroj: The USC-SIPI Image Database. *University of Southern California* [online]. 2016 [cit. 2016-07-19]. URL `http://sipi.usc.edu/database/download.php?vol=misc&img=4.1.08`.

[2] Poznamenejme, že při dopředné simulaci cyklus nemusí být uzavřen navštívením výchozího stavu, hovořit o „délce cyklu“ tedy není zcela přesné.

z předpokladu, že pravidla, která se začnou opakovat příliš brzy, pravděpodobně nemají tak velký potenciál k nalezení stavu vhodného pro kompresi, tento předpoklad však nebyl nijak prokázán. Rozumnějším předpokladem je hledat tímto způsobem pravidla, která se chovají „dostatečně náhodně“[3] a tedy nalézají větší množství různých stavů. Například pro Wolframovo pravidlo 0 je délka cyklu z libovolného neprázdného výchozího stavu 2 (výchozí stav a prázdný stav), pravidlo je tedy triviální.

Vzhledem k rychle rostoucím délkám cyklu některých pravidel test provedeme pro všechna pravidla na náhodných posloupnostech délek 16, 24 a 32 bitů a na textu o délce 4 znaky (32 bitů). Zde testovací data nemusí připomínat data reálná, účelem je pouze základní klasifikace pravidel automatu.

Z výsledků uvedených v tabulce 4.1 vyplývá, že při testování cyklických automatů lze vysledovat tři typy pravidel. Jedním jsou pravidla, která ihned či po konstantním počtu generací skončí nalezením cyklu bez závislosti na velikosti automatu (např. rodina pravidla 0 či 36). Druhým typem jsou pravidla, jejichž délka cyklu je řádově nízká, avšak roste lineárně s velikostí automatu (příkladem je rodina pravidel 2 nebo 3).[4] Nejzajímavějším typem pravidel z tohoto hlediska jsou rodiny pravidel 30, 45 a 106, jejichž průměrná délka cyklu roste až exponenciálně s velikostí automatu.

U acyklických automatů se obecně setkáváme s kratšími cykly (informace na okrajích automatu se ztrácí). Neexistuje zde tedy tak rapidní nárůst délky cyklu jako u cyklických automatů, na rozdíl od nich však můžeme pozorovat jiný fenomén: některá pravidla nalézají výrazně delší cykly pro automaty jiných velikostí než 2^n, namátkou např. rodiny pravidel 90, 105 či 150. U jiných rodin se toto chování projevuje pouze u některých pravidel, např. 26 či 154.

4.3 Zpětný koeficient

Na rozdíl od předchozího testu, jehož uplatnění bylo spíše teoretické, výsledky tohoto jsou pro námi navržený způsob komprese praktičtější. Jeho cílem je roztřídit pravidla podle počtu unikátních stavů (vyjma stavu výchozího) dosažitelných při zpětném běhu z výchozího stavu do předem stanoveného maximálního počtu generací. Počítejme tedy tzv. *zpětný koeficient* jako podíl skutečně dosažitelných stavů a počtu prohledávaných generací, který v tomto případě představuje počet unikátních dosažitelných stavů pro nevětvící se výpočet bez nalezení cyklu. Koeficienty nižší než jedna značí, že výpočet končí či nalezne cyklus ještě před dosažením maximálního počtu generací, naopak koeficienty vyšší než jedna prokazují, že ve výpočtu došlo k větvení.

Test provedeme pro všechna pravidla na náhodných bitových sekvencích délek 16, 24 a 32 a na textových řetězcích délek 4, 8 a 16 znaků, prohledávat budeme

[3] Požadavek na náhodnost či chaotičnost pravidla odpovídá v systému klasifikace elementárních celulárních automatů, jak jej navrhl Wolfram (2002, Kapitola 6), tzv. třídě III.

[4] Podobně i zde lze vysledovat souvislost s Wolframovou klasifikací. Pravidla nalézající cyklus v konstantním čase patří převážně k třídám I a II, pravidla, jejichž čas k nalezení cyklu je lineární, patří obvykle do třídy II.

Cyklická pravidla	**16 bitů**	**24 bitů**	**32 bitů**	**„Lore“**
$\{0, 255\}$	2	2	2	2
$\{2, 16, 191, 247\}$	17	25	33	33
$\{3, 17, 63, 119\}$	33	49	65	65
30	6120	192 048	875 036	918 085
86	4385	188 175	1 052 903	1 057 985
135	4192	186 191	1 043 043	857 120
149	4204	185 196	1 131 853	1 008 630
$\{36, 219\}$	3	3	3	3
45	2816	434 001	27 276 111	1 667 629
75	608	421 188	28 603 926	1 556 803
89	1405	17 523	1 064 657	1 093 782
101	1236	5896	27 066 762	26 933 402
106	2777	7561	236 199	536 121
120	364	24 484	611 709	506 104
169	2767	21 081	515 203	560 955
225	459	9748	233 809	600 149
Acyklická pravidla	**16 bitů**	**24 bitů**	**32 bitů**	**„Lore“**
26	41	2071	94	87
82	43	2067	89	90
167	36	32	38	38
181	31	32	40	47
$\{90, 165\}$	30	2046	62	62
105	30	2046	64	62
150	30	2046	64	63
154	39	2069	92	91
166	16	24	31	33
180	15	23	31	31
210	43	2067	91	90

Pozn: Vstupními daty jsou prefixy testovací bitové posloupnosti o délce 16, 24 a 32 bitů a čtyřznakový řetězec převedený na posloupnost bitů pomocí kódování UTF-8. Tabulka je vertikálně rozdělena po rodinách pravidel. Pravidla, jejichž data jsou shodná, jsou uvedena na jednom řádku.

