Katedra Aplikovanej Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Celulárne automaty s nepravidelnou mriežkou

(Diplomová práca)

Bc. Péter Dobsa

Vedúci: Mgr. Pavel Petrovič, PhD.

Bratislava, 2016

Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Celulárne automaty s nepravidelnou mriežkou

Diplomová práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	2511 Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školitel:	Mgr. Pavel Petrovič, PhD.

Bc. Péter Dobsa
Bratislava, 2016

74396312

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Péter Dobsa
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.9. aplikovaná informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Celulárne automaty s nepravidelnou mriežkou
Cellular Automata with Irregular Mesh

Cieľ: Práca poskytne prehľad existujúcich prístupov k tvorbe celulárnych automatov (CA) s nepravidelnou mriežkou. Zameria sa na použitie CA ako reprezentácie tvarov pre evolučný design. CA budú v práci vytvárané využitím evolučných algoritmov (EA). Študent navrhne vlastné metódy (alebo upraví existujúce) pre vytváranie CA s premenlivou štruktúrou pomocou EA. Vytvorené CA požadovaný tvar dosiahnu iteráciou z počiatočného stavu (embrya) do finálneho fenotypu (zodpovedajúceho požadovaného tvaru). Vytvorené metódy budú experimentálne overené a vyhodnotené.

Literatúra:
Beatens J.M., De Baets, B.: Phenomenological study of irregular cellular automata based on Lyapunov exponents and Jacobians, Chaos, 20, 2010.
Mitchell M.: Computation in Cellular Automata: A Selected Review. In T. Gramss, S. Bornholdt, M. Gross, M. Mitchell, and T. Pellizzari, Nonstandard Computation, pp. 95–140. Weinheim: VCH Verlagsgesellschaft, 1998.
Mitchell M., Crutchfield J.P., Rajarshi D.: Evolving Cellular Automata with Genetic Algorithms: A Review of Recent Work, EuCA'96.

Kľúčové slová: celulárne automaty, evolučný design, nepravidelná mriežka

Vedúci: Mgr. Pavel Petrovič, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. PhDr. Ján Rybár, PhD.

Dátum zadania: 05.10.2014

Dátum schválenia: 26.11.2014

prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

Dobsa
.....
študent

Pavel Petrovič
.....
vedúci práce

Čestne vyhlasujem, že som túto diplomovú prácu vypracoval samostatne s použitím citovaných zdrojov.

. . . . . .

Poďakovanie

Touto cestou by som chcel poďakovať vedúcemu práce Mgr. Pavlovi Petrovičovi, PhD. za konzultácie, obetavý čas, cenné rady a pripomienky počas tvorby práce.

Abstrakt

Diplomová práca je venovaná evolučnému dizajnu používajúcemu celulárne automaty s neregulárnou mriežkou ako reprezentáciu na genetickej úrovni a evolučný algoritmus na spojitú optimalizáciu CMA-ES. Poskytuje prehľad existujúcich prác a výsledkov v oblasti. Jej cieľom je implementácia modulárneho rámcového prostredia na experimentáciu s použitým modelom, ďalej návrh a vyhodnotenie nových postupov. Hlavnými príspevkami diela sú algoritmy na využitie geometrických štruktúr buniek v tvare obdĺžníkov a Voronoiových regiónov, ďalej vylepšená metóda na riadenie topologických zmien počas iterácie automatu.

Kľúčové slová: celulárne automaty, evolučný dizajn, CMA-ES, nepravidelná mriežka

Abstract

This diploma thesis investigates evolutionary design using irregular cellular representation on the genetic level and the state of the art evolutionary algorithm for continuous optimization CMA-ES. It provides an overview of the existing research and results of the field. It's goal is to implement a modular framework for experimentation with the model as well as to propose and evaluate novel approaches. The main contributions of this work are the algorithms using rectangular shapes and Voronoi regions as geometric structures of the cells, furthermore an improved method to handle topologic updates during the iteration of the automaton.

