VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

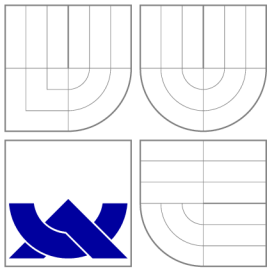
NOVÉ TECHNIKY NÁVRHU CELULÁRNÍCH AUTOMATŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

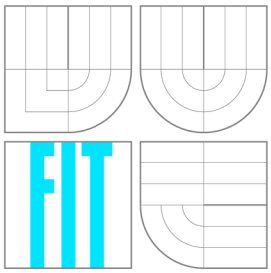
AUTOR PRÁCE
AUTHOR

Bc. MARTIN BALÁŽ

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

NOVÉ TECHNIKY NÁVRHU CELULÁRNÍCH AUTOMATŮ

NEW CELLULAR AUTOMATA DESIGN TECHNIQUES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN BALÁŽ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAL BIDLO, Ph.D.

BRNO 2013

Abstrakt

Cílem této diplomové práce je uvést novou techniku návrhu celulárních automatů, která poskytne lepší možnosti implementace a řešení zadaných problémů v prostředí neuniformních automatů. V rámci této práce byly položeny teoretické základy k problematice celulárních automatů a byly přezkoumány možnosti jejich návrhu pomocí dvou evolučních principů, které se běžně používají — genetický algoritmus a celulární programování. Byly vybrány dva principiálně výrazně odlišné problémy, na kterých byly možnosti a schopnosti těchto technik prověřeny: problém synchronizace a systém implementace logických hradel v prostředí celulárního automatu. Na základě přezkoumání implementačních vlastností a na základě prvních výsledků použití těchto metod byla vytvořena nová metoda návrhu celulárních automatů — celulární evoluce. Celulární evoluce se svojí metodou "predikce budoucího stavu okolních buněk"poskytuje nové možnosti v návrzích automatů, jelikož pracuje se strukturovanými geny, které umožňují, aby byl daný gen aktivní pro více různých celulárních okolí. V závěru práce byly na vybraných dvou problémech porovnané všechny tři metody a jejich schopnosti byly shrnuty v podrobném přehledu.

Abstract

The aim of this master thesis is to introduce a new technique for the design of cellular automata which will provide a better possibilities for the implementation and solving given problems in an environment of non-uniform automata. In this work, the theoretical foundations of cellular automata have been summarized and the possibilities of their design were examined using two evolutionary principles that have commonly been used — genetic algorithm and cellular programming. Two principally different issues were selected on which the possibilities and capabilities of these techniques were proven: the synchronization problem and the system of implementation of logic gates in an environment of cellular automata. Based on a review of the implementation properties and the initial results of usage of these methods a new design method for cellular automata was created — cellular evolution. The cellular evolution with its method of "prediction of the future state of surrounding cells"provides new possibilities in the design of cellular automata since it operates with structured genes which allow the gene to be active for a variety of cellular surroundings. In the conclusion of this work, all three methods were compared on two selected problems and their abilities were summarized in a detailed overview.

Klíčová slova

Celulární automat, genetický algoritmus, celulární programování, úloha synchronizace, systém logických hradel, celulární evoluce, strukturovaný gen, predikce budoucího stavu okolních buněk.

Keywords

Cellular automaton, genetic algorithm, cellular programming, synchronization task, logic gates, cellular evolution, structured gene, prediction of neighboring cell state.

Citace

Martin Baláž: Nové techniky návrhu celulárních automatů, diplomová práce, Brno, FIT VUT v Brně, 2013

Nové techniky návrhu celulárních automatů

Prohlášení

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

.....
Martin Baláž
5. května 2013

Poděkování

Ďakujem vedúcemu práce Ing. Michalovi Bidlovi, Ph.D. za poskytnutie cenných rád, ktoré mi pomohli pri vytváraní tejto práce.

Obsah

Kapitola 1

Úvod

Jadrom tejto diplomovej práce je celulárny automat — diskrétny dynamický model, ktorý má široké možnosti využitia, od modelovania fyzikálnych javov až po simuláciu dopravy či dokonca simuláciu "života". Celulárny automat je jednoduchý model, tvorený mriežkou buniek riadených určitou prechodovou funkciou, ktorý dokáže pracovať s jednoduchými štruktúrami na jednej strane, a zároveň dokáže implementovať štruktúry, ktoré majú silu ekvivalentnú s Turingovými strojmi na strane druhej [12]. Táto práca sa zameriava najmä na druhý koniec spektra využiteľnosti — na neuniformné automaty, ktoré umožňujú návrh veľmi komplexných štruktúr.

Objavenie optimálnej konfigurácie parametrov takéhoto celulárneho automatu tak, aby dokázal uspokojivo riešiť zadaný problém nemusí byť triviálna úloha, keďže stavový priestor, v ktorom sa nachádza riešenie, môže byť obrovský. Preto bude táto práca venovaná možnostiam návrhu neuniformných celulárnych automatov s využitím evolučných techník, ako stochastických optimalizačných metód schopných objaviť vhodné riešenia problémov aj v naozaj obrovských stavových priestoroch.

Výpočtový model evolučných techník ma svoj pôvod v biologickej evolúcií, ktorej úlohou je prispôsobovať či optimalizovať jedincov tak, aby boli schopní prežiť v danom prostredí. Prenesením myšlienok biologickej evolúcie do výpočtového sveta vzniká idea evolučných algoritmov, ktoré získavajú schopnosti prehľadania obrovského stavového priestoru v relatívne "rozumnom"časovom horizonte a zároveň si zachovávajú schopnosť objaviť vhodné riešenie problému.

Cieľom tejto diplomovej práce je vytvoriť úplne novú a originálnu techniku návrhu neuniformných celulárnych automatov, vystavanú na základoch a princípoch evolučných techník, a porovnať ju s metódami, ktoré sa v súčasnosti celkom bežne používajú pri konštrukcií tohto typu automatov.

V prvej fáze riešenia diplomovej práce boli preskúmané bežne používané evolučné techniky používané pri návrhu celulárnych automatov — genetický algoritmus a celulárne programovanie. Schopnosti týchto dvoch metód boli predbežne preverené na dvoch vybraných úlohách, ktoré sú principiálne výrazne odlišné — úloha synchronizácie a systém logických hradiel. Zatiaľčo prvá úloha preverí algoritmy z hľadiska vynucovania globálneho správania, druhá úloha preverí schopnosť vyriešenia problému komplexnej štruktúry.

V druhej fáze projektu bola navrhnutá a implementovaná úplne nová technika umožňujúca návrh celulárnych automatov — získala pomenovanie Celulárna Evolúcia. Táto čiastočne vychádza z vyššie zmienených techník, ale zároveň prináša úplne nový spôsob evolučnej práce s celulárnym automatom a navyše prichádza so zaujimavým rozšírením "predikcie budúceho stavu", ktoré je implementovateľné aj v iných evolučných metódach.

Záverečná fáza práce bola zameraná na experimentovanie s novou metódou na vybraných úlohách a porovnaniu získaných výsledkov so zaužívanými metódami genetického algoritmu a celulárneho programovania.

Štruktúra tejto práce je rozdelená nasledovne: nadchádzajúce tri kapitoly (II., III. a IV.) sú venované popisu celulárnych automatov, vysvetleniu princípov evolučných algoritmov (so zameraním na genetický algoritmus) a objasneniu implementácie metódy celulárneho programovania.

Nasleduje kapitola V. venovaná podrobnému popisu oboch porovnávacích úloh — teda úlohe synchronizácie a systému implementácie logických hradiel v prostredí neuniformného celulárneho automatu.

Kapitola VI. je venovaná samotnému popisu a podrobnému vysvetleniu algoritmu a princípu fungovania najdôležitejších komponént metódy celulárnej evolúcie. Nasleduje kapitola VII. vysvetlujúca konkrétnu implementáciu jednotlivých aplikácií obalujúcich všetky tri zmienené evolučné techniky.

Záverečná časť práce obsahuje kapitolu VIII. venovanú experimentom a porovnaniu všetkých troch metód, vyzdvihnutiu silných a slabých stránok metódy celulárnej evolúcie.

Kapitola 2

Celulárne automaty

Táto diplomová práca je zameraná na hľadanie nových techník návrhu celulárnych automatov (ďalej CA) a preto sa nasledujúca kapitola zaoberá popisom ich vlastností a vysvetľuje základné princípy výpočtov v tomto type výpočtových štruktúr.

Celulárny automat sa spája s menami John von Neumann a Stanislaw Ulam a jeho vznik sa datuje do obdobia 40. rokov 20. storočia, keď títo vedci pracovali vo výskumnom zariadení Los Alamos National Laboratory v Novom Mexiku, USA. Von Neumann sa zaoberal vývojom abstraktného modelu založeného na biologickej sebe-reprodukcií, ktorá bola zaujimavá z pohľadu kybernetiky. Svoju prácu začal úvahami o modele vychádzajúcom z trojrozmerného systémom popísaného pomocou parciálnych diferenciálnych rovníc.

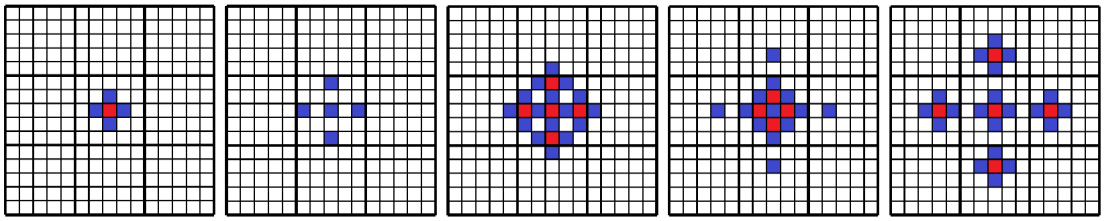
2.1 Neformálny popis

Sipper [12] popisuje celulárny automat ako diskrétny dynamický model, ktorý je tvorený pravidelným, N-rozmerným poľom buniek, ktoré si pre ďalšie použitie označíme pojmom mriežka. Vo svojej podstate môže mať celulárny automat ľubovoľný počet dimenzií, ale v reálnych aplikáciach sa využívajú jedno- až dvoj-dimenzionálné automaty, výnimočne sa môžeme stretnúť s troj-rozmerným automatom. Automaty s väčším počtom dimenzií môžu klásť značné nároky na výpočtový čas, čo je jedným z hlavných dôvodov, prečo sa s nimi nestretávame v praxi príliš často.

Séria obrázkov 2.1 zachytáva príklad vývoja CA, ktorý rieši problém replikácie objektu [1]. V jednotlivých diskrétnych časových krokoch môžeme pozorovať ako sa postupne menia stavy v jednotlivých bunkách mriežky.

Dimenzie mriežky CA teoreticky nie sú obmedzené čo sa veľkosti (počtu buniek) týka, prakticky sa z implementačných dôvodov používajú mriežky s konečnou veľkosťou. Veľkosti dimenzií nie sú na sebe závislé — každá dimenzia teda môže obsahovať rozdielny počet buniek. Konečná veľkosť mriežky však so sebou prináša otázku, ako sa zachovať k bunkám, ktoré tvoria hranicu mriežky. V tomto prípade musíme na okrajoch mriežky počítať s obmedzujúcimi podmienkami, keďže hraničné bunky majú menší počet existujúcich susedných buniek. Na vyriešenie problému chýbajúcich susedov okrajových buniek sa obvykle používajú dve riešenia — v prvom prípade chýbajúcich susedov nahradíme virtuálnymi bunkami

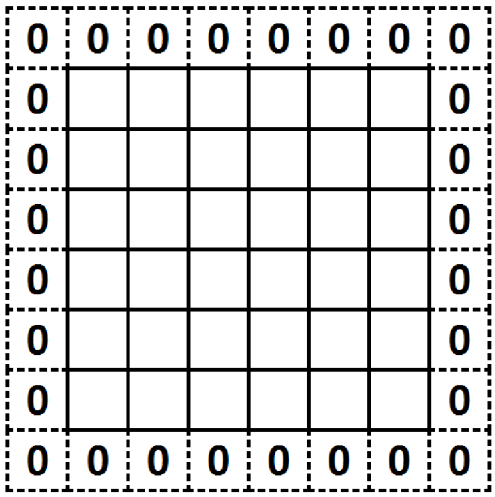
[1] Celulárny automat zachytený na obrázku 2.1 je uniformný (to znamená, že všetky bunky sú riadené pomocou rovnakej lokálnej prechodovej funkcie), bunky môžu nadobúdať jeden z troch možných stavov. Ako celulárne okolie bolo zvolené 5-okolie (viď obrázok 2.5a) a bunečná mriežka je dvojrozmerná s cyklickými okrajovými podmienkami (viď obrázok 2.3).



(a) Iniciálny krok **(b)** 1. krok **(c)** 2. krok **(d)** 3. krok **(e)** 4. krok

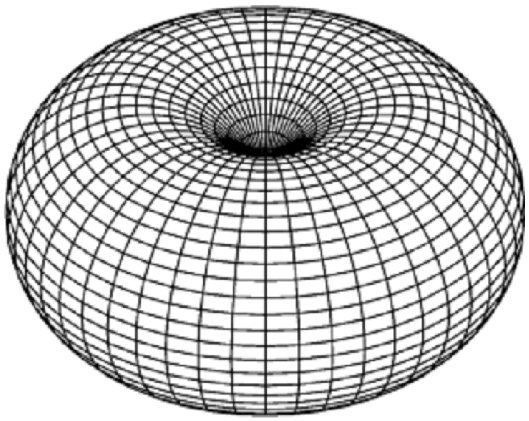
Obrázek 2.1: Príklad vývoja celulárneho automatu riešiaceho problém replikácie objektu. Počiatočný objekt zachytený v iniciálnom kroku CA sa v priebehu niekoľkých nasledujúcich krokov vývoja CA replikuje do niekoľkých kópií.

s pevne zvoleným stavom. Pre dvojrozmerný celulárny automat zobrazuje túto variantu obrázok 2.2 a nazývame ju celulárnym automatom s konštantnými okrajovými podmienkami. Druhou alternatívou je celulárny automat s cyklickými okrajovými podmienkami. U jednorozmerného automatu vznikajú cyklické okrajové podmienky tak, že prvá bunka sa spojí s poslednou, čím vznikne kružnica tvorená bunkami, v ktorom má každá bunka dvoch bezprostredných susedov. V prípade dvojrozmerného automatu spojíme pravý a ľavý okraj mriežky, čím vzniká valec a následne spojíme vrchnú podstavu so spodnou a vzniká objekt, ktorý nazývame torus (obrázok 2.3). Tento spôsob môžeme aplikovať aj na automaty vyšších dimenzií čím získavame cyklický, kvázi-nekonečný model, ktorý nevyžaduje žiadne špeciálne ošetrenie okrajových podmienok pri výpočte stavov jednotlivých buniek na okraji mriežky celulárneho automatu.



Obrázek 2.2: Konštantné okrajové podmienky v 2-dimenzionálnom CA (v tomto prípade nulové okrajové podmienky). Virtuálne bunky sú vyznačené čiarkovanou čiarou.

Každá z buniek celulárneho automatu sa môže nachádzať v jednom z konečného počtu stavov. Použitý počet stavov je daný konkrétnou aplikáciou, najjednoduchšie implementácie využívajú dvojstavové CA, ako je napríklad CA použitý v aplikácií Game of Life [4]. Na druhej strane, automat vytvorený Von Neumannom, ktorý emuloval operácie elektrického

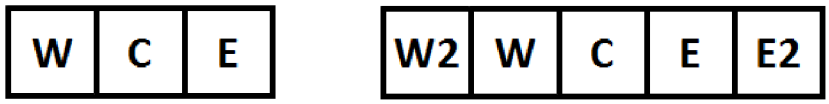


Obrázek 2.3: Cyklické okrajové podmienky transformujú 2-dimenzionálny CA do objektu nazývaného torus. *Prebraté z [10].*

počítača, pracoval s 29 stavmi [13].

Počiatočný stav celulárneho automatu (mriežka CA v čase $t = 0$) je definovaný "ručne", určením konkrétneho stavu každej bunky v mriežke. Tento počiatočný stav je nutné definovať ešte pred zahájením výpočtu a nazýva sa počiatočná konfigurácia celulárneho automatu.

Ďalšou vlastnosťou celulárneho automatu je tzv. celulárne okolie bunky. Stavy všetkých buniek z celulárneho okolia danej bunky sú vstupným parametrom lokálnej prechodovej funkcie (viď nasledujúci text). Celulárnym okolím bunky nazývame množinu okolitých buniek, ktorá je definovaná relatívnym rozmiestnením buniek s ohľadom na centrálnu bunku okolia (centrálnou bunkou okolia nazývame bunku, pre ktorú aktuálne počítame nový stav) a ich vzdialenosťou od centrálnej bunky okolia. Okolia typicky používané u jednorozmerných automatov vidíme na obrázku 2.4. V prípade dvojrozmerného automatu sa typicky používajú okolia znázornené na obrázku 2.5.

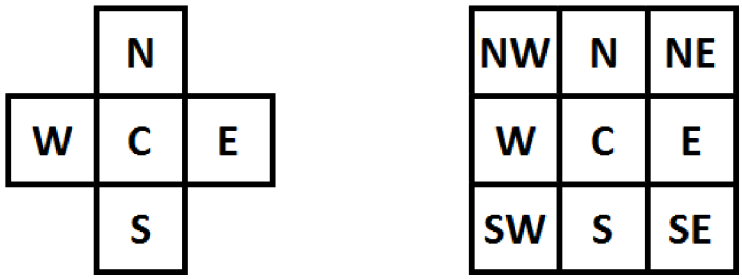


(a) 3-okolie **(b)** 5-okolie

Obrázek 2.4: Bežne používané okolia jednorozmerného CA.

Výpočet v celulárnom automate prebieha v diskrétnych krokoch. V každom časovom kroku sa vypočíta nový stav všetkých buniek v mriežke CA. Nový stav bunky sa určí na základe stavu všetkých buniek z jej celulárneho okolia pomocou lokálnej prechodovej funkcie (v ďalšom texte *prechodová funkcia* alebo LPF).

CA je diskrétny a dynamický model a teda výpočet v celulárnom automate prebieha



(a) Von Neumannovo okolie (5-okolie) **(b)** Moorovo okolie (9-okolie)

Obrázek 2.5: Bežne používané okolia dvojrozmerného CA.

v diskrétnom čase — v každom časovom kroku prebehne paralelne výpočet nového stavu v každej bunke, čím dôjde k aktualizácií stavu všetkých buniek v mriežke CA. Výpočet zaisťujúci aktualizáciu stavu buniek v celulárnom automate je určený pomocou tzv. lokálnej prechodovej funkcie. Vstupným parametrom lokálnej prechodovej funkcie je aktuálny stav bunky, ktorej nový stav práve počítame, a stavy všetkých buniek z definovaného celulárneho okolia tejto bunky. Z hľadiska prechodových funkcií sa celulárne automaty rozdelujú na uniformné a neuniformné. Uniformný CA je charakterizovaný tým, že všetky bunky v mriežke sa riadia podľa rovnakej prechodovej funkcie. Neuniformný automat umožňuje, aby každá bunka bola počas celulárneho výpočtu riadená vlastnou prechodovou funkciou, ktorá všeobecne môže byť pre každú bunku mriežky unikátna.

Prechodová funkcia môže byť zapísaná vo forme tabuľky, ktorá pre každú kombináciu stavov vstupného celulárneho okolia definuje hodnotu stavu centrálnej bunky okolia v nasledujúcom kroku výpočtu CA. Táto tabuľka býva označovaná aj pojmom tabuľka pravidiel lokálnej prechodovej funkcie. Ako príklad tabuľkového zápisu prechodovej funkcie uvádzame tabuľku 2.1 zobrazujúcu tabuľku pravidiel pre tzv. paritné pravidlo (známe aj ako XOR pravidlo) [12]. Celulárny automat využívajúci túto prechodovú funkciu je dvojstavový, dvojrozmerný a využíva celulárne 5-okolie. Paritné pravidlo v tomto prípade priradí stav 1 každej bunke, ktorej okolie obsahuje nepárny počet buniek nachádzajúcich sa v stave 1, inak priradí stav 0. Z tejto tabuľky je jasne vidieť, že každému pravidlu zodpovedá práve jedna kombinácia stavov okolia celulárneho automatu.

Každá možná kombinácia stavov buniek celulárneho okolia musí mať definované pravidlo v tabuľke pravidiel (respektíve musí byť ošetrená lokálnou prechodovou funkciou). Celkový počet pravidiel potom závisí na počte stavov a na počte buniek použitého celulárneho okolia. Nech s je celkový počet stavov a c je počet buniek vo zvolenom okolí. Potom celkový počet variacií s opakovaním, ktoré je možné vyjadriť a teda počet pravidiel v tabuľke pravidiel vyjadruje rovnica:

$$r_{count} = s^c. \tag{2.1}$$

Pri pohľade na konkrétne čísla pre vyššie uvedené typy celulárnych okolí uvidíme nárast počtu pravidiel v závislosti na počte buniek okolia. Napríklad jednodimenzionálny automat

CNWSE	S_{next}	CNWSE	S_{next}	CNWSE	S_{next}	CNWSE	S_{next}
00000	0	01000	1	10000	1	11000	0
00001	1	01001	0	10001	0	11001	1
00010	1	01010	0	10010	0	11010	1
00011	0	01011	1	10011	1	11011	0
00100	1	01100	0	10100	0	11100	1
00101	0	01101	1	10101	1	11101	0
00110	0	01110	1	10110	1	11110	0
00111	1	01111	0	10111	0	11111	1

Tabulka 2.1: Tabuľka pravidiel lokálnej prechodovej funkcie pre paritné pravidlo. CNWSE predstavuje aktuálny stav buniek Von Neumannovho okolia podľa obrázku 2.5a, S_{next} určuje stav centrálnej bunky okolia v nasledujúcom kroku výpočtu.

využívajúci 3 okolie podľa obrázku 2.4a má $r_{count} = 2^3 = 8$ pravidiel, automaty s piatimi bunkami v okolí (obrázky 2.4b a 2.5a) majú celkový počet pravidiel $r_{count} = 2^5 = 32$, pri 9-okolí (obrázok 2.5b) je to už $r_{count} = 2^9 = 512$.

Počet všetkých prechodových funkcií, ktoré je možné nad dvojstavovým celulárnym automatom realizovať, závisí od rozsahu tabuľky pravidiel:

$$f_{count} = 2^{r_{count}}. \tag{2.2}$$

S narastajúcim počtom stavov a počtom buniek v okolí sa značne zvýši celkový počet prechodových funkcií. Už pri jednoduchom automate s 2 stavmi a 5-okolím (a teda s 32 pravidlami podľa vzorca 2.1) je počet prechodových funkcií $f_{count} = 2^{2^5} = 2^{32} = 4\,294\,967\,296$.

Výpočtový model celulárneho automatu je všeobecný a jednoduchý [12]. Všeobecnosť implikuje, že (1) CA podporuje všeobecné výpočty a teda je schopný spracovať akýkoľvek algoritmus [12] a (2) základné jednotky modelu využívajú formu lokálnych interakcií miesto špecializovaných akcií. Jednoduchosť znamená, že výpočtové jednotky – bunky – sú jednoduchšie v porovnaní s Turingovými strojmi. Na miere komplexnosti modelov stoja Turingové stroje na jednom konci a druhý koniec reprezentujú jednoduché modely akými sú konečné automaty. Celulárny automat je jeden z najjednoduchších všeobecných modelov [12].

Kapitola 3

Evolučné algoritmy

V rámci tejto diplomovej práce sú skúmané rôzne prístupy k návrhu celulárnych automatov. Jednou z techník, ktoré sa používajú sú evolučné algoritmy. Evolučný prístup hľadania optimálneho riešenia sa úspešne využíva u problémov, ktoré sú charakteristické obrovským stavovým priestorom možných riešení.

Kapitola 2.1 uvádza, že stavový priestor — počet prechodových funkcií — je v prípade celulárnych automatov značný, pritom počet vhodných riešení, ktoré sú schopné zaistiť požadované správanie automatu môže byť veľmi malý. Evolučný prístup v prípade takejto optimalizačnej úlohy môže viesť k značnému ušetreniu času, potrebného na nájdenie riešenia a na druhej strane môže objaviť unikátne riešenie, aké by sme ručným návrhom prechodovej funkcie pravdepodobne nebrali do úvahy.

Táto kapitola je venovaná princípu evolučného návrhu, popisuje vzor na ktorého základe evolučné techniky stavajú a podrobne rozoberá jeden konkrétny typ — genetický algoritmus, ktorý bude pri návrhu celulárneho automatu použitý.

3.1 Základný princíp evolučných algoritmov

Rodina evolučných algoritmov [9] sa radí k stochastickým optimalizačnýcm metódam. Ich predlohu či inšpiráciu nájdeme v živej prírode — jedná sa o evolúciu, ktorou sa zaoberá moderná biológia a ktorá bola prvýkrát popísaná jedným z najvýznamnejších vedcov 19. storočia, Charlesom Darwinom, keď v roku 1859 publikoval knihu *O pôvode druhov* [2].

Zatiaľčo štandardné optimalizačné metódy sú založené na optimalizácií jediného kandidátneho riešenia, evolučne algoritmy pracujú s množinou kandidátnych riešení — s populáciou. Každé kandidátne riešenie (tzv. fenotyp) predstavuje jedného jedinca populácie a je zakódované do genotypu. Genotyp je teda reprezentáciou fenotypu v evolučnom algoritme.

Evolučný algoritmus pracuje so štruktúrami zvanými genotyp, do ktorých je zakódovaná kompletná genetická informácia jedinca. Prírodná reprezentácia evolúcie má vo väčšine prípadov genotyp rozdelený do samostatných častí — chromozómov, ktoré kódujú jednotlivé vlastnosti fenotypu jedinca. Evolučné techniky obvykle využívajú zjednodušenú formu reprezentácie jedincov – všetky vlastnosti jedinca sú zakódované do jedného genotypu, ktorý je nositeľom všetkej genetickej informácie kandidátneho riešenia. Vďaka tomuto zjednodušeniu je možné miesto slova genotyp používať v prípade evolučných algoritmov aj slová genóm či chromozóm.

Algoritmus 3.1 dokumentuje princíp evolučného algoritmu. Nasledujúce odstavce tejto sekcie sú venované stručnému vysvetleniu fungovania jednotlivých častí tohto algoritmu.

```
gen = 0
gen_max = N
vytvor inicializačnú populáciu P(gen)
ohodnoť každého jedinca v P(gen)
while (nenájdeš vhodné riešenie) && (gen < gen_max) do
  gen += 1
  vyber vhodných jedincov do nasledujúcej generácie P(gen)
  vytvor novú generáciu P(gen) aplikovaním genetických operátorov
  ohodnoť každého jedinca v P(gen)
end while
```

Algoritmus 3.1: Princíp evolučného algoritmu [9].

Pred samotným začiatkom práce evolučného algoritmu je nutné vytvoriť počiatočnú populáciu kandidátnych riešení (tzv. nultá generácia populácie). Táto obvykle vzniká náhodným vygenerovaním genómov. Aby bola zaistená dostatočná diverzita v populácií, je nevyhnutné aby populácia obsahovala dostatočný počet genómov. Tento počet sa obvykle v priebehu evolučnej optimalizácie nemení. Optimalizácia samotná následne priebeha nad celou populáciou genotypov v jednotlivých krokoch evolúcie — v generáciach.

Výpočet jednej generácie teda predstavuje jeden krok evolučného algoritmu. Nasledujúca generácia vzniká z aktuálnej generácie v procese reprodukcie, ktorý je v základnej podobe tvorený štyrmi fázami.

Prvá fáza zaisťuje ohodnotenie každého jedinca z populácie na základe toho, ako kvalitne spĺňa požiadavky kladené na finálne riešenie — tento krok sa nazýva výpočet hodnoty fitness. Čím viac sa ohodnocovaný jedinec blíži k požadovanému riešeniu, tým je kvalitnejší.

S výpočtom fitness hodnoty súvisí otázka mapovania genotypu na fenotyp. Evolučný algoritmus pracuje vo zvyšných troch fázach výhradne s genotypom, avšak v procese výpočtu fitness hodnoty je nevyhnutné vybudovať z genotypu každého jedinca populácie príslušný fenotyp, nad ktorým sa určí ako kvalitne tento jedinec (respektíve toto kandidátne riešenie) spĺňa vlastnosti požadovaného finálneho riešenia. Mapovanie genotypu na fenotyp môže byť priame alebo nepriame. V prípade priameho mapovania je fenotyp priamo zakódovaný do chromozómu, pri použití nepriameho mapovania obsahuje genotyp len predpis, na základe ktorého je vybudovaný fenotyp. Jednoduchšie problemy si vystačia s priamym mapovaním, komplikovanejšie problémy väčšinou vyžadujú nejaký typ kódovania nepriameho. V tomto bode poznamenávame, že niektoré typy evolučných algoritmov nerozoznávajú pojmy genotyp a fenotyp (napríklad genetické programovanie [7]).

Po tom, ako sú všetky kandidátne riešenia ohodnotené a ich fitness je priradená príslušným genómom, preklápa sa evolučný algoritmus do svojej druhej fázy. Táto fáza sa nazýva selekcia chromozómov, ktorá je charakterizovaná tým, že dochádza k výberu vhodných jedincov z aktuálnej generácie (t.j. z generácie, ktorú sme práve ohodnotili vo fáze jedna), ktorí sa budú podieľať na vzniku generácie novej. Na výber vhodných chromozómov existuje mnoho metód, všeobecne môžeme povedať, že čím kvalitnejší jedinec, tým väčšia je pravdepodobnosť výberu jeho genómu.

V tretej fáze sa na vybratých jedincov aplikujú genetické operátory, ktoré majú formálny základ v procesoch biologickej evolúcie. Výber typu genetických operátorov, ako aj ich konkrétna implementácia, závisí jednak na konkrétnom type evolučného algoritmu, ale aj na úlohe, ktorú algoritmom riešime. Typicky používanými operátormi sú kríženie rodičovských genómov a mutácia.

Aplikáciou genetických operátorov na jedincov vybratých procesom selekcie (tzv. ro-

dičia), vznikajú noví jedinci (potomkovia). O vznik nových jedincov sa stará operátor kríženia, vďaka ktorému potomkovia preberajú časť genetického materialu od každého z rodičov, a teda kríženie zaisťuje prenos genetických informacií z aktuálnej generácie do novej populácie. Operátor mutácie má za úlohu doniesť do populácie nové vlastnosti.

V záverečnej fáze dochádza k vytvoreniu novej populácie, ktorá vstupuje do ďalšej generácie evolučného algoritmu — tento proces sa nazýva reprodukcia. Jedinci, ktorí vznikajú aplikovaním genetických operátorov, väčšinou automaticky postupujú do novej generácie. V tejto fáze však musíme vyriešiť problém "prežívania" jedincov z generácie predchádzajúcej. Táto situácia je v evolučných algoritmoch riešená väčšinou dvoma spôsobmi: (1) nová populácia je vytvorená výhradne z jedincov vzniknutých aplikáciou genetických operátorov a teda starí jedinci z aktuálnej populácie vymierajú. Druhou (2) alternatívou je umožniť časti najlepších jedincov aktuálnej populácie prechod do populácie novej bez akejkoľvek zmeny genetického materiálu — jedná sa o tzv. *elitizmus*. Riešenie (1) v sebe skrýva zásadnú nevýhodu v podobe možnej straty veľmi kvalitného riešenia. Genetické operátory negarantujú, že v každej generácií vzniká z kvalitných rodičov minimálne rovnako kvalitný potomok, a teda táto strata genetického materiálu by mohla byť nenávratná. V prípade aplikovania riešenia (2) je zaistené, že najkvalitnejší jedinci sú schopný ovplyvňovať viac ako jednu generáciu [6].

Aplikovaním týchto štyroch fáz na aktuálnu generáciu vzniká generácia nová. Tento postup je periodicky aplikovaný dovtedy, kým nie je objavené riešenie, ktoré by spĺňalo kvalitatívne požiadavky (teda kým nejaký jedinec nedosiahne požadovanú hodnotu fitness), prípadne kým nie je presiahnutá stanovená medz (napríklad čas výpočtu alebo počet generácií).