Tabulka 4.1: Délky cyklů pro zmíněná pravidla ze zvolených počátečních stavů.

do hloubky 100 000 generací zpět. Na rozdíl od testu délky cyklu se v tomto případě prodloužením testovacích textových řetězců pokusíme simulovat reálná data, z důvodu udržení počtu unikátních předchozích stavů u některých pravidel na přijatelných hodnotách však nezvolíme řetězce delší.

Převážná většina cyklických pravidel nabývá ze všech (či z většiny) výchozích stavů nulového zpětného koeficientu a tedy nenalézá žádného předka výchozího stavu. Tato pravidla jsou pro námi navržený způsob komprese bezcenná. Výjimkami jsou rodiny pravidel 30, 45 a 106, jejichž zpětný koeficient je vždy nenulový a v určitých případech přesahuje 1 (jak je vidno z tabulky 4.2). Jedná se o ta pravidla, která zaznamenala úspěch v testu délky cyklu. Potvrzuje se tedy, že tato pravidla jsou schopna z náhodného počátečního stavu nalézt nejen dostatečné množství různých následovníků, ale i velké množství různých předchůdců. Je tedy namístě se jimi dále zabývat. Dalšími pravidly s nenulovými zpětnými koeficienty jsou rodiny pravidel 15, 105, 150, 154 a 170, jejichž zpětný koeficient je lineární v závislosti na velikosti automatu. Při podrobnějším zkoumání můžeme nahlédnout, že například pravidla 15 a 170 vstupní stav pouze rotují a invertují, zvláštním případem je pak pravidlo 51, které v každé generaci pouze invertuje předchozí (či ekvivalentně následující) stav, a tedy vždy nalezne právě jednoho předchůdce. Pro další výzkum jsou tato ostatní pravidla méně zajímavá.

Z acyklických pravidel mají výhradně nenulové zpětné koeficienty (pominemeli opět pravidlo 51) pouze rodiny pravidel 60 a 90, převážně nenulové 15, 30, 45, 105 a 150. Jsou tedy jedinými pravidly z této skupiny, kterými bychom se měli dále zabývat. Přesto však hodnoty jejich koeficientů zdaleka nedosahují nejlepších hodnot získaných u cyklických CA. Příčinou je již zmíněná ztráta informace na okrajích automatu.

4.4 Test obecnými kritérii

Smyslem testu obecnými kritérii je analyzovat, zda pravidla, která jsme vyfiltrovali pomocí předchozích dvou testů a která jsou tedy schopna poskytnout dostatečný počet unikátních stavů, nalézají stavy, které jsou pro kompresi vhodnější než výchozí stav. Zaznamenáváme ohodnocení nejlépe ohodnoceného stavu dle obecných fitness kritérií při zpětné simulaci celulárního automatu po 0, 100, 10 000 a 1 000 000 generacích. Nesmíme však opomenout skutečnost, že tato skupina kritérií hodnotí pouze určitým způsobem uspořádanost stavů. Výsledky tohoto testu tudíž nelze interpretovat jako skutečně dosažitelnou úsporu místa.

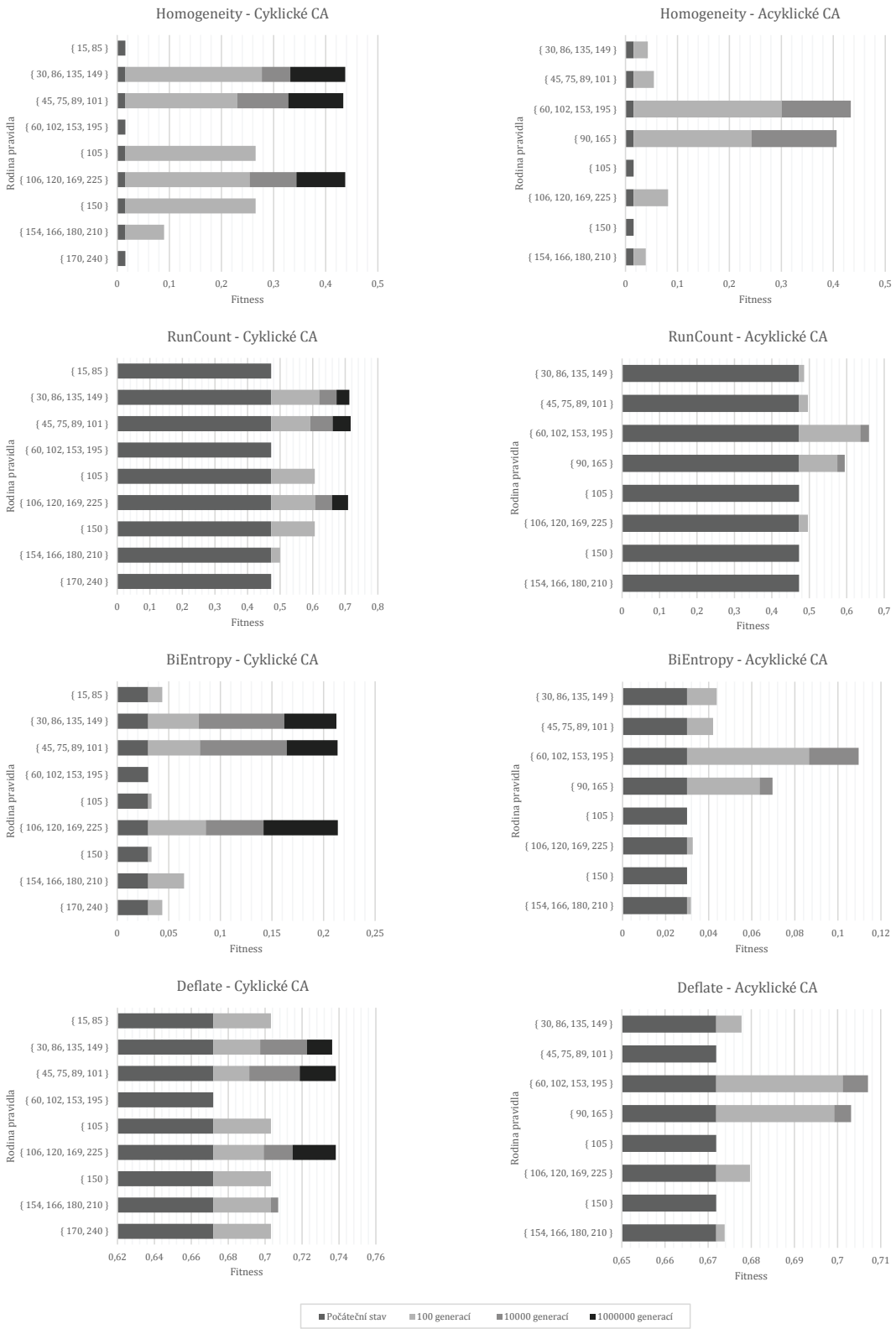
Test provedeme pro všechna pravidla pro textové řetězce délek 4, 8 a 16 znaků. K hodnocení využijeme kritéria Homogeneity, RunCount, Deflate a BiEntropy. Pokud není výslovně zmíněn opak, platí uvedené výsledky pro všechna kritéria.