Keywords: cellular automata, evolutionary design, CMA-ES, irregular mesh

Obsah

1 Úvod **1**

2 Teoretické pozadie **3**

2.1 Celulárne automaty . . . 3

2.1.1 Vlastnosti celulárnych automatov . . . 4

2.1.2 Známe celulárne automaty . . . 6

2.2 Evolučné algoritmy . . . 7

2.2.1 Komponenty evolučných algoritmov . . . 8

2.2.2 Reprezentácia . . . 11

2.2.3 Viacúčelová optimalizácia . . . 12

2.3 CMAES . . . 14

2.3.1 Vzorkovanie novej generácie . . . 14

2.3.2 Parametre a operátory . . . 14

2.3.3 Adaptácia kovariančnej matice . . . 16

2.4 Neurónové siete . . . 17

2.4.1 Perceptrón . . . 17

2.4.2 Viacvrstvový perceptrón . . . 18

2.4.3 Spôsoby učenia . . . 19

2.5 Voronoiov diagram . . . 19

2.5.1 Konštrukcia Voronoiovho diagramu . . . 20

3 Východiská **22**

3.1 Problém vlajky . . . 23

3.2 Organizmus . . . 23

3.3 Zastavenie vývoja . . . 24

3.4 Riadiaci prvok . . . 26

3.5 Reakčný-difúzny prístup . . . 27

3.6 Ďalšie práce . . . 29

4 Formulácia hypotéz **32**
4.1 Geometria buniek 32
4.1.1 Obdĺžníková reprezentácia 33
4.1.2 Voronoiovská reprezentácia 35
4.2 Viacúčelovosť 36
4.3 Ďalšie modifikácie 37

5 Softvér na realizáciu experimentov **40**
5.1 Technologické pozadie 40
5.2 Štruktúra programu 41
5.3 Konfiguračné nastavenia 42
5.3.1 Konfigurácia experimentu 42
5.3.2 Konfigurácia riadiacieho prvku 43
5.3.3 Konfigurácia automatu 44
5.3.4 Konfigurácia optimalizátora 45
5.3.5 Konfigurácia účelových funkcií 45
5.3.6 Konfigurácia kritéria ukončenia iterácie automatu 45

6 Experimenty a výsledky **47**
6.1 Experimenty 48
6.2 Inicializácia 48
6.3 Two-Bands 90 49
6.4 Chess 58
6.5 Two-Bands 45 61

7 Záver **65**

Literatúra **66**

Príloha **69**

KAPITOLA 1

Úvod

Prírodné javy a procesy boli odjakživa nevyčerpateľnou inšpiráciou v rôznych oblastiach ľudskej činnosti, okrem umeleckých disciplín a spoločenských vied ich vplyv našiel svoje uplatnenie aj v technických a inžinierskych smeroch, ako napríklad aj v informatike. Na základe prirodzenej evolúcie vzniklo veľké množstvo metód, známe ako *evolučné algoritmy (evolutionary algorithms)*, ktoré boli úspešne aplikované na riešenie výpočtovo ťažkých úloh a optimalizáciu takých procesov, pri ktorých klasické analytické prístupy zlyhávali.

Mladá, ale perspektívna a rýchlo sa rozvíjajúca podoblasť evolučných výpočtov je *evolučný dizajn (evolutionary design)*, ktorá sa zaoberá automatickým vytvorením komplexných štruktúr, ako napríklad robotické morfológie, topológie matematických objektov, ale aj objektov z reálneho sveta využitím evolučných algoritmov. Evolučný dizajn umožňuje dizajnérovi zjednodušené pátranie po riešení jedine špecifikáciou kritéria na kvalitu (obvykle vo forme funkcie, ktorá mapuje kandidáta na reálne číslo reprezentujúce jeho vhodnosť) a potenciálne preskúmať aj nečakané možnosti, ktoré by možno za normálnych okolností neprišli do úvahy, napriek tomu reprezentujú nádejné alternatívy.

