VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

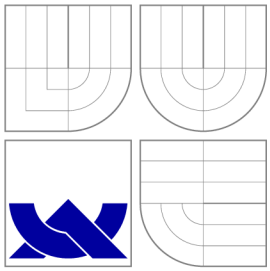
OPTIMALIZACE NÁVRHU CELULÁRNÍCH AUTOMATŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

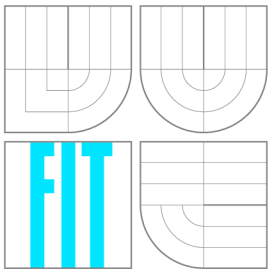
AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ JÍLEK

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

OPTIMALIZACE NÁVRHU CELULÁRNÍCH AUTOMATŮ

CELLULAR AUTOMATA DESIGN OPTIMALIZATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ JÍLEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAL BIDLO, Ph.D.

BRNO 2014

Abstrakt

Tato diplomová práce je zaměřena na evoluční návrh celulárních automatů a jeho optimalizaci. Nejprve jsou popsány evoluční algoritmy, celulární automaty, a poté je rozebrán jeden z možných nových postupů pro reprezentaci a evoluční návrh přechodové funkce automatu - podmínková pravidla. Následuje popis experimentů s podrobnými výsledky. V závěru je prezentováno úspěšné dosažení optimalizace evolučního návrhu celulárních automatů za pomocí podmínkových pravidel pro vybrané problémy.

Abstract

Focus of this master's thesis is on evolutionary design of cellular automata and it's optimalization. There are described Evolutionary algorithms and Cellular automata in first part. Thereafter, one of the new ways of transition function representation and its possible evolutionary design is presented. Name of this method is Conditionally Matching Rules. This is followed by description of realized experiments with detailed results. Finally, success with optimalization for some tasks is presented in last chapter along with discussion.

Klíčová slova

evoluční algoritmus, celulární automat, optimalizace, evoluční návrh, podmínková pravidla

Keywords

evolutionary algorithm, cellular automaton, optimalization, evolutionary Design, Condition Matching Rules

Citace

Tomáš Jílek: Optimalizace návrhu celulárních automatů, diplomová práce, Brno, FIT VUT v Brně, 2014

Optimalizace návrhu celulárních automatů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Michala Bidla, Ph.D.

.....
Tomáš Jílek
25. května 2014

Poděkování

Tímto bych rád poděkoval svému vedoucímu Ing. Michalovi Bidlovi, Ph.D za to, že umožnil vznik této práce svým nadšením pro věc, které mi bylo v průběhu psaní a experimentů inspirací. Neméně si také cením vstřícného a trpělivého přístupu při konzultacích.

Obsah

Kapitola 1

Úvod

V posledních letech se čím dál více pozornosti v rámci informačních technologií obrací k netradičním formám výpočtů a jejich výzkumu. Vyvstávají totiž problémy a úlohy, pro které není známo efektivní řešení s využitím současných technologií a postupů. Tyto snahy úzce souvisí s vývojem oboru nazvaného *softcomputing*, který se snaží oproti tradičním přístupům akceptovat všudypřítomnou nepřesnost a nejistotu reálného světa, pracovat s ní a využívat ji [28]. Algoritmy vycházející ze softcomputingu tedy dokáží pracovat s nepřesností, nejistotou, částečnými či nepřesnými daty a často se jich pak dokonce využívá ve prospěch efektivity. Jako hlavní cíl je kladeno větší přiblížení počítačů k principům fungování přírody a reálného světa. Také se zdá, že tento přístup vede dál ke hlubšímu zkoumání samotné povahy a principu výpočtu, jak ukazuje například Stephen Wolfram [27].

Ruku v ruce s novými přístupy k výpočtům se vyvíjí i hardware, který je výpočty používán. Velká část modelů softcomputingu dokáže využít moderní pro masivně paralelní počítání (neuronové sítě, evoluční algoritmy, fuzzy logika, celulární automaty, ...). Také určité typy hardwaru jsou použitelné a navrhnutelné právě díky softcomputingu, například dynamicky rekonfigurovatelný hardware pomocí evolučních algoritmů [24, 14, 11].

Jedním z takových modelů softcomputingu jsou i *celulární automaty*, které jsou inspirovány biologickými buňkami a jejich interakcemi mezi sebou. Model se skládá z malých na sobě nezávislých částí, které spolu komunikují pouze v rámci svých sousedů. Existuje zde tedy velký potenciál na využití v paralelním hardware, na dobrou škálovatelnost, nezávislost komponent a případně na určitou toleranci k chybám jednotlivých komponent.

Díky své inspiraci buňkami a principu fungování, celulární automaty vykazují velké emergentní vlastnosti, kdy pouze ze znalostí či definice lokálních interakcí mezi komponentami nelze plně předpovědět jak se bude automat jako celek chovat, i když se jedná o plně deterministický výpočet (viz např. *langtonův mravenec* [18] a *teorie chaosu* [2]).

Kvůli této charakteristice celulárního automatu je potřeba mít efektivní techniky pro návrh lokálních interakcí tak, aby řešily zadaný problém. Techniky návrhu celulárních automatů v současné době umí efektivně navrhovat interakce pouze pro jednoduché problémy. Ostatní existující interakce, pro složitější problémy, byly objeveny buď náhodou, nebo se jedná o velké úsilí lidské kreativity.

Nedostatek technik pro více automatizovaný návrh celulárních automatů je jedna z největších překážek rozvoje jejich praktického využití. Motivací pro mojí práci je přispět k úsilí o výzkum těchto technik a posunout celulární automaty o krok blíže praktickému použití, než jsou v tuhle chvíli. Jedná se totiž o fascinující nový model, který má potenciál přispět do různých oborů, zejména do simulací, medicíny [22] a robotiky [1].

Poměrně novým přínosem v oblasti zkoumání celulárních automatů jsou experimenty

s různými reprezentacemi definic lokálních interakcí pro evoluční algoritmy, jako je *reprezentace pomocí instrukcí* [5] či *reprezentace pomocí podmínkových pravidel* [6]. Ukázalo se, že alespoň pro některé problémy jsou navrhované přístupy efektivnější než dosud známé metody [4]. V některých případech lze dokonce najít řešení pro problémy, které dosud nebylo možné standardními metodami navrhovat.

1.1 Obsah práce

Cílem této práce je detailněji prozkoumat metodu návrhu pomocí podmínkových pravidel, implementovat program pro simulaci a provést s podmínkovými pravidly experimenty. První část experimentů se zaměřuje na optimalizaci návrhu transformace vzoru v 2D automatech, původně provedeném v [6]. Poté bude provedeno srovnání efektivity původního experimentu, v této práci optimalizované variaty, a evoluce standardním způsobem bez podmínkových pravidel. Druhá část experimentů bude zaměřena na návrh v 1D celulárních automatech, kde se klade za cíl poznatky a optimalizace z první části využít pro efektivní návrh. Použit bude problém výpočtu druhé mocniny pro číslo, pro který bylo v [27] vytvořeno jedno z možných řešení. Jedná se však o neautomazivaný návrh člověkem. Cílem druhé části experimentů pak bude vytvořit efektivní automatický návrh pro tento problém se srovnáním kvality výsledků s originálním návrhem a efektivity s evolucí bez podmínkových pravidel.

Práce je členěna do kapitol tak, že počáteční kapitoly si kladou za cíl seznámit čtenáře s problematikou celulárních automatů (Kapitola 2) a evolučních algoritmů (Kapitola 3). Následují kapitoly s popisem techniky podmínkových pravidel (Kapitola 4) a specifiky implementace (Kapitola 5). Po nich jsou už uveden popisy a výsledky experimentů spolu se shrnutím jednotlivých fází (Kapitoly 6 a 7). Na závěr jsou zhodnoceny dosažené výsledky s uvedením možných dalších směrů pokračování práce (Kapitola 8).

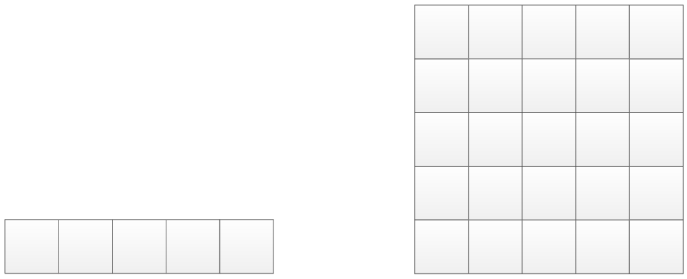
Diplomová práce také navazuje na semestrální projekt stejného tématu. V něm jsem se zabýval hlavně teoretickým studiem tématu a implementací funkčního prototypu aplikace pro provádění experimentů.

Kapitola 2

Celulární automaty

Celulárním automatem (dále jen *CA*) se nazývá diskrétní model, jehož první verzi používali kolem roku 1940 *Stanislaw Ulam* a *John von Neumann* k výzkumům chování složitých systémů [19].

Automat je složen z *buněk* (anglicky „cell“ a „cellular automata“), které jsou uspořádány do čtvercové mřížky. Existuje 1D varianta (obrázek 2.1.a) a 2D varianta (obrázek 2.1.b). Je možné použít automat s jinou než čtvercovou mřížkou nebo např přidat třetí rozměr, jedná se však o velice specifické experimenty či použití a v obecném výzkumu se zatím nevyužívají.



(a) (b)

Obrázek 2.1: Topologie 1D a 2D celulárního automatu

Každá buňka se může nacházet v jednom stavu z konečné množiny stavů, které jsou nejčastěji reprezentovány číslem. Z tohoto pohledu rozlišujeme automaty *binární* se stavy „1“ (živá buňka), „0“ (mrtvá buňka) a *vícestavové* pro automaty s více stavy než dva. V každém kroku výpočtu změní každá buňka svůj stav pouze na základě buněk z okolí. Předpis dle kterého buňka mění svůj stav, nazýváme *přechodovou funkcí*.

Algoritmus 1 ukazuje postup pro základní *synchronní* automat, kdy hodnoty buněk okolí jsou brány vždy z předchozího stavu. Existuje ještě asynchronní varianta, kdy je uchováván pouze jeden aktuální stav, vypočtená změna jakékoli buňky se do něj hned promítne a výpočty okolí používají tento jeden stav. Tímto se ale ztrácí determiničnost automatu a ten se stává závislý na implementaci a průběhu paralelních procesů. Popsaný automat je také nazýván *uniformní*, protože přechodová funkce je stejná pro všechny buňky. Oproti

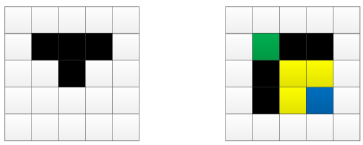
tomu u *neuniformního* automatu se přechodová funkce může lišit pro každou buňku či skupinu buněk. V této práci se budeme zabývat pouze synchronními uniformními celulárními automaty.

Algoritmus 1 Základní synchronní CA

```
t = 0
state(t) = NastavPocatecniStav()
while t < UkoncujiciCas() do
    state(t + 1) = Copy(state(t))
    for cell in state(t) do
        okoli = VratOkoli(cell, state(t))
        newCell = PrechodovaFunkce(okoli)
        state(t + 1)[cell] = newCell
    end for
end while
```

2.1 Zobrazení automatu

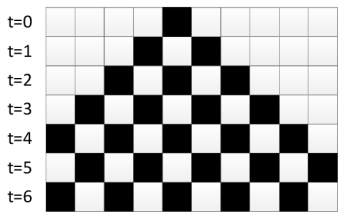
Konvence při zobrazování binárního automatu je zobrazovat buňky ve stavu „0“ bíle (prázdně) a buňky ve stavu „1“ černě (obrázek 2.1.a). Obecněji lze říci, že pro mrtvé buňky se používá barva pozadí, pro živé barva popředí. Pro zobrazení vícestavových automatů se zachovávají konvence binárního zobrazení a přidávají se barvy pro stavy větší než „1“ (obrázek 2.1.b). Tyto barvy nejsou nijak konvencí sjednocené, a různí autoři používají pro označení stavů různé barvy.



(a) (b)

Obrázek 2.2: Vizualizace binárního a vícestavového automatu

Pro 1D automaty se často také používá zobrazení ukazující vývoj v čase. Výsledkem je dvojrozměrná mřížka poskládaná z řádků, z nichž každý reprezentuje stav automatu v čase. Na obrázku 2.3 je příklad vývoje 1D automatu pro časy $t = \{0, 1, 2, 3, 4, 5, 6\}$.



Obrázek 2.3: Vizualizace vývoje 1D automatu v čase

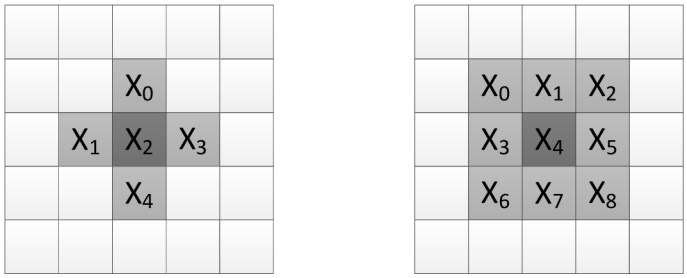
2.2 Okolí buňky

Každá buňka má své *okolí*, které funguje jako vstup pro přechodovou funkci. V 1D automatu (obrázek 2.4.a) je pro buňku X_1 okolím uspořádaná množina $\{X_0, X_1, X_2\}$, tj. buňka samotná a buňka nalevu a napravo. Toto okolí v 1D automatech, na základě jeho velikosti, nazýváme *3-okolí*. Pro 1D automaty lze použít i jiná okolí, například *5-okolí*, kdy přidáme další buňku zleva i zprava (obrázek 2.4.b) tak, že pro buňku X_2 je okolím $\{X_0, X_1, X_2, X_3, X_4\}$. Tento typ a jiné další nejsou až tak často používané jako 3-okolí.

(a) (b)

Obrázek 2.4: 3-okolí (a) a 5-okolí (b) v 1D automatu

Pro 2D automaty se s okolím postupuje podobně a jako hlavní používané typy rozlišujeme *5-okolí* a *9-okolí*. 5-okolí, nazývané též „Von Neumanovské okolí“, tvoří sama buňka a čtyři sousedi přilehlí hranou (obrázek 2.5.a). 9-okolí, které nazýváme „Moorovo okolí“, tvoří pak už buňka a všech osm okolních (obrázek 2.5.a). Z obrázků je také názorně vidět, že konvencí je mít v okolí aktuální buňku uprostřed.



(a) (b)

Obrázek 2.5: 5-okolí (a) a 9-okolí (b) ve 2D automatu

2.3 Přechodová funkce

Přechodová funkce určuje v jakém stavu bude buňka v následujícím kroku výpočtu. Jejím vstupem je okolí buňky a výstupem stav buňky v dalším kroku. Jako základní metoda vyjádření této funkce slouží tabulka o $n+1$ sloupcích, kde prvních n sloupců jsou buňky okolí a poslední sloupec patří nové hodnotě. Pro binární 1D automat s 3-okolím $\{X_1, X_2, X_3\}$ je příklad takové tabulky v tabulce 2.1. Odtuz lze vyčíst, že pro okolí $\{0, 1, 1\}$, čili při hodnotě aktuální buňky „1“, levé buňky „0“ a pravé „1“ bude nová hodnota aktuální buňky „0“.

Řádky v tabulce mohou být poskládány různě, stejně jako okolí buňky, ale na funkcionalitě automatu to nic nezmění. Pokud však následujeme konvence uspořádání okolí dle

obrázků v kapitole 2.2 (zezhora dolu, zleva doprava) a pořadí dle binární inkrementace, je možnost celou tabulku vyjádřit pouze řetězcem nových hodnot. V takovém případě by se pro náš příklad jednalo o řetězec „10101000“. Při určování hodnoty nového stavu z takového řetězce z okolí $\{0, 1, 1\}$ víme, že toto okolí leží v konvenční tabulce na čtvrtém řádku, a proto v řetězci tomuto okolí odpovídá čtvrtá hodnota.

X_1	X_2	X_3	y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Tabulka 2.1: Tabulka přechodové funkce pro okolí 1D automatu se 2 stavy

Vzhledem k délce řetězce pro přechodovou funkci 1D automatu byla Stephenem Wolframem [27, 26] určena další konvence. Pro binární 1D automaty s 3-okolím se přechodové funkce dají označit číslem z intervali $0 - 255$. Binární reprezentace tohoto čísla pak dává binární řetězec, použitý v tabulce pro okolí.

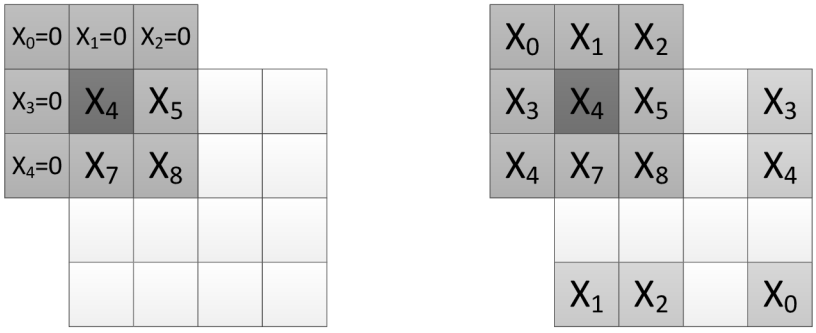
Pro 2D automaty se používá stejná metoda určování nové hodnoty, pouze velikost tabulky poměrně roste. Část tabulky pro binární 2D automat s 9-okolím je v tabulce 2.2, celá by měla $2^9 = 512$ řádků.

X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	y
0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	1	0	0	1
0	0	0	0	0	0	1	0	1	0
...	...	...	...	...	...	...	...	...	...
1	0	1	1	0	0	0	1	0	1

Tabulka 2.2: Část tabulky přechodové funkce pro okolí 2D automatu se 2 stavy

2.4 Okrajové podmínky

Další věcí, kterou je u celulárních automatů nutno řešit, je chování automatu na jeho okrajích. V žádné reálné aplikaci nelze uvažovat nekonečnou plochu pro vývoj. Nejčastějším řešením této situace je uvažovat *nulové* či *cyklické* okrajové podmínky. Nulové podmínky znamenají, že části okolí buňky přesahující hranice automatu, budou mít konstantní hodnotu „0“ po celou dobu běhu automatu (pro levý horní okraj názorně zobrazeno na obrázku 2.6.a). Pro cyklické podmínky platí, že jako hodnota přesahující buňky se použije buňka z opačného konce automatu, jakoby se hrany slepily (pro levý horní roh ukázáno na obrázku 2.6.b).