Cyklická pravidla, která dle předchozího testu dokázala nalézat předcházející stavy, též obvykle nalezla stavy lépe ohodnocené zvolenými kritérii než stav výchozí. U pravidel s lineárně rostoucím, obvykle nízkým, zpětným koeficientem se přirozeně jakékoliv zlepšování zastaví velice brzy. Naopak u pravidel, jejichž zpětný koeficient přesáhl hodnotu 1 (jde o rodiny pravidel 30, 45 a 106), dochází

Cyklická pravidla	**4 znaky**	**8 znaků**	**16 znaků**
$\{15, 85\}$	0,00031	0,00063	0,00127
30	0,01393	0,08445	1
86	4,218	2,0927	1
135	0,00808	1,2706	1
149	0,00107	1,17964	1
45	0,92793	1	1
75	0,21778	1	1
89	0,22182	1	1
101	0,38154	1	1
105	0,00015	0,00031	0,00063
106	0,01657	0,01049	1
120	0,00265	3,62741	1
169	0,00065	1,58854	1
225	0,00032	1,18882	1
150	0,00015	0,00031	0,00063
$\{154, 166, 180, 210\}$	0,00031	0,00063	0,00127
$\{170, 240\}$	0,00031	0,00063	0,00127
Acyklická pravidla	**4 znaky**	**8 znaků**	**16 znaků**
15	0	0	0
85	0,0003	0	0,00002
30	0	0	0
86	0,00014	0	0
135	0,00006	0,00006	0,00006
149	0,00034	0	0,00014
45	0,00006	0,00006	0,00006
75	0	0	0
89	0	0,00002	0,00006
101	0,00002	0	0
$\{60, 102\}$	0,00031	0,00063	0,00127
$\{153, 195\}$	0,00063	0,00127	0,00255
$\{90, 165\}$	0,00061	0,00125	0,00253
105	0,00247	0,00125	0
150	0,00002	0,00125	0

Pozn: Vstupními daty jsou textové řetězce o délce 4, 8 a 16 znaků z testovací „Lorem ipsum“ sekvence, převedené na posloupnost bitů pomocí kódování UTF-8. Tabulka je vertikálně rozdělena po rodinách pravidel. Pravidla, jejichž data jsou shodná, jsou uvedena na jednom řádku.

Tabulka 4.2: Naměřené zpětné koeficienty zmíněných celulárních automatů ze zadaných počátečních stavů do hloubky 100 000 generací zpět.



Obrázek 4.4: Grafy fitness stavů nejlépe hodnocených daným kritériem, získaných zpětným během do hloubky 100, 10 000 a 1 000 000 generací zpět. Zaznamenané fitness je průměrem přes všechna pravidla z rodiny. Výchozím stavem je šestnáctiznakový prefix „Lorem ipsum“ textu, převedený na 128bitovou sekvenci.

ke zlepšování i mezi generacemi 10 000 a 1 000 000.[5] Rozdíl mezi ohodnocením výchozího a nejvhodnějšího stavu se však snižuje s rostoucí velikostí automatu, je tedy otázkou, zda lze dosáhnout takové celkové úspory místa, aby zůstala kladná i při zahrnutí počtu generací.

Jak je patrné z grafů 4.4, zlepšení u všech testovacích vzorků pro všechna kritéria nastalo pouze u již zmíněných rodin pravidel 30, 45 a 106, pro další výzkum se tedy jeví jako nejnadějnější. U rodin 15 a 170 bylo dosaženo zlepšení ve všech případech pouze pro kritéria Deflate a BiEntropy. Kritérium Homogeneity naopak nepřineslo zlepšení nikdy, RunCount pouze sporadicky. Příčinou je fakt, že obě zmíněné rodiny pravidel pouze rotují stav automatu a jedna z nich navíc v každé druhé generaci invertuje všechny buňky. Tyto transformace však neovlivní podíl živých a mrtvých buněk, a počet sledů ovlivní pouze výjimečně. Pro rodiny 60, 105 a 150 bylo zlepšení pozorovatelné převážně u kritérií Homogeneity, RunCount a Deflate, zcela výjimečně naopak u BiEntropy. Rodina 154 zaznamenala zlepšení ve všech případech u kritéria Deflate, ve většině i u zbývajících kritérií.