Ako príklad použitia tejto paradigmy v praxi si môžeme uviesť dizajn štruktúry satelitnej antény v NASA [24], kde riešenie pomocou evolučných metód sa ukázalo byť kompaktnejšie a efektívnejšie ako ľudské dizajny, a ktoré nakoniec bolo zostrojené a poslané do vesmíru.

Predložená diplomová práca predstavuje drobný krok vo výskumno-experimentálnej činnosti, ktorá je venovaná evolučnému dizajnu z inovatívneho pohľadu: použitím celulárnych automatov ako reprezentáciu na genetickej úrovni. Práca silne buduje na sériu na seba nadväzujúcich akademických prác z predchádzajúcich rokov a jej cieľom je rozvíjanie v nich predstavených myšlienok, ďalej implementácia a

experimentálne vyhodnotenie nových postupov.
Práca je štrukturovaná do 7 kapitol nasledujúcim spôsobom:

- kapitola 2 zhrnie dôležité poznámky z oblasti a vybuduje teoretické východisko práce
- kapitola 3 predstavuje predchádzajúci výskum, do detailov popíše práce, ktoré tvoria jadro nášho diela
- popis použitej metodiky a vlastných návrhov na vyhodnotenie sa nachádza v kapitole 4
- implementačné detaily programu a rozpis konfiguračných parametrov sú uvedené v 5. kapitole
- výsledky experimentov spolu s diskusiou obsahuje kapitola 6
- v závere stručne zhrnieme prínosy práce spolu s možnými vylepšeniami

KAPITOLA 2

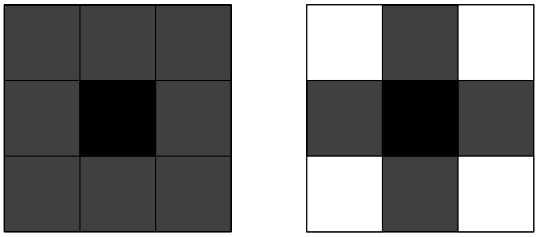
Teoretické pozadie

Nasledujúca kapitola slúži ako súhrn teoretického pozadia diplomovej práce, predstavuje a popíše výpočtové modely a algoritmy z oblasti evolučných výpočtov, umelých neurónovych sietí a celulárnych automatov dôležité na pochopenie problematiky.

2.1 Celulárne automaty

Celulárne automaty (*cellular automata, CA*) sú decentralizované dynamické systémy (t.j. systémy, ktorých aktuálny stav závisí od konečného počtu predchádzajúcich stavov, pričom zmena stavov je určená fixnými pravidlami) diskrétne v čase i v priestore.
Skladajú sa z množiny jednoduchých komponentov (*buniek*), ktoré sú obvykle identické - i keď táto práca sa zaoberá predovšetkým automatmi, ktorých topológia je neregulárna a teda jednotlivé bunky nemusia mať rovnakú geometrickú štruktúru - a i-tá bunka v každom časovom okamihu t je jednoznačne charakterizovaná stavom s_t^i z nejakej konečnej množiny možných stavov Σ, $|\Sigma| = k$, kde s_t^i je nejaká diskrétna veličina. Jednotlivé bunky sú navyše lokálne prepojené, okolie bunky i s ktorým je daná bunka prepojená (vrátane bunky i) sa volá *susedstvo* ζ_i (*neighbourhood*).
Štruktúra buniek je najčastejšie definovaná ako n-dimenzionálna mriežka, bežne $n = 2$, v ktorom sa obyčajne používa *von Neumann* alebo *Moore* susedstvo - prvé obsiahne ortogonálnych susedov, druhé okrem ortogonálnych aj zvyšných štyroch.
Aktualizácia stavu je riadená *prechodovým pravidlom* (*transition rule*) alebo *CA pravidlom*, definované ako funkcia $\phi : \zeta' \mapsto \Sigma$, kde ζ' je množina všetkých susedstiev v danom CA. $\phi(\zeta_i^t)$ pre bunku i vráti stav $s_i^{t+1} \in \Sigma$, teda správanie bunky je určené momentálnym stavom susedných buniek a vlastným stavom; bunky okrem toho žiadu pamäť nemajú. Prechodová funkcia je fixná a globálna, všetky bunky majú

tú istú prechodovú funkciu. Typicky, v každom časovom kroku t každá bunka i prepíše svoj stav synchrónne podľa $\phi(\zeta_i^t)$.