(a) (b)

Obrázek 2.6: Nulové (a) a cyklické (b) okrajové podmínky

2.5 Použití CA

Aplikací CA a experimentů s nimi existuje celá řada. Většinou se týkají výpočtů, simulací a nebo i simulačních hříček. Hlavním problémem využití CA, ale i vlastností kvůli které se CA zkoumají, je pouze lokální interakce mezi buňkami. Problém, úloha či výpočet jsou zpravidla definovány v globálním kontextu, který žádná buňka ani přechodová funkce nezná. Cílem pak je navrhnout, či najít taková pravidla přechodové funkce tak aby celulární automat řešil danou úlohu. Pokud se takové pravidla podaří najít, existuje potenciál úlohu řešit například sadou na sobě nezávislých výpočetních jednotek.

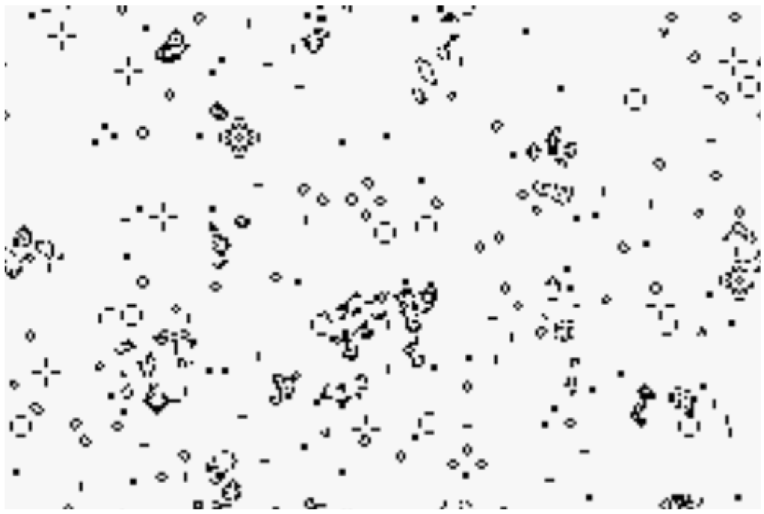
2.5.1 2D CA

Jednu z větších skupin pro studium 2D automatů představují binární automaty založené na *hře života* [10] díky svému chování, které připomíná chování živé kultury buněk. Přechodová funkce tohoto automatu je vyjádřena jednodušší formou než je výše uvedená tabulková metoda tak, že uvažuje vždy součet živých a mrtvých buněk v okolí (tímto slučuje vždy několik pravidel tabulky). Původní přechodová funkce byla definovaná takto:

- Živá buňka s méně jak dvěma živými sousedy zemře na osamělost
- Živá buňka se dvěma či třemi živými sousedy přežije
- Živá buňka s více jako třemi živými sousedy zemře na přemnožení
- Mrtvá buňka s přesně třemi živými sousedy oživne díky reprodukci

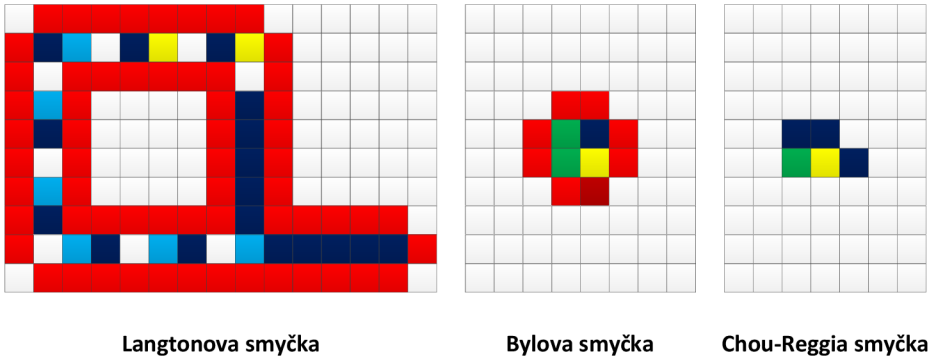
Tato přechodová funkce se také zkráceně zapisuje jako **S23/B3**. Část před lomítkem udává počet okolních buněk, kdy buňka ve stavu „živá“ zachová svůj stav. Část za lomítkem kolik buněk musí být v okolí živých, aby se z mrtvé buňky stala v dalším kroku buňka živá.

Výsledné chování *hry života* se vyznačuje komplexním chaotickým chováním (ukázka na obrázku 2.7), kdy se vytváří celá řada různých struktur. Existují zde statické struktury, cyklické struktury (*oscilátory*) a pohybující se struktury (*kluzáky*). Studuje se chování původního automatu i jeho modifikací, zapsaných jinými čísly v pravidlech (např. **S23/B36** či **S12345/B3**), které mají různá chování.



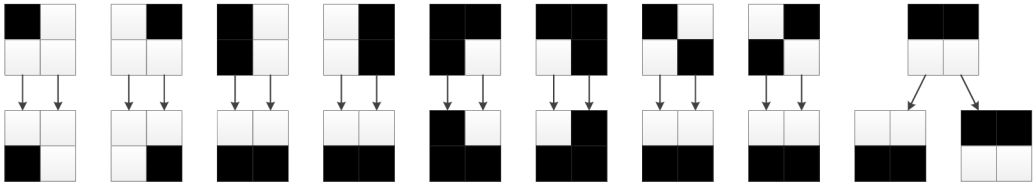
Obrázek 2.7: Hra života

Dalším často zkoumaným jevem je sebe-replikace struktur, hlavně ve 2D vícestavových automatech. Původní takto navržená struktura, Langtonova smyčka [17] (obrázek 2.8), dokázala zkonstruovat vlastní kopii v prostoru vedle tak, že vnitřek smyčky rotoval a postupně vytvářel „výhonek“. Ten se postupem času vždy o 90° zakřivil, až se uzavřel do sebe a vytvořil uzavřenou smyčku. Ta se zase následně zduplikovala do prostoru vedle. V průběhu času byly vyzkoumány smyčky s kratší periodou reprodukce, nebo menší velikostí a počtem potřebných stavů (Bylova smyčka [**?**] či Chou-Reggia smyčka [**?**]).



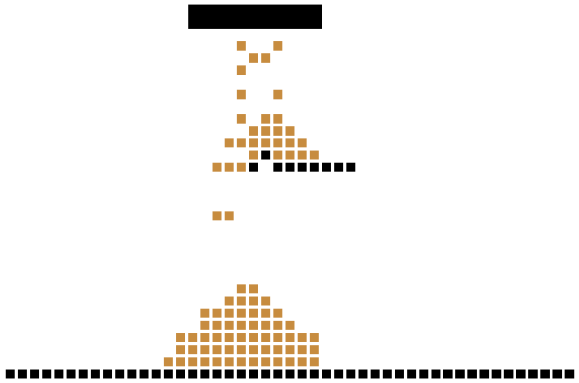
Obrázek 2.8: Ukázky smyček

Jako ukázka použití celulárních automatů v modelování a simulacích může posloužit příklad se jednoduchou simulací sypání písku [8]. Používá se zde upravená přechodová funkce, kdy se postupně střídají liché a sudé 2x2 segmenty a místo určení nového stavu jen jedné buňky se určí nový stav celého segmentu. Vizualizovaná pravidla jsou ukázána na obrázku 2.9.



Obrázek 2.9: Pravidla pro simulaci padání písku

Poslední pravidlo určuje míru „blokování“, kdy segment zůstane ve stejném stavu se zadanou pravděpodobností. Tato pravděpodobnost vyjadřuje „sypkost“ simulovaných částic. Celá simulace pak vypadá jako na obrázku 2.10 (černá = překážka, oranžová = písek)[1]



Obrázek 2.10: Padající písek

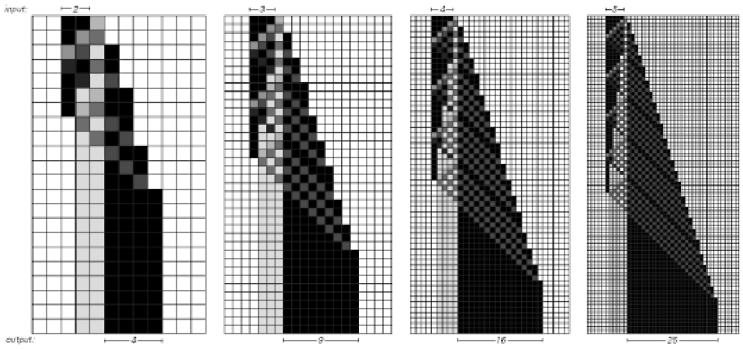
2.5.2 1D CA

Oproti 2D automatům se 1D automaty používají více spíše pro matematické a výpočetní experimenty, zčásti i díky menšímu stavovému prostoru, a tím pádem lepší prozkoumatelnosti. Jeden ze zkoumaných problémů je tzv. „problém většiny“ (z anglického *majority problem*). Ten je zadaný jako 1D binární automat s počátečním stavem, u kterého je cílem aby po konečném počtu kroků byly všechny buňky v takovém stavu, který odpovídá většině v počátečním stavu. Pokud je tedy více buněk ve stavu „1“ než ve stavu „0“ tak by po konečném počtu kroků všechny buňky měly být ve stavu „1“.

Tento problém je dobře prozkoumán a zdokumentován, včetně teoretických studií dokazujících jeho neřešitelnost [16], studií používajících různých modifikací pro vyřešení problému [7, 25] nebo studií řešících problém ve většině případů [9].

Další použití 1D CA lze nalézt pro výpočty. Na obrázku 2.11 je uvedena ukázka výpočtu druhé mocniny čísla pouze pomocí lokálních interakcí v 1D automatu se sedmi stavy navržený Stephenem Wolframem v roce 2002. Vstupní číslo je zakódované jako souvislý blok jednoho stavu o velikosti onoho čísla ukončený blokem jiného stavu. Výsledkem je souvislý blok o velikosti výsledku uvozený blokem stavů o jiné hodnotě.

[1] animováno pomocí http://schuelaw.whitman.edu/JavaApplets/SandPileApplet/



Obrázek 2.11: Výpočet mocniny v 1D automatu [27, str. 639]

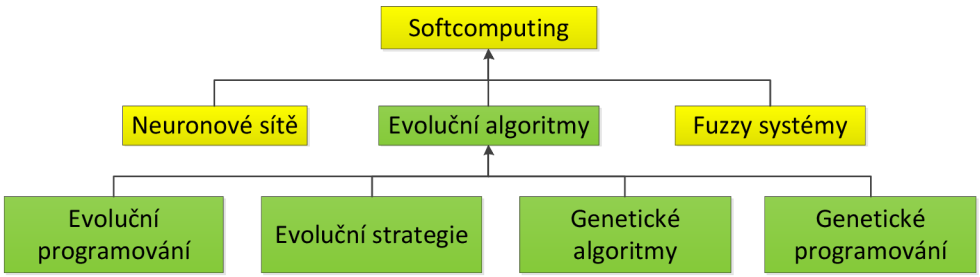
Kapitola 3

Evoluční algoritmy

Evoluční algoritmy je souhrnný název pro stochastické metody prohledávání stavového prostoru obvykle inspirované procesy v přírodě [12] [15, 23]. Podmínkou aplikace je, že musí být možné ohodnotit kvalitu i částečných řešení problému, které částečné řešení je blíže požadovanému výsledku. Fungování evolučních algoritmů je založeno principu evoluce, kdy silnější či lepší jedinci mají větší šanci přežít a rozmnožit se.

V evolučních algoritmech vždy existuje *populace*, což je množina jednotlivých různě kvalitních řešení problému. Tato řešení jsou ohodnocena číslem, které je jim přiřazeno tzv. *fitness funkcí*. V rámci populace se postupem času jednotlivé řešená zlepšují tak, že lépe ohodnocení jedinci mají větší šanci podílet se na replikaci svých částí, než jedinci hůře ohodnocení. Šíření informací o „dobrých“ kusech řešení se v rámci populace děje za pomocí genetických operátorů s obecnými názvy *selekce*, *křížení* a *mutace*.

Na obrázku 3.1 je znázorněno rozdělení evolučních algoritmů na jednotlivé postupy. V dalším textu budou popsány pouze *Genetické algoritmy*, protože byly použity v experimentech.



Obrázek 3.1: Přehled evolučních algoritmů [23, str. 11]

3.1 Algoritmy prohledávání stavového prostoru

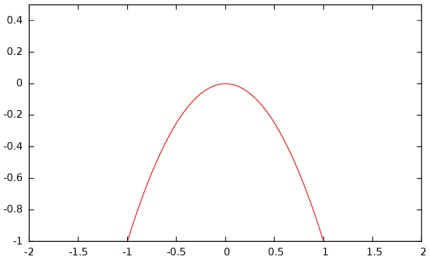
Jako základ pro vysvětlení genetického algoritmu může posloužit slepý prohledávací algoritmus [15] typu *náhodného prohledávání*. Tento algormitmus až do splnění ukončující podmínky (počet iterací, kvalita výsledku, ...) generuje náhodná řešení ze stavového pro-

storu, ohodnotí je, a pokud jsou lepší než stávající nejlepší nalezené řešení, uloží do paměti (Algoritmus 2).

Algoritmus 2 Slepý prohledávací algoritmus

$x_{best} = GenerujNahodneReseni()$
while not $UkoncujiciPodminka$ **do**
  $x = GenerujNahodneReseni()$
  if $Ohodnot(x) > Ohodnot(x_{best})$ **then**
    $x_{best} = x$
  end if
end while

Konkrétním jednoduchým příkladem tohoto algoritmu může být hledání maxima funkce o jedné proměnné (v praxi se pak jedná o funkce mnoha proměnných). Řekněme že chceme najít maximum funkce $f(x) = -x^2$ (obrázek 3.2) u které nebudeme znát její analytické řešení. Dle algoritmu náhodného prohledávání budeme tedy postupovat tak, že například v 1000 cyklech vždy vygenerujeme náhodné číslo v námi zvoleném intervalu (tipneme si, že řešení leží mezi -1000 a 1000), dosadíme do funkce $f(x)$, výsledek porovnáme s uloženým nejlepším dosaženým výsledkem, a pokud je lepší, uložíme si ho. Takto po ukončení iterací získáme výsledek, který bude blízko skutečnému analytickému řešení maxima.



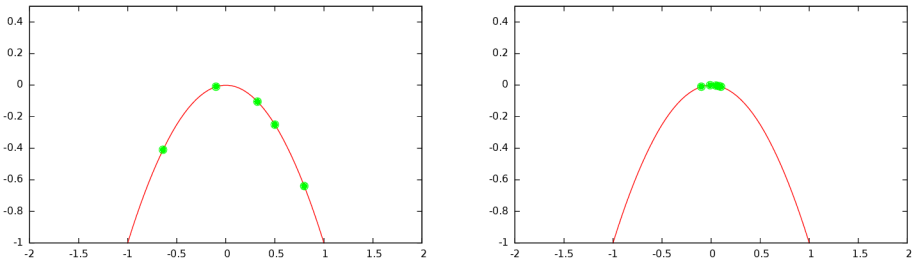
Obrázek 3.2: Příklad maxima funkce

Problémem náhodného prohledávání je špatná škálovatelnost pro zvětšující se stavové prostory, tedy nutnost použít velkého množství iterací. Této vlastnosti se ale i využívá, náhodné prohledávání je mnohdy použito jako referenční algoritmus pro porovnávání rychlosti a efektivity ostatních algoritmů (o kolik rychleji najde algoritmus řešení oproti náhodnému prohledávání). Proto další prohledávací algoritmy využívají různé heuristiky pro generování nových jedinců, či se pokouší odhadnout směr, kterým řešení může ležet (u nich zase hrozí uvíznutí v lokálním maximu). Známými algoritmy jsou např *simulované žíhání* [20] a *horolezecký algoritmus* [21].

3.2 Genetický algoritmus

Genetický algoritmus (algoritmus 3) pracuje s množinou kandidátních řešení nazvanou *populace*. Při použití předchozího příkladu s $f(x) = -x^2$ by populace byla množinou čísel dosazovaných za proměnnou x. V každé iteraci algoritmu nad populací (populace v jednotlivé iteraci se nazývá *generace*) se každý jedinec ohodnotí tzv. *fitness funkcí* (v našem

případě funkce $f(x)$), a jeho hodnota se změní dle genetických operátorů na základě jeho ohodnocení a ohodnocení ostatních členů populace. Typický průběh genetického algoritmu je takový, že původně náhodně vygenerovaná populace se ustálí kolem hledaného maxima (obrázek 3.3, vlevo stav na začátku, vpravo po ukončení).



Obrázek 3.3: Rozložení populace na funkci před a po genetickém algoritmu

Formálněji lze genetický algoritmus popsat pseudoalgoritmem (Algoritmus 3), kde t značí pořadí iterace či generaci a $P(t)$ populaci v generaci t. Vysvětlení ostatních funkcí je v následujícím seznamu:

Algoritmus 3 Základní genetický algoritmus

```
t = 0
P(t) = NahodnaInicializace()
repeat
    Ohodnot(P(t))
    P(t+1) = {}
    repeat
        Rodic_1 = Selekce(P(t))
        Rodic_2 = Selekce(P(t))
        Potomek_1, Potomek_2 = Krizeni(Rodic_1, Rodic_2)
        Potomek_1 = Mutace(Potomek_1)
        Potomek_2 = Mutace(Potomek_2)
        P(t+1) = P(t+1) ∪ {Potomek_1} ∪ {Potomek_2}
    until —P(t+1)— = —P(t)—
    t = t + 1
until UkoncujiciPodminka
```

- **NahodnaInicializace()**: Počáteční populace se inicializuje s náhodnými hodnotami. Také je možné použít heuristiku a vygenerovat již trochu nadějnější kandidáty.
- **Ohodnot(P(t))**: Ohodnotí populaci podle kvality řešení (fitness funkce)
- **Selekce(P(t))**: Vybere z populace jedince, který je použit pro vytvoření jedinců do další generace. Více o tomto operátoru v kapitole 3.2.2. Obecně platí, že lepší jedinci mají větší šanci být vybráni než horší.
- **Krizeni(X, Y))**: Vytvoří dva nové jedince za použití nebo kombinace jedinců vstupních. O tomto operátoru a jeho variantách více v kapitole 3.2.3. Obecně platí, že noví jedinci by měli obsahovat kusy dat z obou vstupních jedinců.