V súčasnosti sa používa viacero prístupov k implementácií evolučných algoritmov, ktoré vznikli rôznou interpretáciou Darwinovej teórie. Tieto implementácie sa od seba viac či menej líšia hoci ich princíp ostáva podobný. Medzi najznámejšie varianty patria genetické algoritmy uvedené Hollandom [5], genetické programovanie, ktorého autorom je Koza [7], evolučné programovanie od Fogela [3] a evolučné stratégie od Bienerta, Rechenberga a Schwefela [11].

3.2 Genetický algoritmus

Nasledujúca sekcia je venovaná jednej z najpoužívanejších techník spomedzi evolučných algoritmov — genetickému algoritmu (ďalej budeme používať skratku GA). Podobne ako existuje mnoho evolučných techník, existuje aj viacero viac či menej odlišných implementácií genetického algoritmu. V tomto texte sa budeme zaoberať základnou podobou genetického algoritmu, ktorý sa označuje aj pojmami jednoduchý či kanonický genetický algoritmus [5].

Autorom genetického algoritmu je John Holland, ktorý sa vo svojich výskumných prácach zaoberal biologickými procesmi a ich implementáciou vo výpočtových systémoch. Na výsledkoch svojích štúdií postavil abstraktný výpočtový model genetického algoritmu [5].

Od svojho vzniku našiel GA uplatnenie vo veľmi širokej škále najrôznejších aplikácií, od optimalizačných úloh až po aplikácie zamerané na skúmanie života [9].

Genetické algoritmy sa radia do skupiny evolučných algoritmov, ktoré rozlišujú genotyp a fenotyp. Genotyp alebo tiež genóm je vytvorený z lineárnej postupnosti génov.

Keďže daná vlastnosť fenotypu je väčšinou kódovaná v genotype len jediným génom, je nevyhnutné, aby tento gén mohol nadobúdať rôzne hodnoty. Každá unikátna hodnota génu sa označuje ako alela. Väčšia škála hodnôt (t.z. väčší počet alél) zaisťuje, že daná vlasnosť sa môže vyskytovať v rôznych podobách (napríklad gén kódujúci farbu očí sa vždy

vyskytuje v rámci genómu na jednom pevne danom mieste, a vďaka rôznym hodnotám ktoré nadobúda je schopný zakódovať viacero rôznych farieb očí). Vďaka tejto vlastnosti je zaistené, že genóm daného druhu, ktorý má pevne danú štruktúru predurčenú poradím jednotlivých génov, je schopný zakódovať mnoho navzájom odlišných fenotypov.

Jednotlivé vlastnosti fenotypu sú teda zakódované v podobe samostatných génov v genotype. Každá variácia konkrétnej vlastnosti fenotypu je reprezentovaná pomocou unikátnej hodnoty (alebo unikátnej postupnosti hodnôt), ktorá ju odlišuje od iných variácií. V jednoduchom GA má genotyp obvykle konštantnú dĺžku.

Jednotlivé gény sú v genotype charakterozované svojou lokalitou — absolútnou polohou v rámci genómu. Táto pozícia sa nazýva lokus. Lokus daného génu predstavujúceho určitú vlastnosť fenotypu je v rámci genómu pevne daný a je nemenný. Jeho premiestnenie na iný lokus by spôsobilo, že daný gén by začal kódovať úplne inú vlastnosť.

Vlastnosť nemeniacich sa, pevne daných lokusov jednotlivých génov v rámci genómu zaručuje, že každá dvojica rôznych genómov rovnakého druhu je navzájom kompatibilná, čo je v genetickom algoritme veľmi dôležité z hľadiska reprodukcie.

Nasledujúci text podrobnejšie rozoberá a vysvetľuje najdôležitejšie časti genetického algoritmu, ktorými sú fitness hodnota, princípy selekcie genómov a dva najčastejšie používané operátory genetického algoritmu — kríženie a mutácia [9].

3.2.1 Fitness

Hoci sa môže zdať ohodnocovanie kvality jedinca ako umelo vykonštruovaná vlastnosť genetického algoritmu, nie je tomu tak. Rovnako ako ostatné časti evolučných algoritmov, má aj fitness hodnota svoj vzor v prírode. Z biologického uhla pohľadu je fitness hodnota definovaná ako relatívna schopnosť jedinca prežiť vo svojom prirodzenom prostredí a zároveň ako schopnosť odovzdať svoj genetický materiál novým jedincom do ďalšej generácie.

Fitness hodnota vo výpočtovom modele evolučného algoritmu plní v podstate úlohu "identifikátoru"kvality jedinca — obvykle je definovaná ako nezáporná hodnota, častokrát sa ale jedná o normalizovanú hodnotu z rozsahu nula až sto percent, ktorá ohodnocuje fenotyp vzhľadom k jeho schopnosti riešiť zadaný problém. Jedinec s fitness 100% v tomto modele predstavuje "dokonalé"riešenie, ktoré je schopné vyriešiť zadaný problém úplne, naopak jedinec s normalizovanou fitness 0% nerieši zadaný problém vôbec.

V skratke môžeme teda povedať, že fitness hodnota je miera umožňujúca relatívne porovnávanie jedincov populácie a umožňujúca určiť, ktorí jedinci sú schopní riešiť zadaný problém lepšie.

Genetický algoritmus je proces iteratívneho charakteru — výpočet prebieha v jednotlivých krokoch, v generáciach. V každej generácií dochádza k ohodnoteniu všetkých jedincov populácie (viď algoritmus 3.1). Evolučné algoritmy obvykle začínajú svoju činnosť s náhodne vygenerovanou populáciou, ktorej priemerná fitness nie je moc vysoká [1]. Cieľom optimalizačného procesu genetického algoritmu je, aby po určitom počte generácií našiel minimálne jedno vyhovujúce riešenie.

Populácia, respektíve priemerná hodnota fitness v populácií genetického algoritmu by teda mala v priebehu generácií konvergovať k ideálnemu riešeniu. Z hľadiska sledovania konvergencie hodnoty fitness je dôležitým pojmom tzv. "povrch fitness", ktorý predstavuje grafickú reprezentáciu hodnôt fitness. Povrch fitness je možné si predstaviť ako funkciu,

[1] Priemerná fitness inicializačnej generácie závisí na mnohých faktoroch, vo všeobecnosti však nedosahuje požadované hodnoty.

ktorej priebeh je určený fitness hodnotami všetkých možných jedincov, v závislosti na parametroch, ktoré sú dané génmi chromozómu. Potom globálne maximum (prípadne minimum, v závislosti na riešenej úlohe) tejto funkcie predstavuje ideálne riešenie zadaného problému. Jednotlivý jedinci populácie sú znázornení ako body na povrchu tejto fitness funkcie.

V jednotlivých generáciach potom môžeme sledovať postupný posun týchto bodov po povrchu fitness funkcie smerom k oblastiam s vyššou hodnotou fitness a teda môžeme pozorovať ako evolučný algoritmus postupne optimalizuje populáciu [9].

3.2.2 Selekcia

Biologickou predlohou pre procesy selekcie — výberu jedincov, ktorí sa budú podieľať na vzniku ďalšej generácie — je jedna z najvýznamejších vedeckých prác modernej éry — Darwinova teória prirodzeného výberu [2].

Práve prirodzený výber — selekcia — je tou hnacou silou, ktorá ženie biologickú evolúciu neustále vpred. Práve tento mechanizmus zaisťuje, že v prírode prežívajú len tí najsilnejší jedinci, ktorí majú ten najlepší genetický materiál, vďaka ktorému sú najlepšie prispôsobení svojmu životnému prostrediu. Selekcia, ktorá si najlepších jedincov populácie "vyberá", umožňuje, aby sa do ďalších generácií rozšíril genetický materiál práve z týchto silných jedincov. Ideálne by samozrejme bolo, ak by sa tento genetický materiál rozšíril do čo možno najväčšieho počtu potomkov, aby sa zvýšila pravdepodobnosť, že vlohy vďaka ktorým sa daní jedinci stali najlepšími vo svojej populácií, sa postupne, generácia za generáciou, rozšíria do väčšiny jedincov. Je prirodzené, že jedinci, ktorí sú lepšie prispôsobení svojmu prostrediu, žijú v priemere dlhšie ako menej prispôsobení jedinci. Vďaka tomuto faktu majú potenciálne viac príležitostí k reprodukcií svojho genetického materiálu v priebehu viacerých generácií.

Biologická evolúcia je schopná zabezpečiť chod svojej selekcie pomocou vlastných mechanizmov, ktoré sú značne komplikované a závisia na mnohých premenných, ktoré vo svojej podstate nie sú úplne predvídateľné. Genetický algoritmus je však v tomto ohľade potrebné "usmerniť", informáciu o schopnostiach a kvalitách jedincov je potrebné dodať do algoritmu externe. Jedincov populácie je teda potrebné usporiadať podľa toho, ako dobre si počínajú vo svojich životných úlohach — čím v prípade genetického algoritmu myslíme, ako dobre riešia zadaný problém. Populáciu teda usporiadame od najlepších jedincov k najhorším, a algoritmus selekcie sa následne pomocou stochastických procesov postará o výber vhodných jedincov k reprodukcií.

Mierou, podľa ktorej vytvoríme usporiadanie jedincov, je práve hodnota fitness, ktorá popisuje ako kvalitne je daný fenotyp schopný zadaný problém riešiť. Výber jedincov v procese selekcie je už potom veľmi podobný prirodzenému výberu — jedinci s vyššou fitness majú väčšiu pravdepodobnosť výberu.

3.2.3 Kríženie

Z hľadiska prenosu genetického materiálu z generácie do generácie zohráva genetický operátor kríženia významnú úlohu. Kríženie totiž zabezpečuje prenos genetického materiálu z rodičov na potomkov tak, že dôjde k vzájomnej výmene časti chromozómov medzi rodičmi.

Základná podoba kríženia sa označuje ako jednobodové kríženie a práve ním sa táto podkapitola zaoberá. A keďže kapitola ako celok je venovaná jednoduchému genetickému algoritmu, budeme ďalej predpokladať konštantnú dĺžku genómu a práve dvoch rodičov vybratých selekciou.

Za takýchto podmienok sa najskôr náhodne vyberie jeden gén (respektíve jeden lokus v rámci genómu), ktorý rozdelí oba rodičovské genómy na dve časti. Potomkovia následne vznikajú tak, že jeden potomok zdedí prvú časť genómu od jedného z rodičov a druhú časť od rodiča druhého. Druhý potomok potom zdedí presne opačné časti genómu rodičov.

Pre ilustráciu uvádza obrázok 3.1a príklad jednobodového kríženia. V tomto obrázku predstavuje spodný pár chromozómov potomkov, ktorí vznikli prekrížením vrchného, rodičovského páru za tretím génom.

Kríženie sa obvykle aplikuje s pomerne vysokou pravdepodobnosťou, [6] uvádza 75 – 95 % pravdepodobnosť aplikácie tohto operátoru. Pokiaľ by kríženie pre niektorí pár rodičov nenastalo, tak potomkovia budú presnými kópiami svojich rodičov.

Z princípu kríženia popísaného v tejto sekcií vyplýva, že tento operátor má zásadný vplyv na prenos "dobrých"vlastností do ďalšej generácie. Umožňuje totiž, aby z dvoch dobrých rodičov vznikoli kombináciou ich genetickej výbavy ešte lepší potomkovia. Keďže sa však jedná o proces stochastický, môže krížením vzniknúť aj potomok, ktorý zdedí menej výhodné časti genómu z oboch rodičov a teda nebude dosahovať kvality rodičovských genómov.

3.2.4 Mutácia

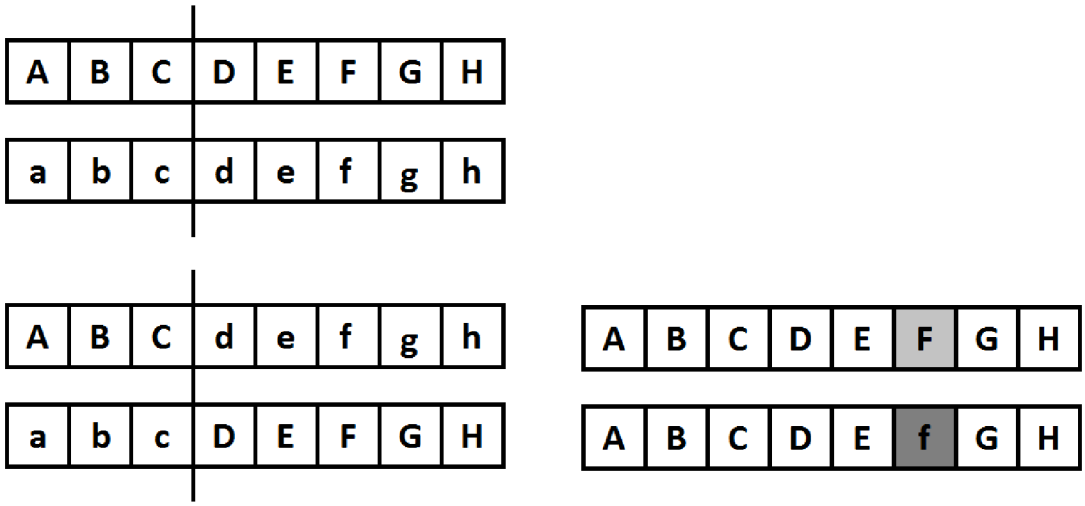
V predchádzajúcich sekciách boli vysvetlené pojmy selekcie, ktorá je hnacím motorom evolúcie, keďže zabezpečuje "odstraňovanie"slabých jedincov, a operátor kríženia umožňujúci vyselektovať najlepšie vlastnosti rodičov a preniesť ich do generácie potomkov vďaka vlastnosti dedenia génov. Tieto dve vlastnosti sami o sebe sú schopné optimalizovať populáciu tak, že jedinci sú generácia za generáciou čoraz lepší a priemerná fitness čím ďalej tým vyššia. Napriek tomu sa môže stať, že genetický algoritmus uviazne v lokálnom maxime povrchu fitness a teda k objaveniu očakávaného riešenia nikdy nedôjde. Dôvod, pre ktorý evolúcia môže uviaznúť, musíme hľadať v počiatočnej populácií — v tomto momente si musíme uvedomiť, že všetka genetická rozmanitosť bola do populácie dodaná už v počiatočnej populácií. Procesy selekcie a kríženia do populácie nedodávajú žiadne nové vlastnosti, len propagujú už existujúce dobré vlohy do ďalších generácií a odstraňujú vlohy nevhodné. Aby genetický algoritmus predišiel uviaznutiu, využíva druhý veľmi dôležitý operátor, ktorý je schopný dodať do genómu populácie nové vlastnosti a tým je genetický operátor mutácia.

Úlohou operátoru mutácie je zabezpečiť, aby sa v populácií objavovali nové vlastnosti počas celého iteračného behu genetického algoritmu, čím bude zabezpečená dostatočná variablita jedincov umožňujúca uniknúť z potenciálneho lokálneho maxima povrchu fitness.

Zabezpečenie variability mutáciou je pomerne jednoduchý proces — v prvom kroku je náhodne vybratý gén (respektíve je náhodne vybratý nejaký lokus) z genómu a následne je jeho hodnota zmenená na inú alelu zo zoznamu prípustných hodnôt. Výber novej hodnoty génu je opäť náhodný proces.

Podobne ako kríženie je aj mutácia stochastický proces, avšak na rozdiel od kríženia prebieha mutácia s veľmi malou pravdepodobnosťou (udáva sa 0.1 – 5 % podľa [6]). Mutácia sa obvykle aplikuje na každý genóm populácie až po operácií kríženia a to tak, že algoritmus mutácie sa aplikuje na každý gén (s ohľadom na malú pravdepodobnosť prebehnutia mutácie, je percento zmenených génov pomerne malé). Podobne ako kríženie, aj mutácia môže mať na popupáciu negatívny vplyv — okrem vnášania nových vlastností je schopná znehodnotiť kvalitné genómy, čo je jedným z dôvodov, prečo sa do genetického algoritmu zavádza elitizmus (viď sekcia 3.2.2), ktorý ochráni najlepšie genómy pred potenciálnymi nežiadúcimi zmenami.

Obrázok 3.1b znázornuje mutáciu génu — šiesty gén s pôvodnou hodnotou "F" zmutoval na novú hodnotu "f".



(a) Kríženie chromozómov. **(b)** Mutácia génu.

Obrázek 3.1: Aplikácia genetických operátorov v jednoduchom GA.

3.3 Genetický algoritmus vo výpočtovom systéme celulárnych automatov

Táto sekcia sa zaoberá kľúčovými myšlienkami využitia genetického algoritmu vo výpočtovom modeli celulárneho automatu. Jednotlivé podsekcie sa zaoberajú developmentom v genetickom algoritme a popisujú princípy nasadenia genetického algoritmu v celulárnom automate.

3.3.1 Development v genetickom algoritme

Development (v preklade *vývin*) v evolučnom algoritme predstavuje transformáciu genetickej informácie, ktorá je zakódovaná v chromozóme na konkrétne správanie sa jedinca — fenotypu.

Myšlienka výpočtového vývinu [8], ako sa development v evolučných algoritmoch nazýva, má svoj pôvod v prírode — jedinec samotný, ale aj jeho vlastnosti, schopnosti a správanie je zakódované v jeho genetickej výbave nepriamo. Genetická informácia z chromozómu slúži na vytvorenie mediátorov, ktoré riadia jednotlivé procesy v živote jedinca.

V živých organizmoch je nositeľom vývoja DNA (deoxyribonukleova kyselina), ktorá vo svojej štruktúre obsahuje gény, na základe ktorých sú syntetizované proteíny v procesoch transkripcie a translácie. Proteíny, ktoré predstavujú vyššie zmienené mediátory, následne ovplyvňujú správanie sa buniek, vo svojej podstate riadia biochemické procesy, ktoré prebiehajú v bunkách [8].

3.3.2 Celulárny automat ako model developmentu

Celulárny automat, ktorého charakteristika je popísaná v kapitole 2, môžeme považovať za jednoduchý model výpočtového vývinu v evolučnom návrhu a to aj napriek tomu, že tabuľku pravidiel sme schopní zakódovať do genómu priamo (čím odpadá nutnosť mediátorov a pokročilého mapovania genotypu na fenotyp). Vývin totiž môžeme sledovať v správaní sa celulárneho automatu, teda vo vývoji stavov buniek v celulárnom priestore.

Cieľom návrhu CA ja nájdenie určitých pravidiel, ktoré zaistia konkrétne chovanie jednotlivých buniek CA v čase

Ako príklad môže uviesť implementáciu celulárneho automatu známu pod názvom "Hra Života"(v originále *The Game of Life*) [4]. V tomto príklade by chromozóm genetického algoritmu kódoval tabuľku pravidiel CA tak, aby objekty vytvorené zo "živých"buniek (napríklad objekt *glider*) vykazovali počas vývoja automatu v čase istý špecifický druh pohybu. Genetickým algoritmom v tomto prípade nehľadáme chromózom, ktorý priamo kóduje riešenie (pohyb objektu), ale ktorý definuje také vlastnosti prostredia (lokálnu prechodovú funkciu celulárneho automatu), aby daný objekt vykazoval v prostredí celulárneho automatu pohyb.

Kapitola 4

Celulárne programovanie

Nasledujúca kapitola úvadza základné princípy jedného z algoritmov evolvovania celulárnych automotov — tzv. *celulárneho programovania* [12] (ďalej v texte CP). CP je druhou spomedzi štandardných techník návrhu celulárnych automatov, ktorá je v rámci textu tejto diplomovej práci skúmaná.

4.1 Algoritmus celulárneho programovania

Celulárne programovanie predstavuje návrhovú techniku obecne neuniformných celulárnych automatov, teda automatov, kde každá bunka má svoju vlastnú prechodovú funkciu.

Pseudokód základnej podoby celulárneho programovania je zobrazný v algoritme 4.1. Nasledujúce odstavce popisujú princíp fungovania tohto algoritmu a stručne upozorňujú na najzásadnejšie rozdiely oproti prístupu evolvovania celulárnych automatov pomocou genetických algoritmov.

Algoritmus celulárneho programovania je postavený na evolučných technikách. Avšak na rozdiel od štandardnej implementácie, kde evolučný algoritmus pracuje s evolvujúcou sa populáciou kandidátnych riešení — teda s populáciou celulárnych automatov — pracuje evolučný algoritmus v celulárnom programovaní len s jedinou inštanciou CA.

Keďže sa jedná o neuniformný automat, má každá bunka svoju vlastnú tabuľku pravidiel, podľa ktorej je riadená v procese vývoja celulárneho automatu. V počiatočnej fáze celulárneho programovania sú tabuľky funkcií pre všetky bunky CA vygenerované náhodne.

Po inicializačnej fáze nastáva evolučný proces, ktorý sa vo všeobecnosti skladá z dvoch krokov (fázi) a podobne ako v evolučnom algoritme (viď algoritmus 3.1) sú tieto fázy zabalené do cyklusu, ktorý predstavuje kroky evolučného algoritmu — generácie. Cyklus prebieha dovtedy, kým nie sú splnené podmienky ukončenia algoritmu.

Evolučný proces algoritmu celulárneho programovania je teda tvorený dvomi fázami, ktoré nie sú rovnocenné čo sa počtu prechodov týka. Prvá fáza — výpočtová — prebehne v C-krát vyššom počte behov ako fáza druhá — evolučná. Táto nerovnomernosť je daná tým, že evolučná fáza prebehne v každej generácií evolúcie len jedenkrát, zatiaľčo výpočtová fáza je v každej generácií spocítaná C-krát, kde C je vstupný parameter algoritmu, ktorý definuje počet náhodne vygenerovaných konfigurácií celulárneho automatu, nad ktorými sa bude určovať kvalita LPF každej bunky CA — fitness hodnota.

Po inicializačnej fáze sa algoritmus CP teda dostáva do fázy "evolučného cyklu", ktorý sme v predchádzajúcom odstavci rozdelili na dva väčšie bloky — výpočtový a evolučný. Výpočtová časť je v hlavnom cykluse CP pred samotným evolučným blokom (viď algoritmus

```
for každá bunka i v CA do in parallel
    inicializuj tabuľku pravidiel bunky i;
    f_i = 0 (vynuluj fitness hodnotu bunky);
end
c = 0 (inicializuj počítadlo konfigurácií);
while nie sú splnené podmienky ukončenia do
    vygeneruj náhodnú inicializačnú konfiguráciu CA;
    vyvíjaj CA na inicializačnej konfigurácií počas M časových krokov;
    for každá bunka i do in parallel
        if bunka i je v korektnom finálnom stave then
            f_i = f_i + 1;
        end
    end
    c = c + 1;
    if c mod C = 0 then
        for každá bunka i v CA do in parallel
            vypočítaj nf_i (počet susedov s vyššou fitness);
            if nf_i(c) = 1 then
                nahraď tabuľku pravidiel i lepšou susednou tabuľkou;
                preveď mutáciu novej tabuľky;
            end
            else if nf_i(c) = 2 then
                nahraď tabuľku pravidiel i výsledom kríženia lepších susedných
                tabuliek;
                preveď mutáciu novej tabuľky;
            end
            else if nf_i(c) > 2 then
                náhodne vyber dve z lepších susedných tabuliek;
                nahraď tabuľku pravidiel i výsledom kríženia lepších susedných
                tabuliek;
                preveď mutáciu novej tabuľky;
            end
            f_i = 0;
        end
    end
end
```

Algoritmus 4.1: Pseudokód algoritmu celulárneho programovania [12].

4.1) a jej kľúčové časti popisujú nasledujúce odstavce.

Vo výpočtovom bloku sa najskôr vygeneruje náhodná konfigurácia stavov všetkých buniek celulárneho automatu. Ako je uvedené vyššie, celulárne programovanie pracuje len s jedinou inštanciou celulárneho automatu, u ktorého sa pomocou evolučného algoritmu vyvíjajú prechodové funkcie jednotlivých buniek CA (po ukončení algoritmu sú očakávaným výstupom prechodové funkcie buniek realizujúce požadované chovanie CA).

Po vygenerovaní náhodnej konfigurácie CA sa zaistí beh celulárneho automatu s aktuálnymi prechodovými funkciami jednotlivých buniek. Vývoj automatu prebieha vo vopred

určenom počte krokov.

Keď vývoj celulárneho automatu skončí, prechádza algoritmus do tretieho výpočtového procesu, v ktorom sa vypočíta fitness hodnota, ktorá určuje kvalitu evolvovaných tabuliek pravidiel ako počet buniek, ktoré sa po záverečnom kroku CA nachádzajú v korektnom stave. Zaujímavosťou výpočtu fitness je, že je určená pre každú bunku automatu zvlášť, teda každá bunka má v celulárnom programovaní nielen vlastnú tabuľku pravidiel, ale aj vlastnú fitness hodnotu.

Je dôležité poznamenať, že v prvom výpočtovom procese algoritmu sú generované len počiatočné konfigurácie buniek CA, zatiaľčo tabuľky pravidiel, ktoré sú asociované s jednotlivými bunkami už nie sú pregenerované. Jednotlivé bunky teda využívajú stále tie isté tabuľky pravidiel, ktoré sú upravované len evolučným procesom v hlavnej fáze algoritmu celulárneho programovania.

V každom opakovaní hlavného cyklusu algoritmu CP teda prebehne výpočtová fáza, ktorá je tvorená tromi vyššie popísanými procesmi práve C-krát, kde C je určený počet počiatočných konfigurácií, na ktorých prebieha vyhodnocovanie chovania CA. Fitness hodnota, vypočítaná pre danú konfiguráciu celulárneho automatu, je pre každú bunku pričítaná k aktuálnej fitness hodnote príslušnej bunky. Tým, že fitness hodnoty sa pre jednotlivé konfigurácie CA sčítajú, je dosiahnuté to, že po prebehnutí vývoja celulárneho automatu pre všetky konfigurácie, bude aktuálna fitness hodnota každej bunky obsahovať informáciu o počte konfigurácií, ktoré dosiahli v záverečnom kroku vývoja CA korektný stav.

Maximálnu fitness hodnotu dosiahne teda tá bunka, ktorá bude mať pre každú náhodne vygenerovanú počiatočnú konfiguráciu CA korektnú hodnotu (stav) po dokončení vývoja celulárneho automatu. Zmyslom opakovaných behov celulárneho automatu nad rôznymi počiatočnými konfiguráciami, ale s nezmenenými tabuľkami pravidiel, je teda overiť fakt, ako dobre si tieto tabuľky budú počínať na rôznych konfiguráciach celulárneho automatu.

Druhá časť v bloku hlavného cyklusu algoritmu CP predstavuje evolučnú fázu algoritmu. Evolučný blok prebehne len raz za C opakovaní hlavného cyklu, teda až v čase, keď prebehne výpočtový blok nad všetkými konfiguráciami celulárneho automatu. V tomto evolučnom bloku sú využívané evolučné princípy na tabuľky pravidiel v jednotlivých bunkách celulárneho automatu, čím sa snažíme dosiahnúť zvýšenie priemernej medzigeneračnej fitness hodnoty. Časť evolučného procesu, výpočet hodnoty fitness, už bola zaistená vo výpočtovej fáze algoritmu, a teda vo fáze evolučnej už priebehajú len tri procesy: selekcia, aplikácia genetických operátorov a modifikácia tabuliek pravidiel v jednotlivých bunkách automatu. Aplikácia týchto procesov prebieha na všetkých bunkách celulárneho automatu a je lokálna — to znamená, že je ovplyvňovaná výhradne susednými bunkami z celulárneho okolia aktuálnej bunky.

V procese selekcie sú preverené fitness hodnoty všetkých buniek z okolia aktuálne vyšetrovanej bunky. Pokiaľ sa v celulárnom okolí nachádzajú viac ako dve bunky (respektíve dve tabuľky pravidiel) s lepšou fitness než má vyšetrovaná bunka, prebehne proces selekci, ktorého princím je popísaný v sekcií 3.2.2 a do procesu genetických operátorov sa "odošlú"dva vybrané genotypy (tabuľky pravidiel celulárneho automatu zakódované do podoby vhodnej pre aplikáciu genetických operátorov). Pokiaľ sa nájde menší počet lepších tabuliek (to znamená dve alebo jedna) sú tieto fenotypy vybrané automaticky.

Druhým evolučným procesom je aplikácia genetických operátorov kríženia a mutácie ako sú popísané v sekciách 3.2.3 a 3.2.4. Kríženie samozrejme prebehne len za predpokladu, že počet vybraných fenotypov v predchádzajúcom procese je rovný dvom. Keďže kríženie generuje dvoch potomkov, je z týchto náhodne vybraný len jeden. Mutácia je aplikovaná

len v prípade, že počet vybraných fenotypov je väčší [1] alebo rovný jednej. Ak sa ale nepodarí v celulárnom okolí nájsť lepšie bunky, postupuje do ďalšej generácie tabuľka pravidiel aktuálne vyšetrovanej bunky bez zmeny. Týmto je zaistený elitizmus.

Proces modifikácie tabuliek zaistí prepísanie aktuálnej tabuľky pravidiel novou, ktorá bola vyevolvovaná v procesoch kríženia a mutácie. Pokiaľ ale tieto procesy neprebehli (to znamená, že v okolí bunky nie sú žiadny lepší susedia), tak sa tabuľka pravidiel nemení. V tomto kroku je zároveň vynulovaná hodnota fitness vo všetkých bunkách, čím je zaistené, že každá generácia začína s nulovou hodnotou fitness.

Tieto dve fázy — výpočtová a evolučná — a celkovo šesť podprocesov, ktoré v nich prebiehajú, definuje jednu generáciu evolučného procesu. Evolučný proces končí nájdením vhodne kvalitného riešenia alebo vyčerpaním zdrojov, ako popisujú odstavce vyššie. Výstupom algoritmu celulárneho programovania sú tabuľky pravidiel všetkých buniek automatu, ktoré zaisťujú vyhovujúce riešenie daného problému, to znamená že tieto výstupné tabuľky predstavujú najlepšiu dosiahnutú fitness hodnotu celulárneho automatu.

[1] V tomto prípade sa o výber jediného fenotypu pre mutáciu postarajú selekcia a kríženie.

Kapitola 5

Vybrané porovnávacie úlohy

Cieľom tejto diplomovej práce je vytvoriť novú techniku či nový prístup k návrhom celulárnych automatov. Tento vlastný prístup, ktorý je prínosom tejto diplomovaj práce, je samozrejme potrebné porovnať s aktuálnymi metódami, ktoré sa v procesoch návrhu celulárnych automatov v súčastnosti bežne používajú.

Táto kapitola uvádza stručné popisy problémov riešiteľných pomocou celulárnych automatov, na ktorých bude porovnanie prevedené. Vybrané boli také úlohy, ktoré nie sú v prostredí celulárneho automatu jednoducho riešiteľné, a to ani pomocou evolučného návrhu, čím sa budeme snažiť zvýrazniť rozdielne schopnosti jednotlivých algoritmov.