Z acyklických pravidel se podařilo dosáhnout lépe ohodnocených stavů pro všechny testovací vzorky a všechna kritéria pouze rodinám pravidel 60 a 90, jsou tedy kandidáty pro další výzkum. Všem ostatním pravidlům, která zaznamenala úspěch v cyklické variantě, se zde podařilo dosáhnout přinejlepším částečných úspěchů, dále se jimi tedy zabývat nebudeme.

4.5 Test kompresními kritérii

Výsledkem testu obecnými kritérii jsou rodiny pravidel se silnými předpoklady pro nalézání stavů vhodných ke kompresi. Nyní tedy zbývá ověřit, zda některá z těchto pravidel skutečně produkují stavy, které budeme schopni zkomprimovat včetně dodatečné informace o generaci, v jaké byl stav získán. Tímto způsobem již jsou implementována kritéria SparseCompression, RLECompression a LZB7Compression.

Test provedeme na rodinách cyklických pravidel 30, 45 a 106 a acyklických pravidel 60 a 90. Testovacími daty budou textové řetězce délek 64 a 128 znaků (512, resp. 1024 bitů).

Kritéria SparseCompression a RLECompression shodně nedokázala pomocí cyklických pravidel nalézt stav, na němž by bylo možné dosáhnout kladné úspory místa, a to pro žádný vstupní řetězec. Naopak, v případě acyklických pravidel 60 a 90 dokázala takové stavy nalézat pro určitý (nutno podotknout, že nepříliš vysoký) počet testovacích dat obě zmíněná kritéria. Výsledky jsou uvedeny v tabulkách 4.3 a 4.4.

Za parametry kritéria LZB7Compression byly zvoleny $x = 9$ a $y = 6$, což

[5] Toto chování se samozřejmě stále odvíjí i od velikosti automatu. Pro stavy délky 32 bitů se zlepšení po dosažení hranice 10 000 generací zpět podařilo dosáhnout pouze rodině pravidla 45, pro 64 a více bitů i rodinám pravidel 30 a 106. Stejně tak lze zvolit dostatečně velký automat, aby i u pravidel s lineárně rostoucím zpětným koeficientem došlo ke zlepšení po 10 000 a více generacích. Rozdíl je tedy spíše kvantitativní, nikoliv kvalitativní.

Pravidlo	A	B	C	D	E	F
60	0	0	0	0	0,0166	0
102	0	0	0,001	0,0068	0,0176	0
153	0	0	0,0049	0	0,0137	0
195	0	0	0	0	0,0127	0
90	0,002	0	0	0	0,0186	0
165	0	0	0,001	0,001	0,0146	0

Pozn: Vstupními daty A až F jsou po sobě jdoucí sekvence o délce 128 znaků (1024 bitů) z „Lorem ipsum“ textu, hodnoceny byly stavy dosažitelné zpětným během do hloubky 100 000 generací zpět. Všechna obsažená pravidla jsou v acyklické variantě.

Tabulka 4.3: Ohodnocení stavů získaných zpětným během rodin pravidel 60 a 90 kritériem RLECompression.

Pravidlo	A	B	C	D	E	F
60	0	0	0	0	0,0156	0
102	0	0	0	0,0068	0,0176	0

Pozn: Vstupními daty A až F jsou po sobě jdoucí sekvence o délce 128 znaků (1024 bitů) z „Lorem ipsum“ textu, hodnoceny byly stavy dosažitelné zpětným během do hloubky 100 000 generací zpět. Všechna obsažená pravidla jsou v acyklické variantě. Vynechaná pravidla nedosáhla nenulového výsledku na žádném testovacím řetězci.

Tabulka 4.4: Ohodnocení stavů získaných zpětným během rodin pravidel 60 a 90 kritériem SparseCompression.

v praxi znamená prohlížecí okno velikosti 511 bitů a aktuální okno velikosti 64 bitů. Touto volbou je maximalizováno jak aktuální okno (implementace větší nepodporuje), tak i okno prohlížecí (vzhledem k velikosti testovacích dat by se větší prohlížecí okno nikdy zcela nezaplnilo). Bude tedy testováno (přepočteme-li a zaokrouhlíme bity na osmibitové znaky) předcházejících 64 znaků na výskyt 8 následujících znaků.

Žádné z pravidel, a to cyklických ani acyklických, z výchozích dat nedokáže tímto kritériem nalézt stav s lepším než nulovým ohodnocením. Nemůžeme vyloučit, že pro dostatečně velké počáteční stavy již takový stav může být nalezen, pak ovšem doba potřebná pro zpětnou simulaci celulárního automatu a ohodnocení všech získaných stavů začne být neúnosná.

Jedinými zbývajícími kandidáty na funkční kompresní algoritmus jsou tak acyklická Wolframova pravidla 60 a 90 spolu s navrženými kompresními algoritmy Sparse Bit a RLE Bit.

4.6 Test celulární komprese

Díky předchozím testům již máme informace potřebné k návrhu vlastního kompresního algoritmu založeného na transformaci dat pomocí elementárního celu-

lárního automatu. Jako testovací data použijme již pouze celé testovací soubory, tedy textový `ipsum.txt` a bitmapový `jelly.bmp`.