- **Mutace(X)**: Provede malou náhodnou změnu v reprezentaci jedince. Více o tomto operátoru v kapitole 3.2.4.
- **UkoncujiciPodminka** - Podmínka ukončení algoritmu. Často používané podmínky jsou počet generací, absolutní kvalita nejlepšího řešení v populaci či míra zlepšení nejlepšího řešení v populaci od poslední iterace.

3.2.1 Reprezentace řešení

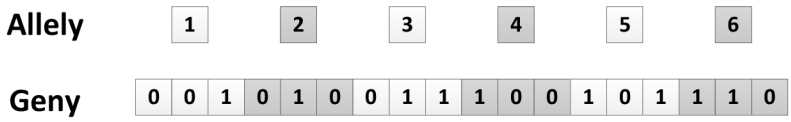
Otázka reprezentace řešení problému je první, kterou je v případě genetického algoritmu třeba řešit. Kandidátní řešení úlohy je obvykle charakterizováno několika parametry (rozměry, váhy). Tato uspořádaná množina parametrů (tj. každá pozice v množině vyjadřuje u všech jedinců stejnou vlastnost) se nazývá *fenotyp* a reprezentuje viditelné vlastnosti jedince. Jednotlivý parametr ve fenotypu se nazývá *alela*. V našem příkladě s maximem jednorozměrné funkce by fenotyp jedince obsahoval jedno číslo $\{x\}$. Pro funkci o třech proměnných by fenotypem byla množina tří alel $\{x, y, z\}$, kterou budeme pro názornost používat v dalších příkladech místo původního příkladu.

Každý fenotyp je pak zakódován do vnitřní reprezentace genetického algoritmu. Tato reprezentace se nazývá *chromozom* (též *genom*, *genotyp*), jehož jednotlivými položkami jsou *geny*. Na tuto vnitřní reprezentaci jsou v průběhu algoritmu aplikovány genetické algoritmy, a tak zvolená reprezentace úlohy velice úzce souvisí s výběrem operátorů.

Vnitřní reprezentace, *genotyp*, se od fenotypu nemusí lišit, ale ve většině případů je záměrem aby se reprezentace lišily a chromozom byl reprezentován například binárním řetězcem. Tato často používaná reprezentace se nazývá *binární reprezentace* a jedná se o řetězec pospojovaný z binárních reprezentací jednotlivých alel z fenotypu. Pro fenotyp $\{1, 2, 3, 4, 5, 6\}$ by binární reprezentace mohla vypadat jako binární řetězec,

001010011100101110

názorně uvedeno na obrázku 3.4.

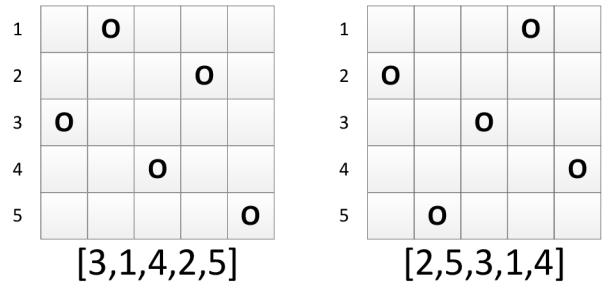


Obrázek 3.4: Ukázka binární reprezentace chromozomu

Pro různé konkrétní problémy existují zakódování (a příslušné genetické operátory), která jsou vhodnější než zmíněné binární zakódování. Jako příklad můžeme uvést problém devíti šachových dam[1], pro který je zvláště vhodné tzv. *permutační kódování* [13]. V něm je každé řešení reprezentováno permutací množiny prvků, v tomto případě permutací pořadových čísel řady (obrázek 3.5), ve které se dáma vyskytuje. Každý prvek reprezentace vyjadřuje jednu řadu. Takto je už přímo způsobem reprezentace zajištěna eliminace řešení, kde jsou dámy v jedné řadě, a efektivně je tak zmenšen prohledávaný stavový prostor o velký

[1] Jedná se o úlohy, kdy je na šachovnici potřeba umístit devět figur dam tak, aby se navzájem neohrožovaly

počet nefungujících řešení. Ve fitness funkci se pak již můžeme zaměřit na ohodnocování pouze na diagonálních konfliktů.



Obrázek 3.5: Příklad permutační reprezentace problému 5 dam [13, str. 94]

3.2.2 Operátor selekce

Operátor selekce [15, 23] (inspirovaný přirozeným výběrem) slouží k vybrání jedinců, kteří se budou „rozmnožovat“. Jeho obecnou vlastností je, že lépe ohodnocení (fitness funkcí) jedinci mají větší šanci být vybráni. K dosažení tohoto cíle se většinou používá jedna ze tří metod - *ruletový výběr*, *selekce uspořádáním* a *turnajový výběr*.

Ruletový výběr získal svůj název podle podobnosti s hazardní hrou jménem *ruleta*. Každý jedinec má na ruletovém kole svoji výseč, větší pro jedince s vyšším ohodnocením, a naopak menší pro jedince s ohodnocením nižším. Pomyslná ruletová kulička má tak větší šanci zastavit se na větších výsečích, a tím pádem jedinci s vyšším ohodnocením mají větší šanci být vybráni. Kvůli tomu, že velikost výseče je proporční vzhledem k velikosti ohodnocení, tento výběr se také někdy nazývá *proporční*.

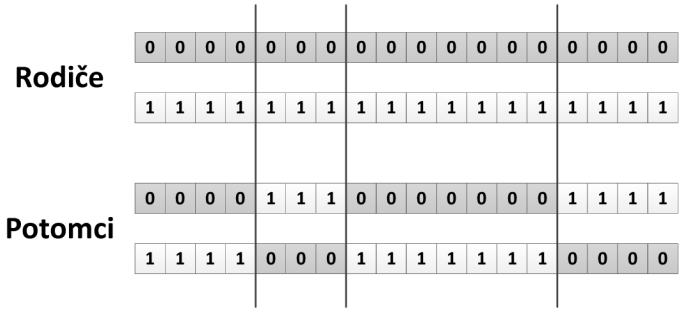
Dalším typem výběru je výběr pomocí uspořádání. Jedinci populace jsou seřazeni dle svého ohodnocení, a následně je podle pořadí přiřazena pravděpodobnost výběru dle funkce. Tato funkce, může být různá většinou se používá buď lineární (šance být vybrán roste lineárně s lepším umístěním) nebo exponenciální (šance být vybrán roste exponenciálně s lepším umístěním).

Posledním běžně používaným typem výběru je turnajová selekce. Ta je inspirována turnaji či souboji tak, že z populace je náhodně vybráno k jedinců (k se nazývá *velikost turnaje*), kteří mezi sebou „soupeří“ tak, že porovnávají svoje ohodnocení. Vybrán je ten, který v turnaji zvítězil - měl ze zúčastněných největší ohodnocení. Tento typ bývá často používán kvůli své rychlosti - není třeba populaci v každém kroku řadit dle ohodnocení jedinců.

3.2.3 Operátor křížení

Operátorem křížení [23, 3] (či rekombinace) rozumíme funkci, která ze dvou jedinců vytvoří dva nové jedince (*potomky*), z nichž každý obsahuje části obou vstupních jedinců (*rodičů*). Nejčastěji je používaný jeden z typů „zlomového“ křížení, kdy se v chromozomu na náhodném místě vytvoří zlom (či více), a genetická informace se „prohodí“ (názorný příklad tříbodového křížení na obrázku 3.6).

Podle počtu bodů zlomu rozeznáváme *jednobodové křížení* (existuje jeden zlom), *vícebodové* (existuje více zlomů, ale malý počet) a *uniformní křížení* (každá dvojice prvků genu má samostatnou pravděpodobnost že bude prohozena - nejčastěji 50%.)



Obrázek 3.6: Ukázka vícebodového křížení [23, str. 18]

Názory na tento operátor se obecně různí, někteří vyzdvihují jeho přínos k výměně informací mezi jedinci, někteří ale tvrdí že zbytečně rozbíjí jednotlivá řešení a tak ho z algoritmu vynechávají. Experimenty ověřující jeho přínos jsou také součástí této práce.

3.2.4 Operátor mutace

Operátor mutace [23, 3] je velice důležitým prvkem genetického algoritmu, jedná se o zdroj nových informací pro populaci (křížení jen kombinuje informace již v populaci existující). Mutace by měla být prováděna s malou pravděpodobností a neměla by dělat velké změny v genu. Každá úloha, reprezentace a velikost chromozomu má různé „ideální“ nastavení pravděpodobnosti, typu a velikosti mutace. Pokud je pravděpodobnost a velikost příliš malá, genetický algoritmus nemá dostatek nových informací a konverguje buď pomalu, nebo vůbec. Příliš velká pravděpodobnost a velikost mutace se zase vyznačuje „rozbíjením“ bloků, genetický algoritmus není schopen v populaci udržet kusy dobrého řešení.

Smyslem tohoto operátoru je náhodně trochu pozměnit genotyp jedince. V binární reprezentaci se často používá inverze náhodného bitu. V číselné reprezentaci (řada za sebou jdoucích celých čísel) můžeme například náhodnému číslu nastavit novou (validní) náhodnou hodnotu. V permutační reprezentaci se zase často používá prohození dvou čísel v genu, což zachová vlastnosti reprezentace (vygenerování náhodného čísla by permutační vlastnosti reprezentace porušilo).

Všechna místa v genotypu nemají stejnou míru dopadu na řešení, mutace může mít pro ohodnocení jedince jak zanedbatelný, tak fatální důsledek. Například mutace nejnižšího versus nejvyššího bitu v reprezentaci celého čísla. První situace číslo změní nepatrně, v druhém drasticky.

I důležitost jednotlivých alel může být různá. Uvažme funkci $f(x, y) = 10000x + 10y$, kde parametr x má na hodnotu funkce mnohem větší dopad než parametr y.

Kapitola 4

Podmínková pravidla

Běžně jsou pravidla CA určena pomocí tabulky (viz kapitola 2.3), kde každý řádek odpovídá jedné z permutací okolí. Pro zvětšující se okolí buňky a zvyšující se počet stavů velikost tabulky poměrně hodně roste. Pro 1D binární automat s 3-okolím máme tabulku o $2^3 = 8$ řádcích a pokud uvážíme stejný automat s pěti stavy, dostaneme $5^3 = 125$ řádků. To ještě nevypadá tak špatně, ale pro takový 2D binární automat s 9-okolím dostaneme už $2^9 = 512$ a pro stejný automat s 3 stavy dokonce $3^9 = 19638$ řádků.

Často se stává, že „funkční“ sada pravidel zabírá jen část tabulky, a její zbytek buď zachovává stavy, nebo se v běhu automatu prakticky neuplatňuje, protože stav na který čekají v automatu nemůže nastat. Tento problém se nejvíce projevuje při pokusech o evoluční návrhy CA, kdy evoluční algoritmus má problémy provést v chromozomu tvořeném řetězcem tabulkové reprezentace nějakou „funkční“ změnu. Kromě existence velkých hluchých částí tabulky je často potřeba pro projevení určité funkční viditelné změny udělat změnu na více místech tabulky.

K popsaným problémům se ještě přidává obrovský stavový prostor CA. Pro klasický binární 2D automat s 9-okolím existuje 2^{512} různých přechodových funkcí. Přidáme jeden stav na celkové 3 možné, a dostaneme 2^{19638} možných funkcí.

Pro efektivní (automatický) návrh přechodových funkcí je proto třeba hledat lepší způsoby reprezentace přechodové funkce. Jedním z takových pokusů je například využití instrukcí pro reprezentaci funkce [5]. Dalším pokusem o vylepšení reprezentace a schopností prohledávat stavový prostor je navržený přístup jménem *podmínková pravidla* [6].

4.1 Popis CMR

Podmínková pravidla jsou založena na myšlence určitého seskupení řádků tabulky tak, že jeden řádek efektivně pokrývá funkčnost více původních řádků. V tabulkovém formátu (viz kapitola 2.3) lze pravidla chápat tak, že u každé hodnoty je podmínka „rovná se“ (==, viz obrázek 4.1) a pravidlo se aplikuje pouze pokud tato podmínka vrátí true při porovnání hodnoty z okolí buňky a hodnoty pravidla.

X_0	X_1	X_2	Y
==0	==0	==0	0
==0	==0	==1	1
==0	==1	==0	0
==0	==1	==1	0
==1	==0	==0	0
==1	==0	==1	1
==1	==1	==0	1
==1	==1	==1	1

Obrázek 4.1: Tabulková reprezentace pravidla za pomocí porovnávací funkce „==“

Podmínková pravidla nahrazují toto pomyslné „rovná se“ různými dalšími funkcemi, například „!=“, „¿=“, „¡=“ (viz obrázek 4.2). Takto upravené pravidlo je schopno zastávat funkci více konvenčních tabulkových pravidel. Celková přechodová funkce je poté charakterizována uspořádanou množinou jednotlivých pravidel. Při vyhodnocení se sekvenčně testuje zda aktuální okolí splňuje podmínkové pravidlo. Pokud splňuje, je vrácena nová hodnota pravidla, pokud ne, otestuje se další pravidlo. Pokud nevyhovuje žádné pravidlo, hodnota buňky zůstává stejná. Podmínkové pravidlo vyhovuje tehdy, pokud testy pro všechny body okolí v pravidlu vrátí hodnotu „true“.

X_0	X_1	X_2	Y
==1	>=0	*	1
>=1	==2	<=2	2
==2	*	*	0

(a)

X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	Y
==1	*	>=0	!=0	*	==1	>=1	<=2	*	1
>=1	==2	<=2	==0	*	*	==1	<=2	==0	2
==2	*	*	==1	==0	*	==1	>=2	!=0	1

(b)

Obrázek 4.2: Podmínková pravidla pro 3-okolí (a) a 9-okolí (b)

Navrhovaným postupem lze většinu tabulkových reprezentací zapsat velmi krátkou formou. Takto zapsaná pravidla jsou vysoce efektivní, není potřeba explicitně uvádět pravidla na zachování hodnot. Zápisem podmínkových pravidel vlastně vyjadřujeme, jak moc se chování automatu liší od chování „ponechej všechny stavy beze změny“.

Pokud uvážíme přechodovu funkci pro 1D automat se třemi stavy vyjádřenou sadou tří

podmínkových pravidel z obrázku 4.2, pak by se pro okolí $\{2, 0, 1\}$ uplatnilo třetí pravidlo (funkce „*“ znamená, že na daném místě může být jakákoli hodnota, vždy vrátí „true“). Pro okolí $\{0, 2, 1\}$ neplatí žádné z dané sady pravidel, a proto by následující hodnota byla 2, hodnota buňky by se nezměnila.

4.2 Tabulkové vyjádření podmínkových pravidel

Vyjádření přechodové funkce pomocí podmínkových pravidel je ekvivalentní jako vyjádření funkce tabulkou, a mezi těmito formami je možné převádět. Tabulka má mnoho různých vyjádření pomocí podmínkových pravidel, ale sada podmínkových pravidel má vždy jedno tabulkové vyjádření. Převod obnáší iteraci přes všechny řádky tabulky a určení nové hodnoty pro toto okolí. Ukázka tohoto algoritmu pro 1D automat se třemi stavy 3-okolím je ukázána na algoritmu 4.

Algoritmus 4 Převod podmínkových pravidel do tabulkové podoby pro 3-okolí a 3 stavy

```
table = []
for (x_0 = 0; x_0 < 3; x_0++) do
    for (x_1 = 0; x_1 < 3; x_1++) do
        for (x_2 = 0; x_2 < 3; x_2++) do
            table[] = getCMRfor(({x_0, x_1, x_2}))
        end for
    end for
end for
return table
```

4.3 Výhody podmínkových pravidel

Jak bylo ukázáno v [6], podmínková pravidla lze úspěšně a efektivně použít pro evoluční návrh poměrně složitých problémů, jako je transformace či replikace vzoru ve 2D binárním automatu. V dosud publikovaných experimentech dokázal evoluční algoritmus mnohem snáze najít řešení problému, než v případě evoluce nad tabulkovou reprezentací. Další způsoby využití, srovnání s evolucí nad tabulkou a optimalizace postupů jsou součástí této práce v dalších kapitolách.

Kapitola 5

Implementace

Implementace experimentů byla provedena v jazyce *Java*. Byly zvažovány a zkoušeny i různé frameworky pro genetické algoritmy typu *GALib*, ale nakonec nebyly použity. Využity byly pouze knihovny s obecnými funkcemi a nástroji jako je např. *Apache Commons*. Pro vizualizace byly použity programy *Gnuplot* (pro grafy) a *Microsoft Visio* (pro ostatní diagramy). Dokumentace byla napsána v prostředí LaTeXs použitím knihovny *czechiso* pro citace.

5.1 Genetický algoritmus

Pro implementaci genetického algoritmu byl zvolen základní algoritmus popsaný v kapitole 3.1. Chromozom je reprezentován polem datového typu **byte**, nebylo potřeba využívat celého rozsahu hodnot typu **int**, to snížilo paměťovou náročnost algoritmu (nejedná se tedy o binární reprezentaci chromozomu). Oproti základnímu algoritmu byla zvolena modifikace, nepovolit v populaci více jedinců se stejným chromozomem. V takovém případě se jedinec do populace nepřidá, zahodí, a dále se pokračuje ve výběru, křížení a mutaci jedinců do nové populace, dokud není dosaženo velikosti původní populace.