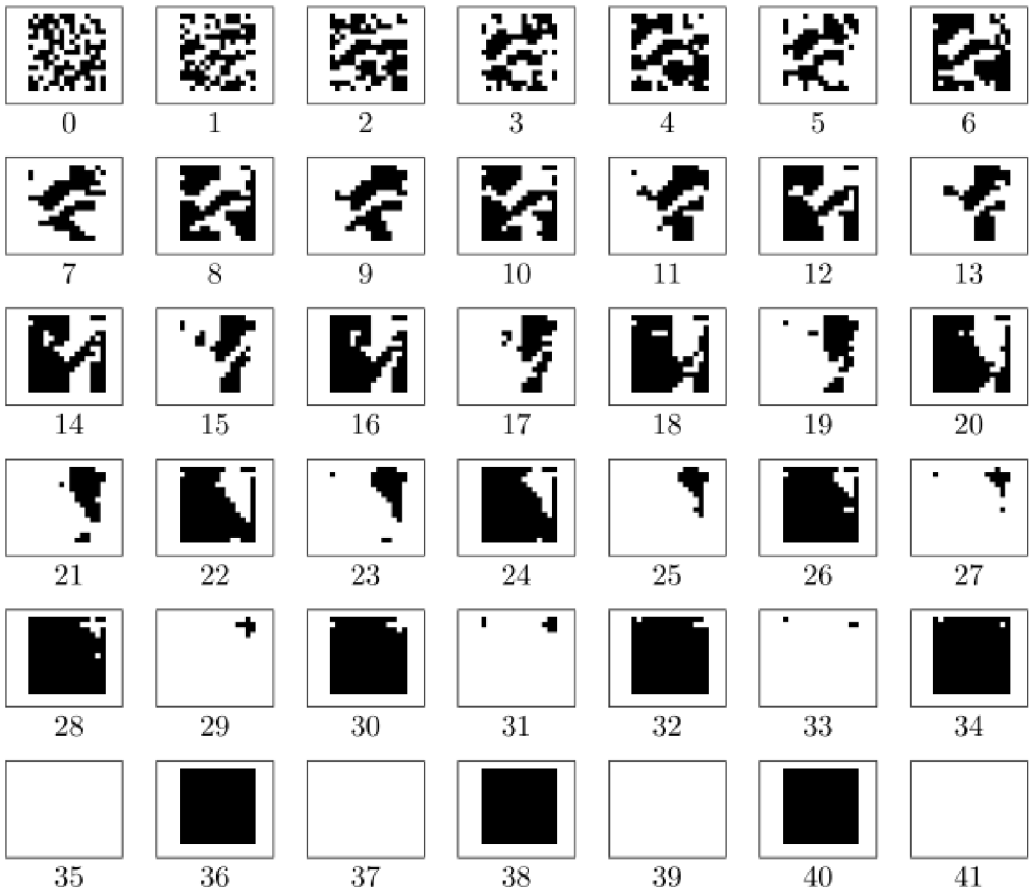
5.1 Problém synchronizácie

Úlohu synchronizácie, riešenú pomocou celulárneho automatu, budeme prezentovať na dvojrozmernom dvojstavovom automate (možné stavy buniek budú 0 a 1), pracujúcom s celulárnym 5-okolím a cyklickými okrajovými podmienkami.

Synchronizácia [12] bude pre potreby tohto textu chápaná nasledovne: ak hustota [1] buniek v prvom kroku nasledujúcom po úvodných M časových krokoch výpočtu CA bude väčšia ako 0.5 (respektíve menšia ako 0.5), tak po uplynutí nasledujúcich štyroch krokoch vývoja automatu (to znamená po uplynutí krokov $M+1..M+4$) bude celulárny automat vykazovať také spravanie, že všetky bunky budú v krokoch nasledujúcich ($M+5$, $M+6$, $M+7$, ...) nadobúdať sekvenciu stavov $0 \to 1 \to 0 \to 1$ (respektíve $1 \to 0 \to 1 \to 0$). Vo všetkých krokoch od $M+5$ by teda mal automat "prepínať"stavy všetkých svojích buniek medzi hodnotami 0 a 1.

Sipper vo svojej práci [12] preveril úlohu synchronizácie na dvojstavovom jednorozmernom automate, ktorý bol uniformný. S ohľadom na skutočnosť, že uniformný automat s veľkosťou celulárneho okolia $r = 1$ má stavový priestor tvorený len $2^8 = 256$ pravidlami, preveril Sipper všetky tieto pravidlá na množine náhodne generovaných inicializačných konfigurácií so záverom, že najlepšie pravidlo dosiahlo normalizovanú hodnotu fitness 0.84. Pokiaľ je ale úloha riešená pomocou neuniformného automatu, malo by byť podľa Sippera možné dosiahnúť "perfektné"riešenie, teda riešenie s normalizovanou fitness hodnotou rovnou 1.00. Je samozrejme potrebné vziať do úvahy fakt, že Sipper prevádzal svoje experimenty na automate o veľkosti 149 buniek, čo znamená že počet všetkých unikátnych

[1] Problém hustoty budeme v prípade tohto automatu chápať ako normalizovaný počet buniek celulárneho automatu nachádzajúcich sa v stave 1. Jedná sa teda o hodnotu súčtu všetkých buniek automatu v požadovanom stave delený celkovým počtom buniek celulárneho automatu.



Obrázek 5.1: Očakávaný výsledok úlohy synchronizácie v dvojrozmernom automate. Obrázky ukazujú, že po istom počte časových krokov vývoja celulárneho automatu sa podarilo dosiahnúť stav, v ktorom sa všetky bunky prepínajú medzi stavmi 0 a 1 v každom ďalšom kroku. *Prebraté z [12].*

konfigurácií celulárneho automatu je 2^{149}. Z tohto dôvodu neboli s riešením, ktoré Sipper objavil, preverené všetky konfigurácie CA.

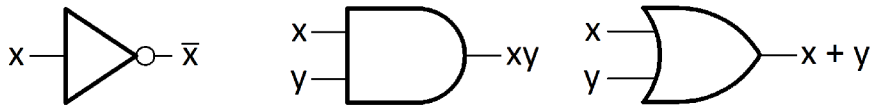
Problém synchronizácie, ako ho popisuje táto sekcia a zobrazuje obrázok 5.1, bol vybraný ako porovnávací model z dôvodu, že v prípade jeho riešenia v celulárnom automate sa jedná o netriviálny problém [12].

Cieľom experimentov spadajúcich do úlohy synchronizácie bude preverenie hrubej sily jednotlivých evolučne zameraných algoritmov. Algoritmy v tomto prípade musia objaviť "globálne–optimálne"prechodové funkcie jednotlivých buniek, ktoré budú schopné ľubovoľnej počiatočnej konfigurácií CA vnútiť (po istom počte krokov vývoja CA) prepínanie stavov 0 a 1 v celej mriežke tak, ako to zobrazuje obrázok 5.1.

5.2 Systém logických hradiel

Úloha logických hradiel bude venovaná problematike univerzálnych výpočtov v prostredí dvojrozmerného binárneho celulárneho automatu, ktorý pracuje s celulárnym 5-okolím a cyklickými okrajovými podmienkami. Primárnym cieľom budú binárne výpočtové štruktúry — logické hradlá — takže automat bude rozlišovať dva rôzne stavy buniek, pre každý logický stav bude vyhradený práve jeden stav CA.

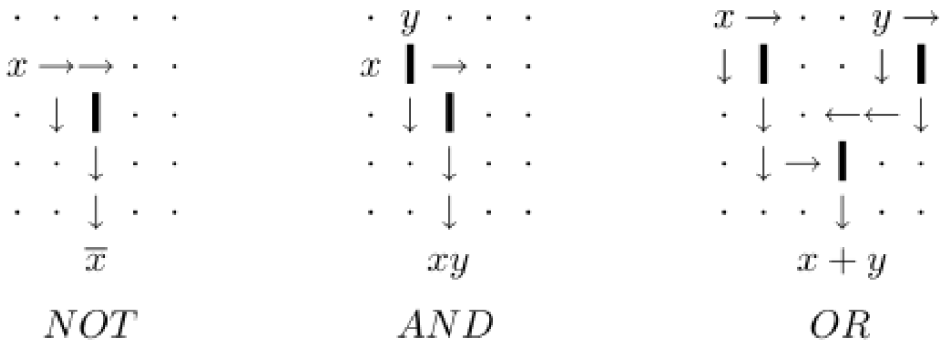
Experimenty tejto sekcie sa konkrétne zamerajú na prácu so základnými logickými hradlami — *AND*, *OR* a *NOT*. Hradlá *AND* a *OR* budú vystavané ako binárne dvojvstupové hradlá s jedným výstupom, *NOT* bude binárne hradlo s jedným vstupom a jedným výstupom. Bežne používané schémy týchto štruktúr sa znázornené na obrázku 5.2.



(a) Hradlo NOT. **(b)** Hradlo AND. **(c)** Hradlo OR.

Obrázek 5.2: Schematické znázornenie jednotlivých hradiel.

S ohľadom na zameranie tejto diplomovej práce budú jednotlivé hradlá reprezentované pomocou vhodne navrhnutého celulárneho automatu. Takýto CA bude obmedzený na určitú plochou (bude teda definovaný počtom buniek mriežky, ktoré bude mať celulárny automat k dispozícií) a v rámci mriežky bude mať pevne určené súradnice význačných buniek automatu [2]. Posledná — a najdôležitejšia — časť definujúca celulárny automat ako logické hradlo, je vhodne zvolená prechodová funkcia automatu, respektíve vhodné prechodové funkcie kľúčových buniek neuniformného celulárneho automatu.



Obrázek 5.3: Implementácia logických hradiel *AND*, *OR* a *NOT* v prostredí dvojrozmerného celulárneho automatu pomocou propagačných pravidiel a pravidla pre výpočet funkcie *NAND*. *Prebraté z [12].*

Cieľom sady experimentov, spadajúcich pod túto kapitolu, bude znovu-objaviť konfiguráciu prechodových funkcií jednotlivých buniek neuniformného celulárneho automatu tak, aby boli tieto evolúciou získané automaty schopné fungovať rovnako, ako automaty, ktoré

[2] Jedná sa o bunky, ktoré do automatu "privádzajú" vstupné hodnoty hradla (viď vstupy A a B na obrázku 5.2) a "odvádzajú" výstupnú hodnotu (viď výstup Y na obrázku 5.2).

vo svojej práci [12] vytvoril Sipper. Grafickú reprezentáciu prechodových funkcií v mriežke automatov, ktoré použil Sipper zobrazuje obrázok 5.3.

Hlavným dôvodom, prečo budú tieto experimenty zamerané výhradne na znovu-objavenie týchto konfigurácií CA, je fakt, že u takto skonštruovaných logických hradiel Sipper ukázal [12], že celulárne automaty tejto konfigurácie, naozaj simulujú činnosť daných logických hradiel.

Sipper vo svojich pokusoch definoval sadu základných primitívnych prechodových funkcií — štyri funkcie definujúce propagáciu stavu bunky (propagácia stavu vpravo, vľavo, hore a dole), jednu funkciu pre logickú operáciu *NAND* a funkciu zachovávajúcu aktuálny stav bunky. Pomocou týchto jednoduchých funkcií bol Sipper schopný skonštruovať na obmedzenej ploche celulárnej mriežky logické hradlá, ktoré sú znázornené na obrázku 5.3. Obrázok 5.4 popisuje všetky typy prechodových funkcií, ktoré vo svojich experimentoch použil Sipper [3].

Designation	Rule	$\mathcal{S}$	Designation	Rule	$\mathcal{S}$
Right propagation cell	$\begin{matrix} & * & \\ x & * & * \\ & * & \end{matrix} \mapsto x$	$\rightarrow$	Exclusive Or (XOR) cell (type a)	$\begin{matrix} & y & \\ x & * & * \\ & * & \end{matrix}$	$\oplus$
Left propagation cell	$\begin{matrix} & * & \\ * & * & x \\ & * & \end{matrix} \mapsto x$	$\leftarrow$	Exclusive Or (XOR) cell (type b)	$\begin{matrix} & x & \\ * & * & y \\ & * & \end{matrix} \mapsto x \oplus y$	$\oplus$
Up propagation cell	$\begin{matrix} & * & \\ * & * & * \\ & x & \end{matrix} \mapsto x$	$\uparrow$	Exclusive Or (XOR) cell (type c)	$\begin{matrix} & * & \\ * & * & x \\ & y & \end{matrix} \mapsto x \oplus y$	$\oplus$
Down propagation cell	$\begin{matrix} & x & \\ * & * & * \\ & * & \end{matrix} \mapsto x$	$\downarrow$	Exclusive Or (XOR) cell (type d)	$\begin{matrix} & * & \\ y & * & * \\ & x & \end{matrix} \mapsto x \oplus y$	$\oplus$
$NAND$ Cell	$\begin{matrix} & y & \\ x & * & * \\ & * & \end{matrix} \mapsto x \mid y$	$\mid$	No Change (NC) cell	$\begin{matrix} & * & \\ * & x & * \\ & * & \end{matrix} \mapsto x$	$\cdot$

Obrázek 5.4: Tabuľka zachytáva grafické reprezentácie všetkých typov prechodových funkcií celulárneho automatu, ktoré vo svojej práci použil Sipper. Jednotlivé stĺpce predstavujú popis pravidla, tvar pravidla v prostredí CA pri využití celulárneho 5-okolia, a symbol, ktorý je využitý v grafickej reprezentácií hradiel na obrázku 5.3. *Prebraté z [12].*

[3] Okrem pravidiel popísaných v texte zobrazuje obrázok 5.4 aj štyri *XOR* pravidlá, ktoré v rámci tejto diplomovej práce neboli využité.

Z obrázku 5.3 je patrné, že výpočet v takto zadefinovaných hradlách v prostredí celulárneho automatu bude prebiehať v niekoľkých diskrétnych krokoch výpočtu CA, ktoré zaistia transformáciu vstupných parametrov na výslednú hodnotu. Jedná sa teda o špecifický výpočetný proces pomocou postupnej propagácie a transformácie hodnôt bunkami CA.

Táto sada experimentov bude teda zameraná na neuniformný celulárny automat a jej primárnym cieľom bude preveriť schopnosti jednotlivých evolučne založených algoritmov pri hľadaní vysoko špecifického chovania na pomerne malej ploche celulárnej mriežky. Aby bolo zaistené korektné fungovanie CA simulujúceho logické hradlá, bude zrejme nevyhnutné definovať (respektíve vyevolvovať) jednotlivé prechodové funkcie buniek natoľko presne, aby boli schopné zaistiť vzájomnú súhru kľúčových oblastí automatu v správnom okamžiku výpočtového procesu hradla.

Kapitola 6

Celulárna evolúcia

V prostredí celulárnych automatov je možné riešiť úlohy rôznej povahy — od skúmania umelého života (napríklad *Game of Life* [4]) po návrh kombinačných obvodov. S ohľadom na povahu úlohy, ktorú chceme v prostredí celulárneho automatu riešiť, môže byť implementácia úlohy netriviálna respektíve objavenie vhodného riešenia môže byť časovo veľmi náročné.

Práve z týchto dôvodov bolo cieľom tejto diplomovej práce vytvorenie novej techniky návrhu celulárnych automatov, so zameraním na využitie evolučných techník. S ohľadom na zadanie konkrétnej úlohy môže byť stavový priestor riešenia v prostredí celulárneho automatu rozsiahly a je teda vhodné využitie evolučných techník.

Nasledujúca kapitola je venovaná popisu techniky užitej k návrhu celulárnych automatov, ktorá bola v rámci tejto diplomovej práce implementovaná. Nová metóda získala názov *celulárna evolúcia* (ďalej v texte pod skratkou CE).

6.1 Primárne zameranie techniky celulárnej evolúcie

Technika *celulárnej evolúcie* bola primárne vytvorená na evolučný návrh celulárnych automatov a špecificky sa zameriava na neuniformné celulárne automaty. Principálne vychádza technika CE z algoritmu predstaveného Bidlom a Vašíčkom [1].

Neuniformné celulárne automaty sú z pohľadu úloh, ktoré v nich môžeme riešiť, zaujímavejšie ako ich uniformné alternatívy. Hlavným dôvodom tohto tvrdenia je fakt, že neuniformný celulárny automat disponuje výrazne väčšou vyjadrovacou silou než automat uniformný [1].

Neuniformný celulárny automat umožňuje vytvárať konštrukcie s heterogénnym správaním jednotlivých blokov — každá bunka celulárneho automatu môže obsahovať svoj vlastný predpis (tabuľku prechodovej funkcie), podľa ktorého sa daná bunka správa počas vývoja automatu. Vďaka tejto vlastnosti je možné konštruovať štruktúry s rôznym stupňom komplexnosti, čo bude nevyhnutné s ohľadom na zadanie porovnávacej úlohy logických hradiel, ktorá je popísaná v sekcií 5.2.

Evolučný návrh riešenia zadaného problému v neuniformnom automate sa javí ako veľmi vhodný, keďže stavový priestor v ktorom hľadáme riešenie môže byť obrovský — narastá

[1] V prostredí dvojrozmerného binárneho neuniformného celulárneho automatu môžeme vybudovať stroj s výpočetnou silou ekvivalentnou s univerzálnym Turingovým strojom [12]. Prvý stroj tohto typu bol vytvorený Von Neumannom [13, 12].

exponenciálne s počtom stavov automatu a počtom buniek celulárneho okolia [2]. S ohľadom na veľkosť tohto priestoru môžu byť aj "štandardne"používané evolučné metódy značne ťažkopádne. Práve preto sa metóda celulárnej evolúcie zameriava primárne na neuniformné celulárne automaty.

Technika celulárnej evolúcie nie je obmedzená dimenzionalitou celulárneho automatu ani počtom stavov automatu, je možné ju nasadiť ako na jednoduché binárne celulárne automaty, tak aj na viacstavové CA. Jej výhody by sa mali výraznejšie prejaviť práve pri komplikovaných automatoch s veľkým počtom stavov respektíve s veľkým počtom buniek celulárneho okolia.

6.2 Algoritmus metódy celulárnej evolúcie

Algoritmus celulárnej evolúcie — ako už z názvu samotného vyplýva — principiálne spadá do skupiny evolučných algoritmov. Operuje teda s bežne zaužívanými pojmami ako sú generácia, populácia, fitness hodnota, selekcia či evolučné (respektíve genetické) operátory mutácie a kríženia.

Základný princíp algoritmu celulárnej evolúcie vychádza priamo z algoritmu celulárneho programovania, ktorý bol podrobne popísaný v kapitole 4. Nasledujúca sekcia sa zameria na popis najdôležitejších miest v algoritme metódy CE. Jednotlivé body budú vysvetlené na algoritme pracujúcom s neuniformným CA.

Je dôležité uvedomiť si spôsob práce algoritmu CE s pojmom populácia. Keďže algoritmus je primárne zameraný na evolučný návrh neuniformných celulárnych automatov, tak podobne ako celulárne programovanie, aj celulárna evolúcia chápe pojem *populácia kandidátnych riešení* inak ako štandardný genetický algoritmus. CE pracuje s jedinou inštanciou celulárneho automatu. Populáciu následne budeme chápať ako počet náhodne vygenerovaných počiatočných konfigurácií tejto inštancie CA.

Najzásadnejším rozdielom medzi algoritmami metód CE a CP je úvodná fáza hlavného cyklu algoritmu, kde dochádza k mapovaniu genetypu (chromozómu evolučného algoritmu) na fenotyp (lokálnu prechodovú funkciu CA) a k výpočtu hodnoty fitness. Práve štruktúra chromozómu a spôsob jej spojenia s prostredím celulárneho automatu je najdôležitejšou časťou metódy CE, ktorá výrazne odlišuje algoritmus CE od CP. Štruktúru chromozómu aj spôsob jej mapovania popisuje sekcia 6.3.

V tomto mieste je možné nechať prebehnúť kompletný vývoj CA v požadovanom počte krokov a následne vypočítať fitness jednotlivých buniek. Druhou možnosťou, ktorú zobrazuje algoritmus 6.1 je počítať "čiastkové"fitness hodnoty buniek po každom kroku vývoja celulárneho automatu. Prvý prístup je vhodný, pokiaľ nás zaujíma len finálny stav automatu, druhý spôsob je s výhodou aplikovateľný pri riešení komplikovaných problémov, u ktorých je dôležitý aj "medzistav"automatu (teda stav buniek celulárnej mriežky v krokoch medzi iniciačným stavom CA a stavom finálnym).

Úloha synchronizácie popísaná v sekcií 5.1 je príkladom problému, u ktorého nás zaujíma len finálny stav mriežky CA, zatiaľčo problém implementácie logických hradiel definovaný

[2] Vzorce pre výpočet veľkosti stavového priestoru uniformného automatu nájdeme v kapitole 2.1. V prípade neuniformného automatu je veľkosť stavového priestoru daná ako súčin počtu buniek celulárneho priestoru so vzorcom (2.2) pre výpočet rozsahu tabuľky pravidiel CA. Každá bunka automatu totiž môže obsahovať jednu z možných prechodových funkcií, ktorých počet vyjadruje práve vzorec 2.2, pričom je nevyhnutné, aby každá bunka neuniformného automatu obsahovala vhodnú prechodovú funkciu, čím sa zaistí požadované správanie automatu ako celku. Pre každú bunku je teda potrebné vybrať z celého rozsahu prechodových funkcií tú najvhodnejšiu.

v sekcií 5.2 vyžaduje výpočet čiastkových fitness po každom kroku CA.

```
for každá bunka i v CA do in parallel
    inicializuj tabuľku pravidiel bunky i;
    f_i = 0 (vynuluj fitness hodnotu bunky);
end
while nie sú splnené podmienky ukončenia do
    c = 0 (inicializuj počítadlo konfigurácií CA);
    while c < c_MAX do
        vygeneruj náhodnú iniciačnú konfiguráciu CA;
        step = 0 (inicializuj počítadlo krokov vývoja CA);
        while step je menšie ako maximálny počet krokov CA do
            preveď jeden krok vývoja CA;
            for každá bunka i do in parallel
                if bunka i je v korektnom finálnom stave then
                    f_i = f_i + 1;
                end
            end
            step = step + 1;
        end
        c = c + 1;
    end
    for každá bunka i v CA do in parallel
        vypočítaj nf_i (počet susedov s vyššou fitness);
        if nf_i(c) = 1 then
            nahraď tabuľku pravidiel i lepšou susednou tabuľkou;
            preveď mutáciu novej tabuľky;
        end
        else if nf_i(c) = 2 then
            nahraď tabuľku pravidiel i výsledom kríženia lepších susedných tabuliek;
            preveď mutáciu novej tabuľky;
        end
        else if nf_i(c) > 2 then
            náhodne vyber dve z lepších susedných tabuliek;
            nahraď tabuľku pravidiel i výsledom kríženia lepších susedných tabuliek;
            preveď mutáciu novej tabuľky;
        end
        f_i = 0;
    end
end
```

Algoritmus 6.1: Pseudokód algoritmu celulárnej evolúcie. Principiálne vychádza z celulárneho programovania predstaveného Sipperom [12], od ktorého sa odlišuje úvodnou fázou hlavného cyklusu. Tu dochádza k výpočtu nového stavu buniek CA a následne k výpočtu hodnoty fitness. Keďže metóda CE používa výrazne odlišnú štruktúru chromozómu, je aj spôsob mapovania genotypu na fenotyp jedinečný.

6.3 Reprezentácia lokálnej prechodovej funkcie celulárnej evolúcie

Technika celulárnej evolúcie je typická unikátnou štruktúru chromozómu, ktorá je odlišná od chromozómov, ktoré sa "štandardne"používaných v návrhoch celulárnych automatov pomocou iných evolučných techník. Z unikátnej štruktúry genotypu vyplýva aj špecifický princíp mapovania tohto genotypu na fenotyp.

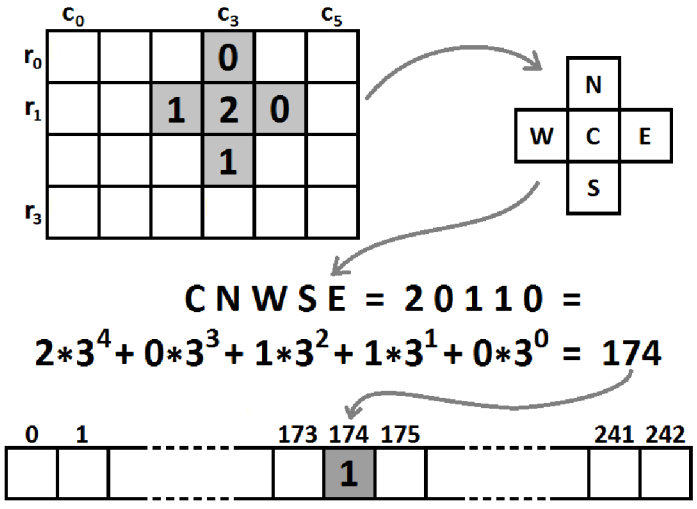
Táto sekcia je zameraná na popis rozdielu medzi bežne používanou štruktúrou chromozómu a chromozómom využitým v algoritme celulárnej evolúcie a vysvetluje princípy mapovania genotypov jednotlivých metód na príslušné fenotypy.

6.3.1 Lineárny chromozóm genetického algoritmu a celulárneho programovania

Táto podsekcia stručne popisuje lineárny chromozóm a princíp jeho spojenia s celulárnym automatom. Body tu uvedené slúžia najmä na lepšie pochopenie odlišností medzi metódou CE (ktorá bude popísaná v ďalšom texte) a inými evolučnými technikami (napríklad GA, CP), ktoré celkom bežne používajú v prostredí celulárneho automatu práve tento typ chromozómu.

Lineárny chromozóm je tvorený postupnosťou určitých číselných hodnôt. Dĺžka tejto postupnosti — teda počet génov chromozómu — je daná počtom stavov a počtom buniek v celulárnom okolí (viď vzorec 2.1). V prípade uniformného automatu je genóm tvorený jediným takýmto chromozómom, zatiaľčo genóm neuniformného automatu tvorí N-násobný počet týchto chromozómov, kde N je počet buniek automatu (každá bunka má svoj vlastný chromozóm).

Pre každú možnú kombináciu stavov buniek z celulárneho okolia, kóduje chromozóm tejto štruktúry nový stav (alelu) bunky CA, pričom index génu (lokus) je daný jednoznačným výpočtom založeným na hodnotách buniek v celulárnom okolí.



Obrázek 6.1: Princíp mapovania informácie medzi stavom buniek celulárneho okolia (fenotyp) a indexom génu jednoduchého lineárneho chromozómu (genotyp). Obrázok približuje výpočet nového stavu bunky s polohou $[r_1, c_3]$ v 3-stavovom automate (stav je reprezentovaný jednou z hodnôt 0, 1, 2), ktorého lokálna prechodová funkcia pracuje s celulárnym 5-okolím.

Princíp tohto jednoznačného výpočtu uvádza obrázok 6.1, ktorý zachytáva mapovanie lineárneho chromozómu na nový stav bunky celulárneho automatu. Využíva sa pritom fakt, že pre každú kombináciu stavov buniek celulárneho okolia existuje v lineárnom chromozóme práve jeden gén, ktorý priamo obsahuje výstupnú hodnotu — nasledujúci stav (alelu) vyšetrovanej bunky CA. Tento výpočet zaisťuje jednoznačné priradenie lokusu génu danému celulárnemu okoliu a zároveň garantuje, že žiadny gén nie je zdieľaný medzi dve navzájom rôzne kombinácie stavov buniek celulárnych okolí.

6.3.2 Chromozóm algoritmu celulárnej evolúcie

Základnou jednotkou genetickej informácie techniky celulárnej evolúcie je štrukturovaný gén [1], ktorého vlastnosti a princípy jeho implementácie a použitia budú vysvetlené v nasledujúcej sekcií na celulárnom automate implementujúcom 5-okolie (viď obrázok 2.5a).

Štrukturovaný gén sa skladá z dvoch častí — prvá časť je tzv. aktivačná jednotka, zatiaľčo časť druhá je výstupná hodnota (alela) génu. Aktivačná jednotka génu je tvorená postupnosťou niekoľkých číselných hodnôt, ktorých počet je rovný počtu buniek celulárneho okolia — hodnoty tejto postupnosti nazveme komponentami aktivačnej jednotky génu (to znamená, že v prípade celulárneho 5-okolia je aktivačná jednotka génu tvorená postupnosťou práve piatich komponént). "Výstupná hodnota"génu (nový stav bunky CA) je tvorená jedinou hodnotou a rovnako ako postupnosť komponént aktivačnej jednotky [3].

Jednotlivé komponenty budú indexované reťazcami CoC, CoN, CoW, CoS, CoE práve v tomto poradí. Výstupnú hodnotu génu budeme indexovať reťazcom O. Grafické znázornenie takejto štruktúry zachytáva obrázok 6.2.

CoC	CoN	CoW	CoS	CoE	O
#	#	#	#	#	#

Obrázek 6.2: Štrukturovaný gén celulárnej evolúcie, ktorého aktivačná jednotka pracuje s celulárnym 5-okolím. Prvých 5 hodnôt tohto chromozómu — CoC, CoN, CoW, CoS a CoE — predstavuje jednotlivé komponenty (podmienky) aktivačnej jednotky, zatiaľčo posledná hodnota (O) reprezentuje výsledný nový stav CA. Znak "#"reprezentuje číslo z oboru celých čísel.

Indexovanie buniek podľa definovaných reťazcov bolo zvolené zámerne s ohľadom na celulárne okolie bunky. Vychádzajúc z 5-okolia ako je definované na obrázku 2.5a, nadobudnú jednotlivé indexy nasledujúci význam: CoC = *Condition C*, CoN = *Condition N*, CoW = *Condition W*, CoS = *Condition S*, CoE = *Condition E*. Každá komponenta aktivačnej jednotky génu je teda viazaná na definovanú bunku celulárneho okolia. Nový stav bunky CA, ktorý predstavuje výstupnú hodnotu génu je indexovaný ako O = *Output*.

Komponenta aktivačnej jednotky génu obsahuje konkrétnu celočíselnú hodnotu, ktorej úlohou je mapovanie aktivačnej podmienky z množiny podmienkových výrazov. Vo svojej základnej verzií, ktorá bola použitá v rámci tejto práce, obsahuje množina podmienkových výrazov jednoduché podmienky typu *rovnosť*, *nerovnosť*, *väčšie ako*, *menšie ako* a *don't care*. Spôsob, akým komponenty aktivačnej jednotky génu mapujú podmienky z množiny podmienkových výrazov, je vysvetlený v sekcií 6.3.3.

[3] Postupnosť komponént, ako aj výstupná hodnota génu (nový stav bunky CA) nadobúdajú hodnoty z oboru celých čísel.

Rozsah hodnôt, ktoré môže komponenta aktivačnej jednotky nadobúdať závisí na mohutnosti množiny podmienkových výrazov (teda na počte dostupných podmienok) a na počte stavov celulárneho automatu. Celkový počet hodnôt vyjadruje nasledujúci vzťah:

$$comp_{card} = states * conditions \tag{6.1}$$

kde $comp_{card}$ je kardinalita (rozsah hodnôt) komponenty aktivačnej jednotky, *states* je počet stavov CA a *condotions* je počet podmienok (mohutnosť množiny podmienkových výrazov). V prípade implementovania piatich podmienok popísaných v predchádzajúcom odstavci a pri troch stavoch celulárneho automatu dostaneme rozsah použiteľných hodnôt v komponente aktivačnej jednotky génu $0-14$, celkovo teda 15 rôznych celočíselných hodnôt.

Výstupná hodnota génu — teda nový stav bunky v celulárnom automate — môže nadobúdať rovnaký rozsah hodnôt ako komponenty aktivačnej jednotky (alela má teda väčšiu kardinalitu hodnôt ako je počet stavov CA). Tento prístup k alele bol zvolený z dôvodu, aby bolo možné so všetkými časťami génu a chromozómu pracovať jednotne (to znamená aby nebolo potrebné riešiť špeciálne postupy pri aplikácií genetických operátorov mutácie a kríženia).

CoC	CoN	CoW	CoS	CoE	O
6	13	1	9	8	3

Obrázek 6.3: Obrázok zobrazuje príklad štrukturovaného génu s konkrétnymi hodnotami. Predpokladá sa 3-stavový CA s 5-okolím a použitie množiny podmienok o mohutnosti piatich podmienok.

Aby vyšetrovaná bunka získala výstupnú hodnotu určitého génu, musia byť splnené všetky podmienky aktivačnej jednotky. Aktivačná jednotka má teda za úlohu zaistiť, aby bol daný štrukturovaný gén aktívny len v určitom prípade, a to práve a jedine vtedy, keď sú splnené podmienky všetkých komponént aktivačnej jednotky génu.

	CoC	CoN	CoW	CoS	CoE	O
0	#	#	#	#	#	#
1	#	#	#	#	#	#
n-1	#	#	#	#	#	#

Obrázek 6.4: Chromozóm celulárnej evolúcie vystavaný z n štrukturovaných génov, ktoré sú usporiadané do vertikálnych vrstiev. Chromozóm je samozrejme možné vybudovať ako lineárnu postupnosť štrukturovaných génov. Čísla *0 až n-1* na ľavej strane génov značia index génu v rámci chromozómu a znak "#"reprezentuje číslo z oboru celých čísel.

Štruktúra zachytená na obrázku 6.2, respektíve na obrázku 6.3, predstavuje jediný gén

chromozómu. Aktivita (propagácia výstupnej hodnoty génu do bunky celulárneho automatu) génu však závisí na vyhodnotení podmienok aktivačnej jednotky — pokiaľ sa nepodarí vyhodnotiť všetky komponenty ako pravdivé, gén je neaktívny. Z tohto obmedzenia teda vyplýva, že jediný gén nemôže pokryť všetky možné kombinácie stavov buniek celulárneho okolia (s výnimkou prípadu, keď budú všetky komponenty mapovať podmienku typu *don't care*).