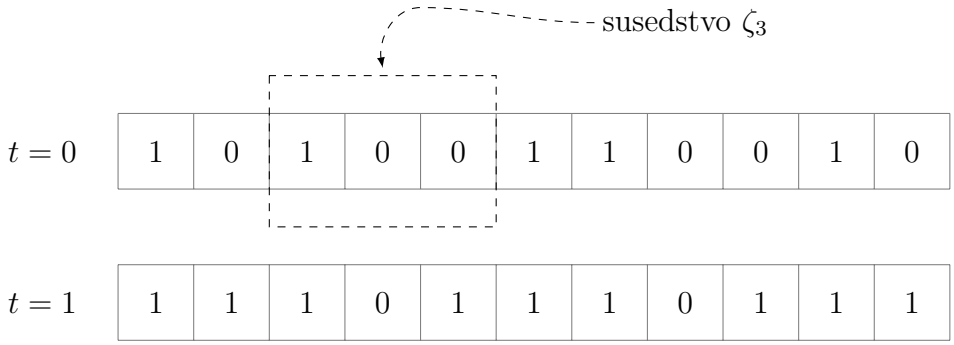


Obr. 2.1: Na ľavej strane je ukážka Moore susedstva (vyplnené tmavým), na pravej von Neumann susedstva. V oboch prípadoch je bunka na aktualizáciu čierna.

Môžeme teda celulárny automat formálne zadefinovať ako štvoricu $\langle L, S, \zeta, \phi \rangle$, kde L je množina buniek, S je množina všetkých možných stavov, ζ je množina susedstiev v danom automate a ϕ je prechodová funkcia.
Celulárne automaty majú potenciál na efektívne vykonanie zložitých výpočtov, tak isto ako aj na modelovanie správania komplexných systémov v prírode. Vďaka tomu celulárne automaty sú predmetom výskumu v matematike a informatike, boli použité na modelovanie fyzických a biologických javov - napr. prietok tekutín a tvorba biologických vzorov [22][17] -, ako paralelná platforma v simuláciách vedecko-technických modelov [9] alebo na spracovanie obrazu [27], ale aj na skúmanie *emergentného výpočtu* (*emergent computation*) v komplexných systémoch [12][13] - t.j. správanie systému, ktoré nie je explicitne zadefinované v elementárnych komponentoch, v prípade CA to znamená nejaké globálne koordinované spracovanie informácie, ktoré je výsledkom lokálnych interakcií buniek.
Celulárne automaty sú dôležité aj z hľadiska teoretickej informatiky. Patria do triedy výpočtových modelov s tzv. *ne-von Neumannovou architektúrou* (*non-von Neumann architecture*) charakterizovaná s výpočtom cez lokálne interakcie bez centrálnej riadiacej jednotky a pamäte - na rozdiel od štandardného modelu zahŕňajúci CPU, globálne prístupnú pamäť, sériové spracovanie, atď. nazvaný aj ako *von Neumannová architektúra* (*von Neumann architecture*).