V experimentech bylo uvažováno (a pro tyto účely implementováno) jednobodové křížení s pravděpodobností 50%. Mutace je použita stejně jak je popsána v článku [6], tj. vybráno je n náhodných čísel v chromozomu, každé s individuální pravděpodobnosti 50% že bude vygenerována nová náhodná validní hodnota. Jako operátor selekce byla použita turnajová selekce, stejně jako v původních experimentech. Velikost turnaje je také součástí experimentů, jinak je nastavena na defaultní hodnotu 4, navrženou v článku. Velikost populace je ve většině experimentů 10, oproti v článku použitých 8.

5.2 Celulární automat

Celulární automaty použité ve fitness funkcích (viz další kapitoly) jsou implementovány jako synchronní uniformní automaty (popsané v kapitole 2) s nulovými okrajovými podmínkami. Použity jsou 1D automaty s 3-okolím a 2D automaty s 9-okolím.

Pro 1D automat je dále implementována optimalizace pro zrychlení běhu autopmatu. V každém kroku je uložen nejmenší a největší index buňky, ve které proběhla změna stavu. Cyklus dalšího kroku neprobíhá přes všechny stavy automatu, ale pouze přes ty od indexu nejmenší minulé změny mínus jedna, do indexu největší minulé změny plus jedna (pro větší

okolí než 3-okolí by se zvětšila i tato konstanta). Neprojité buňky zůstanou stejné jako v minulém stavu.

Tato optimalizace vychází z myšlenky, že pokud se stav nezměnil, v dalším kroku se taktéž nezmění, pokud se nezměnila buňka v jeho okolí. V 3-okolí je okolí buňky definováno jako jedna buňka nalevo a jedna buňka napravo. Testy ukázaly, že tato optimalizace experimenty zrychlila znatelně.

5.3 Podmínková pravidla

Podmínková pravidla jsou implementována v souladu s popisem v kapitole 4, jedná se o uspořádanou množinu jednotlivých pravidel. Pravidla mají pole funkcí a pole hodnot s velikostí shodnou s velikostí okolí daného automatu (3 nebo 9). Dále obsahují novou hodnotu pro případ, že pravidlo odpovídá okolí, a funkci která toto vyhodnocuje. Tato funkce vrátí „true“ pouze pokud pro všechny hodnoty z okolí platí $f_i(x_i, v_i) == true$, kde x_i je hodnota i-té buňky z okolí, v_i je hodnota i-té položky v pravidle a f_i je funkce i-té položky pravidla (viz popsané funkce v tabulce 5.1).

Index	Značka	význam	ignoruje hodnotu pravidla
0	EQ	==	ne
1	NEQ	! =	ne
2	LEQ	<=	ne
3	GEQ	>=	ne
4	*	vždy true	ano
5	ID	> 0	ano
6	NEG	== 0	ano

Tabulka 5.1: Tabulka významu podmínkových funkcí

Oproti popsanému algoritmu je navíc implementována optimalizace v podobě mezipaměti. Pokud již pro nějakou sadu pravidel byla vypočítána hodnota pro nějaké okolí, je tato hodnota uložena do mezipaměti. Při každém dalším požadavku na výpočet nové hodnoty pro toto okolí se místo nového výpočtu vrátí hodnota již jednou vypočítaná a patřící k tomuto okolí.

Popsaná implementace je hlavní sjednocující vnitřní implementací podmínkových pravidel v experimentech, která slouží i jako mezikrok pro převod mezi různými formáty. Jako vstupní formát je často použit textový řetězec tohoto formátu

$<= 0| == 0| <= 0|0; *1| >= 0| <= 0|0; *0| == 1| == 0|1;$

Pravidla jsou oddělena středníkem a části pravidel svislítkem. Příklad je konkrétně pro 1D automat s 3-okolím, je vidět že každé ze tří pravidel obsahuje tři podmínkové části a jednu část pro novou hodnotu.

Druhým formátem je reprezentace polem typu **byte** pro genetický algoritmus. Ve skutečnosti se jedná o dva formáty, které jsou využity v experimentech. Obě varianty reprezentují množinu pravidel jako za sebou poskládaná serializovaná pravidla. Liší se pouze v tom, jak serializují jednotlivá pravidla.

5.3.1 Reprezentace pro GA

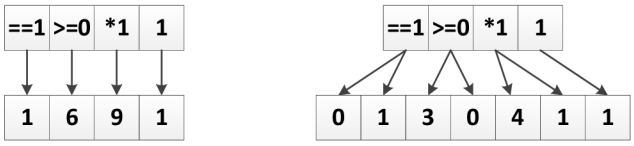
První varianta (obrázek 5.1.a), založená na popisu v originálním článku [6], vyjadřuje podmínku a hodnotu dohromady jedním číslem tak, jak je uvedeno ve vzorci 5.1.

$$x = f_i * v_{count} + v_i \tag{5.1}$$

$$f_i = x / v_{count} \tag{5.2}$$

$$v_i = x \bmod v_{count} \tag{5.3}$$

Platí, žef_i je index i-té funkce v tabulce funkcí (obrázek 5.1), v_{count} je počet možných hodnot stavů a v_i je i-tá hodnota v pravidle. Při zpětném převodu je pak index funkce získán jako celočíselné dělení (vzorec 5.2) a hodnota jako zbytek celočícelného dělení (vzorec 5.3).



(a) (b)

Obrázek 5.1: Porovnání variant implementací serializace podmínkového pravidla do chromozomu

V druhé variantě (obrázek 5.1.b) je podmínková část zakódována vždy dvěma celými čísly. První číslo odpovídá indexu funkce v tabulce, druhé hodnotě. Tato druhá varianta je navržena v této práci a také použita pro experimenty. Předpokladem bylo, že v původní variantě může mít genetický algoritmus problém s mutací a křížením dělat změny pouze ve funkci nebo v hodnotě. Na druhou stranu je pro tuto reprezentaci složitější implementace operátoru mutace, protože na různých místech chromozomu jsou validní různé rozsahy hodnot. Existují zde buď intervaly $(0, f_{max})$ pro funkce, nebo $(0, value_{max})$ pro hodnoty.

Kapitola 6

Experimenty s 2D celulárními automaty

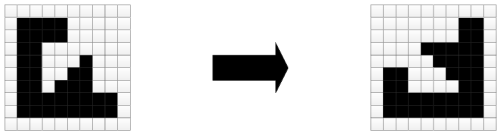
Zpočátku byly provedeny experimenty na základě transformace vzoru z článku [6] ve 2D automatu s parametry, které byly navrženy a zkoušeny tamtéž. Parametry jsou shrnuty v tabulce 6.1, ze které budou v této kapitole vycházet všechny experimenty a uváděny budou vždy pouze odlišné parametry.

název	hodnota
velikost populace GA	10
počet generací GA	500 000
počet zmutovaných čísel	6
pravděpodobost mutace čísla	50%
typ křížení	jednobododové
pravděpodobnost křížení	50%
velikost turnaje	4
startovní populace	Náhodná
typ CA	2D
maximální počet kroků CA	16
počet podmínkových pravidel	22
počet stavů CA	2
podmínkové funkce	EQ, NEQ, GEQ, LEQ, WILD
typ reprezentace pravidla	jedním celým číslem
počet experimentů	100

Tabulka 6.1: Tabulka základních parametrů pro 2D experimenty

Vzor pro transformaci byl zvolen stejný jako v článku, kdy je za úkol stav na obrázku 6.1.a pomocí lokálních přechodových pravidel transformovat na stav na obrázku 6.1.b. Oproti původní transformaci bylo vypuštěno prázdné místo v podobě tří prázdných řádků nad vzorem, a tak zmenšen celkový vzor.

Cílem experimentů této kapitoly je zopakování původního experimentu a následná optimalizace parametrů genetického algoritmu. Získáme tak pokud možno obecnější znalosti o chování podmínkových pravidel v evoluci a které bude možno využít na vyřešeních jiných úloh.



(a) (b)

Obrázek 6.1: Transformace 2D vzoru z (a) na (b)

6.1 Fitness funkce

Fitness funkce pro tuto sadu experimentů má podobu porovnání podoby cílového vzoru a aktuálního. Pro všechny buňky aktuálního stavu CA se porovná, zda odpovídají hodnotě buňek cílového stavu na stejném místě. Celkové skóre stavu je pak číslo z intervalu $< 0.0, 1.0 >$, kde 1.0 odpovídá perfektní shodě s cílovým obrazem a 0.0 odpovídá inverznímu vzoru (všechny buňky mají opačnou hodnotu). V reálné situaci tedy budeme dostávat hodnoty větší než 0.5 (například počáteční stav je vzhledem k cílovému stavu ohodnocen hodnotou 0.68, což znamená, že 68% buněk má správný stav).

Toto ohodnocení je spočítáno pro každý krok automatu od počátečního stavu až do nastaveného maxima počtu kroků, které má CA běžet. Hodnota celé fitness funkce je pak hodnota nejlépe ohodnoceného stavu v rámci celého běhu automatu.

Genetický algoritmus končí ve chvíli, kdy je dosaženo fitness 1.0 nebo pokud je překročen nastavený maximální počet generací pro experiment. Fitness funkce nijak nezvýhodňuje řešení, které k transformaci potřebuje méně kroků, počítá se jakýkoli úspěch v rámci nastaveného počtu kroků.

6.2 Ověření původního experimentu

Před vlastními experimenty bylo potřeba ověřit, zda GA s navrženými parametry skutečně dokáže najít řešení, a rak ověřit, zda je implementace provedena správně a nikde není žádná zásadní chyba.

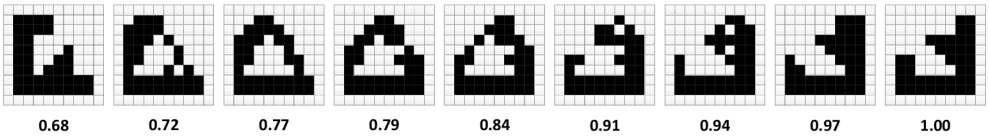
Výsledky tohoto experimentu jsou zapsány v tabulce 6.2, kde sloupec *průměr* značí **průměrný počet generací úspěšných řešení potřebných k výsledku**. Data k původnímu experimentu jsou převzata z výsledků v článku[6]. Data pro nový experiment jsou data z provedeného experimentu s pokud možno co nejvíc stejnými parametry.

	úspěšnost	prům. generací
Původní experiment	64%	158 319
Nový experiment	94%	109 451

Tabulka 6.2: Výsledky úvodního experimentu. Sloupec *prům. generací* vyjadřuje průměrný počet generací potřebný k evoluci úspěšných řešení.

Úvodním experimentem se podařilo dosáhnout mnohem lepšího výsledku než v původním experimentu. Odlišnost v úspěšnosti a počtu generací je pravděpodobně dosažena odlišnou fitness funkcí tak, že je pro transformaci použit menší efektivní prostor (popsané odstranění prázdných řádků). Proto bude pro další účely jako referenční sloužit tento experiment. Pravděpodobně přispívá i rozdílná velikost populace, z původních 8 jedinců na

10 jedinců používaných v těchto experimentech. Ukázka průběhu transformace jednoho z úspěšných řešení je pak na obrázku 6.2.



Obrázek 6.2: Ukázka průběhu výsledku s hodnotami fitness funkce v jednotlivých krocích

6.3 Zakódování chromozomu

Prvním novým experimentem bylo porovnání dvou různých zakódování podmínkových pravidel do chromozomu popsaných v kapitole 5.3. První popsaná implementace slučuje podmínkovou funkci a hodnotu do jednoho celého čísla (budeme nazývat *reprezentace jedním číslek*), druhá obsahuje jedno celé číslo pro podmínkovou funkci a jedno pro hodnotu (budeme nazývat *reprezentace dvěma čísly*). Parametry experimentu jsou shrnuty v tabulce 6.3.

název	hodnota
počet zmutovaných čísel	*parametr experimentu*
typ reprezentace pravidla	*parametr experimentu*

Tabulka 6.3: Parametry experimentu, které se liší od úvodních

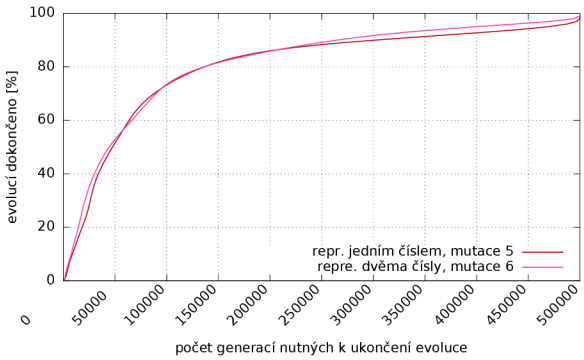
Vzhledem k tomu, že v druhém případě je délka chromozomu skoro dvojnásobná, nelze přímo srovnávat výsledky pro stejnou míru mutace. Pro obě varianty tedy bylo vyzkoušeno více různých měr mutace. Tím získáme pravděpodobnou ideální míru mutace pro oba případy a budeme moci porovnat nejlepší řešení od každé.

Výsledky experimentu jsou uvedeny v tabulce 6.4. Lze z nich vyčíst, že s jinou mírou mutace než byla v původním experimentu, je pro navrženou velikost množiny podmínkových pravidel (22), tj. i pro délku genomu, možné dosáhnout mnohem lepších výsledků. Konkrétně pro menší mutace než 6 dostaneme úspěšnost větší než 90%, s obrovským snížením průměrného počtu potřebných generací.

typ reprezentace	mutace	úspěšnost	prům. generací
jedním číslem	3	95%	98 806
jedním číslem	4	96%	88 241
jedním číslem	5	95%	73 543
jedním číslem	6	64%	158 319
dvěma čísly	4	92%	82 536
dvěma čísly	5	98%	77 159
dvěma čísly	6	97%	75 814
dvěma čísly	7	97%	92 285
dvěma čísly	8	80%	155 460

Tabulka 6.4: Tabulka výsledků experimentu. Sloupec *prům. generací* vyjadřuje průměrný počet generací potřebný k evoluci úspěšných řešení.

Pokud porovnáme nejlepší možnosti obou zakódování (bráno dle počtu generací, úspěšnost se liší minimálně) a složení jejich výsledků, dostaneme graf na obrázku 6.3. Ten zobrazuje, kolik procent výsledků bylo hotovo do učité generace. Z grafu je patrné, že složení výsledků se moc neliší, a pokud zvolíme pro každé zakódování odpovídající míru mutace, zakódování velkou nehraje roli. Vzhledem k mírně zlepšeným mírám úspěšnosti bude v dalších experimentech použito zakódování dvěma čísly.



Obrázek 6.3: Generační složení výsledků experimentu

Na základě grafu se složením výsledků upravíme do příštích experimentů maximální počet generací. Bude nastaven na 200000, protože z grafu je vidět, že poté už evoluce přináší jen malý úspěch. Do tohoto bodu je vytvořeno nad 80% řešení, a výrazně tak snížíme čas běhu jednotlivých experimentů.

6.4 Míra křížení

Dalším experimentem se pokusíme zjistit, zda křížení má nějaký vliv na kvalitu či rychlost výsledků. Existuje totiž domněnka, že křížení dobré stavební bloky spíše rozbíjí. Parametry jsou nastaveny dle tabulky 6.5.

název	hodnota
počet generací GA	200 000
reprezentace pravidla	dvěma čísly
pravděpodobnost křížení	*parametr experimentu*

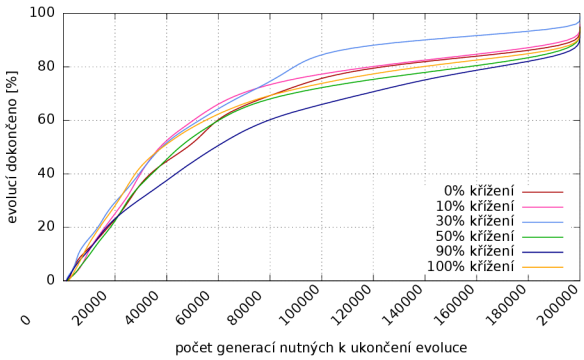
Tabulka 6.5: Parametry experimentu, které se liší od úvodních

Tabulka s výsledky 6.6 ukazuje, že míra křížení nemá žádný kritický vliv na výsledky. Vysledovány byly spíše menší odchylky, konkrétně u hodnoty 30%. Pro odstranění náhodných vlivů bylo provedeno ještě několik dalších běhů a výsledky z tabulky pro 30% byly potvrzeny. Zdá se tedy, že ideální míra křížení je cca třetina případů, a nebo křížení vypustit úplně a ušetřit tak výpočetní čas ke křížení potřebný.

Křížení	prům. generací	úspěšnost
0%	68 438	87%
10%	63 684	87%
20%	69 414	84%
30%	55 118	93%
40%	68 204	84%
50%	72 621	86%
60%	78 516	83%
70%	72 611	85%
80%	74 008	83%
90%	81 074	82%
100%	67 497	84%

Tabulka 6.6: Tabulka výsledků křížení. Sloupec *prům. generací* vyjadřuje průměrný počet generací potřebný k evoluci úspěšných řešení.

Graf generačního složení výsledků (obrázek 6.4) s vybranými výsledky ukazuje, že toto 30% křížení získalo náskok oproti ostatním až v případech, které potřebovaly více generací pro úspěšný vývoj (cca od 80000 výše). Těsně druhém místě je pak 10% křížení, které už však není moc rozlišitelné od ostatních. Výrazně špatně oproti zbylým ostatním si pak stojí 90% křížení.



Obrázek 6.4: Generační složení výsledků experimentu

Na základě těchto výsledků budeme pro další experimenty uvažovat buď 30% křížení, a nebo žádné.

6.5 Množina podmínkových funkcí

V původním experimentu byly použity pouze funkce EQ, NEQ, GEQ, LEQ, WILD pro vyjádření podmínkových pravidel. Proto bylo třeba vyzkoušet, zda neexistuje nějaká lepší množina kombinace funkcí. V experimentu budou vyzkoušeny kombinací uvedených funkcí plus s dvěmi novými funkcemi (jednotlivé funkce jsou popsané v tabulce 5.1 v kapitole 5.3).