Keďže jeden štrukturovaný gén nebude vo väčšine prípadov schopný pokryť všetky kombinácie stavov celulárneho okolia, je chromozóm vybudovaný z postupnosti viacerých génov, ako zobrazuje obrázok 6.4. Postupným aplikovaním evolučných princípov na chromozóm v mnohých generáciach evolúcie je možné získať sadu štrukturovaných génov, ktoré budú schopné pokryť väčšinu — ak nie všetky — kombinácie stavov buniek z celulárneho okolia.

Celková veľkosť chromozómu je teda daná vzťahom:

$$chrom_{size} = (cells_{count} + 1) * genes \tag{6.2}$$

kde $cells_{count}$ predstavuje počet buniek celulárneho okolia, ku ktorým je pričítaná konštanta 1 reprezentujúca výstupnú hodnotu génu a $genes$ je počet génov, z ktorých sa skladá chromozóm.

Chromozóm metódy celulárnej evolúcie je teda vybudovaný z postupnosti štrukturovaných génov. Štrukturovaný gén sa skladá z aktivačnej jednotky a výstupnej hodnoty, ktorá reprezentuje nový stav bunky CA.

Aby bola umožnená propagácia alelu z génu do celulárneho automatu, musí byť daný gén aktívny. O aktivite génu rozhoduje práve jeho aktivačná jednotka, ktorá je implementovaná ako postupnosť číselných hodnôt, ktoré sme nazvali komponentami aktivačnej jednotky. Každá komponenta mapuje práve jednu podmienku z mnoziny podmienkových výrazov.

Gén bude teda aktívny práve a jedine vtedy, ak budú splnené všetky podmienky aktivačnej jednotky génu (teda ak sú všetky komponenty vyhodnoté ako pravdivé). V tomto prípade sa povolí propagácia alely zo štrukturovaného génu.

6.3.3 Princíp mapovania komponenty aktivačnej jednotky na podmienku z množiny podmienkových výrazov

Predchádzajúca sekcia pojednávala o štruktúre chromozómu metódy celulárnej evolúcie. Dozvedeli sme sa, že štrukturovaný gén je vystavaný z aktivačnej jednotky a alely, kde aktivačná jednotka je tvorená postupnosťou komponént, ktoré do aktivačnej jednotky mapujú podmienky rozhodujúce o aktivite daného génu. Táto sekcia popisuje základné typy aktivačných podmienok a najmä spôsob ich namapovania do aktivačnej jednotky génu.

Princíp implementácie podmienok, ako aj ich mapovanie do génu, vychádza z techniky predstavenej Bidlom a Vašíčkom [1].

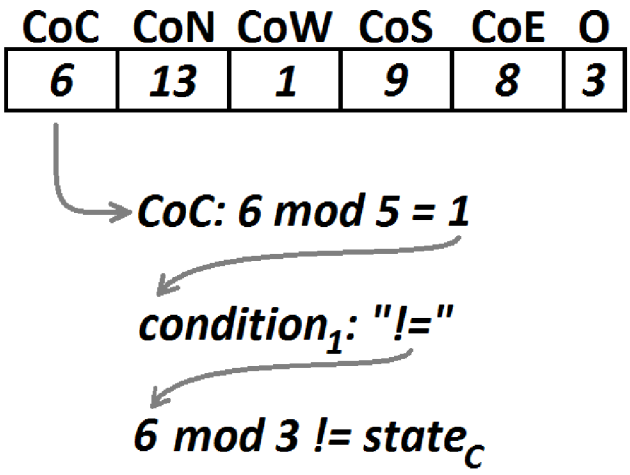
Predpokladajme chromozóm so štruktúrou podľa obrázku 6.4 o veľkosti n štrukturovaných génov, kde $n > 0$. Jednotlivé štrukturované gény budú nadobúdať v rámci chromozómu indexy *0 až n-1*, pričom index i označuje gén ako celok. Ďalej nech je definovaná nasledujúca množina podmienkových výrazov:

- **Condition$_0$** : *rovnosť*: $Co\{cell\}_i$ mod $states_{count} == state_{\{cell\}}$,
- **Condition$_1$** : *nerovnosť*: $Co\{cell\}_i$ mod $states_{count}$!= $state_{\{cell\}}$,
- **Condition$_2$** : *väčšie ako*: $Co\{cell\}_i$ mod $states_{count} > state_{\{cell\}}$,

- **Condition$_3$** : *menšie ako*: $Co\{cell\}_i \bmod states_{count} < state_{\{cell\}}$,
- **Condition$_4$** : *don't care*: *true*,

kde $states_{count}$ predstavuje počet stavov celulárneho automatu, *mod* je funkcia vracajúca zbytok po celočíselnom delení. $Co\{cell\}_i$ značí komponentu aktivačnej jednotky štrukturovaného génu, ktorý má v rámci chromozómu index i a výraz $state_{\{cell\}}$ mapuje stav bunky z celulárneho okolia, pričom reťazec $\{cell\}$ reprezentuje jednu z možností *C, N, W, S, E* — tento reťazec teda mapuje konkrétnu bunku celulárneho okolia, respektíve konkrétnu komponentu aktivačnej jednotky [4]. Zoznam podmienok, v poradí v akom je uvedený vyššie, bude indexovaný hodnotami 0 až 4, to znamená že podmienka *rovnosť* bude označená indexom 0 a *don't care* bude indexovaná ako 4.

Pre lepšiu predstavu konkrétnej podmienky uvádzame nasledujúci príklad pre *nerovnosť*: $CoC_1 \bmod 3 != state_C$. Táto podmienka porovnáva stav bunky C celulárneho okolia so zbytkom po celočíselnom delení, ktorý vzniká keď hodnotu komponenty CoC štrukturovaného génu s indexom 1 delíme počtom stavov celulárneho automatu.



Obrázek 6.5: Obrázok zobrazuje príklad mapovania aktivačnej jednotky pre bunku C celulárneho okolia, ktorá indexuje komponentu CoC aktivačnej jednotky na konkrétnu podmienku z množiny podmienkových výrazov. Predpokladá sa 3-stavový CA s 5-okolím a použitie množiny podmienkových výrazov o mohutnosti piatich podmienok, ktoré boli popísané v odrážkach vyššie. Hodnota komponenty "CoC"sa vydelí počtom podmienok a zbytok po tomto delení predstavuje index konkrétnej podmienky z množiny podmienkových výrazov. Podmienka je následne vybudovaná ako zbytok po celočíselnom delení hodnoty komponenty "CoC"s počtom stavov CA, ktorý sa porovnáva so stavom bunky "C"daného celulárneho okolia.

Samotné mapovanie hodnoty komponenty na podmienku z množiny podmienkových výrazov je surjektívne zobrazenie, to znamená, že viacero hodnôt komponenty môže mapovať rovnakú podmienku. Typ použitej podmienky mapovaný na danú komponentu aktivačnej jednotky génu teda priamo závisí na hodnote čísla uloženého na príslušnom indexe tejto komponenty.

[4] Posledná z uvedených podmienok značí, že bez ohľadu na stav bunky či počet stavov CA bude výsledok tejto podmienky v danej komponente vždy pravdivý.

Výpočet určujúci priradenie podmienky ku kompomente je veľmi jednoduchý — zbytok po celočíselnom delení, ktorý vznikne keď hodnotu komponenty delíme počtom podmienok, predstavuje index podmienky v zozname podmienok. Príklad pre trojstavový CA obohatený o konkrétne číselné hodnoty, zobrazuje obrázok 6.5.

6.3.4 Princíp mapovania genotypu na fenotyp

Sekcia 6.2 pojednáva, že jadro algoritmu celulárnej evolúcie vychádza z celulárneho programovania. Zásadný rozdiel týchto dvoch metód objasnili sekcie 6.3.2 a 6.3.3 venovane štruktúre chromozómu a mapovaniu podmienok metody CE. Jedinečná štruktúra chromozómu tejto metódy samozrejme znamená aj odlišný spôsob výpočtu nového stavu bunky. Nasledujúca sekcia teda popisuje spôsob spojenia celulárneho automatu s evolučnou metódou celulárnaj evolúcie.

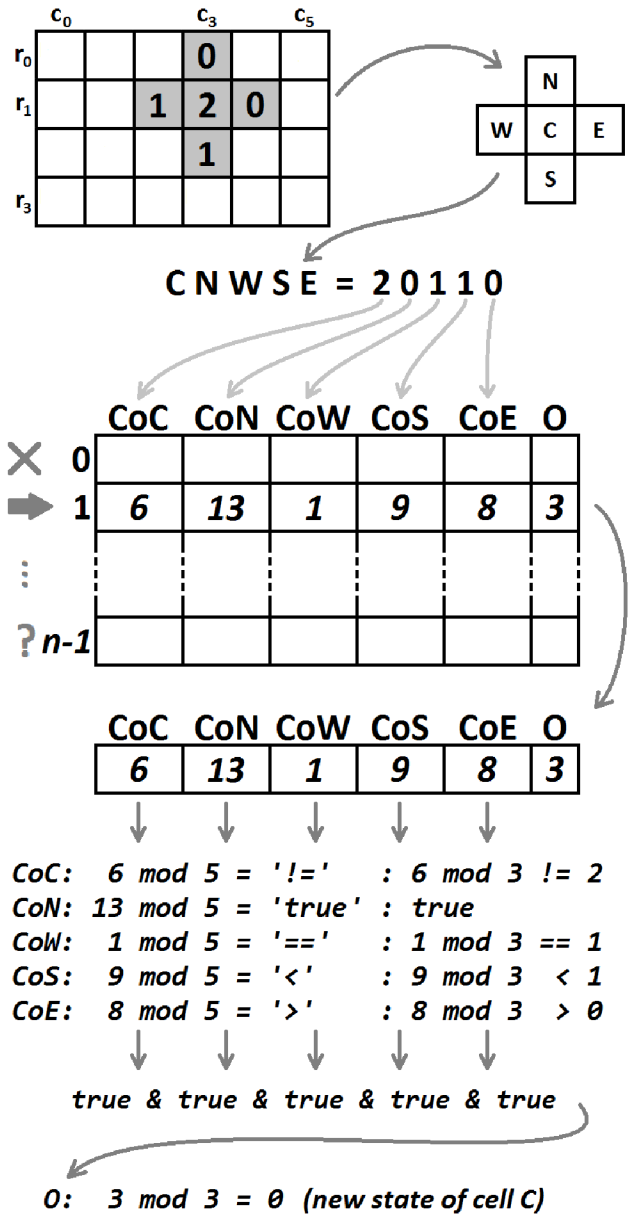
Princíp mapovania medzi genotypom a fenotypom má pôvod v technike predstavenej Bidlom a Vašíčkom [1] a bude objasnený na príklade dvojdimenzionálneho celulárneho automatu, ktorý pracuje s 5-okolím podľa obrázku 2.5a a práve tromi stavmi, ktoré sú rozlíšené číselnými hodnotami *0, 1, 2*. Štruktúru chromozómu celulárnej evolúcie určuje obrázok 6.4 a jednotlivé podmienky sú popísané v sekcií 6.3.3. Popis mapovania, ktorý je rozobratý v niekoľkých nasledujúcich odstavcoch je doložený obrázkom 6.6, ktorý dokladá princíp mapovania medzi genotypom a fenotypom na konkrétnom príklade.

Chromózom celulárnej evolúcie je tvorený z postupnosti n štruktorovaných génov. Vyhodnotenie, ktorý gén bude pre dané celulárne okolie aktívny, prebieha vo cykluse, v ktorom sa postupne berú v potaz jednotlivé gény z chromozómu — výpočet teda začína pre gén s indexom 0 a postupne pokračuje pre gény vyšších indexov, pokiaľ to bude potrebné.

U aktuálne spracovávaného génu sa vyhodnotia všetky komponenty aktivačnej jednotky podľa princípu popísaného v sekcií 6.3.3. Pokiaľ sa všetky komponenty vyhodnotia ako pravdivé, je daný gén aktívny. Ak gén ale aktívny nie je, výpočet sa opakuje pre nasledujúce gény a to až do momentu, než sa nájde taký gén, ktorý bude s ohľadom na dané celulárne okolie aktívny, čím bude umožnené, aby sa jeho hodnota mohla spropagovať ako nový stav bunky do celulárneho automatu. Pokiaľ by sa vyskytol prípad, že žiadny gén chromozómu nebude aktívny, propaguje sa ako preddefinovaná výstupná hodnota chromozómu alela posledného génu (a to bez ohľadu na to, či gén je alebo nie je aktívny).

V momente, keď je vybraná výstupná hodnota chromozómu (alela, ktorá sa použije), je potrebné túto hodnotu znormalizovať, aby mohla byť aplikovaná ako nový stav vyšetrovanej bunky celulárneho automatu. Sekcia 6.3.2 pojednala, že pre zachovanie jednotnej práce s každou hodnotou chromozómu nadobúdajú všetky časti génov (vrátane alely) hodnoty z rozsahu definovaného vzorcom 6.2. Z tohto dôvodu sa prevedie normalizácia alely a to tak, že ako nový stav bunky celulárneho automatu sa aplikuje zvyšok po celočíselnom delení hodnoty alely s počtom stavom celulárneho automatu.

Princíp mapovania genotypu na fenotyp, popísaný v tejto sekcií slovne i graficky, prebieha pre všetky bunky celulárneho automatu. Pri pohľade na algoritmus 6.1 bude predstavený princíp mapovania aplikovaný na riadku algoritmu CE, v ktorom sa počíta nový stav celulárneho automatu — teda na mieste, kde v cykluse krokov celulárneho automatu priebeha vývoj automatu.



Obrázek 6.6: Príklad mapovania genotypu na fenotyp metódy CE: Pri mapovaní sa postupne prechádzajú všetky gény v chromozóme, pričom sa hľadá taký gén, ktorý bude pre dané celulárne okolie aktívny. Gén s indexom 0 bol pre vyhodnotený ako neaktívny, preto sa výpočet presunul ku génu s indexom 1. Aktivačná jednotka génu namapovala podmienky z množiny podmienkových výrazov, ktoré boli následne vyhodnotené vo všetkých komponentách aktivačnej jednotky. Počas vyhodnocovania sa porovnávali hodnoty génu so stavmi buniek z celulárneho okolia. Dokončením tohto výpočtu sa zistilo, že všetky komponenty aktivačnej jednotky génu sú pravdivé, čo znamená, že gén s indexom 1 je pre dané celulárne okolie aktívny. Keďže gén je aktívny, bude jeho výstupná hodnota (alela) normalizovaná, vďaka čomu bude možné alelu spropagovať do celulárneho automatu. Normalizovaná alela reprezentuje nový stav práve vyšetrovanej bunky (teda nový stav bunky C znázorneného celulárneho okolia).

6.4 Celulárna evolúcia s predikciou budúceho stavu okolitých buniek

Nasledujúca sekcia popisuje rozšírené mapovanie genotypu na fenotyp pre metódu celulárnej evolúcie, ktorá bola definovaná v predchádzajúcich sekciách. Úlohou tohto rozšírenia je "predpovedať"správanie sa okolitých buniek, a na základe tohto odhadu upraviť správanie (to znamená budúci stav) práve vyšetrovanej bunky celulárneho automatu.

Metóda celulárnej evolúcie s rozšíreným mapovaním pomocou *"predikcie budúceho stavu okolitých buniek"* (ďalej v texte *CEEx*), je hlavným a najdôležitejším prínosom tejto diplomovej práce.

Rozšírenie *"predikcia budúceho stavu okolitých buniek"* bolo vystavané pre neuniformné celulárne automaty, ktoré sú typické tým, že každá bunka celulárneho automatu obsahuje svoju vlastnú prechodovú funkciu. V automate tohto typu je daná prechodová funkcia "známa"len tej bunke, ktorej životný cyklus riadi (z hľadiska objektového programovania by sme povedali, že prechodová funkcia bunky je privátnou metódou danej bunky), zatiaľčo stav bunky je známy všetkým ostatným bunkám celulárneho automatu (stav je teda verejnou vlastnosťou bunky). Určitá bunka automatu samozrejme "vidí"len stav tých buniek, ktoré sa nachádzajú v jej celulárnom okolí [5].

Majme teda nejakú bunku celulárneho automatu *Cell*, ktorej životný cyklus je riadený jej vlastnou prechodovou funkciou *Cell.TransitionFunction*. Prechodová funkcia je samozrejme implementovaná podľa princípov celulárnej evolúcie — jedná sa teda o chromozóm metódy CE (viď sekcia 6.3.2), ktorého princíp mapovania na celulárny automat popisuje sekcia 6.3.4. Bunka *Cell* je ďalej charakterizovaná svojím stavom *Cell.State* a odkazmi na ostatné bunky zo svojho celulárneho okolia: *Cell.Neighbor.N*, *Cell.Neighbor.W*, *Cell.Neighbor.S*, *Cell.Neighbor.E*. Štruktúra bunky *Cell* teda zapúzdruje svoj stav spolu so svojou prechodovou funkciou a obsahuje odkazy na rovnaké typy štruktúr okolitých buniek.

Predpokladajme teda neuniformný automat zložený z pravidelnej mriežky bunečných štruktúr *Cell*. Keďže v priebehu vyčíslovania svojho nového stavu vidí bunka *Cell* v prostredí takéhoto automatu len aktuálny stav okolitých buniek, ale nepozná ich prechodové funkcie, nie je táto bunka schopná určiť skutočný budúci stav okolitých buniek. Počas výpočtu svojho vlastného nového stavu môže bunka *Cell* vyčísliť budúci stav buniek svojho okolia, a to za použitia svojej vlastnej prechodovej funkcie *Cell.TransitionFunction*.

Tým, že bunka *Cell* aplikuje svoju prechodovú funkciu *Cell.TransitionFunction* na bunky zo svojho okolia, samozrejme nezíska skutočný budúci stav okolitých buniek, ale získa "odhad", ako by sa bunky z jej celulárneho okolia správali, ak by boli riadené prechodovou funkciou náležiacou práve bunke *Cell*. Pri výpočte stavu okolitých buniek sa samozrejme počíta s celulárnym okolím, ktoré náleží týmto susedným bunkám. To znamená, že keď bunka *Cell* odhaduje budúci stav svojho suseda *Cell.Neighbor.N*, predá tejto bunke svoju prechodovú funkciu *Cell.TransitionFunction* a nechá bunku *Cell.Neighbor.N*, aby pomocou nej vypočítala svoj budúci stav na základe svojho vlastného celulárneho okolia. Bunke *Cell.Neighbor.N* je teda dočasne vnútená prechodová funkcia bunky *Cell*.

Dočasným vnútením svojej prechodovej funkcie okolitým bunkám získa aktuálne vyšetrovaná bunka *Cell* odhad možného budúceho stavu okolitých buniek. Bunka *Cell* následne prihliadne na odhady budúcich stavov získané od okolitých buniek a zapracuje ich pri dopočítaní svojho skutočného budúceho stavu. Prihliadnutie k výsledkom okolitých buniek

[5] Ako príklad uveďme celulárne okolie na obrázku 2.5a, ktorého aplikovanie v prostredí dvojrozmerného automatu zobrazujú obrázky 6.1 a 6.6. V prípade tohto okolia sú pre bunku "C"viditeľné stavy len tých buniek, ktoré s ňou bezprostredne susedia (to znamená stavy buniek "N", "W", "S"a "E").

for *každá bunka "Cell"v CellularSpaceIn* **do in parallel**
nextState = 0;
predictThis =
GetNextState(Cell, Cell.TransitionFunction);
predictN =
GetNextState(Cell.Neighbor.N, Cell.TransitionFunction);
predictW =
GetNextState(Cell.Neighbor.W, Cell.TransitionFunction);
predictS =
GetNextState(Cell.Neighbor.S, Cell.TransitionFunction);
predictE =
GetNextState(Cell.Neighbor.E, Cell.TransitionFunction);
$nextState = 0.5 * predictThis + 0.125 * (predictN + predictW + predictS + predictE)$;
CellularSpaceOut.Cell.currentState = nextState $mod\ states_{count}$;
end

Algoritmus 6.2: Pseudokód algoritmu funkcie **CalculateNextCellularSpace(CellularSpaceIn, CellularSpaceOut)**. Táto funkcia prevedie jeden krok vývoja celulárneho automatu, ktorého aktuálna mriežka je uložená v parametry *CellularSpaceIn* a nový stav celulárneho priestoru bude v *CellularSpaceOut*. Algoritmus pracuje s neuniformným celulárnym automatom, ktorý je vybudovaný zo štruktúr typu *Cell*, ktoré zapúzdrujú stav bunky, jej prechodovú funkciu a odkazy na zvyšné bunky z jej celulárneho okolia. Funkcia *GetNextState(CellIn, TransitionFunctionIn)* implementuje princípy celulárnej evolúcie popísané v sekcií 6.3.4 a jej úlohou je vypočítať nový stav bunky *CellIn* pomocou prechodovej funkcie *TransitionFunctionIn*. Vstupný parameter *CellIn* je štruktúra typu *Cell*, parameter *TransitionFunctionIn* definuje prechodovú funkciu (a ako bolo písané vyššie, predstavuje chromozóm metódy CE).

prebieha formou váhovania výsledkov (respektíve odhadov budúcich stavov buniek) a to tak, že predikcií vlastného stavu bunky *Cell* sa priradí váha *50%* a rovnaká váha sa priradí súčtu predikcií budúceho stavu buniek *Cell.Neighbor.N*, *Cell.Neighbor.W*, *Cell.Neighbor.S*, *Cell.Neighbor.E*.

Princíp predikcie budúceho stavu okolitých buniek, ako bol uvedený, dokumentuje algoritmus 6.2. Tento algoritmus zachytáva telo funkcie *CalculateNextCellularSpace(CellularSpaceIn, CellularSpaceOut)*, ktorá zabezpečuje výpočet jedného diskrétneho kroku vo vývoji neuniformného celulárneho automatu. Tento algoritmus sa odkazuje na funkciu *GetNextState(CellIn, TransitionFunctionIn)*, ktorá implementuje princíp mapovania genotypu metódy celulárnej evolúcie na fenotyp neuniformného celulárneho automatu, a to podľa popisu uvedeného v sekcií 6.3.4. Grafické znázornenie tejto funkcie zobrazuje obrázok 6.6.

Popísaná metóda "predpovedania budúceho stavu okolitých buniek"vo svojej podstate rozširuje celulárne okolie danej bunky. Celulárne okolie avšak nie je rozšírené priamo — to znamená, že toto rozšírené okolie nie je priamo mapované do chromozómu, ale je rozdelené do *"sektorov"*, kde každý sektor sa vyčísluje úplne samostatne a nezávisle na ostatných. Jednotlivé sektory reprezentujú celulárne okolia buniek nachádzajúcich sa v okolí aktuálne vyšetrovanej bunky — algoritmus teda pracuje s piatimi sektormi — s celulárnym okolím vyšetrovanej bunky *Cell* a s ďalšími štyrmi okoliami, ktoré náležia bunkám *Cell.Neighbor.N*, *Cell.Neighbor.W*, *Cell.Neighbor.S*, *Cell.Neighbor.E*.

Samostatné vyčíslovanie jednotlivých sektorov je silne paralelizovateľné, čo ukazuje aj algoritmus 6.2 — jednotlivé volania funkcie *GetNextState(CellIn, TransitionFunctionIn)* je možné volať paralelne, čím sa čiastočne vyrovná vyššia výpočtová náročnosť metódy "predpovedania budúceho stavu okolitých buniek"[6].

Metóda CE s rozšírením popísaným v rámci tejto kapitoly by mala byť schopná získať v menšom počte evolučných generácií rovnako kvalitné alebo dokonca lepšie riešenia zadaného problému ako metóda CE bez tohto rozšírenia. Toto tvrdenie vychádza práve z rozšíreného celulárneho okolia a zo spôsobu jeho výpočtu, ktorý je silne paralelizovateľný, vďaka čomu by nároky na reálny výpočtový čas nemali výrazne vzrásť oproti metóde CE bez rozšírenia.

6.5 Prínos celulárnej evolúcie

Predchádzajúce sekcie popísali algoritmus metódy celulárnej evolúcie, zaoberali sa jeho chromozómom i spôsobom mapovania genotypu na fenotyp a zaviedli rozšírenie, ktoré znásobilo možnosti tohto algoritmu. Keďže algoritmus metódy celulárnej evolúcie vychádza z celulárneho programovania, boli veľmi podrobne popísané odlišnosti metód CE a CEEx od CP.

Najdôležitejším prínosom metódy celulárnej evolúcie je práve štruktúra chromozómu, spôsob mapovania genotypu na fenotyp, a najmä rozšírenie, ktorého úlohou je predpovedanie správania sa okolitých buniek celulárneho automatu. Práve vďaka štruktúre chromozómu a jeho mapovaniu na celulárny automat, by malo dôjsť k výraznej zmene v štruktúre, prípadne vo veľkosti stavového priestoru, a teda aj vo veľkosti chromozómu samotného.

Lineárny chromozóm totiž kóduje pre každú kombináciu stavov celulárneho okolia práve jeden gén, pričom žiadny gén chromozómu nie je zdieľaný medzi dve navzájom rôzne okolia. Tento systém síce zaisťuje jednoznačný a veľmi rýchly výpočet nového stavu celulárneho automatu, avšak neúmerne zväčšuje stavový priestor celulárneho automatu — s každým stavom a každou ďalšou bunkou v celulárnom okolí rastie stavový priestor exponenciálne. V prípade neuniformného automatu je rast stavového pristoru ešte znásobený počtom buniek v celulárnej mriežke. Čas potrebný na nájdenie riešenia zadaného problému samozrejme rastie s veľkosťou stavového priestoru.

Práve problém obrovského stavového priestoru komplikovaných automatov a následne značné časové nároky na objavenie riešenia zadaných problémov, má za úlohu vyriešiť metóda celulárnej evolúcie. Vďaka faktu, že metóda CE pracuje so štrukturovanými génmi, ktorých aktivačná jednotka môže daný gén aktivovať pre viacero rôznych kombinácií stavov celulárneho okolia, nemusí byť potrebný tak vysoký počet génov ako v prípade lineárneho chromozómu. Pre komplikované celulárne automaty môže byť chromozóm poskladaný zo štrukturovaných génov výrazne kratší ako lineárny chromozóm. Objektívnou nevýhodou tohto chromozómu je zakomponovanie aktivačnej jednotky do génu, čo zväčšuje dĺžku chromozómu, a kardinalita hodnôt, ktoré sa v chromozóme tejto štruktúry môžu vyskytovať — tá je daná súčinom počtu stavov CA s mohutnosťou množiny podmienkových výrazov. Napriek týmto nevýhodam je prínos tejto metódy návrhu neuniformných celulárnych automatov značný — čo potvrdzujú aj výsledky experimentov popísané v kapitole 8.

[6] Z algoritmu 6.2 výplýva, že pri výpočte nového stavu danej bunky dochádza k vyčíslovaniu piatich rôznych celulárnych okolí, čo znamená päťnásobne väčšie množstvo výpočtov pre určenie stavu každej bunky oproti metóde CE bez tohto rozšírenia. Ďalším extra výpočtom pri určovaní nového stavu bunky je zakomponovanie predpovedaného budúceho stavu okolitých buniek do finálneho stavu práve vyšetrovanej funkcie.

Ako značná výhoda implementácie metódy celulárnej evolúcie sa javí rozšírenie "predpovedania budúceho stavu okolitých buniek". Najvýznamnejším prínosom tejto metódy je fakt, že poskytuje výrazne obsiahlejšie informácie o stave a možnom budúcom vývoji celulárneho automatu. Súbor týchto obsiahlejších informácií zužitkuje každá bunka automatu počas výpočtu svojho nového stavu, vďaka čomu bude schopná progresívnejšie reagovať na aktuálny (respektíve na budúci) vývoj buniek v automate.

Kapitola 7

Princíp implementácie kľúčových častí algoritmov

Táto diplomová práca je venovaná vytvoreniu, implementovaniu a prevereniu schopností novej techniky návrhu celulárnych automatov. Túto novú techniku — náročnosť jej implementácie, nároky na výpočtový čas a hlavne kvalitu získaných výsledkov — je nevyhnutné porovnať s technikami, ktoré sa na riešenie problémov v prostredí celulárnych automatov celkom bežne používajú v súčasnosti.

Úlohou tejto kapitoly je stručne popísať spoločné implementačné aspekty všetkých metód a najmä popísať najdôležitejšie rozdiely. Kapitola je venovaná konkrétnym implementačných vlastnostiam, ktoré boli aplikované pri vytváraní jednotlivých aplikácií pre potreby tejto diplomovej práce.

7.1 Spoločné aspekty implementácie algoritmov

Nasledujúca sekcia a jej jednotlivé podsekcie sú venované stručnému popisu spoločných vlastností jednotlivých metód z hľadiska implementácie. V rámci tejto diplomovaj práce vznikli celkom štyri aplikácie, ktoré implementujú jednotlivé metódy — *genetický algoritmus*, *celulárne programovanie*, *celulárna evolúcia* a *celulárna evolúcia s predikciou budúceho stavu okolitých buniek*.

Všetky štyri aplikácie boli napísané vo vývojovom prostredí *Microsoft Visual C++ 2012* a testované na systéme *Microsoft Windows 7 Ultimate x64*.

7.1.1 Celulárny automat

Všetky metódy využívajú dvojrozmerný dvojstavový celulárny automat, s cyklickými okrajovými podmienkami. Implementované algoritmy ďalej pracujú výhradne s celulárnym 5-okolím, ako bolo definované v sekcií 2.1 a znázornené na obrázku 2.5a. Rozmery mriežky celulárneho automatu ako aj minimálny počet krokov vývoja automatu sú závislé na jednotlivých úlohách, ktoré boli implementované (viď kapitola 5), a konkrétne hodnoty budú uvedené pri jednotlivých experimentoch.

Rozhodnutie použiť v práci výhradne dvojrozmerný automat bolo zvolené z dvoch dôvodov: zadanie úlohy systému logických hradiel vyžaduje prácu v dvojrozmernej celulárnej mriežke (viď kapitola 5). Druhým dôvodom je fakt, že zaistenie požadovaného správania sa automatu je pri dvoch rozmeroch mriežky náročnejšou úlohou pre evolúciu, keďže s rastúcim

celulárnym okolím sa zväčšuje aj veľkosť stavového priestoru zadaného problému. Vďaka tomuto bude hľadanie riešenia úlohy synchronizácie výrazne ťažšou úlohou oproti 1D variante CA, a teda aj porovnanie jednotlivých metód návrhu CA bude zaujímavejšie.

Implementácie všetkých štyroch algoritmov využívajú prostredie neuniformného celulárneho automatu.

7.1.2 Inicializačná konfigurácia celulárneho automatu

V rámci tejto práce boli vybrané dve úlohy, na ktorých majú byť zrovnané schopnosti všetkých metód. Úlohy boli vybrané zámerne tak, aby umožňovali vývoj celulárneho automatu s náhodne vygenerovanými počiatočnými konfiguráciami mriežky celulárneho priestoru.

Prvým krokom vývoja celulárneho automatu je teda vygenerovanie náhodnej počiatočnej konfigurácie CA. Počet takto získaných počiatočných konfiguracií, nad ktorými bude preverovaná každá prechodová funkcia, je daný parametrom *veľkosti populácie*.

7.1.3 Výpočet fitness hodnoty

Implementácia funkcie určenej k výpočtu fitness hodnoty závisí len na konkrétnej riešenej úlohe, to znamená, že všetky metódy zdieľajú úplne rovnaké systémy výpočtu fitness, vďaka čomu bude porovnanie jednotlivých metód v kapitole venovanej experimentom maximálne objektívne.

Úloha synchronizácie

V úlohe synchronizácie sa automat vyvíja v zadanom počte krokov a po ich dokončení sa spustí rutina výpočtu fitness. To znamená, že výpočet hodnoty fitness prebieha až po dokončení vývoja celulárneho automatu.