2.1.1 Vlastnosti celulárnych automatov

Celulárne automaty môžu byť klasifikované podľa rôznych kritérií [28] - ako napr. topológia systému, typy susedstva a prechodových funkcií alebo citlivosti na inicializačné podmienky a náhodné šumy -, my si tu uvedieme heuristickú klasifikáciu Stephen Wolframa podľa typických správaní:



Tabuľka prechodových pravidiel:

susedstvo	000	001	010	011	100	101	110	111
výstup	0	1	1	1	0	1	1	0

Obr. 2.2: Ukážka aktualizácie stavu jednoduchého celulárneho automatu reprezentovaného ako jednorozmerné pole. Bunky môžu nadobúdať binárne hodnoty, t.j. $\Sigma = \{0, 1\}$, susedstvo ζ_i tvoria bunka i s ľavým a pravým susedom, pričom chýbajúci susedia na okrajoch sú doplnení nulami; prechodová funkcia (známa aj ako *"Rule 110"*) je zachytená v tabuľke hore.

- trieda 1) opakovaná aktualizácia (alebo *iterácia*) automatu vždy vedie do stabilného stavu celého automatu - t.j. keď všetky bunky sa dostali do stabilného stavu a žiadny počet ďalších aktualizacií ich stav nemení - a to nezávisle od inicializačných stavov
- trieda 2) opakovaná aktualizácia automatu vedie k periodickej konfigurácii - t.j. keď isté vzory sú pravidelne znova a znova vygenerované
- trieda 3) opakovaná aktualizácia automatu vedie k chaotickému alebo aperiodickému správaniu, teda žiadna periodickosť a stabilnosť nie je pozorovateľná vo vygenerovaných vzoroch
- trieda 4) opakovaná aktualizácia automatu vedie k štruktúre, ktorá generuje perzistentné, komplexné vzory, ktoré môžu navzájom interagovať nejakým zaujímavým spôsobom. Zriedkavo majú potenciál sa dostať do triedy 2) a oscilovať medzi istými vzormi, ale to zvyčajne vyžaduje veľký počet aktualizácií. Príklad v sekcii 2.1.2.

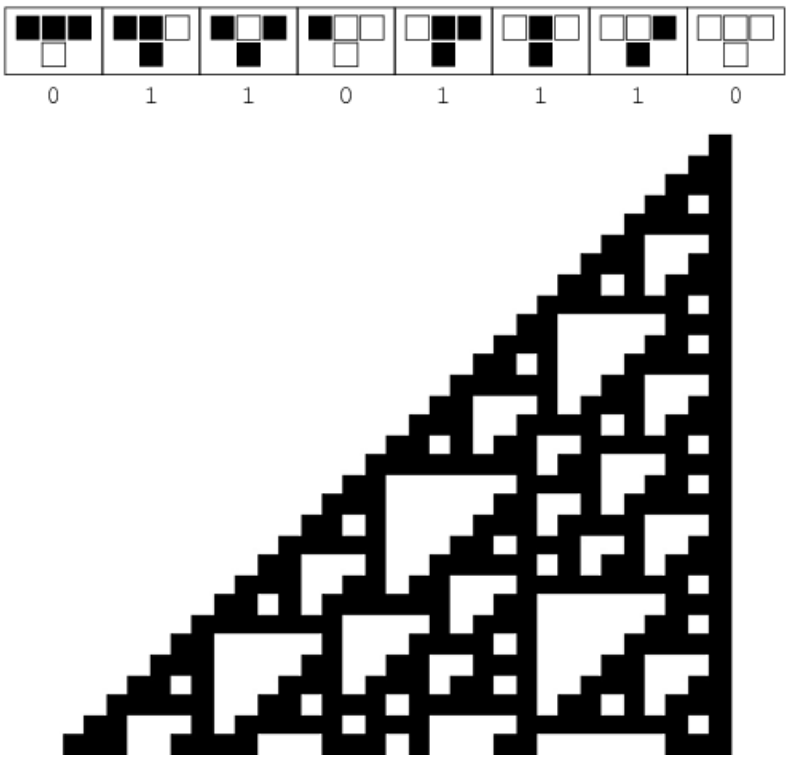
Celulárny automat je *reverzibilný* (*reversible*) ak stav systému v ľubovoľnom čase t je zrekonštruovateľný ak poznáme stav v čase $t + 1$. Bolo ukázané [5], že ak CA je reverzibilný, potom funkcia, ktorá mapuje stav automatu v čase t na stav v