První novou funkcí je funkce identity (*ID*), která testuje zda je hodnota buňky rovná hodnotě 1, tj. ignoruje hodnotu pravidla. Pro pozdější vícestavové automaty je definovaná jako „hodnota buňky je rozdílná od nuly“.

Druhou novou funkcí je funkce negace identity (*NID*), která testuje zda je hodnota buňky rovná hodnotě 0. Pro vícestavové automaty je pak definovaná jako „hodnota buňky je rovna nule“. Ostatní parametry experimentu jsou uvedeny v tabulce 6.7.

název	hodnota
počet generací GA	200 000
pravděpodobnost křížení	30%
reprezentace pravidla	dvěma čísly
podmínkové funkce	*parametr experimentu*

Tabulka 6.7: Parametry experimentu, které se liší od úvodních

Následující tabulka 6.8 shrnuje výsledky experimentu, a je seřazena dle úspěšnosti sestupně, ne podle pořadí experimentů. Dále je zde zvýrazněna původní použitá množina funkcí.

Použité podmínkové funkce							prům. generací	úspěšnost
EQ	NEQ			GEQ	LEQ		35 090	97%
EQ	NEQ		NID	GEQ	LEQ		44 286	96%
EQ				GEQ	LEQ		42 051	92%
EQ			NID	GEQ	LEQ		48 137	91%
	NEQ			GEQ	LEQ		45 503	89%
EQ	NEQ			GEQ	LEQ	WILD	46 361	87%
EQ	NEQ	ID	NID	GEQ	LEQ		44 770	86%
			NID	GEQ	LEQ		55 384	86%
EQ	NEQ	ID	NID	GEQ	LEQ	WILD	46 284	85%
		ID	NID	GEQ	LEQ		44 692	84%
EQ	NEQ	ID		GEQ	LEQ		51 903	83%
		ID	NID	GEQ	LEQ	WILD	54 154	82%
EQ					LEQ		44 190	79%
EQ		ID		GEQ	LEQ		64 069	78%
EQ				GEQ			56 756	76%
				GEQ	LEQ		50 125	75%
		ID		GEQ	LEQ		66 539	62%
EQ	NEQ			GEQ			52 969	60%
		ID	NID				51 378	8%
EQ	NEQ						—	0%

Tabulka 6.8: Tabulka výsledků podmínkových funkcí. Sloupec *prům. generací* vyjadřuje průměrný počet generací potřebný k evoluci úspěšných řešení.

Je vidět, že oproti původní množine existují lepší i horší kombinace funkcí, respektive původní množina je jedna z těch lepších. Zajímavé je, že se na lepších místech neumístilo žádné řešení s funkcí WILD. Na tento fenomén ukazují i další různé předběžné experimenty, kdy se zdálo že tato funkce výsledky spíše kazí. Dalším zajímavým faktem je úspěch funkce negace identity (NID), u které se zdá, že dost často množině funkcí prospívá. Pro zkoumaný problém je pravděpodobně výhodné mít funkci „buňka není nulová“.

Nejlepších výsledků pak bylo dosaženo pouze odstraněním funkce WILD z původní množiny, a s touto upravenou množinou budou prováděny další experimenty.

6.6 Velikost turnajové selekce

Dalším testovanýn parametrem byla zvolena velikost turnaje v selekci. V původním navrženém postupu byla velikost turnaje 4, což bylo nejspíš myšleno jako polovina populace (ta čítala 8 jedinců). Pokud by původní předpoklad byl správný, mělo by platit, že nejlepší velikostí turnaje a tedy selekčního tlaku, bude pro náš případ hodnota okolo čísla 5. Experiment byl proveden s parametry z tabulky 6.9.

název	hodnota
počet generací GA	200 000
pravděpodobnost křížení	0%
podmínkové funkce	EQ, NEQ, GEQ, LEQ
reprezentace pravidla	dvěma čísly
velikost turnaje	experiment

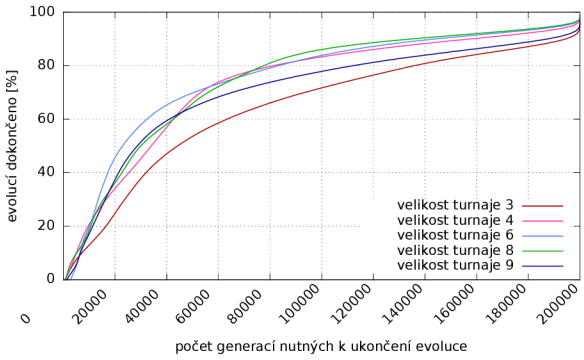
Tabulka 6.9: Parametry experimentu, které se liší od úvodních

Jak ukazují výsledky v tabulce 6.10, na velikosti turnaje až tolik nezáleží, pokud je dostatečně velká (v našem případě > 3). Dá se ale vypozorovat určitý trend, že je mírně lepší volit sudé velikosti turnaje.

velikost turnaje	úspěšnost	prům. generací
2	4%	160 273
3	88%	51 924
4	93%	40 108
5	95%	49 344
6	94%	37 363
7	89%	43 719
8	94%	38 808
9	90%	41 773
10	91%	44 568

Tabulka 6.10: Tabulka výsledků turnajové selekce. Sloupec *prům. generací* vyjadřuje průměrný počet generací potřebný k evoluci úspěšných řešení.

Pokud si zobrazíme generační skladbu vybraných výsledků (obrázek 6.5), nelze vypozorovat žádnou větší odchylku hodnot větších než 3, kromě poměrně rychlého nástupu hodnoty 6. Není ovšem tak významná, aby se jednalo o výrazné zlepšení. Na základě výše uvedené tabulky a grafu bude v dalších experimentech používána velikost turnaje původních 4 nebo nových 6.



Obrázek 6.5: Generační složení výsledků experimentu

6.7 Počáteční populace

Pro genetický algoritmus nemusí být výhodné pokaždé začínat s náhodně vygenerovanou populací. Místo toho by mohlo být pro určité problémy vhodné, začínat s populací jejíž geny budou nulové. Speciálně u podmínkových pravidel, kdy nulová pravidla znamenají pravidla, která nic nedělají. Mohlo by tak být snažší najít řešení, když by genetický algoritmus vždy jen přidával chování do prázdných, stav neměnících pravidel. Místo toho, aby se snažil modifikovat již nějak fungující systém pravidel. Na základě tohoto předpokladu byl realizován experiment s parametry v tabulce 6.11.

název	hodnota
počet generací GA	200 000
pravděpodobnost křížení	0%
reprezentace pravidla	dvěma čísly
ystartovní populace	*parametr experimentu*
podmínkové funkce	*parametr experimentu*

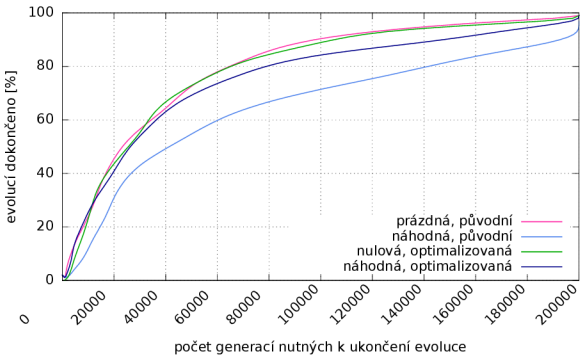
Tabulka 6.11: Parametry experimentu, které se liší od úvodních

Ve výsledcích (tabulka 6.12) se předpoklad potvrdil ale se zajímavým vedlejším efektem. Testovaná byla jak původní množina funkcí (EQ, NEQ, GEQ, LEQ, WILD), tak vylepšená z předchozí kapitoly (EQ, NEQ, GEQ, LEQ). Původní množina získala velké zlepšení prázdnou počáteční konfigurací, oproti náhodné (je znát i na grafu 6.6). Zatímco pro množinu bez funkce WILD nedošlo změnou počáteční populace až tak k velkému zlepšení.

populace	podm. funk.	úspěšnost	prům. generací
náhodná	optimalizované	90%	54 077
prázdná	optimalizované	98%	35 385
náhodná	původní	96%	40 566
prázdná	původní	97%	35 246

Tabulka 6.12: Tabulka výsledků počáteční populace. Sloupec *prům. generací* vyjadřuje průměrný počet generací potřebný k evoluci úspěšných řešení. *Původní* podmínkové funkce vyjadřují množinu $\{EQ, NEQ, GEQ, LEQ, WILD\}$, optimalizované pak množinu $\{EQ, NEQ, GEQ, LEQ\}$

Dalo by se tedy říci, že počáteční populace inicializovaná jako prázdná pravidla, vyvažuje negativní dopad funkce WILD. Dále pak nulová inicializace zlepšuje místo náhodné pro tento problém čas pro nalezení řešení.



Obrázek 6.6: Generační složení výsledků experimentu

6.8 Počet podmínkových pravidel a míra mutace

V počátečním experimentu bylo zběžně nastíněno téma velikosti mutace v závislosti na velikosti chromozomu. V následujícím experimentu bude provedeno přesnější měření úspěšnosti evoluce v závislosti na velikosti mutace a počtu podmínkových pravidel.

S každým přidaným podmínkovým pravidlem se prodlužuje chromozom, a tak intuitivně cítíme, že bude potřeba zvětšit i mutaci. Otázkou je, při jakém zvětšení chromozomu je potřeba zvětšit i mutaci. Parametry prováděného experimentu jsou uvedeny v tabulce 6.13.

název	hodnota
počet generací GA	200 000
pravděpodobnost křížení	30%
podmínkové funkce	EQ, NEQ, GEQ, LEQ
reprezentace pravidla	dvěma čísly
počet zmutovaných čísel	*parametr experimentu*
počet podmínkových pravidel	*parametr experimentu*

Tabulka 6.13: Parametry experimentu, které se liší od úvodních

V tabulce 6.14 jsou popsány výsledky experimentu tak, že jednotlivé sloupce vyjadřují míru mutace a řádky počet podmínkových pravidel. V levé tabulce je procentuální míra úspěšnosti evoluce, na pravé pak průměrný počet generací úspěšných řešení v tisících. Pro lepší čitelnost jsou hodnoty vyznačeny barvami, kde červená značí horší výsledek než zelená. Hodnoty -1 znamenají, že pro dané parametry nebylo měření prováděno.

počet podmínkových pravidel \ velikost mutace	1	3	5	7	9	11
10	36	60	4	-1	-1	-1
13	34	79	56	4	-1	-1
16	35	82	94	24	4	-1
19	28	82	94	84	0	0
22	44	88	100	92	52	0
25	44	68	92	88	80	16
28	28	76	100	100	96	20
31	48	72	96	100	96	48
34	20	48	92	100	100	72
37	48	76	92	100	96	80
40	16	64	92	100	96	88

úspěšnost evoluce v %

počet podmínkových pravidel \ velikost mutace	1	3	5	7	9	11
10	17k	19k	46k	-1	-1	-1
13	80k	58k	74k	27k	-1	-1
16	65k	54k	61k	26k	48k	-1
19	12k	65k	44k	18k	-1	-1
22	22k	13k	10k	14k	23k	-1
25	17k	13k	9k	9k	19k	18k
28	9k	20k	11k	5k	12k	19k
31	18k	21k	7k	10k	10k	19k
34	14k	17k	11k	8k	9k	21k
37	21k	15k	10k	6k	10k	19k
40	28k	14k	13k	7k	7k	23k

prům. počet generací v tisicích

Tabulka 6.14: Tabulka výsledků mutace

Jak se dalo očekávat, se zvyšujícím se počtem podmínkových pravidel je potřeba zvyšovat míru mutace. I když se zdá, že stačí navyšovat jen velice pozvolna a také že vyšší počty podmínkových pravidel úspěšně zvládají i větší rozpětí hodnot. Taktéž je vidět, že po překonání určité hranice nejmenšího ideálního počtu pravidel pro úspěšné řešení úlohy (zde okolo 22 pravidel), zvyšující se počet pravidel nemá už moc vliv na snižování počtu generací nutných k řešení, ani na úspěšnost.

Pro další experimenty budeme dále uvažovat s počtem podmínkových pravidel 22, protože dle tabulky s výsledky se jedná o jednu z nejmenších hodnot pro dosažení úspěšnosti blízké 100% a zároveň s nízkým počtem generací.

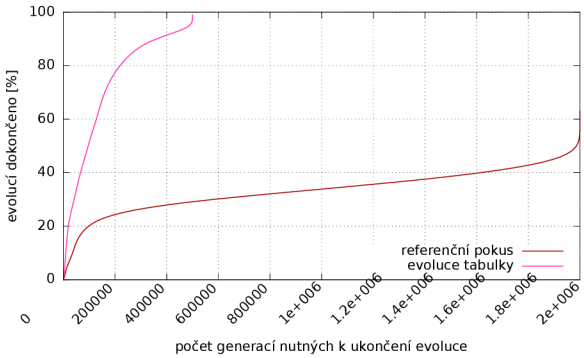
6.9 Evoluce tabulkových pravidel

Pro zjištění skutečného přínosu metody a optimalitací, je potřeba je srovnat s použitím standardních metod, v tomto případě s evolucí pravidel přechodové funkce v tabulkové formě (popsáno v kapitole 2.3). Použitý chromozom je pole typu **byte**[] o délce $2^9 = 512$, protože máme automat se 2 stavy a 9-okolím. Ostatní parametry experimentu jsou shrnuty v tabulce 6.7.

název	hodnota
počet generací GA	2 000 000
počet zmutovaných čísel	10
pravděpodobnost křížení	30%
podmínkové funkce	EQ, NEQ, GEQ, LEQ

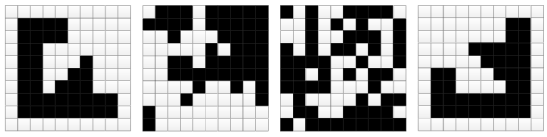
Obrázek 6.7: Parametry experimentu, které se liší od úvodních

Evoluce s popsanými parametry dokázala nacházet řešení, i když za mnohem delší čas (počet generací) než v případě použití podmínkových pravidel. Úspěšnost byla kolem **60%**, a průměrný počet generací úspěšných řešení kolem **300 000**. Srovnání spůvodním referenčním experimentem (který byl omezen *500 000* generacemi) je uvedeno na obrázku 6.8, kde je vidět rozdíl efektivit.



Obrázek 6.8: Generační složení výsledků experimentu

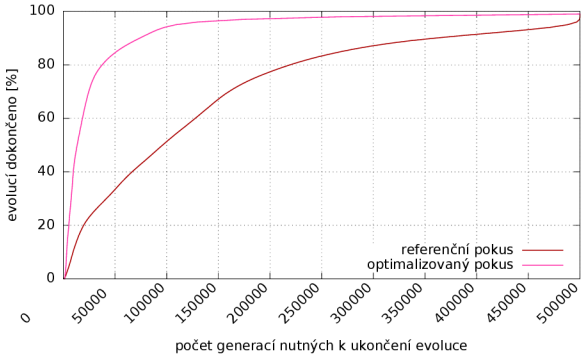
Zajímavé jsou podoby výsledných transformací. Jednak v průměru potřebují méně kroků než transformace používající podmínková pravidla (dalo by se případně upravit fitness funkcí, aby zohledňovala počet kroků). Za druhé vypadají úplně jinak než transformace pomocí podmínkových pravidel. Ukázku typického řešení pomocí tabulkových pravidel nalezneme na obrázku 6.9 (srovnej s obrázkem 6.2 v kapitole 6.1, kde je typický průběh výsledku pomocí podmínkových pravidel).



Obrázek 6.9: Průběh typické transformace dle tabulkových pravidel

6.10 Závěr

Experimenty v této kapitole se týkaly rozšíření původního experimentu s podmínkovými pravidly a jeho optimalizací. Už při pokusu o opakování experimentu s původními parametry byla naměřena větší úspěšnost než publikovaná, což je způsobeno odlišnostmi implementace. Při použití tohoto úvodního experimentu jako referenčního se oproti němu povedlo průběh evoluce velice významně vylepšit a zoptimalizovat nastavením parametrů. Srovnání původního a optimalizovaného experimentu je uvedeno na obrázku 6.10.



Obrázek 6.10: Generační složení výsledků srovnání optimalizace

Změna byla provedena hlavně v použitých podmínkových funkcích, kdy bylo zjištěno že funkce WILD výsledky kazí. Následně byla ještě upravena pravděpodobost křížení a startovní populace nastavena na nulovou místo náhodné (viz tabulka parametrů 6.11).

název	hodnota
počet generací GA	200 000
pravděpodobnost křížení	30%
velikost turnaje	6
startovní populace	Nulová
podmínkové funkce	EQ, NEQ, GEQ, LEQ
reprezentace pravidla	dvěma čísly

Obrázek 6.11: Optimalizované parametry

Při srovnání s konvenčním přístupem, evolucí tabulkových pravidel, vyšlo najevo, že podmínková pravidla sice dokáží návrh hodně zefektivnit, je to však možné i bez nich (nebo případně použít jiné optimalizace evoluce). Také se ukázalo, že řešení navrhnutá konvenčními způsoby a pomocí pomínkových pravidel vypadají úplně jinak, co se průběhu transformace týče.

Transformace dosažené pomocí podmínkových pravidel dosahují vyšší uspořádanosti tak, že lze vidět jak se vzor postupně trasformuje či je možné vidět alespoň lépe rozpoznatelné tvary pro lidské oko a vnímání. V transformaci vzniklé při použití evoluce tabulkových pravidel se jedná spíše o chaos mezi cílovým a počátačním stavem, ve kterém nelze lidským okem zachytit žádné podobné tvary.

Kapitola 7

Experimenty s 1D celulárními automaty

Dalším problémem, kterým jsem se zabýval, byla aplikace podmínkových pravidel na 1D celulární automaty, konkrétně na výpočty které mohou provádět. Základním problémem je zde to, jakým způsobem reprezentovat vstup a výstup, protože pro celulární automat jsou „čitelnější“ jiné formy než pro člověka. Experimenty v této kapitole vychází z 1D automatu pro výpočet mocniny, kterou v [27] navrhl Stephen Wolfram (obrázek 2.11 v kapitole 2.5.2).