Nejprve rozdělíme vstupní soubor do sekvencí o délce n bajtů (bude tedy nutno otestovat i účinnost komprese v závislosti na parametru n). Každou z těchto sekvencí pak použijeme jako výchozí stav acyklického CA pravidla 60 či 90 a nalezneme nejlépe ohodnocený předcházející stav dle kritéria SparseCompression nebo RLECompression do maximálního počtu t generací zpět.[6] Následně zapíšeme generaci tohoto stavu pomocí Omega kódu a stav samotný komprimovaný Sparse Bit, resp. RLE Bit kompresí. V případě, že ve vstupním souboru v poslední sekvenci zbyde méně než n bajtů, zmenšíme pro tuto sekvenci celulární automat na odpovídající velikost.

Blok automatu	**60S**	**60R**	**90S**	**90R**
128 bajtů	77 469	78 982	79 505	80 390
256 bajtů	75 875	76 828	78 810	78 396
512 bajtů	74 058	74 394	78 335	75 437
1024 bajtů	73 065	73 245	78 026	74 564
2048 bajtů	72 434	72 591	79 323	73 060
4096 bajtů	71 984	71 982	79 634	72 428
128 bajtů	223 155	224 706	224 412	226 275
256 bajtů	224 164	225 515	224 820	226 113
512 bajtů	224 728	226 056	224 728	226 056
1024 bajtů	224 678	226 005	224 678	226 005
2048 bajtů	224 652	225 980	224 652	225 980
4096 bajtů	224 640	225 966	224 640	225 966

Pozn: Číslo v záhlaví sloupců značí použité acyklické pravidlo celulárního automatu, následující písmeno určuje, zda byl užit Sparse Bit (S) či RLE Bit (R) kompresní algoritmus. První oddíl popisuje kompresi `ipsum.txt` (původní velikost 72 931 bajtů), druhý oddíl kompresi `jelly.bmp` (původní velikost 196 662 bajtů). Velikosti po kompresi jsou uvedeny v bajtech.

Tabulka 4.5: Velikosti souborů `ipsum.txt` a `jelly.bmp` po kompresi v závislosti na parametrech.

Z výsledků uvedených v tabulce 4.5 vyplývá, že pro velké množství bitových sekvencí je výsledný zápis i po aplikaci Sparse Bit či RLE Bit komprese větší, než zápis původní, komprese bylo dosaženo pouze pro `ipsum.txt` s bloky o velikosti 2048 bajtů a většími. Přesto lze vysledovat, že zatímco pravidlo 60 vykazuje podobné chování s oběma typy komprese, pravidlo 90 je obzvlášť neúčinné v kombinaci se Sparse přístupem. Upravme nyní význam zapsané generace tak, aby zapsaná nultá generace značila, že následuje binární zápis výchozího stavu bez aplikované komprese. Touto modifikací (nazývejme ji *skipping*) zajistíme, že každý nekomprimovatelný stav zvětší zkomprimovaný soubor pouze o jeden bit nutný k zápisu nuly Omega kódem.

Jak vyplývá z výsledků uvedených v tabulce 4.6, komprese nyní již bylo dosa-

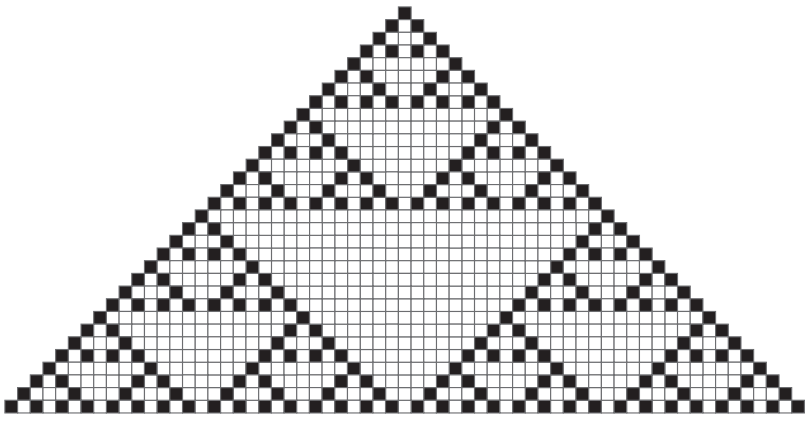
[6] Z testu zpětného koeficientu vyplývá, že v naprosté většině případů nepřekročí počet unikátních předcházejících stavů acyklických pravidel 60 a 90 dvojnásobek délky automatu, můžeme tedy touto hodnotou maximální počet generací omezit.

Blok automatu	**60S**	**60R**	**90S**	**90R**
128 bajtů	72 082	72 062	72 265	72 213
256 bajtů	71 996	71 982	72 232	72 095
512 bajtů	71 954	71 942	72 226	72 042
1024 bajtů	71 959	71 952	72 259	72 009
2048 bajtů	71 936	71 932	72 312	71 959
4096 bajtů	71 984	71 982	72 359	71 988
128 bajtů	196 687	196 687	196 789	196 853
256 bajtů	196 713	196 713	196 746	196 757
512 bajtů	196 711	196 711	196 711	196 711
1024 bajtů	196 687	196 687	196 687	196 687
2048 bajtů	196 675	196 675	196 675	196 675
4096 bajtů	196 669	196 669	196 669	196 669

Pozn: Číslo v záhlaví sloupců značí použité acyklické pravidlo celulárního automatu, následující písmeno určuje, zda byl užit Sparse Bit (S) či RLE Bit (R) kompresní algoritmus. První oddíl popisuje kompresi `ipsum.txt` (původní velikost 72 931 bajtů), druhý oddíl kompresi `jelly.bmp` (původní velikost 196 662 bajtů). Velikosti po kompresi jsou uvedeny v bajtech.