Samotný výpočet je určený vzájomným porovnaním posledných dvoch krokov vývoja automatu, kde sa podľa zadania zo sekcie 5.1 vyžaduje, aby v jednotlivých krokoch vývoja dochádzalo k striedaniu hodnôt 0 a 1 vo všetkých bunkách.

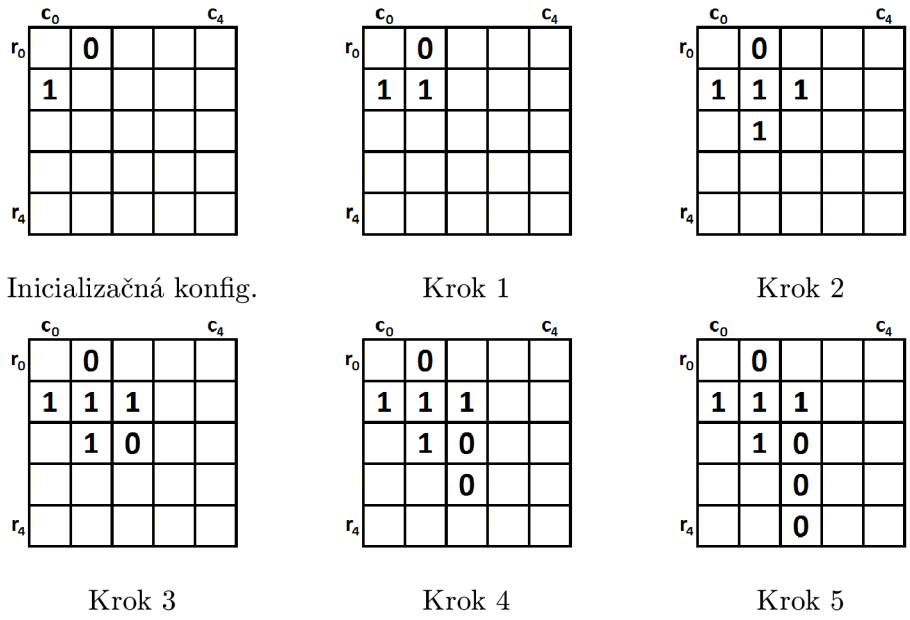
Na začiatku výpočtu fitness sa určí, či stav všetkých buniek v poslednom kroku mal byť 0 alebo 1 (toto rozhodnotie závisí na počte buniek v danom stave, teda na rozhodnotí problému majority — ak je väčšina buniek posledného kroku v stave 1, tak stav všetkých buniek v poslednom kroku mal byť práve 1), čím sa zároveň určí očakávaný stav všetkých buniek v predposlednom kroku vývoja (to znamená, že ak stav všetkých buniek v poslednom kroku má byť 1, tak očakávaný stav všetkých buniek v predposlednom kroku je 0). Následne sa vypočíta jednoduchá diferencia medzi skutočným stavom automatu a očakávaným výsledkom, ktorá určuje počet buniek v nesprávnom stave. Cieľom je dosiahnúť diferenciu rovnú nule, čo zaistí maximálnu hodnotu fitness.

Systém logických hradiel

Na rozdiel od úlohy synchronizácie, v systéme logických hradiel dochádza k výpočtu čiastkových fitness hodnôt po každom kroku vývoja celulárneho automatu. Toto špecifikum výpočtu fitness hodnoty je dané zadaním tejto úlohy, kde úlohou evolučných techník je znovu-objaviť v prostredí celulárneho automatu úplne rovnakú štruktúru logických hradiel, akú ručne navrhol Sipper vo svojej práci [12] a ktorá je zobrazená na obrázku 5.3.

Pri výpočte fitness v tejto úlohe sú dôležité len tie bunky, ktoré vykonávajú "užitočný"výpočet. To znamená, že výpočet fitness hodnoty je obmedzený na tie bunky auto-

matu, ktoré podľa schémy 5.3 obsahujú iný symbol ako "**.**". Bunky so symbolom "**.**"nemajú žiadny vplyv na výslednú funkciu hradla, pretože obsahujú prechodovú funkciu propagujúcu pôvodnú hodnotu danej bunky bez zmeny.



Obrázek 7.1: Séria obrázkov znázorňuje príklad vývoja celulárneho automatu riešiaceho logické hradlo AND definované na obrázku 5.2b. Hradlo pracuje so vstupnými hodnotami $x = 1$ a $y = 0$, očakávaný výsledok je $x\ AND\ y = 0$. Jednotlivé obrázky zachytávajú práve tie bunky automatu, ktoré sú dôležité pre výpočet čiastkovej fitness hodnoty daného kroku vývoja CA. Bunky, ktoré neobsahujú žiadnu hodnotu, nie sú pre výpočet fitness v danom kroku dôležité a môžu nadobúdať ľubovoľné hodnoty.

V jednotlivých krokoch vývoja automatu sa užitočné bunky zapájajú do činnosti výpočtu hradla postupne, ako sa propagujú hodnoty vstupných parametrov x a y a ich transformácií naprieč bunkami automatu. Pre lepšiu predstavu tejto postupnej propagácie hodnôt cez užitočné bunky znázorňuje obrázok 7.1 jednotlivé kroky vývoja automatu riešiaceho hradlo AND.

Výpočet fitness hodnoty v úlohách logických hradiel prebieha postupne, po každom kroku vývoja automatu, čím sa po každom kroku získa čiastková fitness hodnota, ktorá sa pripočíta k celkovej výslednej fitness. Pri výpočte čiastkovej fitness hodnoty sú dôležité stavy užitočných buniek predstavujúcich štruktúru daného hradla. Čiastková fitness hodnota je rovná počtu užitočných buniek, ktoré v danom kroku automatu majú správny stav.

Príklad na obrázku 7.1 znázorňuje korektný výpočet v hradle AND pre vstupné hodnoty $x = 1$ a $y = 0$. Čiastková fitness *Inicializačnej konfigurácie* je v tomto príklade rovná hodnote 2, pre *Krok 1* je fitness rovná 3, pre *Krok 4* to bude 5, atď. Celková fitness je daná ako súčet všetkých čiastkových fitness, čo bude pre hradlo AND $2 + 3 + 5 + 6 + 7 + 8 = 31$. Keďže sa jedná o úplný a korektný výpočet v tomto hradle, je fitness v tomto príklade 100%.

7.1.4 Operátory mutácie a kríženia

Keďže všetky algoritmy — *genetický algoritmus*, *celulárne programovanie* a *celulárna evolúcia* — sú postavené na evolučných základoch, využívajú tieto algoritmy operátory mutácie a kríženia ako prostriedkov na zachovanie diverzity a dedičnosti genetickej informácie.

Algoritmy zdieľajú spoločnú implementáciu týchto operátorov s tým, že metóda celulárnej evolúcie, ktorá má inú štruktúru chromozómu, pracuje s iným parametrom kardinality hodnôt ako genetický algoritmus a celulárne programovanie.

Kríženie je implementované ako jednoduché, jednobodové kríženie, v ktorom sa náhodne vygeneruje index, v ktorom nastane prekríženie rodičovských chromozómov. Samotná implementácia je zhodná s popisom uvedeným v sekcií 3.2.3. Pravdepodobnosť, že dôjde ku kríženiu je určená parametrom so štandardnou hodnotou 75%.

Operátor mutácie vychádza z popisu uvedeného v sekcií 3.2.4. Operátor je implementovaný tak, že pre každú hodnotu chromozómu sa s malou pravdepodobnosťou (predefinovaná hodnota 6.5%) náhodne vygeneruje nová hodnota alely.

7.2 Genetický algoritmus

Genetický algoritmus, využitý v rámci tejto práce, vychádza z popisu uvedeného v sekcií 3.2, a jeho konkrétnu implementáciu znázorňuje pseudokód uvedený v algoritme 7.1.

Implementovaný genetický algoritmus využíva lineárny chromozóm, ktorého štruktúra i spôsob mapovania na fenotyp boli popísané v sekcií 6.3.1. Chromozóm (respektíve prechodová funkcia) teda kóduje všetky možné kombinácie stavov celulárneho okolia, pričom akákoľvek kombinácia stavov celulárneho okolia je jednoznačne mapovaná na určitý lokus chromozómu, ktorý nie je zdieľaný s inými kombináciami stavov.

Algoritmus 7.1 znázorňuje zásadný rozdiel použitého GA oproti jeho štandardným implementáciam — schopnosti každého chromozómu sú preverené na množine počiatočných konfigurácií celulárneho automatu, ktoré boli náhodne vygenerované. Algoritmus GA v tomto prípade teda pracuje s jedným interným cyklom navyše, ktorého zmyslom je preveriť každý chromozóm z populácie chromozómov nad mnohými rôznymi konfiguráciami CA.

Najvýraznejším rozdielom implementácie tejto metódy oproti ostatným dvom je proces selekcie. Selekcia je implementovaná ako turnajová selekcia, ktorá pozostáva z dvoch fáz — v prvej fáze sa náhodne vyberie skupina k jedincov populácie, v druhej fáze prebehne samotný turnaj. Za víťaza turnaja je vyhlásený jedinec s najvyššou hodnotou fitness, tento sa zúčastní na procese vytvorenia novej generácie. Číslo k, ktoré predstavuje veľkosť turnaja, bolo stanovené na hodnotu 2. Vyššia hodnota totiž zvyšuje selekčný tlak kladený na jedincov — ak by sa v jednom kole turnaja vyskytlo viacero dobrých jedincov, vybraný by bol len jeden z nich, čo by mohlo mať kontraproduktívny učinok na rýchlosť konvergencie genetického algoritmu [6]. Proces selekcie implementuje elitizmus, ktorý zaistí, že najlepší jedinec populácie vždy postúpi do nasledujúcej generácie bez zmeny, na jeho chromozóm teda nie je aplikovaný proces mutácie.

```
for evo_run = 1 to evo_run_max do
  inicializuj GA a množinu prechodových funkcií (chromozómov)
  for gen = 1 to gen_max do
    for chrom = 1 to chrom_count do
      namapuj chromozóm chrom z množiny prechodových funkcií do CA
      for pop = 1 to pop_count do
        reinicializuj CA (náhodnou počiatočnou konfiguráciou)
        vykonaj step_max krokov výpočtu v CA
        vypočítaj fitness po kroku step_max
        pričítaj aktuálnu fitness k celkovej fitness populácie
      end for
    end for
    preveď selekciu
    preveď kríženie
    preveď mutáciu
    doplň populáciu novými jedincami
  end for
  exportuj najlepší chromozóm z aktuálneho behu
end for
exportuj najlepší chromozóm spomedzi všetkých behov evolúcie
```

Algoritmus 7.1: Pseudokód implementácie genetického algoritmu využitého v rámci tejto práce.

Vlastnosti celulárneho automatu a najdôležitejších evolučných častí, ako aj spôsob implementácie a vyhodnocovania výbraných úloh, ktoré boli implementované v aplikácií genetického algoritmu popisuje sekcia 7.1.

7.3 Celulárne programovanie

Algoritmus aplikácie celulárneho programovania bol implemenetovaný presne podľa popisu uvedeného v kapitole 4. Podobne ako v aplikácií genetického algoritmu, aj aplikácia CP využíva lineárny chromozóm, ktorého popis spolu so spôsobom jeho mapovania na celulárny automat popisuje sekcia 6.3.1.

Vlastnosti celulárneho automatu a najdôležitejších evolučných častí, ako aj spôsob implementácie a vyhodnocovania výbraných úloh, ktoré boli implementované v aplikácií celulárneho programovania, popisuje sekcia 7.1.

7.4 Celulárna evolúcia

Spôsob implementácie celulárnej evolúcie v prostredí neuniformného celulárneho automatu bol veľmi podrobne zmapovaný v kapitole 6, ktorá je celá venovaná práve tejto novej technike návrhu CA. V prípade celulárnej evolúcie vznikli dve aplikácie — jedna, ktorá aplikuje princípy CE bez rozšírenia, a druhá, ktorá implementuje rozšírenie "predikcie budúceho stavu okolitých buniek"(CEEx).

Keďže aplikácie CE pracujú s dvojstavovým automatom a celulárnym 5-okolím, obsahuje aktivačná jednotka štruktorovaného génu práve päť komponént. Každá z týchto komponént môže namapovať práve jednu z piatich možných podmienok — v rámci tejto práce sú implementované podmienky *rovnosti*, *nerovnosti*, *väčšie ako*, *menšie ako* a *don't care*, ktoré

boli popísané v sekcií 6.3.3. Keďže všetky implementované úlohy pracujú s práve dvomi rôznymi stavmi, je kardinalita možných hodnôt v chromozóme *0 - 9*.

Veľkosť chromozómu — teda počet štrukturovaných génov — bola experimentálne stanovená na päť-násobok počtu stavov celulárneho automatu, celkovo teda chromozóm metódy CE v implementácií tejto práce obsahuje *10* štrukturovaných génov, pričom každý gén je tvorený celkovo šiestimi číselnými hodnotami (päť komponént aktivačnej jednotky génu a nový stav bunky CA). Z uvedených čísel plynie fakt, že celková dĺžka chromozómu je tvorená postupnosťou *60* číselných hodnôt z rozsahu *0 - 9* [1]. Presný počet génov chromozómu použitých pri experimentoch bude uvedený priamo v popise príslušného experimentu.

Zvyšné aspekty v implementáciach celulárnej evolúcie odpovedajú popisu, ktorý je uvedený v jednotlivých sekciách kapitoly 6 venovanej celulárnej evolúcií.

Vlastnosti celulárneho automatu a najdôležitejších evolučných častí, ako aj spôsob implementácie a vyhodnocovania vybraných úloh, ktoré boli implementované v aplikácií celulárnej evolúcie, popisuje sekcia 7.1.

[1] Výpočty vychádzajú zo vzorcov uvedených v sekcií 6.3.2.

Kapitola 8

Experimenty

Cieľom tejto diplomovej práce bolo vytvoriť novú techniku umožňujúcu návrhovať v prostredí celulárneho automatu riešenia najrôznejších problémov — vznikla teda technika celulárnej evolúcie, ktorá je popísaná v kapitole 6. Túto novo-vytvorenú techniku, respektíve jej kvality a schopnosti, je samozrejme nutné porovnať s už existujúcimi technikami, ktoré sa v súčasnosti celkom bežne používajú. Za účelom tohto zrovnania boli vybrané boli metódy genetického algoritmu a celulárneho programovania. Ďalej boli vybrané dve úlohy, ktoré poslúžili na porovnanie existujúcich techník s novým prístupom — úloha synchronizácie a systém implementácie logických hradiel. Tieto úlohy boli popísané v kapitole 5.

Je potrebné podotknúť, že vyhodnotenie na niekoľkých vybraných úlohách nemôže poskytnúť informácie o schopnostiach týchto metód obecne, avšak povahy úloh ponúkajú prehľad o typických výhodách a nevýhodách jednotlivých prístupov pri riešení problémov podobného charakteru.

Úlohou tejto kapitoly je previesť sadu experimentov, v ktorých budeme pomocou jednotlivých techník riešiť zadané problémy, a na základe výsledkov, ktoré budú z pokusov získané, zhodnotíme schopnosti, silné a slabé stránky týchto techník a najmä porovnáme "efektivitu"s akou sú zabehnuté metódy schopné riešiť zadané problémy v porovnaní s novou technikou celulárnej evolúcie.

8.1 Úloha synchronizácie

Úloha synchronizácie bola vybraná ako demonštrácia schopnosti vynútenia daného "globálneho"správania sa celulárneho automatu. V praxi toto tvrdenie znamená, že nechceme pomocou evolučných metód len objaviť prechodovú funkciu neuniformného CA, vykazujúcu určité špecifické správanie v danej bunke mriežky, ale chceme túto funkciu rozšíriť do všetkých buniek tak, aby sa všetky bunky správali a reagovali rovnako, a to bez ohľadu na počiatočnú konfiguráciu mriežky CA.

Cieľom týchto experimentov je globálne vynútiť (to znamená v celej mriežke neuniformného automatu) také správanie, ktoré zabezpečí vzájomnú synchronizáciu všetkých buniek v určitom momente vývoja CA.

V tejto úlohe sa teda nesnažíme o dosiahnutie vysoko špecifického správania sa vybraných buniek mriežky, ale naopak — chceme, aby všetky bunky vykazovali jednotné a pomerne jednoduché správanie — teda aby boli riadené v podstate rovnakou funkciou.

8.1.1 Nastavenie parametrov úlohy

Jednotlivé sekcie tejto podkapitoly popisujú najdôležitejšia spoločné konfiguračné parametre jednotlivých metód, použité v experimentoch s úlohou synchronizácie.

Celulárny automat

Experimenty spadajúce pod túto úlohu boli vykonané v prostredí 2-rozmerného 2-stavového neuniformného celulárneho automatu, využívajúceho celulárne 5-okolie podľa obrázku 2.5a a cyklické okrajové podmienky.

V úlohe synchronizácie sa dá očakávať, že kvalita získaných výsledkov v danom počte generácií evolúcie bude silne závisieť na veľkosti celulárnej mriežky. Z tohto dôvodu boli prevedené dve sady experimentov líšiace sa veľkosťou celulárnej mriežky:

- **malá mriežka**: *rozmery mriežky (šírka x výška):* 4 x 4 bunky,
- **veľká mriežka**: *rozmery mriežky (šírka x výška):* 8 x 8 buniek.

V oboch mriežkach boli prevedené testy s nastavením *5 a 10 krokov vývoja* celulárneho automatu. Úloha má teda preveriť schopnosť zosynchronizovať bunky v pomerne malom počte krokov vývoja CA. Podobne ako pri veľkosti mriežky, aj pri rôznych nastaveniach počtu krokov môžeme očakávať výsledky rôznej kvality — predpokladáme, že menší počet krokov vývoja automatu bude na získanie korektného riešenia náročnejší, takže v prípade 5 krokov vývoja CA dosiahne evolúcia v danom maximálnom počte krokov horšie výsledky.

Základné nastavenia evolúcie

Keďže všetky algoritmy sú postavené na evolučných princípoch, uvádza táto sekcia základné nastavenia evolučného procesu. Pretože evolučné algoritmy sú navrhnuté ako stochastické optimalizačné procesy, je nevyhnutné previesť istú minimálnu sadu nezávislých behov evolúcie s nasledujúcimi nastaveniami:

Počet nezávislých behov evolúcie	100
Maximálny počet generácií	1 000
Počet náhodných konfigurácií CA	50
Mutácia — pravdepodobnosť	6.5 %
Kríženie — pravdepodobnosť	75.0 %

Tabulka 8.1: Základné nastavenia evolučných algoritmov spoločné pre všetky tri metódy.

8.1.2 Malá mriežka celulárneho automatu

Experimentovanie s malou mriežkou (*4 x 4 bunky*) celulárneho automatu má za úlohu demonštrovať rýchlosť jednotlivých metód, s akou sa budú schopné dopracovať k požadovaným výsledkom v pomerne jednoduchej úlohe. Celá mriežka CA totiž obsahuje len 16 buniek, a teda v prípade neuniformného automatu len 16 prechodových funkcií, ktoré riadia celý automat.

Môžeme teda predpokladať, že objavenie vhodnej prechodovej funkcie v nejakej bunke bude trvať dlhšie (pretože automat má menej buniek, a teda aj menej prechodových funkcií, čo znamená menšiu šancu, že nejaká bunka bude mať v určitom okamžiku vývoja vhodnú

prechodovú funkciu) ako v prípade veľkého automatu, avšak na strane druhej bude malá celulárna mriežka výhodnejšia v momente, keď sa vlastnosti vhodných prechodových funkcií budú šíriť do ostatných buniek mriežky.

Môžeme očakávať, že sada experimentov s 5 krokmi vývoja CA by mala byť pre evolučné procesy náročnejšou úlohou oproti experimentovaniu s 10 krokmi vývoja, keďže vyžaduje progresívnejšie prechodové funkcie, ktoré budú schopné zaistiť požadované správanie automatu v menšom počte krokov vývoja CA.

Primárnym cieľom experimentov na malej mriežke CA teda bude preveriť vplyv počtu krokov vývoja celulárneho automatu na kvalitu výsledkov získaných z jednotlivých evolučných procesov.

Preverenie funkcií, ktoré evolučné procesy jednotlivých metód "prehlásia"za najlepšie objavené výsledky, bude vykonané na veľkom množstve náhodne generovaných počiatočných konfigurácií celulárneho automatu. Cieľom tohto procesu bude preveriť, ako sa tieto najlepšie funkcie budú správať na množine konfigurácií CA, z ktorých značná časť nebola použitá pri evolučnom vývoji funkcií [1].

Experimenty s 5 krokmi vývoja celulárneho automatu

Tabuľka 8.2 zachytáva najlepšie objavené riešenia z jednotlivých metód, zatiaľčo grafy na obrázku 8.1 zobrazujú priebeh nárastu fitness týchto najlepších riešení. Z tabuľky môžeme vyčítať najlepšiu fitness [2], ktorú sa jednotlivým metódam podarilo získať ako aj počet generácií, potrebných na dosiahnutie tohto najlepšieho riešenia.

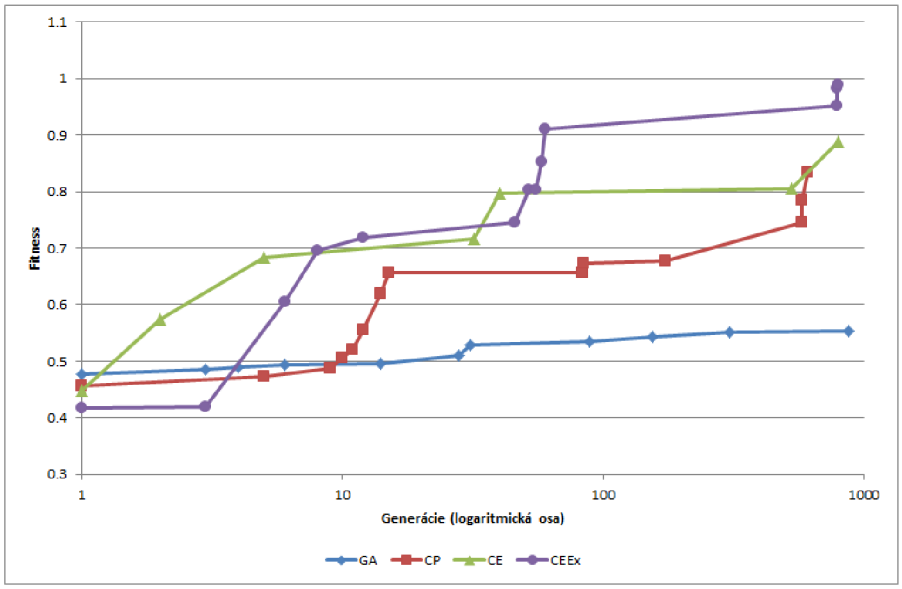
Evolučná metóda	Evolúcia		Testovanie
	Max. fitness	Generácia	Priemerná fitness
GA	0.55375	872.0	0.498456
CP	0.834375	608.0	0.765012
CE	0.88875	798.0	0.905694
CEEx	0.988125	791.0	0.978694

Tabulka 8.2: Najlepšie výsledky dosiahnuté pomocou jednotlivých metód pri riešení úlohy synchronizácie pre 5 krokov vývoja CA, v prostredí malej mriežky. Stĺpec *Evolúcia* vyjadruje najlepšie hodnoty získané z evolučného procesu. Tieto najlepšie prechodové funkcie dosiahli pri preverovaní kvality fitness, ktorá je evedená v stĺpci *Testovanie*, pričom testy prebehli na celkovo 5 000 náhodne vygenerovaných počiatočných konfiguráciach CA.

Môžeme si všimnúť, že úlohu synchronizácie v prostredí malého automatu s nízkym počtom krokov vývoja CA zvládli jednotlivé metódy vyriešiť na minimálne 75 %. Týmto môžeme prvotný predpoklad, že experiment s malou mriežkou CA by nemal byť pre jednotlivé metódy problematický, považovať za splnený. Výrazným odklonom od tohto tvrdenia je

[1] Na základe tabuľky evolučných nastavení 8.1 budú jednotlivé metódy evolvovať riešenia na množine 50 náhodných konfigurácií CA. To znamená, že v každej generácií evolučného procesu sa vygeneruje práve 50 konfigurácií, na ktorých budú jednotlivé metódy evolvovať prechodové funkcie tak, aby získali vhodné riešenia zadaného problému. Keďže preverenie schopností najlepších objavených riešení bude prebiehať na mnohonásobne väčšom počte náhodných konfigurácií CA, môžeme s ohľadom na veľkosť stavového priestoru CA predpokladať, že väčšina testovacích konfigurácií nebola použitá v procese evolučného hľadania riešenia.

[2] Fitness uvedená v stĺpci *Evolúcia* bola dosiahnutá na sade náhodných konfigurácií CA v mnohých generáciach vývoja evolučného algoritmu. Toto riešenie, ktoré bolo objavené pomocou evolučného procesu, a ktorého fitness je prezentovaná práve v stĺpci *Evolúcia*, bolo následne otestované na veľkom počte náhodne vygenerovaných konfigurácií CA, pričom výsledok tohto testu — priemernú fitness — zachytáva stĺpec *Testovanie*.



Obrázek 8.1: Grafy na obrázku zachytávajú priebeh nárastu fitness jednotlivých metód v priebehu generácií evolučného procesu. Zobrazené grafy odpovedajú evolúcií najlepších prechodových funkcií jednotlivých metód, ktorých parametre prezentuje tabuľka 8.2.

genetický algoritmus, ktorý mal k dispozícií až 50 jedincov v populácií, a teda výrazne väčší počet LPF (celkový počet LPF je pre GA daný súčinom veľkosti populácie a počtom buniek neuniformného CA, v tomto prípade až *50 * 16 = 800* LPF). Ostatné metódy pracovali len s jedinou inštanciou neuniformného CA a teda s celkom 16 LPF. Napriek tomu sa GA svojími výsledkami nedokázal priblížiť ostatným metódam.

Zaujímavejšie je porovnanie metód, ktoré vznikli špeciálne pre neuniformné automaty — CP, CE a CEEx. Tu môžeme pozorovať značné odstupňovanie v kvalitách získaných riešení. Metódy CE a CEEx dosiahli lepšie výsledky ako štandardný CP, i keď rozdiel medzi CP a CE je veľmi malý. Ešte zaujímavejšie sa javí porovnanie metódy CE s metódou CEEx obsahujúcou rozšírenie "predikcie budúceho stavu okolitých buniek"— rozšírená metóda dosiahla v približne rovnakom počte generácií značne lepšieho výsledku.

V grafe priebehu fitness (diď obrázok 8.1) môžeme vidieť, že jednotlivé metódy zaznamenali najmasívnejší nárast fitness v prvej stovke generácií evolučného vývoja. Metódy CP, CE a CEEx dokázali zabezpečiť už od prvých generácií vývoja veľmi prudký nárast fitness hodnoty v prvej približne stovke generácií a priebežný rast v nasledujúcom vývoji. Metóda GA oproti nim ukázala minimálny rast fitness počas celého vývoja.

Tabuľka 8.2 ďalej zobrazuje výsledky testovania najlepších riešení jednotlivých metód na množine 5 000 náhodne vygenerovaných konfigurácií CA. Na výsledkoch môžeme pozorovať, že u všetkých metód s výnimkou CE došlo pri tomto testovaní k poklesu priemernej fitness hodnoty v rádovo jednotkách percent.

Pri testovaní bol dosiahnutý výraznejší rozdiel jednotlivých metód: veľký skok môžeme pozorovať najmä medzi metódami celulárneho programovania a celulárnej evolúcie — CE dokáže generovať o približne 14 % lepšie riešenia a CEEx pridáva ešte ďalších 7 % navrch, celkovo je teda metóda CEEx až o približne 20 % lepšia oproti CP a o takmer 40 % lepšie oproti GA. V tomto bode je už prínos metódy celulárnej evolúcie, a najmä jej rozšírenej formy veľmi viditeľný.

Evolučná metóda	Priem. fitness	Priem. generácia	Smerodatná odchýlka
GA	0.521359	485.675	0.01212
CP	0.713609	519.225	0.04509
CE	0.824844	548.225	0.06724
CEEx	0.946266	441.65	0.04062

Tabulka 8.3: Priemerné výsledky dosiahnuté pomocou jednotlivých metód pri riešení úlohy synchronizácie pre 5 krokov vývoja CA v prostredí malej mriežky. Výsledky odpovedajú priemeru 100 nezávislých behov evolúcie.

Tabuľka 8.3 zachytáva priemerné výsledky, ktoré sa jednotlivým metódam podarilo dosiahnúť v priebehu 100 nezávislých behov. V tejto tabuľke môžeme sledovať paralelu s najlepšími výsledkami z tabuľky 8.2, čo sa zrovnania kvalít jednotlivých metód týka.

Veľmi zaujimavo z priemerných výsledkov vychádza porovnanie metód GA a CP s CEEx. Nová metóda s rozšírením bola schopná v menšom počte generácií získať v priemere výrazne lepšie riešenia ako štandardné metódy GA a CP. Rozdiel priemerných výsledkov medzi GA a CEEx ukazujú viac ako 40 % nárast fitness a rozdiel medzi CP a CEEx je viac ako 23 %, čo vypovedá veľmi jasne o schopnostiach tejto novej metódy.

Experimenty s 10 krokmi vývoja celulárneho automatu

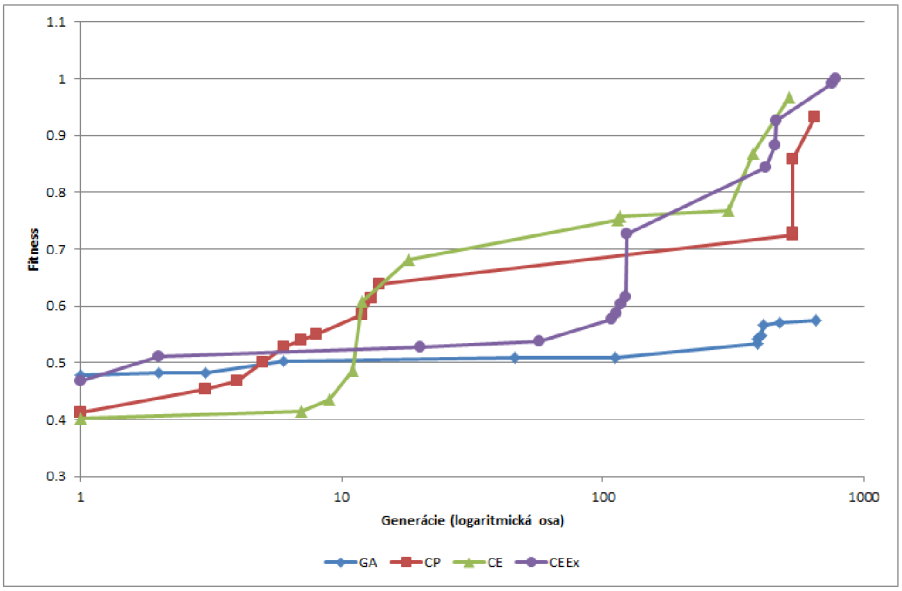
Pri pohľade do tabuľky 8.4 môžeme pozorovať najlepšia získané výsledky pri experimentovaní s 10 krokmi vývoja CA. Podľa očakávaní vyslovených v úvode tejto sady experimentov, pomohlo zvýšenie počtu krokov vývoja CA z pôvodných 5 na novú hodnotu 10 krokov celkom výrazne.

Evolučná metóda	Evolúcia		Testovanie
	Max. fitness	Generácia	Priemerná fitness
GA	0.57375	654.0	0.5257
CP	0.9325	645.0	0.932362
CE	0.966875	517.0	0.973969
CEEx	1.0	775.0	0.994244

Tabulka 8.4: Najlepšie výsledky dosiahnuté pomocou jednotlivých metód pri riešení úlohy synchronizácie pre 10 krokov vývoja CA v prostredí malej celulárnej mriežky. Stĺpec *Evolúcia* vyjadruje najlepšie hodnoty získané z evolučného procesu. Tieto najlepšie prechodové funkcie dosiahli pri preverovaní kvality fitness, ktorá je evedená v stĺpci *Testovanie*, pričom testy prebehli na celkovo 5 000 náhodne vygenerovaných počiatočných konfiguráciach CA.