Experimenty se pokusíme o nalezení řešení výpočtu mocniny se změněnou reprezentací tak, že nebude potřeba mít pro vstup ani výstup před vlastním číslem ještě další blok buněk jiného stavu. Číslo tak bude reprezentováno pouze souvislým blokem stejných stavů o velikosti hodnoty čísla. Řešení bude hledáno genetickým algoritmem s využitím podmínkových pravidel.

7.1 Fitness funkce

Základním kritériem navržené fitness funkce pro tento problém je to, že vývoj celulárního automatu musí jednou skončit sám tak, že se po sobě vyskytnou dva stejné stavy (a tak by se tento stav opakoval do nekonečna). Pokud takový případ nenastane do nastaveného limitu, kandidátní řešení je ohodnoceno paušálně hodnotou 0.0. Pokud automat úspěšně skončí, je konečný stav ohodnocen dle toho, jak blízko se nachází k požadované hodnotě.

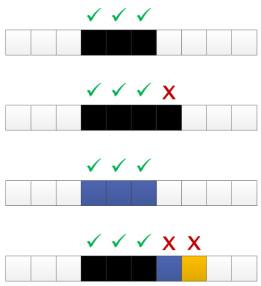
Tato hodnota je určena jako souvislý blok **jakéhokoli nenulového** stavu. Přidaná je pak penalizace za odlišnost velikosti bloku od očekávané hodnoty a také za jakékoli buňky ve stavu > 0, které se vyskytují mimo hodnotový blok (vizuálně demonstrováno na obrázku 7.1). Celý vzoreček pak vypadá jako rovnice 7.1, kde n_t je cílové číslo, n_b je velikost největšího jednobarevného bloku buněk, n_c je počet buněk ve stavu > 0 a w je šířka automatu.

$$fitness = 1 - \frac{|n_t - n_b| + (n_c - n_b)}{w - n_c} \tag{7.1}$$

V původním návrhu od Stephena Wolframa je výsledek výpočtu blok stavů stejné hodnoty, jakou mají vstupní stavy (konkrétně hodnota 1 - černá). Ve zde navržené fitness funkci není toto omezení, výsledek může být reprezentován blokem jiných stavů, než stavy hodnoty 1 či jinými stavy než jakými byl reprezentován vstup. Zároveň také platí, že výsledek

mocniny různých čísel může být vyjádřen bloky různých stavů. Dokud je pro každé číslo výsledek vždy jednobarevný blok, jedná se o validní řešení.

Chceme tak dát genetickému algoritmu větší možnosti a více volnosti. Čitelnost řešení taktéž neutrpí, je pro nás stejně snadné spočítat blok buněk o hodnotě > 0 jako spočítat blok buněk jen určitého stavu.



Obrázek 7.1: Příklad hodnocení reprezentace čísla 3 fitness funkcí

Jako kódování vstupní hodnoty je v této práci **vždy použit** souvislý blok stavů o hodnotě 1 (ve vizuálních reprezentacích se jedná o černou barvu), který má na svých obou stranách dostatečný velký počet nulových stavů (tj. nejedná se o okraj automatu). Velikost tohoto bloku je rovná hodnotě reprezentovaného čísla. Například pro číslo 3 se jedná o tři černé buňky vedle sebe.

7.2 Určování obecného řešení

Výše popsané ohodnocení běhu automatu se pro každého jedince (reprezentovaného sadou podmínkových pravidel) testuje pro více vstupních hodnot. V základní verzi experimentu je součástí fitness funkce ohodnocení výpočtů pro vstupní čísla 0 až 5. Po úspěšném skončení evoluce (automat prování správně výpočet pro všechny vstupy testované ve fitness funkci) následuje test pro vstupy 0 až 100. **Pokud testem projde, je řešení pro účely této práce považováno za obecné.**

V průběhu experimentů byl na náhodné výsledky pouštěn test pro vstupy 0 až 200, který už byl více náročnější na čas. Žádný z těchto testů neukázal že by obecnost řešení z prvního testu selhala na intervalu 101 až 200. Z toho bylo usouzeno, že první test je pro účely této práce dostatečný. Na výpočty mocnin vyšších čísel nebyla řešení v této práci testována.

7.3 Parametry experimentu

Parametry experimentů této kapitoly jsou uvedeny v tabulce 7.1, některé jsou založeny na poznatcích z minulé kapitoly o optimalizace 2D transformace. U jednotlivých případů budou vždy uvedeny pouze rozdílné parametry oproti této tabulce. Oproti předchozím hodnotám byla upravena hodnota mutace na 4, protože i chromozom 1D automatu je kratší, než chromozom 2D automatu. Další úpravy této hodnoty pak budou součástí experimentování. Maximální počet kroků CA je taktéž modifikován.

Pro zrychlení běhu genetického algoritmu je maximální počet kroků automatu určen dynamicky pro každý vstup - je jasné, že pro výpočet mocniny čísla 1 bude potřeba méně

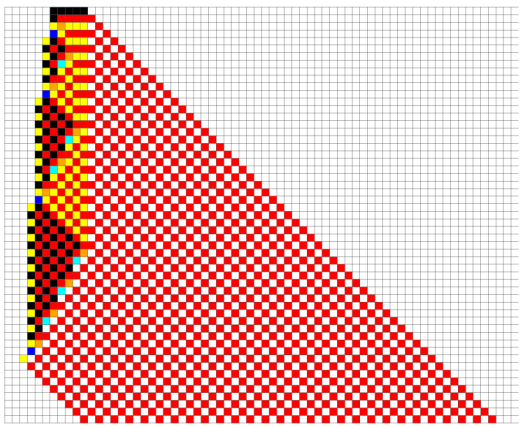
kroků než pro výpočet mocniny čísla 5. Proto je zavedeno omezení maximálního počtu kroků závislé na vstupní hodnotě. Počet stavů CA je nastaven na 8, dle původního návrhu, ale toto číslo není konečné a bude též měněno v rámci experimentů.

název	hodnota
velikost populace GA	10
počet generací GA	500 000
počet zmutovaných čísel	4
pravděpodobost mutace čísla	50%
typ křížení	jednobododové
pravděpodobnost křížení	30%
velikost turnaje	6
startovní populace	Náhodná
typ CA	1D
maximální počet kroků CA	$x^2 * 2$
šířka automatu	100
vstupy hodnocené ve fitness	0-5
počet podmínkových pravidel	22
počet stavů CA	7
podmínkové funkce	EQ, NEQ, GEQ, LEQ
reprezentace pravidla	dvěma čísly
počet experimentů	100

Tabulka 7.1: Tabulka základních parametrů pro 1D experimenty

7.4 Reprezentace výsledku výpočtu součtem stavů

Zpočátku bylo experimentováno s nastavením fitness funkce, a zkoumány různé reprezentace výsledku. Jedním z takových předběžných pokusů je zkouška reprezentace výsledku jako **součtu všech nenulových buněk**. Jak je vidět na obrázku 7.2, výpočet se sice podařilo realizovat (červených buněk je 25), ale automat je nutno zastavit ručně po dosažení určitého počtu kroků.



Obrázek 7.2: Výsledek první verze fitness funkce

Dalšími kroky se sice řešení nemění, ale nejedná se o dva stejné po sobě jdoucí stavy.

Automat by tak pokračoval do nekonečna, a nebo do dosažení konce automatu, kde by se řešení začalo ztrácet. Mít nutnost automatu potenciálně nekonečné šířky by nebylo implementačně praktické.

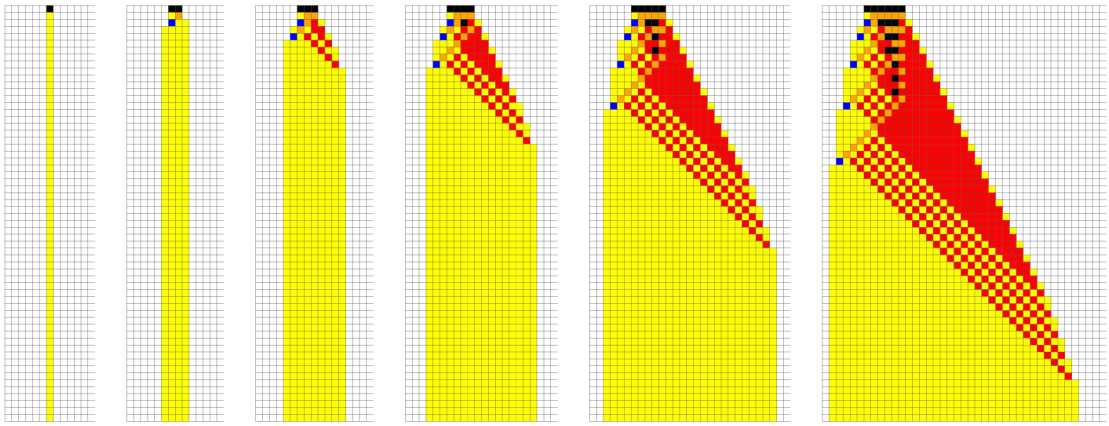
7.5 Reprezentace výsledku výpočtu blokem stavů

Dalším experimentováním s fitness funkcí bylo dosaženo formy, která je popsána v předchozích odstavcích, kdy výsledkem je souvislý blok nenulových stavů stejné hodnoty (s nastavením experimentu dle tabulky 7.2). Takto bylo úspěšně dosaženo výsledku zobrazeného na obrázku 7.3, kde jsou vizualizovaná řešení pro mocniny čísel $1 - 6$. Evoluce probíhala pouze pro čísla $0 - 5$, řešení ale funguje i pro ostatní čísla (testováno výpočtem mocniny do čísla 200).

název	hodnota
počet podmínkových pravidel	30
počet stavů CA	8

Tabulka 7.2: Parametry experimentu, které se liší od úvodních

Podmínková pravidla a tabulková reprezentace přechodové funkce tohoto konkrétního příkladu řešení jsou přiloženy v příloze A.



Obrázek 7.3: Výsledný výpočet mocniny s blokovou reprezentací

7.6 Množina podmínkových funkcí

Podobně jako v experimentech s 2D transformací vzoru, je třeba zjistit zda existuje nějaká množina podmínkových funkcí, která by pro řešení problému byla než jiné. Na základě již provedených experimentů budeme uvažovat menší pokusnou množinu tak, abychom pokryli jak dost možností, tak abychom nezkoušeli možnosti sobě moc podobné.

název	hodnota
podmínkové funkce	*parametr experimentu*

Tabulka 7.3: Parametry experimentu, které se liší od úvodních

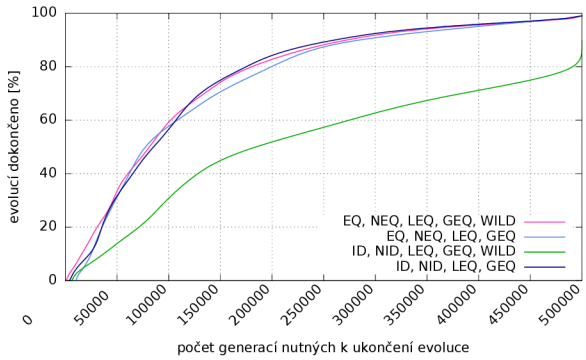
Z výsledků v tabulce 7.4 je vidět, že původně použitá množina funkcí EQ, NEQ, GEQ, LEQ (s WILD i bez) sice má velkou úspěšnost řešení, ale malou úspěšnost obecného řešení (**jedná se o procento obecných řešení z celkových úspěšných řešení evoluce**). Množina funkcí, obsahující funkce ID, NID dosahuje mnohem lepších obecných výsledků. Také to vypadá, že funkce WILD zase kazí úspěšnost evoluce, stejně jako v experimentech s 2D transformacemi. V dalších experimentech proto budeme uvažovat množinu **ID, NID, GEQ, LEQ**.

Použité podmínkové funkce							prům. generací	úsp.	obec. úsp.
EQ	NEQ			GEQ	LEQ	WILD	104 971	98%	23%
EQ	NEQ			GEQ	LEQ		112 416	98%	22%
		ID	NID	GEQ	LEQ	WILD	161 847	77%	79%
		ID	NID	GEQ	LEQ		105 321	98%	80%

Tabulka 7.4: Tabulka výsledků experimentu s podmínkovými funkcemi. Sloupec *prům. generací* vyjadřuje průměrný počet generací potřebný k evoluci úspěšných řešení.

Možným vysvětlením rozdílné úspěšnosti množiny ID, NID oproti EQ, NEQ může být to, že průběh výpočtu pomocí každé z množin vypadají vizuálně podobně. Je tak možné dosáhnout stejného efektu např funkce podmíkového pravidla pro buňku IDX i funkcí EQ1. u ID se ignoruje hodnota pravidla, kdežto EQ potřebuje hodnotu 1, aby funkci ID simulovala. Tato konkrétní podoba podmínky EQ se ale objeví pouze ve zlomku případů, v závislosti na celkovém počtu možných stavů v automatu.

V dalších experimentech budeme také na základě grafu generačního složení výsledků (obrázek 6.8) uvažovat s nižším počtem generací pro evoluci, konkrétně s 200000. Pro uvažovaou množinu funkcí je v této generaci hotovo 80% výsledků.



Obrázek 7.4: Graf generačního složení výsledků

7.7 Počet ohodnocených výpočtů fitness funkcí

Ve fitness funkci bylo zatím uvažování ohodnocování výpočtů pro čísla $0 - 5$. Nicméně vyvstává otázka, zda by nebylo možné snížením počtu těchto ohodnocení zrychlit evoluci při zachování podobné úspěšnosti, nebo přidáním více čísel dosáhout zpomalením větší úspěšnosti. Zrychlení a zpomalení je způsobeno časem, který fitness funkce potřebuje pro ohodnocení kandidátního řešení. Čím více hodnot testuje, tím více simulací CA provádí a tím déle jí to trvá. Počet generací zůstává neovlivněn.

název	hodnota
počet generací GA	200 000
šířka automatu	200
podmínkové funkce	EQ, NEQ, LEQ, GEQ
vstupy hodnocené ve fitness	*parametr experimentu*

Tabulka 7.5: Parametry experimentu, které se liší od úvodních

Byly provedeny experimenty pro intervaly $0-3$ až $0-7$, jejichž výsledky jsou shrnuty v tabulce 7.6. Pro menší intervaly než $0-5$ je sice úspěšnost evoluce 100% a počet generací nutný k řešení malý, ale úspěšnost obecného řešení drasticky klesá. Při evoluci s intervalem $0-3$ se v experimentech nepodařilo najít žádné obecné řešení.

interval	prům. generací	úsp.	obec. úsp.
$0-3$	9 511	100%	0%
$0-4$	35 971	100%	14%
$0-5$	87 244	71%	73%
$0-6$	80 974	71%	99%
$0-7$	94 021	10%	100%

Tabulka 7.6: Tabulka výsledků experimentu s obecným řešením. Sloupec *prům. generací* vyjadřuje průměrný počet generací potřebný k evoluci úspěšných řešení.

Na základě dosažených výsledků budeme v dalších experimentech ve fitness funkci uvažovat s ohodnocováním intervalu $0-6$. Jedná se o první z testovaných intervalů s téměř jistou obecnou úspěšností. Zároveň má rozumnou úspěšnost evoluce tak, aby byl reálně použitelný.

7.8 Výpočet za použití méně stavů

Z předchozích experimentů se zdálo, že občas vycházejí podobná řešení, ale s různě prohozenými barvami stavů. Je proto možné, že původních 8 stavů je naddimenzovaná hodnota a řešení by bylo možné dosáhnout i s méně stavy. Konkrétní nastavení pro tento experiment je uvedeno v tabulce 7.7.

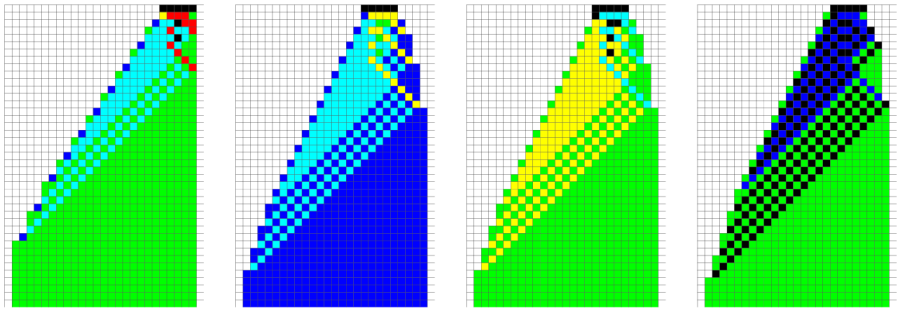
název	hodnota
počet generací GA	1 000 000
vstupy hodnocené ve fitness	0-6
podmínkové funkce	EQ, NEQ, ID, NID, GEQ, LEQ
počet stavů CA	*parametr experimentu*

Tabulka 7.7: Parametry experimentu, které se liší od úvodních

Množina podmínkových funkcí obsahuje sloučenou množinu funkcí (vynechána funkce WILD, která zatím vždy řešení kazila) proto, že genetickému algoritmu chceme dát více možností pro vyjádření a nevíme, zda by pro řešení s menším počtem stavů nějaká funkce nechyběla. Takto zvětšený stavový prostor je kompenzován zvýšením počtu generací. Také zde nejde o změření úspěšnosti ale o nalezení alespoň jednoho obecného řešení s daným počtem stavů automatu.

Postupně bylo dosaženo výsledných výpočtů až pomocí 4 stavů (obrázek 7.5). Všechna takto získaná řešení jsou dost podobná předchozím řešením pro 8 stavů (byla pozorována i různá jiná řešení, jedná se však spíše o kosmetické odlišnosti, typově všechna pozorovaná řešení vypadají stejně). Nejmenší řešení, tj. pro 4 stavy, se nepodařilo navrhnout jiné, než je na zmíněném obrázku. Několik nezávislých běhů experimentu našlo toto jedno konkrétní řešení.

Pro automat se 3 stavy žádné řešení nalezeno nebylo, a populace nejevila žádné známky konvergence ani zlepšování se od určité hraniční hodnoty fitness funkce.