Tabulka 4.6: Velikosti souborů `ipsum.txt` a `jelly.bmp` po kompresi v závislosti na parametrech s přeskakováním případných nekomprimovatelných bloků.

ženo pro `ipsum.txt` ve všech případech, přičemž pravidlo 60 se obecně jeví jako vhodnější. Úspora místa je však zanedbatelná (v nejlepším případě 1,4 %). Soubor `jelly.bmp` se stále komprimovat nepodařilo. Zkusme se tedy blíže zabývat využívanými acyklickými pravidly 60 a 90. V obou případech se jedná o pravidla, která z počátečního stavu s jedinou živou buňkou generují fraktální útvar známý jako Sierpińského trojúhelník (viz obrázek 4.5).



Obrázek 4.5: Sierpińského trojúhelník vygenerovaný prvními 32 generacemi Wolframova pravidla 90 z počátečního stavu s jedinou živou buňkou.

Z počátečního stavu s libovolnou konfigurací délky 2^n je pak dopředným během automatu získán stav vzniklý dvojím opakováním této konfigurace v generaci 2^{n-1} v případě pravidla 90 a v generaci 2^n v případě pravidla 60. Pro novou konfigu-

raci platí totéž, v generaci $2^{n-1} + 2^n = 3 \times 2^{n-1}$, respektive $2^n + 2^{n+1} = 3 \times 2^n$ se tedy původní konfigurace opakuje již čtyřikrát. Při zpětném běhu pak přirozeně pozorujeme, že z dvojího opakování konfigurace délky 2^n se po 2^{n-1}, resp. 2^n generacích stává pouze jediný výskyt této konfigurace doplněný mrtvými buňkami (či nulovými bity). To je obzvlášť výhodné pro námi navržené kompresní algoritmy Sparse Bit a RLE Bit, neboť tím vzniká delší sekvence nulových bitů (RLE Bit) a tyto pak svým počtem výrazně převyšují bity jedničkové (Sparse Bit). Dalšího zlepšení by tedy mohlo být dosaženo předchozí úpravou vstupních dat tak, abychom maximalizovali výskyt takovýchto opakujících se sekvencí. Vzhledem ke sdružování stejných znaků (a tedy častému opakování osmibitových skupin) se nabízí využití Burrows-Wheelerovy transformace jako preprocesoru pro následnou aplikaci navržené komprese. Samotná Burrows-Whelerova transformace po blocích o velikosti v bajtů vstupní soubor velikosti V bajtů zvětší o $\lceil V/v \rceil \times \lceil \log_2 v \rceil$ bitů, výhodnější uspořádání dat by však mělo tento nedostatek vyvážit. Otestujme tedy navrženou kompresi na souborech transformovaných Burrows-Wheelerovou transformací s různou velikostí bloku.

Blok BW	**60S**	**60R**	**90S**	**90R**
256 bajtů	67 390	67 226	67 461	69 103
4096 bajtů	46 274	46 179	46 071	48 193
65 536 bajtů	34 140	33 966	33 917	34 987
256 bajtů	168 937	168 891	169 992	174 707
4096 bajtů	160 997	160 920	161 798	167 125
65 536 bajtů	157 794	157 766	159 078	164 549

Pozn: Číslo v záhlaví sloupců značí použité acyklické pravidlo celulárního automatu, následující písmeno určuje, zda byl užit Sparse Bit (S) či RLE Bit (R) kompresní algoritmus. Velikost bloku se týká pouze Burrows-Wheelerovy transformace, komprese celulárním automatem probíhala vždy po 32 bajtech. První oddíl popisuje kompresi `ipsum.txt` (původní velikost 72 931 bajtů), druhý oddíl kompresi `jelly.bmp` (původní velikost 196 662 bajtů). Velikosti po kompresi jsou uvedeny v bajtech.

Tabulka 4.7: Velikosti souborů `ipsum.txt` a `jelly.bmp` po Burrows-Wheelerově transformaci o různé velikosti bloků a následné celulární kompresi v závislosti na parametrech.

Z tabulky 4.7 lze vyčíst, že komprese již byla úspěšná pro oba soubory se všemi parametry. Můžeme nahlédnout, že výsledná úspora místa se zlepšuje s rostoucí velikostí bloku Burrows-Wheelerovy transformace. Kombinace Wolframova pravidla 90 a RLE komprese se ukazuje nejméně účinná, nejúčinnější se naopak jeví Wolframovo pravidlo 60 spolu s RLE kompresí (tato kombinace dosáhla nejlepšího výsledku ve 4 ze 6 případů). Uvažujme tedy již pouze tuto variantu.

Pro soubor `ipsum.txt` jsme dosáhli v nejlepším případě úspory místa 53,4 %, pro soubor `jelly.bmp` 19,8 %. Námi navržená komprese je tedy funkční a dosahuje nezanedbatelné úspory místa na skutečných datech. Nyní je třeba ji srovnat s tradičními kompresními algoritmy.

4.7 Srovnání kompresních algoritmů

V posledním testu srovnáme navržený kompresní algoritmus s ostatními implementovanými kompresními algoritmy z hlediska kompresního poměru, respektive dosažené úspory místa. Test provedeme na testovacích souborech `ipsum.txt` a `jelly.bmp` a jejich variantách transformovaných Burrows-Wheelerovou transformací po blocích o velikosti 65 536 bajtů.