čase $t + 1$ je injektívna a zároveň aj surjektívna. Z toho potom vyplýva, že takýto automat neobsahuje vzor *Garden of Eden*, t.j. taká konfigurácia CA, ktorá nie je vygenerovateľná pomocou danej prechodovej funkcie, nezávisle od inicializačných podmienok. Ak automat je definovaný vo viac ako jednorozmernom prostredí, zistenie, či existuje stav Garden of Eden je nerozhodnuteľný problém, rovnako aj zistenie reverzibility.
Celulárny automat je *univerzálny* (*universal*) [8] ak má rovnakú výpočtovú silu ako univerzálny Turingov stroj, teda ak dokáže simulovať výpočet ľubovoľného Turingovho stroja. Prvá ukážka takého CA pochádza z roku 1966 od matematika Johna von Neumanna [26], ktorý ho zostrojil za účelom nájdenia automatu, ktorý dokáže reprodukovať svoju vlastnú štruktúru. Aby vylúčil z možností triviálnych riešení, vyžadoval univerzálnosť. Výsledok bol dvojdimenzionálny CA v nekonečnom priestore s pevne danou prechodovou funkciou a počiatočnou konfiguráciou, dokopy s 29 možnými stavmi, ktorý dokázal vygenerovať nové vzory buniek a tak vytvoriť kópiu seba samého ale aj ľubovoľného iného celulárneho automatu. Automat bol ďalej zjednodušený najskôr na osemstavový (Codd, 1967 [10]), potom na štvorstavový (1971, Banks [3]).

2.1.2 Známe celulárne automaty

Medzi často skúmané celulárne automaty patria Wolframové jednoduché jednorozmerné automaty [31] (známe aj ako elementárne automaty). Sú definované na jednorozmernej mriežke, každá bunka má 2 možné stavy (0 a 1) a prechodová funkcia berie do úvahy danú bunku a susedov na oboch stranách (ako na obrázku 2.2). Teda v takýchto automatoch existuje $2^3 = 8$ rôznych možných susedstiev, a - keďže prechodová funkcia k nim môže priradiť 2 rôzne hodnoty - $2^8 = 256$ rôznych možných prechodových funkcií (a tým pádom existujú 256 rôzne elementárne automaty) - každý indexovaný pomocou 8 bitového binárneho čísla, pričom číslo zároveň reprezentuje aj hodnoty prechodovej funkcie pre dané poradie susedstiev (napr. $110 = 01101110_2$ pre pravidlo 110).

Ďalším široko známym automatom je Conwayov CA [6]: *hra život* (*Conway's game of life*). Je taktiež dvojstavový (živé a mŕtve stavy) ale definovaný na dvojrozmernej mriežke patriace do štvrtej triedy podľa Wolframovej klasifikácie, ktorý so svojím správaním pripomína vývoj spoločnosti živých organizmov. Bol vymyslený von Neumannom, pričom Conway dokázal jeho univerzálnosť. Prechodové pravidlo je definované nasledovne:



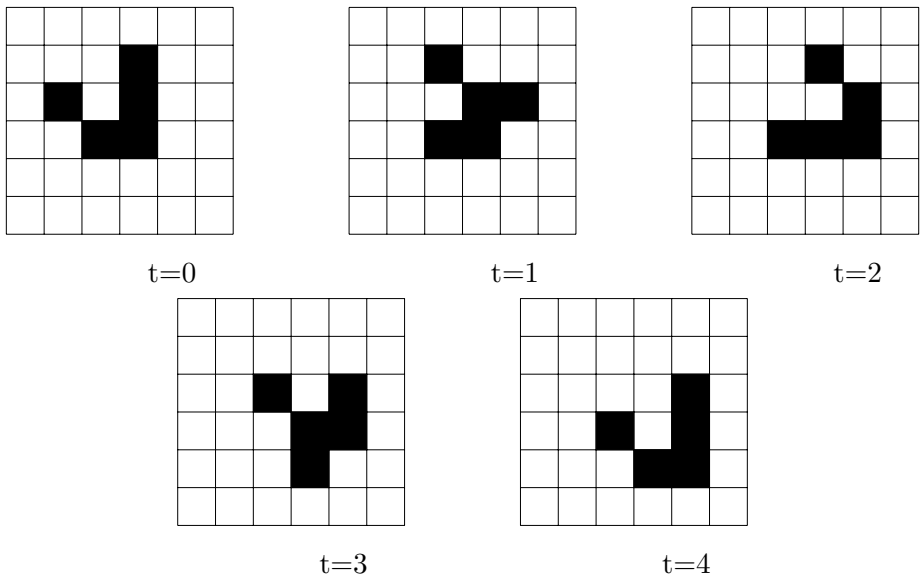
Obr. 2.3: Pravidlo 110 - každý riadok reprezentuje jednu iteráciu, $t = 0$ je najvrchší riadok, začiatočná konfigurácia sa skladá z jednej 1 (čierny štvorec), zdroj: [30]

- každá aktívna, živá bunka, ktorá má menej ako dvoch susedov umiera z osamelosti
- každá aktívna, živá bunka, ktorá má viac ako troch susedov umiera kvôli nedostatku priestoru
- mŕtva bunka s presne troma živými susedmi ožije
- živá bunka s dvoma alebo troma susedmi ostáva živá aj v ďalšej iterácii

Hra v tomto zmysle teda neznamená klasickú hru, keďže hráč so systémom vie interagovať jedine nastavením počiatočnej konfigurácie, ďalej len pozoruje ako sa automat evolvuje.

2.2 Evolučné algoritmy

Pojem '*evolučný algoritmus*' (*evolutionary algorithm*, EA) zahŕňa pomerne široké spektrum optimalizačných a heuristických metód, ale ich základná idea je spoločná: prehľadávanie priestoru riešení, kde každý bod priestoru reprezentuje riešenie pre

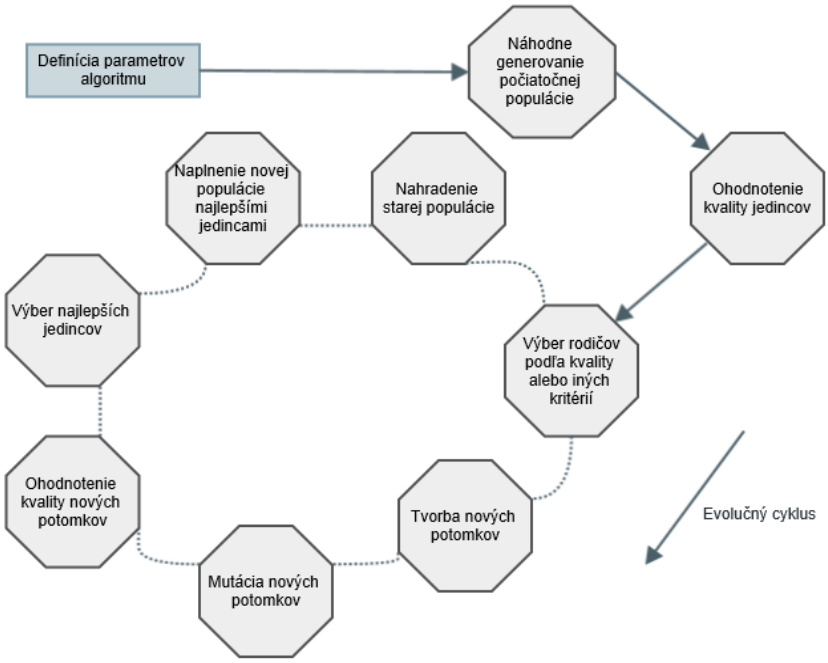


Obr. 2.4: Špeciálna štruktúra *glider* v Conwayovom automate. V každom štvrtom časovom kroku sa hýbe diagonálne o bunku