Obrázek 7.5: Ukázky výpočtu mocniny řešením s méně stavy pro číslo 5

Lze tedy s velkou pravděpodobností tvrdit, že pro použitou reprezentaci se nejspíš jedná o řešení s minimálním možným počtem stavů. Úspěšnost nalezení je ale velice nízká, každým ubraným možným stavem se drasticky snižuje. Podmínková pravidla a tabulka přechodové funkce tohoto dosaženého řešení se 4 stavy jsou přiloženy v příloze B.

7.9 Počet podmínkových pravidel a míra mutace

Jako u 2D transformace, bylo i zde potřeba prověřit závislost úspěšnosti na míře mutace a počtu použitých podmínkových pravidel. Navíc k samotné úspěšnosti evoluce byl přidán ke sledování parametr obecné úspěšnosti, která se stejně jako v předchozích 1D experimentech měří jako procento obecných řešení z úspěšných řešení evoluce. Parametry experimentu jsou uvedeny v tabulce 7.8

název	hodnota
počet generací GA	200 000
vstupy hodnocené ve fitness	0-6
podmínkové funkce	ID, NID, GEQ, LEQ
počet podmínkových pravidel	*parametr experimentu*
počet zmutovaných čísel	*parametr experimentu*

Tabulka 7.8: Parametry experimentu, které se liší od úvodních

V tabulce 7.9 jsou uvedeny naměřené hodnoty. Ve sloupcích jsou zaneseny míry mutace, v řádcích pak počet podmínkových pravidel.

Jak se očekávalo, při přidávání podmínkových pravidel je potřeba zvyšovat míru mutace, abychom dosahovali dobré úspěšnosti (levá tabulka). Oproti 2D transformaci mají vyšší počty podmínkových pravidel o trochu menší toleranci na rozpětí možných dobrých mutací.

V pravé tabulce úspěšnosti obecných řešení se ale vyskytl zajímavý úkaz. Pokud je mutace nižší než by pro daný počet podmínkových pravidel měla být, klesá jak úspěšnost evoluce, tak i úspěšnost obecných řešení v úspěšných řešení evoluce. Pokud je ale míra mutace vyšší než by bylo pro daný počet podmínkových pravidel patřičné, úspěšnost evoluce sice opět klesá, **ale úspěšnost obecných řešení zůstává kolem 100%!**

úspěšnost evoluce v %

počet podmínkových pravidel \ velikost mutace	1	2	3	4	5	6
9	18	58	14	0	0	0
11	14	74	51	4	0	0
13	31	66	79	38	9	0
15	21	58	84	59	12	0
17	-1	56	76	80	28	2
19	-1	47	79	84	62	15
21	-1	58	76	88	68	33
23	-1	55	62	83	81	58
25	-1	64	62	80	83	72
27	-1	55	76	70	83	81
29	-1	44	67	67	81	80

úspěšnost obecného řešení v %

počet podmínkových pravidel \ velikost mutace	1	2	3	4	5	6
9	61	100	100	0	0	0
11	42	98	100	100	0	0
13	16	83	100	100	100	0
15	0	77	97	98	100	0
17	-1	44	93	100	96	100
19	-1	14	84	97	100	100
21	-1	18	71	96	98	100
23	-1	12	69	93	95	98
25	-1	20	37	88	96	97
27	-1	12	39	74	96	100
29	-1	2	17	73	93	97

Tabulka 7.9: Tabulka úspěšností řešení pro různé míry mutace

Vyplývá z toho tedy jasný poznatek, že pokud si v experimentu nejsme jisti správným nastavením mutace, raději zvolit vyšší hodnotu než nižší. V tabulce 7.10, která zobrazuje průměrný počet generací úspěšných řešení, se nevyskytuje nic neobvyklého. Pokud nastavíme mutaci na jinou než optimální míru, zvyšuje se počet generací nutných k řešení. Při správné míře mutace je pak počet generací vždy podobný pro různý počet podmínmkových pravidel (69 000 - 75 000). Trochu větší počet generací se vyskytuje pouze u velmi malých počtů podmínkových pravidel, které nejspíš hraničí s možností vyjádřit potřebná pravidla.

počet podmínkových pravidel \ velikost mutace	1	2	3	4	5	6
9	104k	102k	80k	-1	-1	-1
11	81k	87k	76k	112k	-1	-1
13	112k	103k	79k	100k	69k	-1
15	102k	88k	76k	90k	89k	-1
17	-1	94k	68k	80k	101k	78k
19	-1	93k	83k	73k	90k	110k
21	-1	93k	91k	70k	86k	98k
23	-1	97k	100k	74k	71k	89k
25	-1	107k	98k	84k	85k	82k
27	-1	102k	81k	83k	73k	81k
29	-1	99k	87k	86k	75k	73k

Tabulka 7.10: Prům. počet generací v tisících

7.10 Evoluce tabulkových pravidel

V závěru je třeba ještě vyzkoušet, zda podobných vysledů nelze dosáhnout také konvenčními metodami. Byl tedy proveden experiment s evolucí pomocí tabulkových pravidel. Jako chromozom slouží pole typu **byte**[] obsahující hodnoty nových stavů pro pravidlo na patřičném místě tabulky přechodové funkce (popsáno v kapitole 2.3). Velikost chromozomu je tedy $8^3 = 512$, protože uvažujeme 8 stavů a 3-okolí. Jinak jsou parametry experimentu shrnuty v tabulce 7.11 s dodatkem, že se nepoužívají podmínková pravidla ale tabulková reprezentace.

název	hodnota
velikost populace GA	10
počet generací GA	2 000 000
počet zmutovaných čísel	10-100
šířka automatu	30-100
vstupy hodnocené ve fitness	0-5 (0-4, 0-3)
počet stavů CA	7

Tabulka 7.11: Parametry experimentu

Pro počáteční nastavení, kdy fitness funkce testovala a ohodnocovala výpočty mocnin čísel 0-5, evoluce nenalezla žádné řešení, ať už obecné nebo jen pro čísla testovaná fitness funkcí. Protože to mohlo být malou či velkou mírou mutace, bylo testováno více hodnot mutace (v rozsahu $5 - 100$), žádná nepřinesla výsledek (větší mutace než 100 nebylo třeba zkoušet, už tato dost rozbíjela potenciální řešení).

Částečných výsledků bylo dosaženo při nastavení fitness funkce tak, aby testovala pouze mocniny $0 - 3$, kdy občas bylo nalezeno neobecné řešení (řešení jen pro testované hodnoty), ale i tak s velmi malou pravděpodobností (menší než 5%).

Jednoduchým konvenčním testem bez optimalizací bylo potvrzeno, že takto získaná řešení pro tento problém se ani kvalitou ani rychlostí neblíží kvalitě a rychlosti získané pomocí podmínkových pravidel.

7.11 Závěr

Experimenty v této kapitole byl úspěšně navržen obecný výpočet mocniny v 1D celulárním automatu za použití podmínkových pravidel. Byla použita upravená a zjednodušená reprezentace problému oproti původní wolframově verzi. I s touto úpravou se povedlo dosahovat úspěšných automatických návrhů v rozumném čase a s dobrou úspěšností. Řešení se povedlo zmenšit až na automat používající 4 stavy (oproti původním 8).

Při porovnání přístupu za použití podmínkových pravidel a konvenčního přístupu pomocí tabulkových pravidel se ukázalo, že konvenčním přístupem se nepodařilo dosáhnput obecného řešení za žádného nastavení parametrů. Dosáhnout se nepodařilo ani řešení pro výpočet mocniny intervalů $0 - 5$ či $0 - 4$. Podařila se jen evoluce pro interval $0 - 3$, a i to jen s velmi malou pravděpodobností.

Prostorová složitost nalezených řešení je ve třídě $O(n^2)$, konkrétně $x^2 + 2$ buněk celulárního automatu. Této hodnoty je dosaženo v posledním kroku a jedná se o velikost reprezentace výsledku výpočtu. Na každé straně je pak potřeba mít alespoň jednu další buňku (v závislosti na okrajových podmnínkách. Pokud budou nulové, lze vynechat).

V průběhu výpočtu žádný ze sledovaných automatů uvedenou hodnotu prostorové složitosti nepřekročil.

Časová složitost výpočtu spadá do třídy $O(n^2)$ a nejčastěji se pohybuje okolo $2x^2 - 3x$ (složitost pro řešení se 4 stavy z kapitoly 7.8, která byla získána interpolací programem *gnuplot* z dat o počtu kroků mocnin čísel $1 - 100$). Lepší třídy časové složitosti nelze při uvažované reprezentaci problému dosáhnout kvůli vlastnosti 1D CA, který se v každém kroku může rozšířit maximálně o 2 buňky (platí pro 3-okolí). V uvažované reprezentaci je vstup zakódován jako nepřerušená posloupnost buněk, a při nutnosti rozšířit vstup na blok o velikosti druhé mocniny narážíme na tento limit.

Pro získání lepší časové složitosti by bylo možné zkusit vstup kódovat například s mezerami, či udělat takovou reprezentaci výsledku, která by byla méně náročná na počet buněk automatu.

Kapitola 8

Závěr

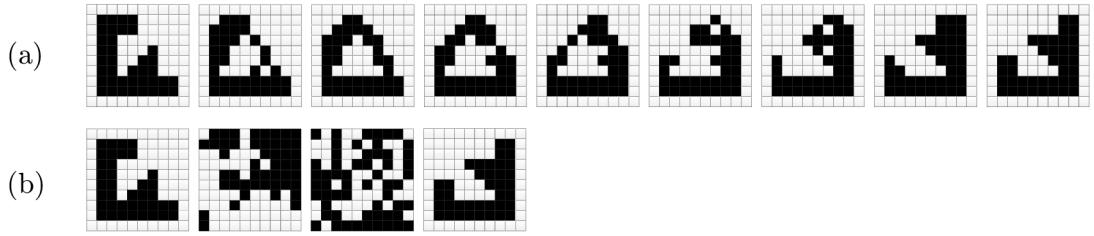
V práci byl čtenář seznámen se základy problematiky celulárních automatů a genetických algoritmů. Dále byla popsána perspektivní metoda návrhu CA pomocí *podmínkových pravidel*. Pro následné experimenty byla implementována aplikace v jazyce Java, která obsahovala jednoduchou implementaci genetických algoritmů, 1D a 2D celulárních automatů a podmínkových pravidel. Dále obsahovala prostředky pro import/export výsledků včetně jejich vizualizace. Tímto nástrojem byly provedeny dva okruhy experimentů, s 2D automaty a 1D automaty. Jejich podrobné výsledky jsou uvedeny v předchozích kapitolách, zde bude uvedeno shrnutí, zhodnocení a diskuze. Pro účely vložení vizualizací výsledků do dokumentace byl navíc vytvořen jednoduchý vizualizátor 1D automatů v jazycích html/javascript, který výsledky exportoval do vektorového formátu svg.

Experimenty s 2D automaty

Ve 2D experimentech byl úspěšně zopakován experiment z původního článku o podmínkových pravidlech s podobnými výsledky jako byly publikované [6]. Poté bylo vyzkoušeno několik různých nastavení genetického algoritmu a podmínkových pravidel tak, že postupně **došlo k velmi znatelnému zefektivnění** evoluce. Největšího zlepšení dosáhla optimalizace množiny použitých podmínkových funkcí, kdy byla odstraněna funkce *WILD* a tím se dosáhlo zefektivnění o cca. **15%**. Druhým nejvíce ovlivňujícím faktorem byla míra mutace v závislosti na počtu použitých pravidel. Po analýze této závislosti vyšlo najevo, že původní experiment neměl nastavenou vhodnou míru mutace pro použitý počet pravidel, a tím ztrácel na efektivitě.

Celkově bylo u tohoto experimentu dosaženo zlepšení o cca. **80%** při porovnání plně optimalizovaného experimentu s referenčním (z průměrných 109 451 generací na 23 816 generací). Za referenční experiment je považován experiment se stejnými parametry jako byly původně publikované, ale provedený implementací použitou v této práci (podrobněji popsáno v kapitole 6.2).

Následně byl proveden evoluční návrh, kdy přechodová funkce byla reprezentována standardní tabulkou. Ukázalo se, že standardní metodou je možné nalézt řešení sledovaného transformačního problému, ale s mnohem menší efektivitou návrhu. Typické průběhy výsledných transformací obou metod vůbec nevypadají podobně, jak lze vidět z obrázku 8.1.

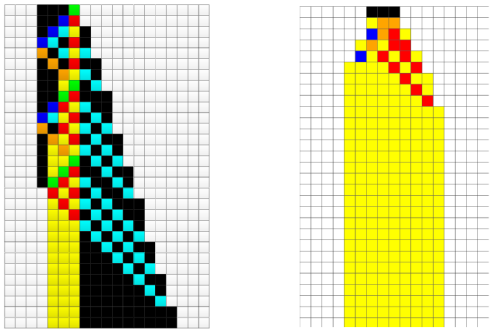


Obrázek 8.1: Typický průběh transformace získaný evolucí podle podmínkových pravidel (a) a evolucí tabulkové reprezentace (b)

Experimenty s 1D automaty

V druhé části prováděných experimentů, byla na základě návrhu Stephena Wolframa [27] navržena nová, zjednoduššená reprezentace výpočtu druhé mocniny pomocí 1D automatu. Pro tuto reprezentaci se povedlo pomocí podmínkových pravidel provést úspěšný evoluční návrh fungujícího výpočtu druhé mocniny, u kterého byla **ověřena jeho obecnost do výpočtu mocniny čísla 200**. Následně byly provedeny experimenty s cílem tento prvotní úspěch zoptimalizovat. Z provedených experimentů stojí za zmínku výsledek, kterým se podařilo nalézt řešení pro automat pouze se čtyřmi stavy, oproti původním osmi.

Pro výpočet druhé mocniny čísla 3 je rozdíl mezi wolframovým návrhem a jedním z výsledků této práce uveden na obrázku 8.2. Jedná o šestistavový automat, který lze oproti čtyřstavovému automaticky navrhovat v rozumném čase. Lze pozorovat mnohem větší efektivitu výpočtu než v původním wolframově návrhu. Jak časovou (počet kroků), tak v reprezentaci čísla (v původním je složitě kódováno, v nově navrženém se vždy jedná o jeden jednobarevný blok). **Při použití metod a parametrů popsaných v této práci lze zde ukázané řešení navrhovat automaticky v rozumném čase**.



Obrázek 8.2: Ukázka rozdílu průběhu výpočtu druhé mocniny pro číslo 3 mezi wolframovým [27, str. 639] návrhem (a) a jedním z výsledků této práce (b)

Jako u předchozího bloku experimentů bylo i zde provedeno srovnání se standardní metodou evolučního návrhu, evolucí tabulkové reprezentace. Pomocí ní se vůbec nepovedlo získat jakékoli obecné (ve smyslu výpočtů do čísla 200) řešení výpočtu druhé mocniny. Dosaženo bylo pouze výpočtu pro čísla $0 - 3$, které byly hodnoceny ve fitness funkci. Pro čísla $0 - 4$ se už řešení za pomocí použitých parametrů najít nepovedlo. Zjištěno bylo,

že standardní metoda evolučního návrhu nemá ani zdaleka efektivitu potřebnou pro tento problém, a tedy že evoluce pomocí podmínkových pravidel je zdaleka více efektivní.

8.1 Možnosti rozšíření

Podmínková pravidla se ukazují jako efektivní způsob reprezentace přechodové funkce pro širokou škálu problémů, a proto by bylo přirozeným dalším vývojem jejich aplikování na další problémy. U 2D automatů je zatím pořád prakticky nemožné automaticky navrhovat pro vícestavové automaty, ve kterých stavový prostor roste obrovským tempem. Přitom právě ve vícestavových automatech se vyskytují problémy, pro které by nás automatický návrh jejich řešení skutečně zajímal. S použitím podmínkových pravidel a jejich optimalizací je už teoreticky možné vrhnout se na experimenty s návrhy těchto složitějších problémů. V rámci této práce byly provedeny letmé experimenty s replikací smyček, nicméně nebyla vymyšlena žádná vhodná a efektivní fitness funkce. Tyto letmé experimenty nepřinesly žádné výsledky a proto se jimi práce nijak nezabývala. Dalším možným krokem by bylo zaměřit se na efektivní fitness funkce, které by dobře spolupracovaly s podmínkovými pravidly a umožnily krok právě k vícestavovým 2D automatům.

V rámci experimentů s výpočty v 1D automatech bylo dosaženo poměrně dobrých výsledků a efektivity pro výpočet druhé mocniny. Dalším možných pokračováním by mohlo být vyzkoušení dalších operací, jako je sčítání, násobení, včetně výzkumu lepších a efektivnějších způsobů reprezentace. Jak bylo diskutováno v závěru 1D experimentů, reprezentace blokem o velikosti čísla není pro výsledek mocniny moc efektivní, prostor potřebný pro reprezentaci výsledku roste velice rychle.

Literatura

[1] Ahmed, S.; Akhter, A.; Kunwar, F.: Cellular automata based real time path planning for mobile robots. In *Control Automation Robotics Vision (ICARCV), 2012 12th International Conference on*, Prosinec 2012, s. 142–147.

[2] Alligood; Kathleen: *Chaos : an introduction to dynamical systems*. New York: Springer, 1997, ISBN 0387946772.

[3] Bentley, P.: *Evolutionary Design by Computers*. číslo díl 1 in Evolutionary Design by Computers, Morgan Kaufman Publishers, 1999, ISBN 9781558606050.

[4] Bidlo, M.: Evolution of Computational Processes in Uniform Cellular Automata. In *2014 IEEE World Congress on Computational Intelligence*, IEEE Press, 2014 (přijato k publikaci).

[5] Bidlo, M.; Vašíček, Z.: Evolution of Cellular Automata Using Instruction-Based Approach. In *2012 IEEE World Congress on Computational Intelligence*, Institute of Electrical and Electronics Engineers, 2012, ISBN 978-1-4673-1508-1, s. 1060–1067.

[6] Bidlo, M.; Vašíček, Z.: Evolution of Cellular Automata with Conditionally Matching Rules. In *2013 IEEE Congress on Evolutionary Computation (CEC 2013)*, IEEE Computer Society, 2013, ISBN 978-1-4799-0452-5, s. 1178–1185.