Metódy CP, CE a CEEx si s rovnakým nastavením evolučných parametrov viedli pri vyššom počte krokov CA lepšie — v zásade môžeme usúdiť, že čím horšie si daná metóda viedla pri 5 krokoch, o to väčší nárast fitness dosiahla pri krokoch desiatich. Metóda CEEx síce dosiahla v 1 % prípadov fitness 1.0, ale v porovnaní s výsledkom z 5 krokov CA bolo zaznamenané zlepšenie len o necelé 2 %. Zato metóda CP si polepšila o približne 10 % a metóda CE o 8 % oproti experimentom s 5 krokmi vývoja CA. Najmenej významný prírastok dosiahla metóda GA a to o rovné 2 %.

Zvýšenie počtu krokov CA pri zachovaní všetkých ostatných parametrov teda viedlo k celkovému posunu všetkých metód ku kvalitnejším riešeniam a zároveň sa rozdiely medzi jednotlivými metóda zmenšili.



Obrázek 8.2: Grafy na obrázku zachytávajú priebeh nárastu fitness jednotlivých metód v priebehu generácií evolučného procesu. Zobrazené grafy odpovedajú evolúcií najlepších prechodových funkcií jednotlivých metód, ktorých parametre prezentuje tabuľka 8.4.

Grafy na obrázku 8.2 zachytávajú priebeh nárastu fitness týchto najlepších riešení v priebehu generácií evolučného vývoja. Pri pohľade na priebeh fitness môžeme vidieť paralelu s experimentami s 5 krokmi vývoja CA — jednotlivé grafy majú relatívne podobné priebehy. Avšak oproti pokusom s 5 krokmi vývoja CA (viď obrázok 8.1), kde bol zaznamenaný prudký nárast fitness v pomerne krátkom čase (a to už od prvých generácií evolúcie), nie je vidieť v tejto sade experimentov tak prudké zvýšenie fitness — nárast fitness je pri 10 krokoch vývoja CA skôr pozvoľný než nárazový, pričom najvýraznejší prírastok nastal vo všetkých metódach až na konci evolučného vývoja.

Pri pohľade do tabuľky obsahujúcej výsledky testovania (viď 8.4) opäť usúdime fakt z predchádzajúceho experimentu — testovanie najlepších získaných riešení na náhodne generovanej množine 5 000 konfigurácií CA viedlo k mierne nižšej priemernej fitness.

Evolučná metóda	Priem. fitness	Priem. generácia	Smerodatná odchýlka
GA	0.530297	579.0	0.01007
CP	0.815625	513.35	0.04333
CE	0.859719	428.925	0.09306
CEEx	0.912375	562.175	0.08266

Tabulka 8.5: Priemerné výsledky dosiahnuté pomocou jednotlivých metód pri riešení úlohy synchronizácie pre 10 krokov vývoja CA v prostredí malej celulárnej mriežky. Výsledky odpovedajú priemeru 100 nezávislých behov evolúcie.

V priemere si s novým nastavením krokov CA vedú jednotlivé metódy lepšie, čo sa kvality získaných prechodových funkcií týka (viď tabuľka 8.5). Napríklad metóda CP si v priemere polepšila až o 10 % (oproti experimentom s 5 krokmi CA) a to len vďaka vyššiemu počtu krokov CA. Celkovo tak došlo k zmenšeniu rozdielu medzi jednotlivými metódami — pri 5 krokoch vývoja CA bol rozdiel medzi CP a CEEx až 23 %, pri 10 krokoch je priemerný

rozdiel týchto dvoch metód len približne 10 %.

Celkovo teda môžeme zhrnúť, že predpoklad vyslovený na začiatku podkapitoly experimentov s úlohou synchronizácie v malej mriežke CA — 10 krokov vývoja CA oproti piatim zabezpečí, že v danom čase bude evolúcia schopná získať kvalitnejšie riešenia — sa ukázal ako pravdivý. Zvýšenie počtu krokov CA pri rovnakých nastaveniach evolúcie naozaj viedlo ku kvalitnejším výsledkom. Zároveň môžeme konštatovať, že pri vyššom počte krokov CA sa zmenšili rozdiely v kvalite výsledkov získaných z rôznych metód, avšak nová metóda (CE a najmä CEEx) zostala, čo sa kvality týka, neprekonaná.

8.1.3 Veľká mriežka celulárneho automatu

V prípade experimentovania s veľkou mriežkou (*8 x 8 buniek*) automatu môžeme predpokladať, že objavenie prvej vhodnej prechodovej funkcie, ktorá zaistí nejakej bunke korektné správanie, bude jednoduchšie, ako v prípade malého automatu. Toto tvrdenie sa opiera o fakt, že veľká mriežka automatu obsahuje 64 buniek, a teda v prípade neuniformného automatu 64 prechodových funkcií, ktoré budú evolučným procesom optimalizované. Väčší počet kandidátnych riešení by teda mal znamenať aj väčšiu pravdepodobnosť, že prvé vhodné riešenie (t.z. prechodová funkcia nejakej bunky) bude objavené v kratšom čase ako v prípade malého automatu. Na strane druhej môžeme predpokladať, že veľká mriežka bude omnoho výraznejšou prekážkou v procese evolučného šírenia (napríklad pomocou kríženia chromozómov) vhodných vlastností z kvalitných prechodových funkcií naprieč zvyšnými bunkami neuniformného automatu. Celkovo teda veľká mriežka bude predstavovať výrazné negatívum pri šírení dobrých vlastností, a môžeme teda očakávať menej kvalitné výsledky ako v malej mriežke CA.

Z dôvodu predpokladanej výrazne väčšej náročnosti objavenia vhodného automatu, respektíve všetkých jeho prechodových funkcií, bude experimentovanie s veľkou mriežkou obmedzené len na použitie 10 krokov vývoja celulárneho automatu. Experimenty s malou mriežkou celulárneho automatu totiž ukázali, že pri rovnakých nastaveniach evolučných parametrov dokáže vyšší počet krokov vývoja CA výrazne uľahčiť evolučnému procesu objavenie vhodného riešenia problému synchronizácie.

Môžeme však predpokladať, že 1 000 generácií evolúcie nebude ani pre 10 krokov vývoja CA dostačujúcich, preto budú prevedené dve sady experimentov — jedna bude ohraničená 1 000 generáciami, druhá sada bude mať k dispozícií až 10 000 generácií evolučného vývoja.

Cieľom experimentovania s veľkou mriežkou teda bude preveriť vplyv maximálneho počtu generácií na kvalitu výsledkov, získaných z jednotlivých evolučných procesov.

Preverenie kvality najlepších získaných riešení z jednotlivých metód bude opäť prevedené na veľkom množstve náhodne generovaných konfigurácií CA, podobne ako to bolo v prípade experimentov s malou mriežkou CA.

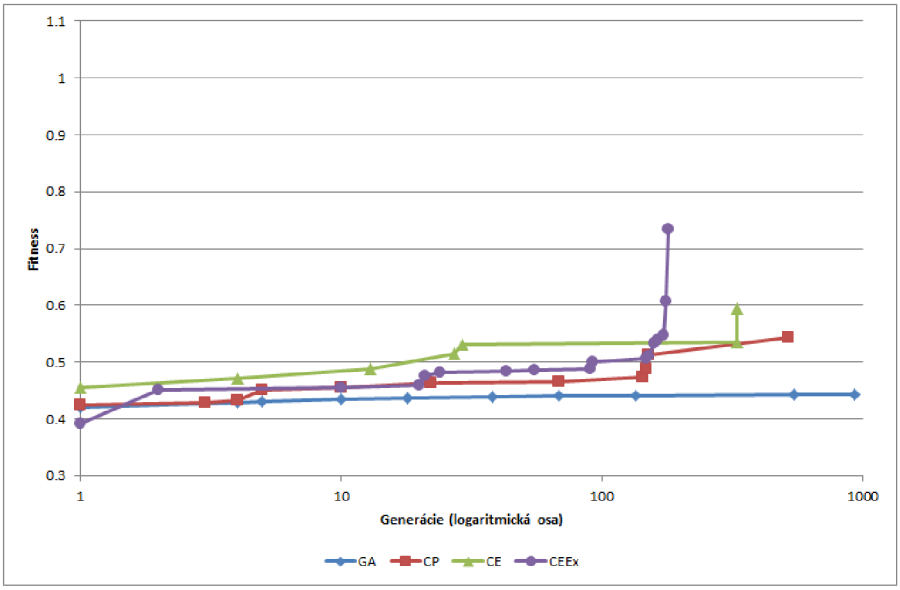
Experimenty s 1 000 generáciami evolučného vývoja

Pohľad do tabuľky 8.6 prezradí, že prvotný predpoklad o výrazne vyššej náročnosti úlohy synchronizácie vo veľkej mriežke CA bol naplnený — najlepšie získané výsledky sú rádovo o desiatky percent horšie ako výsledky z experimentov s malou mriežkou — metóda CEEx klesla o približne 27 %, metóda CE o 37 %, algoritmus CP dokonca o 39 %, len genetický algoritmus zaznamenal približne 13 % pokles kvality výsledkov oproti experimentom s malej mriežke CA s rovnakými nastaveniami ostatných parametrov.

Môžeme teda usúdiť, že v neuniformnom prostredí kladie väčšia mriežka automatu výrazne väčšie nároky na evolučný proces, najmä ak je evolúcia ohraničená nízkym počtom

Evolučná metóda	Evolúcia		Testovanie
	Max. fitness	Generácia	Priemerná fitness
GA	0.443437	925.0	0.424714
CP	0.54375	515.0	0.522125
CE	0.593594	330.0	0.591105
CEEx	0.733906	180.0	0.705714

Tabulka 8.6: Najlepšie výsledky dosiahnuté pomocou jednotlivých metód pri riešení úlohy synchronizácie pre 10 krokov vývoja CA v maximálne 1 000 generáciach evolúcie vo veľkej celulárnej mriežke. Stĺpec *Evolúcia* vyjadruje najlepšie hodnoty získané z evolučného procesu. Tieto najlepšie prechodové funkcie dosiahli pri preverovaní kvality fitness, ktorá je evedená v stĺpci *Testovanie*, pričom testy prebehli na celkovo 5 000 náhodne vygenerovaných počiatočných konfiguráciach CA.



Obrázek 8.3: Grafy na obrázku zachytávajú priebeh nárastu fitness jednotlivých metód v priebehu generácií evolučného procesu. Zobrazené grafy odpovedajú evolúcií najlepších prechodových funkcií jednotlivých metód, ktorých parametre prezentuje tabuľka 8.6.

generácií.

Grafy priebehu fitness na obrázku 8.3 prezrádzajú, že nárast fitness v priebehu generácií bol u metód CP, CE i CEEx veľmi pozvoľný, zatiaľčo u metódy GA takmer nulový, a to počas celého evolučného procesu.

Tabuľka 8.6 zachytáva výsledky pretestovania najlepších získaných výsledkov z veľkej celulárnej mriežky na množine 5 000 náhodne generovaných konfigurácií. Podobne, ako pri experimentoch s malou mriežkou vidíme mierny pokles priemernej fitness daný tým, že mnoho z testovacích konfigurácií nebolo použitých počas evolučného procesu. Výsledky testovania neuniformných metód nám ukazujú značne lepšie výsledky metódy CEEx oproti CE a najmä oproti CP, kde CEEx dosahuje až o 18 % lepšie výsledky. Pri porovnaním s GA vidíme o dokonca 28 % lepšie výsledky metódy CEEx.

Tabuľka priemerných výsledkov 8.7 odhaľuje veľmi vyrovnané spektrum fitness vo všetkých neuniformných metódach. Výsledky potvrdzujú domnienku z úvodu tejto sekcie — pre neuniformné metódy je väčšia celulárna mriežka výraznou prekážkou k ceste za kvalitnými

Evolučná metóda	Priem. fitness	Priem. generácia	Smerodatná odchýlka
GA	0.438746	622.0	0.00644
CP	0.51598	500.275	0.01752
CE	0.562523	496.275	0.01669
CEEx	0.571656	512.075	0.10411

Tabulka 8.7: Priemerné výsledky dosiahnuté pomocou jednotlivých metód pri riešení úlohy synchronizácie pre 10 krokov vývoja CA v maximálne 1 000 generáciach evolúcie vo veľkej celulárnej mriežke. Výsledky odpovedajú priemeru 100 nezávislých behov evolúcie.

výsledkami. Rozdiely jednotlivých metód sú avšak natoľko malé, že z priemerných výsledkov nemožno vyvodiť obecné závery o kvalitách týchto metód. Výrazne horšie výsledky oproti ostatným metódam dosiahol jedine algoritmus GA, u ktorého bol rozdiel až takmer 14 % oproti najlepšej metóde CEEx.

Môžeme však pozorovať jeden zaujímavý fakt, ktorý nastal pri metóde CEEx — diferencia medzi priemernými riešeniami a riešením najlepším je približne 15 %, zatiaľčo tieto diferencie u metód CE a CP sú približne 3 %. Môžeme teda usúdiť, že metóda CEEx je schopná generovať riešenia s väčším rozptylom, a preto je schopná svoje najlepšie riešenie výraznejšie odlíšiť od riešení priemerných.

Celkové zhrnutie výsledkov tejto sady experimentov — veľká mriežka a 10 krokov vývoja CA spolu s ohraničením evolúcie na 1 000 generácií — dopadá v neuniformných metódach opäť v prospech metódy CEEx, i keď rozdiel oproti zvyšným metódam už nie je tak zásadný, ako v experimentoch s malou mriežkou CA.

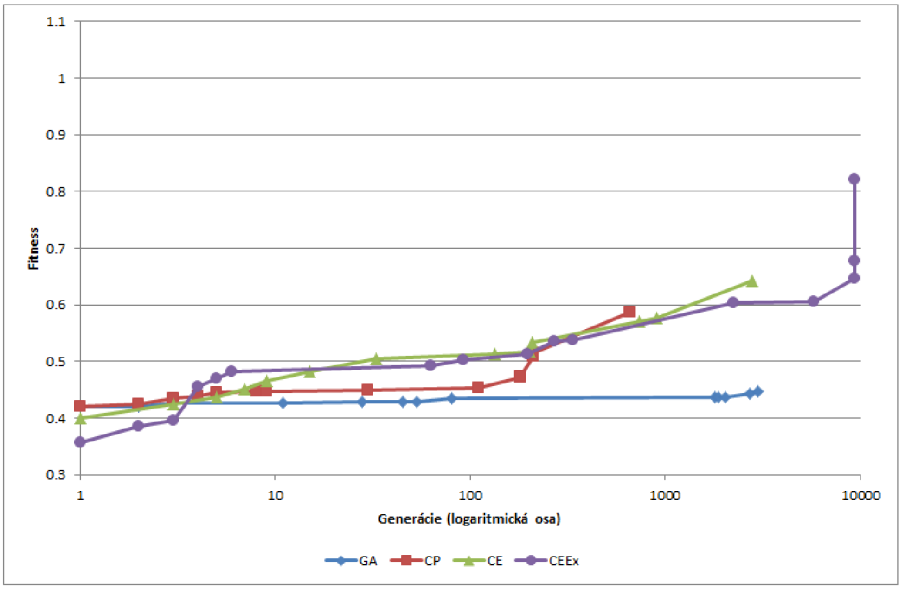
Experimenty s 10 000 generáciami evolučného vývoja

Z tabuľky 8.8 je patrné, že 10-násobné zvýšenia maximálneho počtu generácií z pôvodnej hodnoty 1 000 na nových 10 000 prinieslo vo všetkých metódach zlepšenie výsledkov len o rádovo jednotky percent. Najvýraznejšie zlepšenie zaznamenala metóda CEEx, u ktorej zvýšenie počtu generácií viedlo k získaniu o približne 9 % lepších výsledkov, ako dosiahla táto metóda v pokusoch s 1 000 generáciami. Najmenší prírastok zaznamenala metóda GA, ktorá si polepšila len o približne 0.5 % oproti experimentu s 1 000 generáciami.

Evolučná metóda	Evolúcia		Testovanie
	Max. fitness	Generácia	Priemerná fitness
GA	0.448125	2993.0	0.420445
CP	0.587656	658.0	0.562288
CE	0.6425	2802.0	0.628236
CEEx	0.822031	9330.0	0.792747

Tabulka 8.8: Najlepšie výsledky dosiahnuté pomocou jednotlivých metód pri riešení úlohy synchronizácie pre 10 krokov vývoja CA v maximálne 10 000 generáciach evolúcie vo veľkej celulárnej mriežke. Stĺpec *Evolúcia* vyjadruje najlepšie hodnoty získané z evolučného procesu. Tieto najlepšie prechodové funkcie dosiahli pri preverovaní kvality fitness, ktorá je evedená v stĺpci *Testovanie*, pričom testy prebehli na celkovo 5 000 náhodne vygenerovaných počiatočných konfiguráciach CA.

Grafy na obrázku 8.4 zachytávajú priebehy fitness pre najlepšie získané riešenia. Pri experimentovaní s veľkou mriežkou môžeme vidieť paralely medzi grafmi z tejto sady experimentov a s grafmi získanými na 1 000 generáciach, ktoré sú zachytené na obrázku 8.3.



Obrázek 8.4: Grafy na obrázku zachytávajú priebeh nárastu fitness jednotlivých metód v priebehu generácií evolučného procesu. Zobrazené grafy odpovedajú evolúcií najlepších prechodových funkcií jednotlivých metód, ktorých parametre prezentuje tabuľka 8.8.

Metóda GA, rovnako ako v predchádzajúcom experimente, nedosiahla počas celého vývoja fakticky žiadne prírastky, zatiaľčo metódy CP, CE a CEEx dosahovali počas vývoja aspoň minimálne prírastky fitness, pričom najväčší rozmach zaznamenali na konci vývoja. Najlepšie tento fakt dokumentuje metóda CEEx, ktorá v oboch pokusoch s veľkou mriežkou dosiahla práve na konci evolučného procesu veľmi prudký nárast fitness v krátkom čase.

Už v predchádzajúcich experimentoch bola vyslovená domnienka, že metóda CEEx dosahuje výrazne väčšieho rozptylu fitness hodnôt od priemeru. Tento fakt je doložený práve na grafoch experimentov s veľkou mriežkou, kde pri pohľade do tabuliek s priemernými hodnotami fitness môžeme naozaj pozorovať, že rozdiel medzi najkvalitnejším získaným riešením a priemernými riešeniami je práve u metódy CEEx najväčší zo všetkých metód.

Tabuľka 8.8 ďalej zachytáva výsledky testovania najlepších získaných riešení na množine 5 000 náhodne vygenerovaných konfigurácií celulárneho automatu. Algoritmus GA nezaznamenal fakticky žiadny rozdiel oproti experimentu s 1 000 generáciami — desať-násobné zvýšenie počtu generácií nepomohlo tejto metóde takmer vôbec. CP, CE a CEEx si počas testovania riešení z vyššieho počtu generácií polepšili o pár percent s tým, že najväčší prínos z nárastu počtu generácií zaznamenala metóda CEEx, ktorá oproti testovaniu s 1 000 generáciami zaznamenala o 9 % lepšie výsledky.

Zaujimavé je zrovnanie z testovania jednotlivých metód v rámci tohto pokusu s počtom generácií rovným 10 000 — metóda CEEx dosiahla oproti najslabšej GA o približne 38 % a proti CP o celých 23 % kvalitnejšie výsledky. Pre zrovnanie — pri pokusoch s 1 000 generáciami bol rozdiel medzi CEEx a GA 29 % a rozdiel medzi CEEx a CP 18 % v prospech CEEx.

Tabuľka 8.9 dokladá pomerne vyrovnaný pomer síl medzi neuniformnými metódami, a to ako v kvalite riešení, tak aj v počte generácií potrebných na dosiahnutie týchto výsledkov. Najlepšie umiestnenie získala CEEx, metóda CE sa opäť zaradila svojimi kvalitami medzi metódy CP a CEEx. V tomto experimente sa v ešte väčšej miere prejavila schopnosť metódy CEEx generovať výsledky s väčším rozptylom od priemeru, čo môžeme pozorovať

Evolučná metóda	Priem. fitness	Priem. generácia	Smerodatná odchýlka
GA	0.44225	5153.32	0.03867
CP	0.554723	5270.45	0.05304
CE	0.597266	4340.15	0.01924
CEEx	0.641211	5484.18	0.13101

Tabulka 8.9: Priemerné výsledky dosiahnuté pomocou jednotlivých metód pri riešení úlohy synchronizácie pre 10 krokov vývoja CA v maximálne 10 000 generáciach evolúcie vo veľkej celulárnej mriežke. Výsledky odpovedajú priemeru 100 nezávislých behov evolúcie.

pri porovnaní najlepších riešení s riešeniami priemernými.

Experimentovanie s veľkou mriežkou celulárneho automatu ukázalo, že nárast počtu buniek je značnou prekážkou v evolúcií neuniformného riešenia úlohy synchronizácie — všetky preverované metódy dosahovali výrazne horších výsledkov v porovnaní s experimentami v malej mriežke.

Zvýšenie limitu 1 000 evolučných generácií na novú hodnotu 10 000 síce zabezpečilo zlepšenie kvality výsledkov vo všetkých metódach, avšak nárast fitness nebol až tak veľký — 10-násobné zvýšenie generácií umožnilo nárast fitness o 9 % v metóde CEEx, pričom zvyšné metódy dosiahli výrazne menšieho nárastu — len okolo 3 až 5 % oproti pokusom s 1 000 generáciami evolúcie, metóda GA dokonca nezaznamenala takmer žiadne zlepšenie výsledkov.

Celkovo dosiahla opäť najlepšie výsledky metóda CEEx — jednak v experimente s 10 000 generáciami dosiahla najkvalitnejšie riešenia, a za druhé dokázala najviac vyťažiť zo zvýšenia počtu generácií.

8.1.4 Diskusia

V rámci tejto sekcie boli prevedené experimenty s úlohou synchronizácie, na ktorej sme chceli demonštrovať schopnosti jednotlivých metód pri vynucovaní pomerne jednoduchého správania — úlohy synchronizácie — v prostredí celého automatu.

Boli prevedené dve sady experimentov — prvá na malej mriežke celulárneho automatu (*4 x 4 bunky*), kde sme preverovali vplyv počtu krokov vývoja CA na kvalitu získaných výsledkov, a druhú sadu, ktorá sa zamerala na prostredie veľkého celulárneho automatu (*8 x 8 buniek*), kde prebiehalo preverenie vplyvu maximálneho počtu generácií v jednotlivých evolučných metódach.

Experimenty zamerané na malú mriežku celulárneho automatu boli obmedzené na 1 000 generácií vývoja evolúcie, čím sme sa snažili zdôrazniť schopnosť evolúcie objaviť vhodné riešenie v krátkom časovom úseku. Napriek tomuto obmedzeniu dokázali všetky štyri metódy získať v danom čase zaujímavé riešenia, a navyše sa podarilo zachytiť vplyv počtu krokov CA na kvality získaných výsledkov.

Porovnanie jednotlivých metód ukázalo značnú prevahu nových prístupov — CE a predovšedkým CEEx — nad bežnými metódami GA a CP. V priemere ukázala metóda CEEx nad CP prevahu až o 23 %, a oproti GA dokonca až o 42 % (diferencie priemernej fitness v malom počte krokov vývoja CA). Experimentovanie s veľkým počtom krokov vývoja CA v malej celulárnej mriežke vyrovnalo schopnosti jednotlivých metód — CEEx si však stále udržala o približne 10 % lepšie priemerné výsledky oproti metóde CP a o približne 38 % oproti GA.

Pri spriemerovaní získaných výsledkov zistíme, že najlepšia metóda CEEx ukázala nad CP prevahu až o 16 %, a oproti GA dokonca až o 40 % (priemer všetkých experimentov v malej mriežke CA). V týchto prípadoch sa metóda CE umiestnila čo do kvality výsledkov medzi metódy CP a CEEx. Zaujimavé sa javí zistenie, že všetky štyri metódy si v priemere vystačili s približne rovnakým počtom generácií evolučného vývoja.

Experimenty s veľkou mriežkou celulárneho automatu boli obmedzené počtom krokov vývoja automatu — na základe výsledkov získaných z malej mriežky bola zvolená hodnota 10 krokov, čím bol automatu ponechaný dostatočný čas na vývoj konfigurácií. Zároveň bol týmto obmedzený tlak na prechodové funkcie, ktoré neboli nútené zaistiť vývoj synchronizácie v minimálnom počte krokov vývoja CA.

Prvá sada experimentov bola obmedzená na 1 000 evolučných generácií, pričom jednotlivé neuniformné metódy ukázali vyrovnanú priemernú kvalitu výsledkov — metóda CEEx predviedla len o približne 6 % kvalitnejšie riešenia oproti CP a o približne 13 % kvalitnejšie riešenia oproti GA. Zaujimavé ale bolo zistenie, že riešenia metódy CEEx majú väčší rozptyl kvality, vďaka čomu bolo najlepšie riešenie z CEEx schopné prekonať najlepšie riešenie CP o celých 19 % a najlepšie riešenie z GA bolo prekonané dokonca o 39 %.

Po desať-násobnom zvýšení počtu generácií z pôvodných 1 000 na 10 000, dosiahli priemerné výsledky jednotlivých metód väčšie rozdiely — CEEx získala o približne 9 % kvalitnejšie riešenia oproti CP a o približne 20 % si polepšila oproti GA. V tejto sade experimentov sa však ešte viac zdôraznila schopnosť metódy CEEx generovať výsledky s väčším rozptylom od priemeru — najlepšie riešenie získané z metódy CEEx takto prekonalo najlepšie riešenie z CP až o 23 % a najlepšie riešenie GA bolo prekonané o 38 %.

Výsledky tejto sady experimentov ukázali, že najmä metóda CEEx si v zrovnaní s CP vedie veľmi dobre, a je schopná za preverovaných okolností získať výrazne lepšie riešenia v približne rovnakom počte generácií. Ešte výraznejší sa ukázal rozdiel medzi metódami CEEx a GA, kde rozdiel v kvalite získaných riešení dosahoval až desiatky percent.

Sila metódy CEEx sa ukázala najmä v malom prostredí obmedzenom nízkym počtom generácií evolúciea krokov vývoja CA a následne v prostredí veľkom, v ktorom je na vývoj k dispozícií veľké množstvo generácií evolúcie. Teda na jednej strane je prostredie so značnými obmedzeniami, ale s menším stavovým priestorom, a na strane druhej je prostredie s veľkou mierou voľnosti, avšak s obrovským stavovým priestorom. V týchto dvoch prípadoch dokázala metóda CEEx získať výrazne kvalitnejšie výsledky ako štandardné metódy GA a CP.

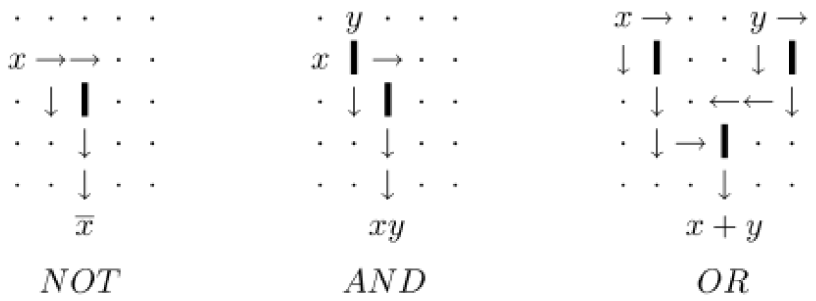
Druhá silná stránka metódy CEEx oproti ostatným metódam sa ukázala pri "zmenšení miery obmedzení"— teda v momente, keď sme sa rozhodli evolučnému procesu uľahčiť prácu odstránením, respektíve zmiernením niektorých príliš reštriktívnych obmedzení — napríklad keď sme zvýšili počet krokov vývoja CA alebo zvýšili celkový počet generácií evolučného vývoja. V oboch týchto prípadoch dokázala metóda CEEx získať zo zmiernenia týchto obmedzení najväčší prírastok fitness, teda najväčší nárast kvality objavených riešení.

Veľmi zaujímavá je domnienka o väčšom rozptyle riešení, ktoré generuje metóda CEEx. Vďaka tejto schopnosti sú najkvalitnejšie riešenia tejto metódy výrazne lepšie, ako priemer všetkých získaných riešení.

Čo sa rýchlosti jednotlivých metód týka, sú výsledky vo všetkých experimentoch pomerne vyrovnané — žiadna z metód nedosiahla výrazne lepšie výsledky ako metódy ostatné — priemerný počet generácií, potrebných na objavenie najkvalitnejšieho riešenia, je teda pomerne stabilný naprieč jednotlivými neuniformnými metódami.

8.2 Systém logických hradiel

Na rozdiel od úlohy synchronizácie, kde sme sa snažili "globálne"vynútiť pomerne jednoduché správanie sa buniek v prostredí celého automatu, je úloha implementácie systému logických hradiel výrazne odlišná. Systém logických hradiel má za úlohu znovu-objaviť pomocou evolučného procesu štruktúry logických hradiel *AND, OR a NOT* v prostredí neuniformného celulárneho automatu tak, ako tieto hradlá popísal Sipper vo svojej práci [12]. Implementáciu týchto hradiel zobrazuje obrázok 8.5, na ktorom môžeme pozorovať pravidelnú dvojrozmernú bunečnú mriežku, kde jednotlivé symboly v mriežke reprezentujú určitú špecifickú prechodovú funkciu danej bunky.



Obrázek 8.5: Implementácia logických hradiel *AND*, *OR* a *NOT* v prostredí dvojrozmerného celulárneho automatu pomocou propagačných pravidiel a pravidla pre výpočet funkcie *NAND*. *Prebraté z [12].*

Aby celulárny automat simulujúci činnosť takto definovaných logických hradiel bol schopný korektne fungovať, bude nevyhnutné, aby evolučné procesy objavili vo všetkých "užitočných"bunkách automatu prechodové funkcie natoľko presne, aby sa užitočné bunky, simulujúce činnosti jednotlivých častí hradla, boli schopné v správnom okamžiku zapojiť do činnosti hradla [3]. V opačnom prípade hradlo nebude schopné korektne fungovať pre akúkoľvek počiatočnú konfiguráciu celulárneho automatu.

Cieľom experimentov spadajúcich pod túto sekciu je preverenie schopnosti jednotlivých techník, "zamerať"svoju pozornosť na určité špecifické časti automatu a v nich objaviť veľmi špecifické prechodové funkcie, zatiaľčo ostatné bunky nie sú pre činnosť automatu, a teda ani pre evolučný proces zaujimavé.

8.2.1 Nastavenie parametrov úlohy

Nasledujúce sekcie tejto podkapitoly popisujú nastavenia najdôležitejších parametrov jednotlivých aplikácií. V sekciách nájdeme nastavenie spoločných parametrov i špecifiká príslušných metód.

[3] Činnosť hradla prebieha v jednotlivých diskrétnych krokoch vývoja celulárneho automatu tak, že užitočné bunky si postupne predávajú čiastkové výsledky výpočtu — to znamená, že bunky v kroku $n + 1$ preberú stav z užitočných buniek, ktoré previedli časť výpočtu hradla v kroku n vývoja CA. Postupné zapájanie užitočných buniek do výpočtu hradla (to znamená aktivovanie buniek hradla) v prostredí automatu simulujúceho hradlo AND je znázornené v sérií obrázkov 7.1.

Celulárny automat

Experimentovanie so systémom implementácie logických hradiel bolo prevedené na dvojrozmernom, 2-stavovom celulárnom automate, ktorý využíva celulárne 5-okolie ako bolo definované na obrázku 2.5a a cyklické okrajové podmienky. Automat je neuniformný, keďže úloha logických hradiel vyslovene vyžaduje rozličné typy prechodových funkcií v jednotlivých bunkách.

Minimálna veľkosť mriežky CA potrebná pre riešenie tejto úlohy je daná samotnými rozmermi štruktúr logických hradiel, ako boli definované na obrázku 8.5. S ohľadom na výsledky experimentov úlohy synchronizácie je očakávateľné, že väčšia mriežka automatu by mohla výrazne skomplikovať úlohu evolučného procesu, a preto bola zvolená minimálna mriežka, ktorá svojou plochou pokrýva štruktúru daného logického hradla:

- **Hradlo AND**: *rozmery mriežky (šírka x výška):* 5 x 5 buniek,
- **Hradlo NOT**: *rozmery mriežky (šírka x výška):* 5 x 5 buniek,
- **Hradlo OR**: *rozmery mriežky (šírka x výška):* 6 x 5 buniek.