Srovnejme nejprve navržené kompresní algoritmy, tedy Sparse Bit, RLE Bit, LZ77, LZ78 a jejich bitové varianty, bez jakékoliv aplikace celulárních automatů. Předpokládáme, že zatímco Sparse Bit a RLE Bit nepřinesou žádnou úsporu místa, LZ77 a LZ78 jakožto zavedené kompresní algoritmy ano. LZ77 Bit a LZ78 Bit pravděpodobně též dosáhnou kladné úspory místa, avšak z důvodu jejich bitové orientace oproti bajtové orientaci testovacích dat nejspíše ne tak vysoké jako jejich bajtově orientované předlohy.

Algoritmus	`ipsum.txt`	`ipsum.bw_16`	`jelly.bmp`	`jelly.bw_16`
Sparse Bit	−12,13 %	−15,21 %	−14,22 %	−14,26 %
RLE Bit	−16,00 %	−12,95 %	−14,89 %	−15,15 %
LZ78	37,33 %	48,44 %	−23,20 %	−7,57 %
LZ78 Bit	−40,31 %	−2,26 %	−83,26 %	−59,66 %
LZ77 $(9, 6)$	28,21 %	62,31 %	−19,09 %	3,59 %
LZ77 $(11, 3)$	50,37 %	52,00 %	12,50 %	20,49 %
LZ77 $(14, 3)$	54,41 %	48,31 %	16,27 %	18,86 %
LZ77 $(15, 4)$	60,67 %	56,32 %	13,26 %	15,51 %
LZ77 Bit $(12, 6)$	27,55 %	50,30 %	−13,21 %	10,13 %
LZ77 Bit $(15, 6)$	53,18 %	48,44 %	9,82 %	16,32 %
LZ77 Bit $(16, 6)$	54,66 %	47,30 %	11,21 %	16,02 %

Pozn: Soubory s příponou `.bw_16` vznikly Burrows-Wheelerovou transformací po blocích o velikosti 65 536 bajtů. Parametry LZ77 a LZ77 Bit algoritmů byly určeny empiricky (pro LZ77 z $x \in \{3 \dots 18\}$, $y \in \{2 \dots x - 1\}$, pro LZ77 Bit z $x \in \{7 \dots 16\}$, $y \in \{3 \dots 6\}$) pro každý ze vstupních souborů tak, aby byl komprimován co nejlépe právě s těmito parametry.

Tabulka 4.8: Úspora místa dosažená na souborech `ipsum.txt` a `jelly.bmp` a jejich variantách transformovaných Burrows-Wheelerovou transformací pomocí klasických kompresních algoritmů.

Tabulka 4.8 potvrzuje naše předpoklady, že Sparse Bit ani RLE Bit žádnou úsporu místa nepřináší. Tradiční LZ77 a LZ78 obstojně komprimují textová data, obrazová však jen LZ77 s vhodně zvolenými parametry. Oproti našemu očekávání však z jejich bitových variant uspěl pouze LZ77 Bit; LZ78 Bit úspory nedosáhl na žádném ze vstupních souborů. Nyní porovnejme výsledky komprese těchto čtyř souborů pomocí Wolframova pravidla 60 a RLE Bit.

Z dat uvedených v tabulce 4.9 lze vyčíst, že na původních souborech celulární komprese nedosahuje příliš dobrých výsledků. Po Burrows-Wheelerově transformaci těchto souborů je však co do kompresního poměru srovnatelná s naší implementací slovníkových algoritmů LZ77 a LZ78, v případě `jelly.bmp` dokonce dosáhla lepšího kompresního poměru než jakýkoliv jiný testovaný algoritmus. Zdá

Blok automatu	`ipsum.txt`	`ipsum.bw_16`	`jelly.bmp`	`jelly.bw_16`
32 bajtů	−0,09 %	53,08 %	−0,22 %	19,78 %
64 bajtů	0,24 %	57,05 %	0,03 %	21,70 %
128 bajtů	0,45 %	59,16 %	−0,01 %	22,61 %
256 bajtů	0,56 %	60,21 %	−0,03 %	22,96 %
512 bajtů	0,62 %	60,61 %	−0,02 %	23,09 %
1024 bajtů	0,61 %	60,78 %	−0,01 %	23,26 %
2048 bajtů	0,63 %	60,92 %	−0,01 %	23,30 %
4096 bajtů	0,56 %	61,00 %	0,00 %	23,27 %

Pozn: Soubory s příponou `.bw_16` vznikly Burrows-Wheelerovou transformací po blocích o velikosti 65 536 bajtů.

Tabulka 4.9: Úspora místa dosažená na souborech `ipsum.txt` a `jelly.bmp` a jejich variantách transformovaných Burrows-Wheelerovou transformací pomocí celulární komprese Wolframovým pravidlem 60 a bitovou RLE kompresí po blocích různé velikosti.

se též, že kompresní poměr se zlepšuje s rostoucí velikostí bloku celulární komprese, teoreticky by tedy bylo možné dalším zvětšováním bloku produkovat menší výstup za cenu déle trvajícího výpočtu. Je ovšem nutno také poznamenat, že téhož výsledku lze dosáhnout zvětšováním bloku Burrows-Wheelerovy transformace (viz tabulka 4.7), a to s nižší výpočetní náročností.

Největší překážkou pak zůstává rychlost celulární komprese. Průměrná časová složitost algoritmu tak, jak je implementován, je lineární vzhledem k součinu velikosti vstupního souboru a velikosti bloku pro celulární automat. Dvojnásobné zvětšení vstupního souboru tedy lze kompenzovat zmenšením bloku na polovinu za cenu možného zhoršení kompresního poměru.

Na testovací sestavě s procesorem Intel Core i5-3470 a 8 GB RAM trvala komprese jednoho „kB^2“, neboli vstupního souboru o velikosti 1 kB po blocích délky 1024 bajtů, přibližně 8 sekund. V praxi je tedy takto navržený algoritmus téměř nepoužitelný.