[7] Byl, J.: Self-reproduction in small cellular automata. *Physica D: Nonlinear Phenomena*, ročník 34, č. 1–2, 1989: s. 295–299, ISSN 0167-2789.

[8] Capcarrere, M. S.; Sipper, M.; Tomassini, M.: Two-state, r = 1 Cellular Automaton that Classifies Density. *Physical Review Letters*, ročník 77, Prosinec 1996: s. 4969–4971.

[9] Chopard, B.; Droz, M.: *Cellular Automata Modeling of Physical Systems*. Cambridge University Press, 1998, ISBN 0-521-46168-5.

[10] Gach: One-Dimensional Uniform Arrays That Wash Out Finite Islands. *Probl. Peredachi Inf.*, ročník 14, č. 3, 1978: s. 92–96.

[11] Gardner, M.: The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, ročník 223, Říjen 1970: s. 120–123.

[12] Higuchi, T.: *Evolvable hardware*. New York London: Springer, 2006, ISBN 9780387312385.

[13] Holland, J. H.: *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI, USA: University of Michigan Press, 1975.

[14] Hynek, J.: *Genetické algoritmy a genetické programování.* Průvodce (Grada), Grada, 2008, ISBN 9788024726953.

[15] Kajitani, I.; Hoshino, T.; Nishikawa, D.; aj.: A Gate-Level EHW Chip: Implementing GA Operations and Reconfigurable Hardware on a Single LSI. In *ICES*, 1998, s. 1–12.

[16] Kvasnička, V.; Pospíchal, J.; Tiňo, P.: *Evolučné algoritmy.* Edícia vysokoškolskỳch učebníc / Slovenská technická univerzita, Slovenská technická univerzita, 2000, ISBN 9788022713771.

[17] Land, M.; Belew, R. K.: No Perfect Two-State Cellular Automata for Density Classification Exists. *Physical Review Letters*, ročník 74, Červen 1995: s. 5148–5150.

[18] Langton, C. G.: Self-reproduction in cellular automata. *Physica D: Nonlinear Phenomena*, ročník 10, č. 1–2, 1984: s. 135–144, ISSN 0167-2789.

[19] Langton, C. G.: Studying artificial life with cellular automata. *Physica D: Nonlinear Phenomena*, ročník 22, č. 1–3, 1986: s. 120–149, ISSN 0167-2789, proceedings of the Fifth Annual International Conference.

[20] Neumann, J. V.: *Theory of Self-Reproducing Automata.* Champaign, IL, USA: University of Illinois Press, 1966.

[21] Pham, D. T.; Karaboga, D.: *Intelligent Optimisation Techniques: Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks.* Secaucus, NJ, USA: Springer-Verlag New York, Inc., první vydání, 1998, ISBN 1852330287.

[22] Reggia, J. A.; Chou, H.-H.; Lohn, J. D.: Cellular Automata Models of Self-Replicating Systems. *Advances in Computers*, ročník 47, 1998: s. 141–183.

[23] Russell, S. J.; Norvig, P.: *Artificial Intelligence: A Modern Approach.* Pearson Education, druhé vydání, 2003, ISBN 0137903952.

[24] Salcido, A.: *Cellular automata: innovative modelling for science and engineering.* Rijeka: InTech, 2011, ISBN 9789533071725.

[25] Schwarz, J.; Sekanina, L.: *Aplikované evoluční algoritmy, Studijní opora předmětu EVO.* FIT VUT v Brně, 2006.

[26] Sekanina, L.; Vašíček, Z.; Růžička, R.; aj.: *Evoluční hardware: od automatickeého generování patentovatelných invencí k sebemodifikujícím se strojům.* Praha: Academia, 2009, ISBN 9788020017291.

[27] Sukumar, N.: Effect of Boundary Conditions on Cellular Automata that Classify Density. *Contributions to Mineralogy and Petrology*, Duben 1998: str. 4001.

[28] Wolfram, S.: Statistical mechanics of cellular automata. *Reviews of Modern Physics*, ročník 55, Červenec 1983: s. 601–644.

[29] Wolfram, S.: *A New Kind of Science.* Champaign, Ilinois, US, United States: Wolfram Media Inc., 2002, ISBN 1-57955-008-8.

[30] Zadeh; Lofti, A.: Soft Computing and Fuzzy Logic. *IEEE Softw.*, ročník 11, č. 6, Listopad 1994: s. 48–56, ISSN 0740-7459.

Příloha A

Řešení 1D výpočtu (8 stavů)

Následují podmínková pravidla (tabulka A.1) a tabulková reprezentace (tabulka A.2) přechodové funkce 1D CA s 3-okolím pro obecný výpočet mocniny čísla.

X_0	X_1	X_2	Y
$<= 5$	$== 0$	$!= 3$	0
$== 4$	$>= 4$	$== 7$	7
$!= 1$	$== 6$	$== 2$	3
$== 2$	$== 2$	$>= 3$	3
$== 0$	$<= 7$	$>= 6$	3
$== 1$	$<= 7$	$>= 0$	7
$<= 5$	$!= 3$	$<= 6$	4
$>= 0$	$== 5$	$<= 4$	6
$!= 0$	$<= 0$	$!= 6$	4
$<= 0$	$!= 1$	$!= 3$	2

(1. část)

X_0	X_1	X_2	Y
$== 5$	$== 0$	$!= 3$	5
$<= 7$	$<= 7$	$<= 6$	6
$>= 3$	$!= 0$	$!= 0$	1
$>= 4$	$<= 3$	$!= 2$	1
$<= 4$	$== 6$	$>= 3$	7
$>= 3$	$== 1$	$<= 3$	2
$== 5$	$>= 0$	$<= 1$	7
$!= 7$	$>= 2$	$<= 5$	2
$== 4$	$!= 3$	$== 6$	4
$== 7$	$!= 1$	$== 0$	1

(2. část)

X_0	X_1	X_2	Y
$== 0$	$== 5$	$>= 5$	0
$>= 6$	$<= 6$	$== 2$	3
$>= 1$	$<= 6$	$== 0$	3
$!= 1$	$<= 7$	$!= 6$	0
$== 5$	$!= 7$	$>= 5$	5
$>= 0$	$== 6$	$== 4$	6
$!= 6$	$!= 1$	$!= 2$	4
$<= 7$	$<= 3$	$>= 4$	5
$== 7$	$>= 0$	$>= 3$	3
$!= 7$	$<= 7$	$>= 4$	5

(3. část)

Tabulka A.1: Podmínková pravidla

[0, 0, 0, 4, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 3, 3, 4, 4, 4, 4, 4, 4, 3, 3, 2, 2, 2, 6, 2, 2, 3, 3, 4, 4, 4,
4, 4, 4, 3, 3, 4, 4, 4, 4, 4, 4, 3, 3, 4, 4, 3, 4, 4, 4, 3, 3, 4, 4, 4, 4, 4, 4, 3, 3, 0, 0, 0, 7, 0, 0, 0,
0, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 0, 0, 0, 4, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4,
5, 4, 4, 4, 3, 3, 3, 3, 3, 6, 6, 6, 6, 6, 6, 6, 5, 4, 4, 4, 4, 4, 4, 4, 5, 4, 4, 4, 4, 4, 4, 4, 5, 4, 4, 3,
4, 4, 4, 4, 7, 4, 4, 4, 4, 4, 4, 4, 5, 0, 0, 0, 4, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4,
1, 6, 6, 6, 6, 6, 6, 6, 1, 4, 4, 4, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4, 1, 4, 4, 3, 4, 4, 4, 4, 1, 4, 4, 4,
4, 4, 4, 4, 1, 0, 0, 0, 4, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4, 1, 6, 6, 6, 6, 6, 6, 6,
1, 4, 4, 4, 4, 4, 4, 4, 7, 4, 4, 4, 4, 4, 4, 4, 7, 4, 4, 3, 4, 4, 4, 4, 7, 4, 4, 4, 4, 4, 4, 4, 7, 0, 0, 0,
4, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4, 1, 6, 6, 6, 6, 6, 6, 6, 1, 4, 4, 4, 4, 4, 4, 4,
1, 4, 4, 4, 4, 4, 4, 4, 1, 4, 4, 3, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 6, 4, 6, 6, 6,
6, 6, 6, 6, 1, 6, 6, 6, 6, 6, 6, 6, 1, 6, 6, 6, 6, 6, 6, 6, 1, 6, 6, 6, 6, 6, 6, 6, 1, 6, 6, 6, 6, 6, 6, 6,
1, 6, 6, 3, 6, 6, 6, 6, 1, 6, 6, 6, 6, 6, 6, 6, 1, 4, 4, 4, 4, 4, 4, 6, 4, 6, 6, 6, 6, 6, 6, 6, 1, 6, 6, 6,
6, 6, 6, 6, 1, 6, 6, 6, 6, 6, 6, 6, 1, 6, 6, 6, 6, 6, 6, 6, 1, 6, 6, 6, 6, 6, 6, 6, 1, 6, 6, 3, 6, 6, 6, 6,
1, 6, 6, 6, 6, 6, 6, 6, 1]

Tabulka A.2: Pole nových hodnot buňky pro tabulkovou reprezentaci

Příloha B

Řešení 1D výpočtu (4 stavy)

X_0	X_1	X_2	Y	X_0	X_1	X_2	Y
$!= 3$	$X3$	$== 2$	2	$!= 1$	$X3$	$== 3$	3
$>= 3$	$!X1$	$<= 1$	1	$== 0$	$X0$	$>= 0$	1
$X0$	$== 0$	$!X3$	0	$!= 2$	$X2$	$>= 0$	3
$!X2$	$!= 2$	$!X3$	0	$<= 3$	$>= 3$	$!= 2$	2
$!= 0$	$== 1$	$!= 0$	3	$<= 3$	$!X2$	$!X3$	2
$== 2$	$== 3$	$== 2$	3	$!= 3$	$!= 0$	$X3$	3
$== 3$	$!= 0$	$!X2$	3	$!= 3$	$X3$	$>= 3$	2
$<= 3$	$<= 3$	$!X2$	2	$!X2$	$== 1$	$>= 3$	0
$!= 1$	$== 0$	$!= 2$	2	$<= 2$	$>= 0$	$>= 3$	2
$!X0$	$!X2$	$X3$	0	$!X1$	$== 1$	$X1$	2
$>= 1$	$>= 1$	$!= 2$	1				
(1. část)				(2. část)			

Tabulka B.1: Podmínková pravidla

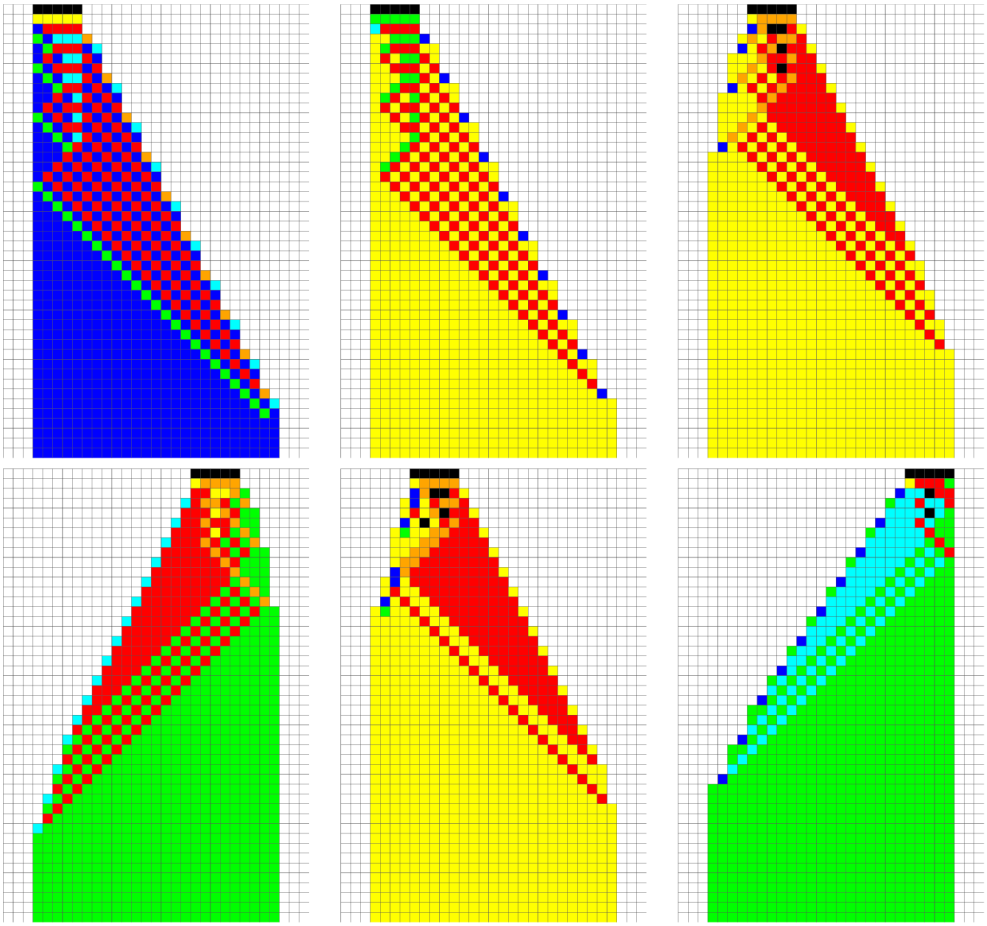
[0, 2, 0, 2, 2, 2, 2, 2, 0, 1, 2, 3, 1, 1, 1, 1, 2, 1, 2, 3, 1, 1, 1, 1, 0, 1, 2, 3, 1, 1, 1, 1, 0, 1, 2, 3, 1, 1, 1, 1, 0, 1, 2, 3, 1, 1, 1, 1, 0, 1, 2, 3, 1, 1, 1, 1, 0, 1, 2, 3, 1, 1, 1, 1, 0, 0, 0, 2, 2, 2, 2, 2, 2, 3, 2, 3, 3, 3, 3, 3, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 0, 2, 0, 2, 2, 2, 2, 2, 2, 3, 2, 3, 3, 3, 3, 3, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 1, 1, 1, 1, 1, 3, 1, 3, 1, 1, 1, 1, 1, 3, 1, 3, 1, 1, 1, 1, 1, 3, 1, 3, 1, 1, 1, 1, 1, 3, 1, 3, 1, 1, 1, 1, 1, 3, 1, 3, 1, 1, 1, 1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 1, 3, 2, 3, 3, 3, 3, 3, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 1, 3, 2, 3, 3, 3, 3, 3, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 1, 3, 2, 3, 3, 3, 3, 3, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 1, 3, 2, 3, 3, 3, 3, 3, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1]

Tabulka B.2: Pole nových hodnot buňky pro tabulkovou reprezentaci

Příloha C

Ukázky různých řešení 1D výpočtu

V této příloze je uvedeno několik příkladů výpočtu mocniny vzniklých v průběhu psaní práce pro srovnání a hrubou představu, jak se od sebe jednotlivá řešení liší. Obrázky mají sloužit jako ilustrační příklad, a proto nejsou uvedeny žádné další podrobnosti.



Obrázek C.1: Vizualizace různých řešení výpočtu mocniny pro číslo 5

Příloha D

Stručný návod k aplikaci

Implementovaná aplikace obsahuje dvě části. První částí je Java simulátor ve kterém se provádí evoluce, vyhodnocování atd. Druhou částí je jednoduchá html/javascript aplikace pro vizualizaci 1D automatů.

HTML vizualizátor

Tato část aplikace je ve složce jménem **simpleCellular**. Jejím hlavním souborem je *./public_html/index.html*, ve kterém se nastavují akce, co provést. Nejčastěji se používá tak, že nastavíme počet kroků automatu, podmínková pravidla a počáteční stav. Tímse na výstup vizualizuje průběh daného automatu.

```
var steps = 45;
var string = "<=5|X3|>=4|5;<=5|X5|!=1|2;!X0|X1|==5|0;==4|!=3|>=3|2;...";
ruleSet.init(string);
cellular.init("00000000000000000000000011111000", ruleSet, svgPrinter);
```

Obrázek D.1: Ukázka použití vizualizátoru.

Tato aplikace také byla využívána na zkoumání a vizualizaci tabulkových pravidel. Lze podle nich vizualizovat a tabulková pravidla z podmínkových vytvářet.

```
var string = "<=5|X3|>=4|5;<=5|X5|!=1|2;!X0|X1|==5|0;==4|!=3|>=3|2;...";
ruleSet.init(string);
tableSet.setTable(ruleSet.createTable());
tableSet.printTableArray(8);
```

Obrázek D.2: Ukázka převedení podmínkových pravidel na tabulkové.

JAVA simulátor

Hlavní část aplikace je implementace simulací pro experimenty, a nachází se ve složce **Diplomka**. Jedná se o Java aplikaci s několika závislostmi na externí knihovny. Nejjednodušší

způsob zprovoznění je otevřít/importovat zdrojové soubory v nějakém IDE Netbeans/Eclipse/Idea a přidat do projektu přiložené knihovny (složka **Diplomka/libs**).

Projekt obsahuje několik hlavních tříd (v balíku *cz.arcanis.dip.celular*), která každá slouží k jinému účelu. SimpleMain je třídou pro spouštění experimentů, MainVisualize zase slouží pro vizualizaci výsledků. Ostatní hlavní třídy mají různé pomocné a formátovací funkce, některé byly použity jen pro vývoj.

V SimpleMain je potřeba nastavit kolik experimentů budeme provádět, kolik paralelních jader procesoru budeme používat, a pak třídu s konfigurací experimentu. Tyto konfigurace (balík *cz.arcanis.dip.celular.config*, všechny dědí od třídy *EvolutionConfig*) obsahují různé parametry experimentů dle kterých se simulace řídí.

O průběhu evoluce informuje okno prostředí Swing, kde je také zobrazen uběhlý čas a pravděpodobný zbývající čas experimentu. Výsledky se vypisují do konzole a také do souboru *log1.txt* ve složce projektu. Podmínková pravidla pro každé řešení se kvůli přehlednosti nevypisují do konzole ale jen do souboru.