Experimenty boli prevádzané s minimálnym počtom krokov vývoja CA potrebných na spropagovanie (a stransformovanie) vstupných parametrov do výstupnej bunky:

- **Hradlo AND**: *vývoj CA:* 6 krokov,
- **Hradlo NOT**: *vývoj CA:* 6 krokov,
- **Hradlo OR**: *vývoj CA:* 8 krokov.

Hradlá AND a OR majú dva vstupné parametre, hradlo typu NOT len jeden. Keďže sa jedná o parametre, musí evolučný proces počítať s tým, že na tieto vstupy môže byť privedená ľubovoľná kombinácia logických stavov 0 *a* 1. Z tohto dôvodu bude každá z náhodne generovaných konfigurácií CA (teda z konfigurácií, ktoré sa používajú v evolučnom procese, respektíve v procese testovania najkvalitnejšieho riešenia) použitá celkom štyrikrát (respektíve dvakrát pre hradlo NOT), aby došlo k prevereniu danej konfigurácie CA pre všetky možné kombinácie vstupných parametrov.

To znamená, že veľkosť populácie (teda počet náhodných konfigurácií CA použitých v metódach GA, CP, CE a CEEx) bude interne (v hlavnom cykle algoritmu) štvornásobne väčší, ako určuje parameter veľkosti populácie [4].

Základné nastavenia evolúcie

S ohľadom na fakt, že všetky metódy využívajú k objaveniu riešenia evolučné princípy, uvádza tabuľka 8.10 nastavenia najdôležitejších evolučných parametrov.

Oproti úlohe synchronizácie bol zvolený menší počet jedincov v populácií, respektíve menší počet náhodných počiatočných konfigurácií CA použitý v každej generácií evolučného procesu. Reálny počet však bude v prípade hradiel AND a OR štvornásobne väčší, keďže každá z týchto konfigurácií bude preverená pre všetky možné kombinácie vstupných parametrov. V prípade hradla OR bude reálny počet konfigurácií väčší dvojnásobne, keďže toto hradlo pracuje len s jediným vstupným parametrom.

[4] To znamená, že pri veľkosti populácie 25 jedincov bude v procese evolučného hľadania riešenia vygenerovaných 25 náhodných konfigurácií CA v každej generácií, avšak každá z týchto konfigurácií bude použitá celkom štyrikrát, zakaždým ale s rôznou kombináciou buniek so vstupnými parametrami hradla.

Počet nezávislých behov evolúcie	100
Maximálny počet generácií	100 000
Počet náhodných konfigurácií CA	25
Mutácia — pravdepodobnosť	6.5 %
Kríženie — pravdepodobnosť	75.0 %

Tabulka 8.10: Základné nastavenia evolučných algoritmov spoločné pre všetky tri metódy.

8.2.2 Experiment s hradlom AND

Logické hradlo AND, ako ho definoval Sipper vo svojej práci [12], vyžaduje celkovú plochu celulárneho automatu 25 buniek rozdelených do mriežky *5 x 5* buniek. Na tejto ploche sa nachádza celkom 8 užitočných buniek:

- **Down Propagation**: 3 bunky,
- **Right Propagation**: 1 bunka,
- **NAND**: 2 bunky,
- **No Change**: 2 bunky *(vstupné parametre hradla "x" a "y").*

Hradlo AND je teda tvorené celkom štyrmi druhmi užitočných buniek, z nich tri sú prechodové funkcie propagačného typu a jeden druh je funkcia zaisťujúca logickú funkciu NAND. Môžeme predpokladať, že najkomplikovanejšou časťou bude objavenie práve tejto prechodovej funkcie, ktorá má v zrovnaní s propagačnými funkciami komplikovanejšiu štruktúru.

Prechodové funkcie zvyšných 17 buniek zadefinoval Sipper vo svojom hradle ako "No Change", teda "bez zmeny". Tieto bunky by teda počas výpočtu hradla nemali nijako meniť svoj stav. Prvé experimenty ale ukázali, že vyžadovať zachovanie stavu vo zvyšných bunkých je pre evolučný proces veľmi náročná úloha. Preto bol v tejto sade experimentov zvolený pozvoľnejší prístup umožňujúci ľubovoľný stav ostatných buniek automatu. Jadro úlohy však ostalo zachované — objaviť štruktúru hradla zloženú z užitočných buniek ako ju definoval Sipper. Vďaka tomuto uľahčeniu úlohy môžeme očakávať, že evolúcia sústredí svoje sily na dôležité oblasti celulárneho automatu, čo by v danom počte generácií mohlo viesť ku kvalitnejším výsledkom.

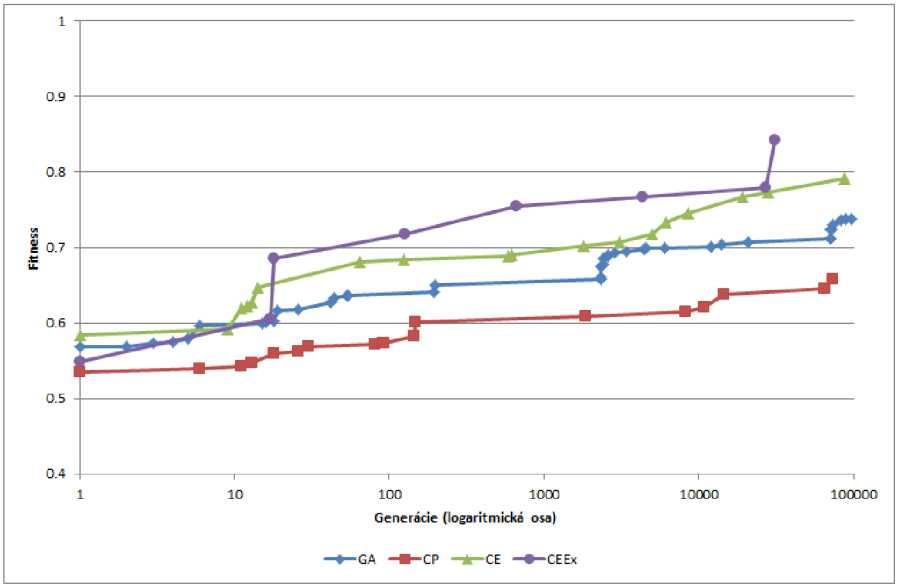
Výsledky experimentov

Tabuľka 8.11 zobrazuje najlepšie dosiahnuté výsledky jednotlivých neuniformných metód riešiacich úlohu logického hradla AND. Na rozdiel od pomerne jednoduchej úlohy synchronizácie, ktorej výsledky boli popísané v sekcií 8.1, môžeme pozorovať značný pokles kvality výsledkov, a to aj napriek tomu, že experimenty s hradlom AND mali k dispozícií viac generácií evolučného procesu a dokonca aj menšiu mriežku ako úloha synchronizácie.

Môžeme pozorovať, že rozdiel medzi najlepším výsledkom metódy CEEx oproti CP je až takmer 19 %. Môžeme si všimnúť, že metóda CEEx dosiahla o približne 5 % lepšie výsledky ako jej sesterská metóda CE, ktorá však neovplýva rozšírením "predikcie budúceho stavu okolitých buniek". Zaujimavé je pozorovanie, že metóda CP získala horšie výsledky ako metóda neuniformného GA.

Evolučná metóda	Evolúcia		Testovanie
	Max. fitness	Generácia	Priemerná fitness
GA	0.737903	96737.0	0.675722
CP	0.657258	73639.0	0.604823
CE	0.791129	87260.0	0.781266
CEEx	0.842742	31254.0	0.841734

Tabulka 8.11: Najlepšie výsledky dosiahnuté pomocou jednotlivých metód pri riešení úlohy logického hradla AND. Stĺpec *Evolúcia* vyjadruje najlepšie hodnoty získané z evolučného procesu. Tieto najlepšie prechodové funkcie dosiahli pri preverovaní kvality fitness, ktorá je evedená v stĺpci *Testovanie*, pričom testy prebehli na celkovo 10 000 náhodne vygenerovaných počiatočných konfiguráciach CA.



Obrázek 8.6: Grafy na obrázku zachytávajú priebeh nárastu fitness jednotlivých metód v priebehu generácií evolučného procesu. Zobrazené grafy odpovedajú evolúcií najlepších prechodových funkcií jednotlivých metód, ktorých parametre prezentuje tabuľka 8.11.

Z tohto môžeme potvrdiť domnienku, že evolvovanie takto komplexných štruktúr, ktoré pre svoju činnosť vyžadujú veľmi presne definované prechodové funkcie, je pre evolučný proces veľmi náročnou úlohou.

Grafy zachytené na obrázku 8.6 prezentujú priebeh fitness najlepších objavených riešení. Môžeme pozorovať veľmi pozvoľný nárast fitness hodnôt všetkých štyroch metód v priebehu celého procesu evolučného vývoja. Grafy dokladajú, že získanie riešení bolo veľmi náročnou úlohou, keďže jediné výraznejšie prírastky k fitness zaznamenali len metódy CE a CEEx, a aj to len na začiatku evolučného vývoja.

Ďalej si môžeme všimnúť, že výraznejšiu prevahu mala metóda CEEx nad CE vo fáze evolučného vývoja ohraničeného približne generáciami 10 až 10 000 a v oblasti záveru evolučného procesu. To je rozdiel oproti pokusom s úlohou synchronizácie, kde metóda CEEx častokrát dosahovala prevahu na začiatku evolučného procesu a najmä v jeho závere, zatiaľčo stredná oblasť evolučného vývoja patrila — čo sa kvality riešení týka — metóde CE.

Tabuľka 8.11 prezentuje výsledky testovania najlepších získaných riešení. Testovanie prebiehalo na množine 2 500 náhodne generovaných konfigurácií CA, avšak každá konfigurácia bola testovaná so všetkými kombináciami vstupných parametrov hradla AND, takže celkový počet konfigurácií, na ktorých prebehlo preverenie kvality, je až 10 000.

V tomto bode je veľmi zaujimé pozorovanie, že počas testovania došlo k niekoľkopercentnému poklesu priemernej fitness vo všetkých metódach, okrem metódy CEEx. Riešenie, ktoré táto metóda získala, sa zrejme najviac blíži k očakávanému, keďže z náhodých konfigurácií, z ktorých väčšina určite nebola použitá v evolučnom procese, sme získali takmer rovnako kvalitné výsledky ako z evolučných konfigurácií CA.

Evolučná metóda	Priem. fitness	Priem. generácia	Smerodatná odchýlka
GA	0.720887	55581.2	0.01336
CP	0.649839	74621.4	0.00572
CE	0.777177	68940.2	0.01386
CEEx	0.823468	68309.8	0.02012

Tabulka 8.12: Priemerné výsledky dosiahnuté pomocou jednotlivých metód pri riešení úlohy logického hradla AND. Výsledky odpovedajú priemeru 100 nezávislých behov evolúcie.

Pri pohľade do tabuľky 8.12 môžeme pozorovať, že spriemerované výsledky jednotlivých nezávislých behov evolúcie sa príliš nelíšia od najkvalitnejších získaných riešení. Tento fakt je podporovaný aj grafmi priebehu fitness z obrázku 8.6, ktoré ukazujú pozvoľný nárast fitness v priebehu celého vývoja.

Na základe výsledkov môžeme usúdiť, že jednotlivé metódy si s evolučným procesom hľadania konfigurácie hradla AND poradili pomerne slušne. Výsledky, ktoré dosiahli sú lepšie, ako výsledky úlohy synchronizácie vo veľkom celulárnom automate, avšak výrazne horšie ako v automate malom. Opäť sa teda potvrdzuje domnienka, že veľkosť automatu v prípade neuniformnej úlohy dokáže veľmi ovplyvniť kvalitu výsledkov. Druhým faktorom ovplyvňujúcim kvalitu je počet generácií evolúcie, ktoré sú k dispozícií. V prípade úlohy hradla AND sme pracovali s mriežkou o počte 25 buniek, ktorá čo do veľkosti zapadá medzi malú (celkom 16 buniek) a veľkú (celkom 64 buniek) mriežku úlohy synchronizácie, pričom počet generácií evolúcie bol 100-násobne vyšší oproti malej mriežke a 10-násobne vyšší oproti veľkej mriežke. Dôvodom, prečo ani jedna metóda nedokázala prekonať malú mriežku synchronizačnej úlohy je požadovaná štruktúra prechodových funkcií — zatiaľčo úloha synchronizácie vyžadovala v podstate jediný typ prechodovej funkcie s pomerne jednoduchým správaním vo všetkých bunkách, hradlo AND vyžaduje špecializované prechodové funkcie v užitočných bunkách mriežky.

Rovnako ako v úlohách synchronizácie, aj pri riešení logického hradla typu AND v neuniformnom automate sa ako najlepšia metóda ukázala CEEx, ktorá dokázala prekonať všetky ostatné neuniformné metódy. Čo do kvality sa jej dokázala najviac priblížiť metóda CE, ktorá získala o približne 5 % horšie výsledky. Metóda CEEx však ukázala jednoznačne lepšie výsledky oproti štandardne používanej metóde CP, kde rozdiel priemeru najkvalitnejších riešení je až 17 %.

8.2.3 Experiment s hradlom NOT

Podobne ako u logického hradla AND, aj hradlo NOT definované Sipperom v práci [12] vyžaduje celkovú plochu neuniformného automatu 25 buniek v mriežke *5 x 5* buniek. Táto plocha však obsahuje len 7 užitočných buniek:

- **Down Propagation**: 3 bunky,
- **Right Propagation**: 2 bunky,
- **NAND**: 1 bunka,
- **No Change**: 1 bunka *(vstupný parameter hradla "x").*

Na základe Sipperov štruktúry logického hradla NOT, by zvyšných 18 buniek automatu malo mať prechodovú funkciu "No Change". V experimentoch s hradlom AND bolo definované, že tieto bunky môže nadobúdať ľubovoľnú prechodovú funkciu a teda ľubovoľný stav. Keďže hradlo typu NOT má jednoduchšiu štruktúru ako hradlo AND, na základe čoho by sa dalo očakávať, že získanie kvalitných riešení pomocou evolučného procesu nebude tak náročnou úlohou. Preto bolo rozhodnoté, že v tejto sade experimentov budeme vyžadovať zachovanie stavu ostatných buniek automatu.

Pri výpočte fitness bol väčší dôraz kladený na užitočné bunky hradla a to tak, že fitness užitočnej bunky má 4-krát väčšiu váhu, ako fitness bunky neužitočnej.

Tým, že v tejto sade experimentov budeme vyžadovať zachovanie stavu ostatných buniek, môžeme oproti experimentom s hradlom AND očakávať horšie výsledky a to aj napriek tomu, že hradlo má jednoduchšiu štruktúru: skladá sa z menšieho počtu užitočných buniek (rozdiel celkovo jednej bunky) a navyše pre svoju činnosť vyžaduje vyevolvovanie len jedinej prechodovej funkcie typu *NAND*, ktorú môžeme považovať za najnáročnejšiu spomedzi všetkých typov prechodových funkcií potrebných na vybudovanie logických hradiel.

Výsledky experimentov

Tabuľka 8.13 zobrazuje najlepšie dosiahnuté výsledky jednotlivých metód pri riešení neuniformnej úlohy logického hradla NOT.

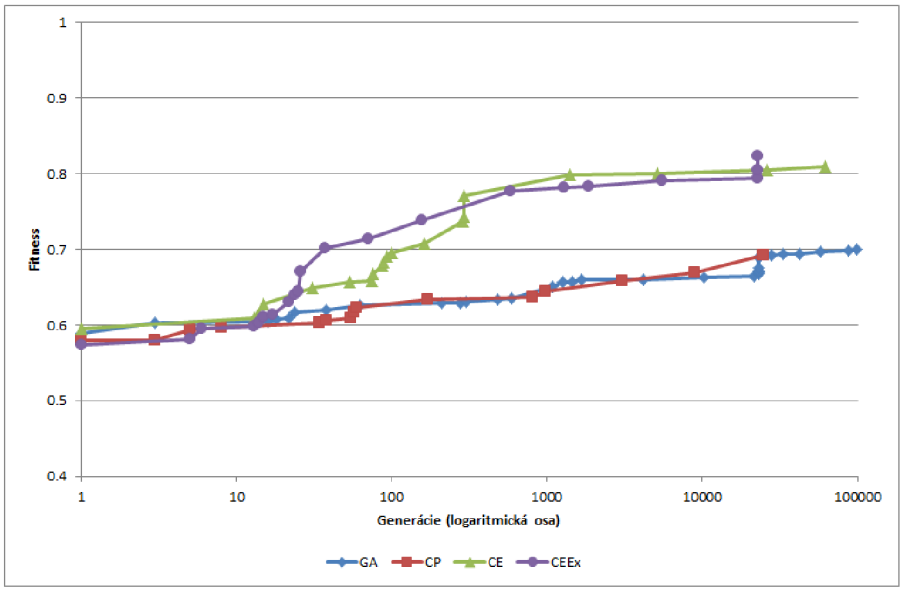
Evolučná metóda	Evolúcia		Testovanie
	Max. fitness	Generácia	Priemerná fitness
GA	0.7	98994.0	0.653877
CP	0.692308	24799.0	0.67642
CE	0.809423	61772.0	0.803238
CEEx	0.823269	22814.0	0.820363

Tabulka 8.13: Najlepšie výsledky dosiahnuté pomocou jednotlivých metód pri riešení úlohy logického hradla NOT. Stĺpec *Evolúcia* vyjadruje najlepšie hodnoty získané z evolučného procesu. Tieto najlepšie prechodové funkcie dosiahli pri preverovaní kvality fitness, ktorá je evedená v stĺpci *Testovanie*, pričom testy prebehli na celkovo 5 000 náhodne vygenerovaných počiatočných konfiguráciach CA.

Predpoklad, že evolúciou hradla NOT nezískame tak kvalitné výsledky ako v prípade hradla AND sa potvrdil. Hradlo NOT naozaj má jednoduchšiu štruktúru, avšak tým, že evolučný proces sa musel zamerať nielen na užitočné bunky hradla samotného, ale aj na bunky ostatné, nepodarilo sa mu získať tak kvalitné výsledky, ako v úlohe hradla AND. Rozdiel medzi najkvalitnejšími výsledkami z jednotlivých metód medzi hradlami AND a NOT ale nie je moc veľký — pohybuje sa v rádovo jednotkách percent.

Zaujímavé sa javí zistenie, že výsledky jednotlivých metód sa v tejto úlohe vyrovnali — rozdiel medzi štandardnými metódami GA a CP je len necelé 1 %, rozdiel medzi CE

a CEEx sú necelé 2 %. Obe nové metódy si však udržujú viac ako 10 % rozdiel od metód štandardných.



Obrázek 8.7: Grafy na obrázku zachytávajú priebeh nárastu fitness jednotlivých metód v priebehu generácií evolučného procesu. Zobrazené grafy odpovedajú evolúcií najlepších prechodových funkcií jednotlivých metód, ktorých parametre prezentuje tabuľka 8.13.

Grafy na obrázku 8.7 potvrdzujú, že výraznejší rozdiel metód CE a CEEx oproti metódam GA a CP bol udržiavaný takmer počas celého evolučného vývoja. Grafy zároveň prezrádzajú, že v úlohe hradla NOT, kde bol dôraz kladený nielen na užitočné bunky hradla, ale aj na bunky ostatné (teda v úlohe, kde sa evolučný proces zameral na celý automat), podávali metódy CE a CEEx veľmi vyrovnané výsledky — jediným miestom, kde mala metóda CEEx výraznejšiu prevahu je oblasť približne medzi generáciami 10 a 1000 a oblasť záveru evolučného procesu.

Tabuľka 8.13 prezentuje výsledky testovania najlepších získaných riešení. Testovanie prebiehalo na množine 2 500 náhodne generovaných konfigurácií CA, avšak každá z konfigurácií bola použitá celkom dvakrát — hradlo NOT má totiž jeden vstupný parameter, a preto bola každá z konfigurácií preverená na všetkých možných kombináciach tohto vstupného parametru, preto bol reálny počet testovacích konfigurácií celkom 5 000.

Môžeme pozorovať, že výraznejší rozdiel medzi evolvovanou fitness a fitness testovacou zaznamenali štandardné metódy GA a CP. Metódy CE a CEEx dosiahli fakticky rovnako kvalitné výsledky pri testovaní, ako pri evolvovaní. Týmto sa opäť potvrdzuje domnienka vyslovená v experimentoch s úlohou hradla AND — riešenia z metód CE a CEEx sú kvalitnejšie, keďže sú schopné udržať si pri testovaní na neznámych konfiguráciach rovnako kvalitné výsledky ako počas evolučného procesu.

Tabuľka 8.14 potvrdzuje vyrovnanosť riešení medzi metódami GA a CP a medzi metódami CE a CEEx — jednotlivé dvojice dosiahli de-facto rovnaké priemerné výsledky. Výraznejšiu odchýlku zaznamenala len metóda GA a to v priemernom počte generácií, ktorý je približne dvojnásobne vyšší ako u všetkých ostatných metód.

Celkovo dosiahla v úlohe znovu-objavenia štruktúry logického hradla NOT najlepšie výsledky metóda CEEx, i keď rozdiel oproti CE je zanedbateľne malý. Avšak obe nové

Evolučná metóda	Priem. fitness	Priem. generácia	Smerodatná odchýlka
GA	0.670077	68944.2	0.01711
CP	0.679788	33143.8	0.00774
CE	0.800385	32161.2	0.01006
CEEx	0.809519	34231.0	0.01267

Tabulka 8.14: Priemerné výsledky dosiahnuté pomocou jednotlivých metód pri riešení úlohy logického hradla NOT. Výsledky odpovedajú priemeru 100 nezávislých behov evolúcie.

metódy dosiahli o približne 13 % kvalitnejšie výsledky ako metódy GA a CP.

Hradlo NOT má teda jednoduchšiu štruktúru, na základe čoho sme predpokladali kvalitnejší výsledok oproti hradlu AND s rovnako nastavenými parametrami evolučného procesu. Fakt, že hradlo NOT nedosiahlo lepšie výsledky teda môžeme prisúdiť práve tomu, že v neužitočných bunkách hradla sme vynucovali zachovanie pôvodného stavu bunky.

Môžeme teda konštatovať, že v prípade úloh, ktoré vyžadujú, aby sa evolučný proces zameral na určitú dôležitú časť celulárneho automatu (čo sú v tomto prípade užitočné bunky hradla), zatiaľčo ostatné bunky automatu nie sú veľmi dôležité (avšak sú do evolučného procesu zahrnuté) nie je prínos rozšírenia "predikcie budúceho stavu okolitých buniek"veľmi výrazný.

V úlohe, kde sa evolúcia zamerala výhradne na dôležité bunky (to znamená úloha hradla AND) priniesla rozšírená metóda CEEx výrazne kvalitnejšie výsledky. Rovnako aj pri úlohe synchronizácie, kde sa evolučný proces zameral na celý automat (pretože všetky bunky boli dôležité) dosiahlo rozšírenie výrazný prínos k získaniu kvalitnejších výsledkov.

8.2.4 Experiment s hradlom OR

Na rozdiel od predchádzajúcich hradiel AND a NOT, vyžaduje hradlo OR, ako ho definoval Sipper v práci [12], plochu až 30 buniek automatu rozdelených do mriežky *6 x 5* buniek. Táto plocha obsahuje celkovo až 16 užitočných buniek:

- **Down Propagation**: 6 buniek,
- **Right Propagation**: 3 bunky,
- **Left Propagation**: 2 bunky,
- **NAND**: 3 bunky,
- **No Change**: 2 bunky *(vstupné parametre hradla "x" a "y").*

Štruktúrou najkomplikovanejšie hradlo, tvorené až 16 užitočnými bunkami, ktoré sú riadené až 5 rôznymi typmi prechodových funkcií. Pre svoju korektnú činnosť vyžaduje až tri bunky riadené funkciou typu NAND a celkovo 13 buniek riadených rôznymi typmi propagačných funkcií. Jedná sa o jediné hradlo, kde počet užitočných buniek tvorí viac ako polovicu celkového počtu buniek celulárneho automatu.

Zvyšných 14 buniek automatu by podľa Sippera malo byť riadených funkciou "No Change". V prípade hradla NOT sme ale úlohu upravili tak, že budeme vyžadovať vynulovanie stavu týchto buniek — cieľom teda bude dosiahnúť v týchto bunkách stav 0. Dôvod k tomuto rozhodnutiu je pomerne jednoduchý — prechodová funkcia, ktorej jedinou

úlohou je (bez ohľadu na konfiguráciu buniek celulárneho okolia) vynulovať stav aktuálnej bunky, bude mať výrazne jednoduchšiu štruktúru ako funkcia, ktorá musí stav bunky zachovať. V tomto prípade sme teda evolučnému procesu uľahčili úlohu.

Pri výpočte fitness bol väčší dôraz kladený na užitočné bunky hradla a to tak, že fitness užitočnej bunky má 4-krát väčšiu váhu, ako fitness bunky neužitočnej.

Na základe týchto faktov môžeme usúdiť, že sa jedná o absolútne najnáročnejšiu úlohu riešenú v rámci tejto práce. Môžeme teda očakávať, že kvalita výsledkov nedosiahne na žiadnu z predchádzajúcich úloh. S ohľadom na závery experimentu s hradlom NOT môžeme ďalej očakávať, že prínos metódy CEEx proti CE nebude veľmi výrazný a neprinesie výrazne lepšie výsledky, ako sme videli v experimentoch s hradlom AND.

Výsledky experimentov

Tabuľka 8.15 dokumentuje najlepšie získané riešenia v experimentoch s logickým hradlom OR. Výsledky zobrazujú pomerne výrazný pokles kvality získaných riešení oproti experimentov s hradlami AND a OR. Týmto sa potvrdzujú predpoklady, že výrazne komplikovanejšia štruktúra hradla OR, ktorá bola zasadená do väčšieho automatu, bola pre evolučný proces výrazne komplikovanejšou úlohou. Na základe výsledkov s hradlom NOT, kde sme vynucovali zachovanie stavu neužitočných buniek hradla, čo viedlo k zhoršeniu kvality výsledkov, môžeme ďalej tvrdiť, že komplikovanosť hradla OR samotného v kombinácií s vynucuvaním stavu 0, tiež prispela k menej kvalitným výsledkom.

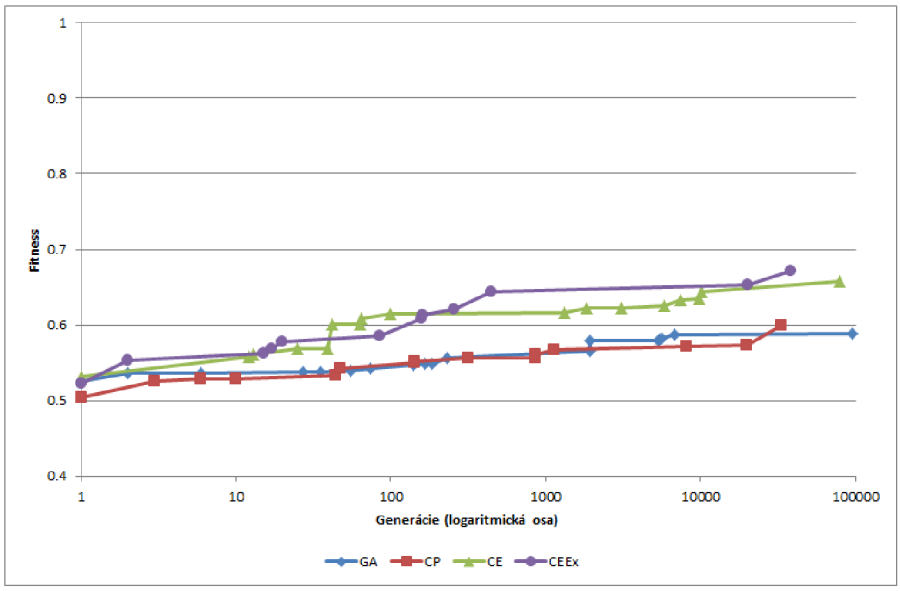
Evolučná metóda	Evolúcia		Testovanie
	Max. fitness	Generácia	Priemerná fitness
GA	0.588896	96307.0	0.555962
CP	0.59955	33561.0	0.588061
CE	0.656982	79200.0	0.640248
CEEx	0.670923	38067.0	0.671266

Tabulka 8.15: Najlepšie výsledky dosiahnuté pomocou jednotlivých metód pri riešení úlohy logického hradla OR. Stĺpec *Evolúcia* vyjadruje najlepšie hodnoty získané z evolučného procesu. Tieto najlepšie prechodové funkcie dosiahli pri preverovaní kvality fitness, ktorá je evedená v stĺpci *Testovanie*, pričom testy prebehli na celkovo 10 000 náhodne vygenerovaných počiatočných konfiguráciach CA.

Rovnako ako pri hradle NOT môžeme pozorovať minimálne rozdiely v kvalite riešení získanými z metód CE a CEEx rozdiel je opäť necelé 2 %. Rovnako skončila dvojica metód GA a CP. Rozdiel medzi novými metódami oproti metódam GA a CP je približne 15 % — v tomto prípade je rozdiel väčší ako v úlohe hradla NOT. Toto je dané tým, že štruktúra hradla OR je výrazne komplikovanejšia, čo pomohlo novým metódam vďaka ich unikátnej štruktúre prechodovej funkcie.

V grafoch priebehu fitness na obrázku 8.8 môžeme vidieť paralelu s grafmi z experimentovania s hradlom NOT. V oboch prípadoch pozorujeme, že metódy GA a CP dosahujú počas celého evolučného procesu veľmi podobné výsledky, a rovnako aj v prípade dvojice metód CE a CEEx.

Rovnako ako v experimentoch s hradlami AND a NOT, aj pri hradle OR konštatujeme, že testovanie najlepších riešení na množine náhodných konfigurácií lepšie zvládli riešenia získané z metód CE a CEEx, u ktorých nebol zaznamenaný v podstate žiadny pokles hodnoty fitness, ako dokladá tabuľka 8.15.



Obrázek 8.8: Grafy na obrázku zachytávajú priebeh rastu fitness jednotlivých metód v priebehu generácií evolučného procesu. Zobrazené grafy odpovedajú evolúcií najlepších prechodových funkcií jednotlivých metód, ktorých parametre prezentuje tabuľka 8.15.

Evolučná metóda	Priem. fitness	Priem. generácia	Smerodatná odchýlka
GA	0.57986	39636.0	0.00796
CP	0.591054	34927.2	0.00546
CE	0.649527	60488.8	0.00425
CEEx	0.65241	38944.0	0.01114

Tabulka 8.16: Priemerné výsledky dosiahnuté pomocou jednotlivých metód pri riešení úlohy logického hradla OR. Výsledky odpovedajú priemeru 100 nezávislých behov evolúcie.

Najlepšie získané výsledky sa od priemerných, ktoré zobrazuje tabuľka 8.16 líšia len minimálne — môžeme teda konštatovať, že úloha znovu-objavenia hradla OR bola pre evolučné procesy veľmi komplikovanou úlohou. Evolučné procesy totiž neboli v danom počte generácií schopné získať výrazne lepšie riešenia od priemerných výsledkov. Tento jav sme pozorovali vo všetkých troch experimentoch s logickými hradlami, a je výrazne odlišný od konštatovania z úlohy synchronizácie, kde najmä metóda CEEx bola schopná v závere evolučného procesu výrazne zvýšiť hodnotu fitness v pomerne malom počte generácií.

8.2.5 Diskusia

Táto sekcia bola venovaná experimentom zamerným na znovu-objavenie štruktúr logických hradiel AND, NOT a OR v prostredí neuniformného celulárneho automatu. Cieľom týchto experimentov bolo preveriť schopnosti jednotlivých metód, pri hľadaní komplexnej štruktúry logického hradla s pomerne sofistikovaným chovaním. Jednotlivé metódy mali zamerať svoju pozornosť na tie časti celulárneho automatu, ktoré obsahovali tzv. "užitočné"bunky hradla — teda bunky, ktoré sa na výpočte prebiehajúcom v hradle priamo podieľajú. Práve v týchto bunkách bolo potrebné objaviť zmienené sofistikované lokálne prechodové funkcie, zaisťujúce rôzne typy propagácie hodnôt z buniek celulárneho okolia, prípadne funkcie

počítajúce logickú funkciu NAND.