5. Závěr

5.1 Dosažené cíle

V této práci jsme prozkoumali některé základní vlastnosti elementárních celulárních automatů. Implementovali jsme algoritmy pro jejich simulaci jak dopřednou, tak zpětnou. Pomocí sady testů a kritérií jsme roztřídili přechodové funkce elementárních celulárních automatů podle počtu a uspořádanosti jimi produkovaných stavů. Na základě těchto výsledků jsme pak implementovali bezeztrátový kompresní algoritmus využívající transformaci acyklickým celulárním automatem s využitím Wolframova pravidla 60 a 90. Ten jsme posléze srovnali s klasickými kompresními metodami.

Navržený algoritmus dosahuje kompresních poměrů srovnatelných se slovníkovými algoritmy LZ77 nebo LZ78. Pro tuto účinnost musí však být vstupní data nejprve transformována pomocí Burrows-Wheelerovy transformace. Aplikací konvenčních kompresních algoritmů, které kromě slovníkových přístupů implementují často i právě tuto transformaci či další metody pro optimalizaci komprese dat (např. Huffmanovo kódování), bychom dosáhli lepších výsledků. Vyvinutý algoritmus navíc pro každý blok vstupních dat prohledává a analyzuje velké množství jeho předchůdců ve zvoleném celulárním automatu, časová složitost je tedy vzhledem k dosaženému výsledku neúměrná.

Náš algoritmus není vhodný pro užití v praxi, postačuje však k prokázání úvodní hypotézy, že celulární automaty obecně jsou schopny transformovat data způsobem vhodným pro bezeztrátovou kompresi.

5.2 Náměty pro další výzkum

Žádoucím vylepšením stávajícího algoritmu by bylo jeho zrychlení tak, aby byl použitelný i na souborech obvyklé velikosti. Toho můžeme docílit efektivnějším prohledáváním předchozích stavů celulárního automatu, popřípadě implementací vhodné heuristiky.

Dále je možné zabývat se i jinými než elementárními celulárními automaty. Jak bylo zmíněno, z důvodu bajtové orientace běžných dat nemusí být tyto dvoustavové automaty nejvhodnějšími kandidáty pro jejich bezeztrátovou kompresi. Bylo by možné prozkoumat jednorozměrné či vícerozměrné celulární automaty např. s 256 stavy a tedy s možností operovat přímo nad jednotlivými bajty vstupních dat.

Pro budoucí praktickou využitelnost navrženého přístupu k bezeztrátové kompresi dat je však v každém případě další výzkum v této oblasti nutný.

Seznam použité literatury

Aron, J. (2013). First replicating creature spawned in life simulator. *New Scientist*.

Burrows, M. a Wheeler, D. J. (1994). A Block-sorting Lossless Data Compression Algorithm. Technical report, DIGITAL SRC RESEARCH REPORT.

Cook, M. (2004). Universality in Elementary Cellular Automata. *Complex systems*, **15**(1), 1–40.

Croll, G. J. (2013). BiEntropy-The Approximate Entropy of a Finite Binary String. *arXiv preprint arXiv:1305.0954*.

Gardner, M. (1970). Mathematical games: The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, **223**(4), 120–123.

Gosper, R. W. (1984). Exploiting regularities in large cellular spaces. *Physica D: Nonlinear Phenomena*, **10**(1), 75–80.

Haukeli, M. (2012). Lossless Data Compression Using Cellular Automata. Master's thesis, University of Oslo.

Jeras, I. a Dobnikar, A. (2007). Algorithms for computing preimages of cellular automata configurations. *Physica D: Nonlinear Phenomena*, **233**(2), 95–111.

Khan, A. R., Choudhury, P., Dihidar, K. a Verma, R. (1999). Text Compression Using Two-Dimensional Cellular Automata. *Computers & Mathematics with Applications*, **37**(6), 115–127.

Lafe, O. (1996). Data Compression and Encryption Using Cellular Automata Transforms. In *Intelligence and Systems, 1996., IEEE International Joint Symposia on*, pages 234–241. IEEE.

Rendell, P. (2002). Turing universality of the game of life. In *Collision-based computing*, pages 513–539. Springer.

Sayood, K. (2012). *Introduction to Data Compression*. Newnes.

Shannon, C. E. (2001). A Mathematical Theory of Communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, **5**(1), 3–55. ISSN 1559-1662. doi: 10.1145/584091.584093. URL `http://doi.acm.org/10.1145/584091.584093`.

Wolfram, S. (2002). *A New Kind of Science*, volume 5. Wolfram media Champaign.

Seznam obrázků

Seznam tabulek

Příloha A

Obsah přiloženého DVD

- `program.zip` - archiv obsahující spustitelný program `CellularAutomata-Visualization.exe` pro vizualizaci simulace elementárních celulárních automatů, spustitelný program `CellularAutomataCompression.exe` pro testování implementovaných kompresních a transformačních metod, a PDF soubor `dokumentace.pdf` obsahující uživatelskou dokumentaci k oběma spustitelným programům
- `testfiles.zip` - archiv obsahující soubory `ipsum.txt` a `jelly.bmp`, na kterých byla v této práci komprese testována
- `code.zip` - archiv obsahující projekt Microsoft Visual Studia 2013 se zdrojovými kódy k oběma spustitelným programům
- `data.zip` - archiv obsahující naměřená data z jednotlivých experimentů uvedených v kapitole Experimenty
- `thesis.pdf` - PDF soubor s touto bakalářskou prací