Prvé experimenty prebehli na logickom hradle AND, ktorého štruktúra vyžaduje automat o rozmere *5 x 5* buniek a celkovo 6 krokov vývoja automatu na dokončenie výpočtu v hradle. Hradlo je v tomto prípade tvorené 8 užitočnými bunkami, stavy (a teda ani prechodové funkcie) zvyšných buniek neboli prané v úvahu.

Výsledky ukázali prevahu metód CE a CEEx nad štandardnými metódami GA a CP — riešenie najlepšej metódy CEEx dosiahlo prevahu až 19 % nad metódou CP. Podobne ako pri experimentoch s úlohou synchronizácie, aj pri logickom hradle AND skončila metóda CE medzi algoritmami CEEx a CP — dosiahla o približne 5 % horšie výsledky ako vedúca metóda CEEx. Prekvapujúce sú výsledky z GA a CP — metóda CP, ktorá principiálne vychádza z GA a je zameraná na neuniformné automaty, dosiahla horšie výsledky ako neuniformná alternatíva genetického algoritmu.

Druhá sada experimentov sa zamerala na logické hradlo NOT. Rozborom štruktúry tohto hradla sme konštatovali, že je jednoduchšie ako hradlo AND, a teda by mala evolúcia vo všetkých metódach získať kvalitnejšie výsledky. Tento experiment sme však doplnili požiadavkou, aby si všetky neužitočné bunky hradla zachovali svoj pôvodný stav — týmto sme zvýšili nároky kladené na evolučné procesy, ktoré sa okrem užitočných buniek hradla museli zamerať aj na bunky ostatné.

Vďaka doplnovej požiadavke na stav neužitočných buniek, dosiahli všetky metódy podľa očakávania horšie výsledky oproti hradlu AND. Tým, že sa evolučný proces musel zamerať na všetkých 25 buniek automatu (hoci bunky užitočné boli preferované vďaka váhovaniu fitness), nebola schopná žiadna z evolučných metód získať lepšie riešenia, než aké objavili pri hradle AND.

Experimenty s hradlom NOT však objavili veľmi zaujimavé správanie sa jednotlivých metód — za prvé došlo k výraznej separácií fitness hodnôt — metódy CE a CEEx mali takmer počas celého evolučného vývoja výrazne lepšiu fitness ako GA a CP. Pritom fitness CE a CEEx boli počas celého vývoja temer totožné, rovnaké konštatovanie platí pre dvojicu algoritmov GA a CP (viď obrázok 8.7).

Ďalej sme konštatovali, že zadanie úlohy s váhovanou fitness (teda úloha, kde časť buniek je dôležitejšia a pri výpočte fitness má väčšiu váhu, zatiaľčo zvyšok buniek až tak veľmi dôležitý nie je, a teda má menšiu váhu fitness), takmer úplne zmazalo výhody rozšírenia "predikcie budúceho stavu okolitých buniek", ktoré sme pozorovali pri iných úlohách — metóda CEEx dosiahla oproti CE len o 1 či 2 % lepšie výsledky. Pri úlohách, kde sa evolučný proces CEEx sústredil výhradne na dôležité bunky automatu, bola metóda CEEx schopná priniesť výrazne kvalitnejšie riešenia oproti CE.

Posledné experimenty prebehli s hradlom typu OR, ktoré má výrazne komplikovanejšiu štruktúru ako AND a NOT a jeho užitočné bunky vyžadujú automat s väčšou plochou — až *6 x 5* buniek. Podobne ako v úlohe hradla NOT sme definovali, že evolúcia má vynútiť v neužitočných bunkách automatu stav 0, čím sme chceli potvrdiť závery výsledkov predchádzajúceho experimentu.

Výsledky získané s hradlom OR potvrdili konštatovanie objavené pri analýze výsledkov hradla NOT — opäť došlo k výraznej separácií priebehu fitness metód CE a CEEx oproti GA a CP, i keď rozdiel už nebol tak viditeľný.

Veľmi dôležité bolo potvrdenie predpokladu o strate prínosu rozšírenia "predikcie budúceho stavu okolitých buniek"v momente, keď evolučný proces musí rozdeliť svoje sily medzi

užitočné bunky s väčšou váhou fitness a bunky ostatné, ktoré majú na celkovej fitness váhu menšiu.

Metóda CEEx síce opäť získala najkvalitnejšie výsledky, ale rozdiel oproti CE bol opäť len 2 %. Navyše sme mohli pozorovať, že s narastajúcou komplikovanosťou zadania úlohy sa zmenšujú rozdiely vo výsledoch najlepšej metódy oproti metóde najhoršej. Zatiaľčo pri hradle AND bol rozdiel medzi CEEx a CP približne 19 % v prospech metódy CEEx, pri hradle NOT už len 13 % a konečne pri hradle OR získala metóda CEEx len o 7 % lepšie výsledky ako CP.

Pri pohľade na prezentované výsledky konštatujeme, že neexistuje jednoznačná odpoveď na otázku časovej efektivity porovnávaných metód. Experimenty s hradlom AND ukázali veľmi vyrovnané výsledky, čo sa počtu generácií týka — najlepšie skončila metóda GA, ktorá potrebovala o približne 10 % menší počet generácií ako CE či CEEx, avšak metóda CP si vyžiadala o takmer 10 % vyšší počet generácií oproti CE a CEEx. Hradlo NOT však ukázalo úplne vyrovnané nároky metód CP, CE a CEEx, zatiaľčo metóda GA potrebovala takmer dvojnásobný počet generácií. Pri hradle OR dosiahli vyrovnané výsledky metódy GA, CP a CEEx, zatiaľčo metóda CE potrebovala o približne 50 % generácií viac k získaniu najkvalitnejších riešení. Jedinou metódou, ktorá v žiadnom z prezentovaných experimentov nedosahovala výrazne horšie výsledky ako metódy ostatné, tak ostala CEEx.

Kapitola 9

Zhodnotenie vlastností novej metódy

V rámci tejto práce vznikli dve metódy návrhu celulárnych automatov, špeciálne zamerané na neuniformné automaty — metóda *celulárnej evolúcie* (CE) a jej alternatíva: *celulárna evolúcia s rozšírením "predikcie budúceho stavu okolitých buniek"* (CEEx). Obe tieto metódy boli porovnané na dvoch princiálne odlišných úlohach — na úlohe synchronizácie a na systéme logických hradiel — s bežne používanými evolučnými metódami genetického algoritmu (GA) a celulárneho programovania (CP).

9.1 Úloha synchronizácie

Úloha synchronizácie predstavuje príklad, v ktorom vyžadujeme vynútenie pomerne jednoduchého správania vo všetkých bunkách neuniformného celulárneho automatu. Úlohou evolučného procesu teda bolo nielen objaviť vhodnú prechodovú funkciu, ktorá zaistí požadované chovanie v jednej či v niekoľkých bunkách, ale aj rozšíriť dobré vlastnosti takýchto prechodových funkcií do celého automatu (napríklad pomocou evolučného operátoru kríženia).

Cieľom týchto pokusov nebolo len porovnanie jednotlivých metód v úlohe "globálneho vynucovania" určitého správania v celom prostredí CA, ale aj preverenie vplyvu počtu krokov vývoja CA, maximálneho počtu dostupných generácií evolučného procesu a veľkosti mriežky CA na kvalitu získaných výsledkov.

Pokusy v neuniformnom CA ukázali výrazne lepšie výsledky metód CE a CEEx oproti GA a CP. V prípade malej mriežky CA boli kvalitnejšie výsledky z týchto metód získavané takmer počas celého evolučného procesu. Na druhej strane vo veľkej mriežke automatu boli výsledky takmer počas celého evolučného procesu veľmi vyrovnané, ale záverečné fázy evolúcie ukázali zaujimavú schopnosť metód CE a najmä CEEx — získať vysoký prírastok fitness v pomerne malom počte generácií. Táto schopnosť metódy CEEx je daná výrazne väčším rozptylom fitness hodnôt od priemeru — a práve vďaka tomuto väčšiemu rozptylu je metóda CEEx schopná vygenerovať najlepšie riešenie zadaného problému, ktoré je výrazne lepšie ako priemer fitness hodnôt riešení, získaných zo všetkých nezávislých behov evolučného procesu.

Dvojnásobné zvýšenie počtu krokov vývoja CA (pri zachovaní všetkých parametrov evolučného procesu) prinieslo pozitívny efekt vo všetkých metódach. Analýzou výsledkov bolo konštatované, že všetky metódy ťažili zo zvýšenia počtu krokov približne rovnako.

Porovnaním výsledkov získaných z evolučných procesov ohraničených maximálne 1 000 generáciami oproti výsledkom ohraničených 10 000 generáciami ukázalo, že zväčšenie dostupného počtu generácií prinieslo najväčší úžitok metóde CEEx, ktorá bola schopná zo zvýšenia počtu generácií získať najväčší prírastok fitness hodnoty.

Pri experimentoch bolo zistené, že zvýšenie počtu buniek v mriežke CA má negatívne dopady na všetky neuniformné metódy. Najviac postihnutá bola metóda CP, ktorej výsledky sa po štvornásobnom zväčšení mriežky CA prepadli o temer 40 %, zatiaľčo metódy CE a CEEx zaznamenali zhoršenie len o 20 % (pri rovnakých parametroch evolučného procesu). Najmenší postih bol v tomto prípade zaznamenaný u metódy GA, a to o približne 10 %.

Vhodné použitie metód CE a CEEx

Celkovo môžeme túto sadu experimentov zhodnotiť tak, že prínos metód CE a CEEx je v úlohach, kde je vynucované určité globálne správanie vo všetkých bunkách neuniformného CA, značný — najmä metóda CEEx je schopná získať za všetkých okolností výrazne lepšie výsledky ako štandardné CP. CEEx je najmenej postihnutá, ak dôjde k zťaženiu podmienok (napríklad zväčšenie mriežky CA) a zároveň je schopná najviac získať, ak sa podmienky pre evolučný proces uľahčia.

V tomto type úloh je metóda CEEx najvhodnejšia najmä vtedy, keď je evolučný proces výrazne obmedzený — nízko nastavenou hranicou maximálneho počtu generácií a malým počtom krokov vývoja CA. Je teda zvlášť vhodný, keď v krátkom čase potrebujeme riešenie na pomerne malej ploche mriežky CA, ktoré by malo byť funkčné v minimálnom počte krokov vývoja CA — v týchto prípadoch generuje výrazne lepšie výsledky ako zvyšné metódy.

9.2 Úloha návrhu logických hradiel

Druhá sada experimentov sa zamerala na implementáciu systému logických hradiel v prostredí neuniformného celulárneho automatu. Cieľom tejto sady experimentov bolo znovuobjaviť konfiguráciu prechodových funkcií v bunkách neuniformného automatu tak, aby CA bol schopný simulovať činnosť logických hradiel AND, NOT a OR ako ich ručne navrhol Sipper. Jednotlivé hradlá boli ohraničené pomerne malou plochou CA a nízkym počtom krokov vývoja CA potrebných na získanie výsledku z daného hradla. V týchto experimetoch bolo úlohou evolučného procesu sústrediť sa na pomerne malý počet buniek a v nich objaviť relatívne špecifické prechodové funkcie.

Hradlá AND a NOT majú pomerne jednoduchú štruktúru — celkovo sú tvorené 8, respektíve 7 bunkami CA (tieto bunky sme nazvali "užitočnými" bunkami CA, keďže len tieto bunky sa podieľajú na výpočte hradla, zatiaľčo ostatné bunky CA nie sú pre činnosť hradla dôležité ani potrebné), pričom celá mriežka automatu má 25 buniek. Najkomplikovanejšiu štruktúru má jednoznačne hradlo typu OR, ktoré je tvorené 16 užitočnými bunkami rozloženými na ploche 30 buniek mriežky CA. V experimentoch s hradlom AND bolo úlohou evolučného procesu objavit výhradne konfiguráciu užitočných buniek hradla, zatiaľčo stavy a prechodové funkcie ostatných buniek neboli evolúciou vôbec brané do úvahy. Pri hradlách NOT a OR bolo úlohou evolučného procesu nielen získanie správnych prechodových funkcií užitočných buniek, ale evolúcia mala zaistiť aj vynulovanie stavu ostatných buniek automatu (hradlo OR), respektíve zachovať stav ostatných buniek bez zmeny (hradlo NOT), čím bola úloha výrazne náročnejšia ako pri hradle AND. Je dôležité poznamenať, že jednotlivé bunky sa na celkovej fitness hodnote podieľali s rôznymi váhami — fitness užitočných

buniek mala až štyri-krát vyššiu váhu.

Výsledky pokusov s hradlom AND potvrdili závery vyvodené v úlohe synchronizácie — metódy CE a CEEx boli schopné, na pomerne malej ploche mriežky CA, obmedzenej nízkym počtom krokov vývoja CA, získať výrazne kvalitnejšie výsledky ako GA a CP — rozdiel fitness medzi najlepšou metódou CEEx a najhoršou CP bol takmer 19 %. Výraznú diferenciu v kvalite si CEEx udržal takmer od začiatku evolučného vývoja až do jeho konca.

Experimenty s hradlami NOT a OR ukázali zaujimavý jav — veľmi výraznú separáciu metód CE a CEEx oproti metódam GA a CP. Jednotlivé dvojice (to znamená CE spolu s CEEx, a GA spolu s CP) metód mali takmer od začiatku evolučného procesu v podstate rovnakú hodnotu fitness. Zároveň dvojica CE a CEEx si oproti GA a CP takmer počas celého vývoja udržala veľmi veľký rozdiel fitness hodnôt. Tento jav je pripísaný tomu, že GA a CP majú lineárnu štruktúru chromozómu s jednoduchým mapovaním, zatiaľčo CE a CEEx používajú štrukturovaný chromozóm s pomerne komplikovaným mapovaním na fenotyp. Tento výrazne odlišný spôsob mapovania genotypu na fenotyp umožnil, že metódy CE a CEEx mali takmer počas celého evolučného vývoja o približne 15 % vyššiu fitness ako GA a CP.

Experimenty zároveň odhalili veľmi zaujimavé zistenie — pokiaľ je evolučný proces v metóde CEEx nútený optimalizovať bunky, ktoré sa na fitness podieľajú rôznou váhou, úplne sa stráca výhoda rozšírenia "predikcie budúceho stavu okolitých buniek". Tento fakt bol potvrdený v oboch experimentoch s váhovanou fitness, avšak neexistuje jednoznačné vysvetlenie tohto fenoménu. Je pravdepodobné, že rozšírené celulárne okolie metódy CEEx má negatívny vplyv v procese selekcie tých buniek, ktoré majú nižšiu váhu fitness (to znamená, že nejaká bunka s väčšiu váhou fitness ľahšie získa vysokú fitness, čo spôsobí, že bunka s menšou váhou preberie jej prechodovú funkciu, hoci je táto funkcia pre danú bunku absolútne nevhodná).

Vhodné použitie metód CE a CEEx

Celkovo môžeme tieto experimenty zhodnotiť tak, že metódy CE a najmä CEEx potvrdili závery z úlohy synchronizácie — v malej mriežke CA, s nízkym počtom krokov vývoja CA, ukazujú jednoznačnú dominanciu nad metódami GA a CP. Táto dominancia sa teda prejavila aj v úlohách, kde evolučný proces musel sústrediť svoje sily len na niekoľko dôležitých buniek mriežky CA (ktoré vyžadovali pomerne špecifické prechodové funkcie), zatiaľčo zvyšné bunky boli ignorované.

Je dôležité upozorniť, že metóda CEEx sa ukázala ako nevhodná, ak pracujeme s automatom, ktorý má váhovanú fitness — teda niektoré bunky sa podiaľajú na fitness s vyššou váhou ako iné. V takomto prípade sa absolútne stráca zmysel rozšírenia "predikcie budúceho stavu okolitých buniek"a je vhodnejšie použiť menej náročný algoritmus CE.

9.3 Implementačná náročnosť

Metódy CE a CEEx sú implementačne mierne náročnejšie ako štandardné metódy GA a CP. Metóda CEEx navyše pridáva rozšírenie oproti algoritmu CE, čím sa implementačne stáva najnáročnejšou metódou.

Najvýraznejší rozdiel je v spôsobe mapovania genotypu na fenotyp, kde metódy GA a CP obvykle používajú lineárny chromozóm a jednoduché mapovanie, v ktorom lokus alely génu je daný jednoduchým a jednoznačným výpočtom na základe stavov buniek celulárneho okolia. Tento spôsob mapovania bol implementovaný aj v tejto práci.

Metódy CE a CEEx používajú chromozóm vybudovaný z postupnosti štrukturovaných génov, kde jednotlivé komponenty aktivačnej jednotky génu mapujú podmienku z množiny podmienkových výrazov. Aby bol gén aktívny, musia byť splnené podmienky všetkých komponént aktivačnej jednotky génu. V opačnom prípade je gén neaktívny a bude sa vyhodnocovať aktivita nasledujúceho génu postupnosti.

Metóda CEEx zaviedla rozšírenie "predikcie budúceho stavu okolitých buniek", ktoré vo väčšine prípadov dokáže generovať spomedzi všetkých metód, uvažovaných v tejto práci, jednoznačne najlepšie výsledky. Samotné rozšírenie je implementačne veľmi jednoduché — pri výpočte nového stavu bunky *Cell* sa zároveň vypočítajú aj budúce stavy buniek z jej celulárneho okolia, ale pomocou prechodovej funkcie náležiacej bunke *Cell*, tak ako to ukazuje algoritmus 6.2. Získané "odhadované budúce"stavy sa následne váhujú, čím sa získa nový stav bunky *Cell*.

Obrovskou výhodou metódy CEEx je fakt, že rozšírenie "predikcie budúceho stavu okolitých buniek"je silne paralelizovateľné, vďaka čomu budú nároky na strojový čas v podstate rovnaké, ako u metódy CEEx.

Celkovo sú teda metódy CE a CEEx implementačne náročnejšie ako metódy GA a CP, avšak zvýšená náročnosť implementácie, spolu s výrazne odlišným princípom fungovania, so sebou priniesli výrazne lepšie výsledky.

Kapitola 10

Záver

Táto diplomová práca naväzuje na semestrálny projekt, v rámci ktorého boli preskúmané teoretické podklady tejto práce — celulárne automaty, evolučné algoritmy (z nich predovšetkým genetický algoritmus) a celulárne programovanie. Semestrálny projekt zahŕňal aj výber dvoch vhodných úloh — úlohy synchronizácie a systému implementácie logických hradiel — pomocou ktorých boli zrovnané schopnosti jednotlivých algoritmov.

Celulárny automat predstavuje základny aspekt — jadro — tejto práce, a preto boli preštudované nielen jeho vlastnosti a história, ale predovšetkým boli preskúmané možnosti implementácie rôznych štruktúr, ktoré výpočtový model CA poskytuje. Na základe tohto skúmania bolo rozhodnuté, že v rámci tejto práce bude najvhodnejším modelom neuniformný, dvojrozmerný, dvojstavový celulárny automat, ktorý poskytuje dostatočnú vyjadrovaciu silu na pokrytie vybraných úloh, a zároveň nie je výpočtovo veľmi náročný.

Keďže bol v tejto práci kladený dôraz na možnosti "automatizovaného"návrhu celulárnych automatov, boli ako najvhodnejšie metódy návrhu CA zvolené evolučné metódy. Z nich bolo najväčšie zameranie kladené na dva principálne rôzne prístupy k aplikácií evolučných prístupov — na genetické algoritmy a na celulárne programovovanie.

Obe tieto evolučné metódy návrhu celulárnych automatov boli implementované do prostredia neuniformného automatu, kde boli prevedené prvé experimenty s evolučným riešením vybraných úloh — teda úlohy synchronizácie a systému logických hradiel.

Cieľom práce bolo vytvoriť vlastný prístup návrhu CA, preto bola, na základe získaných výsledkov, navrhnutá a implementovaná metóda *celulárnej evolúcie*, ktorá principiálne vychádza z algoritmu predstaveného Bidlom [1]. Metóda celulárnej evolúcie v tejto diplomovej práci však bola špeciálne zameraná na návrh neuniformných celulárnych automatov.

Najdôležitejším prínosom tejto diplomovej práce je rozšírenie celulárnej evolúcie nazvané "*predikcia budúceho stavu okolitých buniek*", ktoré ďalej rozšírilo schopnosti celulárnej evolúcie. Toto rozšírenie je navyše natoľko obecné, že je možné ho pomerne ľahko implementovať aj v prostredí iných automatov a iných evolučných metód. Celkovo teda v rámci tejto práce vznikli štyri aplikácie — pre genetický algoritmus, celulárne programovanie, celulárnu evolúciu a celulárnu evolúciu s rozšírením predikcie budúceho stavu okolitých buniek.

S jednotlivými metódami bolo prevedených niekoľko sád experimentov, ktorých primárnym účelom bolo porovnať schopnosti novo-navrhnutých metód oproti metódam štandardným. Výsledky boli zhrnuté v prehľadných tabuľkách a grafoch s podrobným vysvetlením získaných výsledkov. Na základe týchto výsledkov boli vyvodené vlastnosti týchto nových metód, ich silné a slabé stránky.

Literatura

[1] BIDLO, M.; VAŠÍČEK, Z.: Evolution of Cellular Automata with Conditionally Matching Rules. In *Proceedings of the 2013 IEEE Congress on Evolutionary Computation*, IEEE Computer Society, 2013 (v tisku).

[2] DARWIN, C.: *On the Origin of Species by Means of Natural Selection*. Dover Publications, 2006, ISBN 978-0486450063.

[3] FOGEL, L. J.; OWENS, A. J.; WALSH, M. J.: *Artificial Intelligence through Simulated Evolution*. New York: Wiley, 1966.

[4] GARDNER, M.: Mathematical games: The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, , č. 223, Október 1970: s. 120–123.
URL http://www.ibiblio.org/lifepatterns/october1970.html

[5] HOLLAND, J. H.: *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press, 1975.

[6] HYNEK, J.: *Genetické algoritmy a genetické programování*. Praha: Grada, 2008, ISBN 978-80-247-2695-3.

[7] KOZA, J. R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Boston: The MIT Press, 1992.

[8] KUMAR, S.; BENTLEY, P.: *On growth, form and computers*. Boston: Elsevier Press, 2003, ISBN 0-12-428765-4.

[9] KVASNIČKA, V.; POSPÍCHAL, J.; TIŇO, P.: *Evolučné Algoritmy*. Bratislava: Slovenská Technická Univerzita Bratislava, 2000, ISBN 80-227-1377-5.

[10] NIERHAUS, G.: *Algorithmic Composition: Paradigms of Automated Music Generation*. Vienna: Springer, 2008, ISBN 978-3211755396.

[11] SCHWEFEL, H. P.: *Kybernetische evolution als strategie der experimentellen forschung in der stroemungstechnik*. Diplomová práce, Technische Universitaet Berlin, Berlin, 1965.

[12] SIPPER, M.: *Evolution of Parallel Cellular Machines: The Cellular Programming Approach*. Heidelberg: Springer, 1997, ISBN 978-3540626138.
URL http://www.moshesipper.com/pcm/

[13] WOLFRAM, S.: *A New Kind of Scince*. Champaign: Wolfram Media, 2002, ISBN 1-57955-008-8.
URL http://www.wolframscience.com/nksonline/toc.html

Příloha A

Obsah CD

Priložené CD obsahuje:

/app/bin/CellularEvolution.exe	spustiteľný súbor aplikácie *celulárnej evolúcie* určený pre systém Windows
/app/bin/CellularEvolutionEx.exe	spustiteľný súbor aplikácie *celulárnej evolúcie s rozšírením "predikcie budúceho stavu okolitých buniek"* určený pre systém Windows
/app/bin/CellularProgramming.exe	spustiteľný súbor aplikácie *celulárneho programovania* určený pre systém Windows
/app/bin/GeneticAlgorithm.exe	spustiteľný súbor aplikácie *genetického algoritmu* určený pre systém Windows
/app/symbols/	symboly určené k ladeniu priložených aplikácií
/app/src/DIP_2012.sln	hlavný projektový súbor zdrojových kódov určený pre *Microsoft Visual Studio 2012*
/app/src/DIP_2010.sln	hlavný projektový súbor zdrojových kódov určený pre *Microsoft Visual Studio 2010*
/app/src/CellularEvolution/	zdrojové súbory aplikácie *celulárnej evolúcie*
/app/src/CellularEvolutionEx/	zdrojové súbory aplikácie *celulárnej evolúcie s rozšírením "predikcie budúceho stavu okolitých buniek"*
/app/src/CellularProgramming/	zdrojové súbory aplikácie *celulárneho programovania*
/app/src/GeneticAlgorithm/	zdrojové súbory aplikácie *genetického algoritmu*
/app/src/SharedSources/	zdrojové súbory zdieľané medzi aplikáciami
/app/src/sources.zip	zdrojové súbory všetkých aplikácií skomprimované do zip súboru
/app/doc/	programová dokumentácia
/text/thesis-ing.pdf	text diplomovej práce
/text/src/tex/	zdrojové súbory textu diplomovej práce (LaTeX)
/text/src/thesis-ing.zip	zdrojové súbory textu skomprimované do zip súboru

Příloha B

Návod k použitiu

V rámci tejto diplomovej práce vznikli celkom štyri aplikácie implementujúce jednotlivé metódy návrhu celulárnych automatov:

- **GeneticAlgorithm** — implementuje uniformný i neuniformný genetický algoritmus,
- **CellularProgramming** — neuniformná metóda celulárneho programovania,
- **CellularEvolution** — implementuje neuniformnú metódu celulárnej evolúcie,
- **CellularEvolutionEx** — neuniformná metóda celulárnej evolúcie s rozšírením "predikcie budúceho stavu okolitých buniek".

Všetky štyri aplikácie sú konzolové, a boli implementované výhradne pre prostredie operačného systému *Microsoft Windows*.

B.1 Kompilácia aplikácií

Všetky štyri aplikácie boli naprogramované vo vývojovom prostredí *Microsoft Visual Studio 2012*, konkrétne v programovacom jazyku *Microsoft Visual C++*.

CD priložené k textu práce obsahuje adresár */app/src/*, v ktorom sa nachádzajú všetky zdrojové kódy vytvorené v rámci tejto diplomovej práce. Každá z aplikácií sa nachádza vo svojom vlastnom adresári, zdrojové kódy, ktoré sú medzi aplikáciami zdieľané sa nachádzajú vo zvláštom adresari (viď adresárovú štruktúru CD popísanú v predchádzajúcej kapitole). Ďalej v tomto adresári nájdeme hlavný projektový súbor [1] určený na otvorenie celého projektu (to znamená na otvorenie zdrojových kódov všetkých štyroch aplikácií) v prostredí *Microsoft Visual Studio*. Jedná sa o súbor *DIP_2012.sln*, ktorý je určený pre prostredie *Microsoft Visual Studio 2012* a súbor *DIP_2010.sln*, určený na otvorenie projektu v *Microsoft Visual Studio 2010*.

Zdrojové kódy aplikácií otvoríme v prostredí *Microsoft Visual Studio* [2] práve pomocou projektového súboru, a to buď poklepaním na projektový súbor, alebo jeho otvorením cez dialóg *Súbor > Otvoriť*, priamo z aplikácie *Microsoft Visual Studio*.

[1] V anglickej jazykovej mutácií prostredia *Microsoft Visual Studio* sa *Hlavný projektový súbor* označuje ako *Solution*. V ďalšom texte bude používaný tento výraz.

[2] Pred samotným otvorením hlavného projektového súboru je nevyhnutné skopírovať kompletný obsah adresáru */app/src/* na diskovú lokalitu, ktorá umožňuje zápis.

Po otvorení hlavného projektového súboru sa v okne aplikácie *Microsoft Visual Studio* zobrazí zoznam jednotlivých aplikácií v prehľadnej stromovej štruktúre, ktorú môžeme prechádzať, a v ktorej nájdeme všetky zdrojové súbory jednotlivých aplikácií.

Samotné skompilovanie projektov prevedieme výberom príslušného projektu (prípadne výberom hlavného projektového súboru) a v menu *Kompilovať / Build* vyberieme príslušný príkaz na preklad vybraného projektu, alebo na preklad celého *Solution*.

Po úspešnom skompilovaní nájdeme výsledné binárne súbory v adresári */app/src/Build/Bin/<platforma>/<konfigurácia>/*, kde *<platforma>* značí jednu z možností *win32|x64* (to znamená binárne súbory buď pre 32 alebo pre 64 bitovú platformu) a *<konfigurácia>* značí jedno z *Release|Debug* (teda binárne súbory určené pre ladenie alebo pre nasadenie).

B.2 Podporované parametre aplikácií

Všetky štyri aplikácie sú nastaviteľné pomocou voliteľných parametrov príkazovej riadky:

- *<Application.exe>*$[-task\ TASK_{TYPE}]$ $[-rep\ REPETITIONS]$
  $[-gen\ GENERATIONS]$ $[-pop\ POPULATION]$
  $[-step\ CA_{STEPS}]$ $[-width\ CA_{WIDTH}]$ $[-height\ CA_{HEIGHT}]$
  $[-uniform\ UNIFORMITY]$.

Jednotlivé parametre majú nasledujúci význam a môžu nadobúdať hodnoty, ktoré sú uvedené pod vysvetleniami významu jednotlivých parametrov:

- parameter: **-task** — slúži na výber úlohy, ktorá má byť spracovaná:
  - **SYNC** — bude riešená úloha synchronizácia,
  - **AND** — úloha logického hradla AND, vyžaduje minimálne rozmery mriežky *5 x 5 buniek* a minimálne *6 krokov* vývoja CA,
  - **OR** — úloha logického hradla OR, vyžaduje minimálne rozmery mriežky *6 x 5 buniek* a minimálne *8 krokov* vývoja CA,
  - **NOT** — úloha logického hradla NOT, vyžaduje minimálne rozmery mriežky *5 x 5 buniek* a minimálne *6 krokov* vývoja CA,
  - *Predefinovaná hodnota:* **SYNC**.
- parameter: **-rep** — nastaví počet nezávislých behov evolučného procesu:
  - celočíselná hodnota v rozmedzí **1** − **100**,
  - *Predefinovaná hodnota:* **10**.
- parameter: **-gen** — maximálny počet generácií evolučného procesu
  - celočíselná hodnota v rozmedzí **10** − **10 000 000**,
  - *Predefinovaná hodnota:* **10 000**.
- parameter: **-pop** — veľkosť populácie v chápaní danej metódy
  - celočíselná hodnota v rozmedzí **3** − **1 000**,
  - *Predefinovaná hodnota:* **20**.

- parameter: **-step** — počet krokov vývoja celulárneho automatu
  - celočíselná hodnota v rozmedzí **1** − **20**,
  - *Predefinovaná hodnota:* **6**.
- parameter: **-width** — šírka (v počte buniek) mriežky celulárneho automatu
  - celočíselná hodnota v rozmedzí **1** − **20**,
  - *Predefinovaná hodnota:* **5**.
- parameter: **-height** — výška (v počte buniek) mriežky celulárneho automatu
  - celočíselná hodnota v rozmedzí **1** − **20**,
  - *Predefinovaná hodnota:* **5**.
- parameter: **-uniform** — prepínač nastaví režim automatu (uniformný, neuniformný). Je implementovaný len v metóde *GA*, ostatné metódy pracujú výhradne s neuniformným automatom. Podporované hodnoty:
  - **YES** — automat bude pracovať ako uniformný,
  - **NO** — automat bude neuniformný,
  - *Predefinovaná hodnota:* **YES**.
- parameter: **-help** — zobrazí nápovedu k aplikácií.

Príklad spustenia aplikácie:
CellularEvolutionEx.exe -task SYNC -rep 5 -gen 1000 -pop 25 -step 8 -width 4 -height 4.

B.3 Export výsledkov

Všetky štyri aplikácie zapisujú výsledky výhradne na štandardný textový výstup *std::out*, teda do konzolového okna z ktorého bola aplikácia spustená [3]. Pokiaľ chceme zapísať výstup aplikácie do textového súboru, je potrebné presmerovať výstup aplikácie do príslušného súboru, napríklad: *CellularEvolutionEx.exe -task SYNC >output.txt.*

[3] Pre korektné zobrazenie výsledkov odporúčame šírku konzolového okna aspoň 150 znakov.