daný problém. Priestor potenciálnych riešení môže byť až tak obrovský, že klasické prehľadávacie algoritmy a analytické metódy optimálne riešenie nedokážu nájsť v prijateľnom čase. Evolučné algoritmy sa používajú práve na riešenie takýchto problémov.
Na vybraný podpriestor tohto priestoru môžeme pozerať ako na populáciu jedincov, kde každý jedinec danej poplácie je kandidát na riešenie problému. Počiatočná populácia je generovaná náhodne. Populácia sa časom mení, nevhodné jedince (t.j. najhoršie riešenia) sú odstránené z populácie aby sa uvoľnilo miesto pre nových, lepších jedincov. Tieto procesy sa iterujú v *evolučnom cykle* po tzv. *generáciách*, kým sa nenájde dostatočne kvalitné riešenie alebo nedosiahneme prednastavenú hranicu počtu generácií - príp. algoritmus môže skončiť aj v prípadoch ak nájdené najlepšie riešenie sa už dlhý čas nezlepšilo.

2.2.1 Komponenty evolučných algoritmov

- **reprezentácia** - prvým krokom algoritmu je riešenie mapovania medzi priestorom problému a priestorom algoritmu. Vzhľadom na inšpiráciu z genetiky, jedinec v kontexte pôvodného problému sa volá *fenotyp* a jeho kódovanie v EA sa volá *genotyp*. Tieto dve reprezentácie môžu byť veľmi odlišné, nakoľko verzia s ktorou pracuje algoritmus musí byť ľahko modifikovateľná aj



Obr. 2.5: Všeobecný cyklus evolučného algoritmu - podľa [32]

v závislosti od iných genotypov (viď kríženie a mutácia) - napr. binarný reťazec. Hotové riešenie dostaneme jeho dekódovaním na fenotyp, na ktorom sa testuje aj kvalita riešenia.

- **vyhodnocovacia funkcia (účelová funkcia)** - slúži na vyhodnotenie kvality daného riešenia. Každý jedinec má tzv. *fitness hodnotu*, čím kvalitnejšie je riešenie, tým väčšia (príp. menšia ak sa funkcia minimalizuje) je hodnota. Teda pre účelovú funkciu F platí: $F : P \mapsto \mathbb{R}^+$, kde P je populácia jedincov.

- **selekcia** - proces na začiatku každej generácie počas ktorého sú vybratí rodičia danej generácie, zvyčajne pomocou pravdepodobnosti - lepšia fitness hodnota znamená väčšiu pravdepodobnosť. Keďže výber výlučne najlepších jedincov (greedy search) by viedol k *predčasnej konvergencii* (*premature convergence*), čiže stagnácii na lokálnom optime, EA s nejakou nenulovou pravdepodobnosťou vyberajú aj jedince s nižšou fitness, čím umožnia populácii objaviť vzdialenejšie oblasti prehľadávacieho priestoru.
  Typické selekčné mechanizmy [2]:

  - *ruletová selekcia (roulette wheel selection / fitness proportionate selection)* - individuá sú namapované na spojitý segment nejakej čiary a veľkosť tohto segmentu je úmerná fitness hodnote. Generuje sa náhodné

číslo podľa veľkosti čiary a bude vybratý jedinec podľa toho, že do ktorého segmentu padne toto číslo - takže veľkosť pravdepodobnosti selekcie jedinca závisí od veľkosti jeho fitness.

- *turnajová selekcia (tournament selection)* - z populácie je náhodne vybratá množina k jedincov (obvykle $k = 2$) a následne z tejto množiny je vybratý jedinec s najlepšou fitness. Je to jedna z najpopulárnejších metód, jednak kvôli udržaniu selekčného tlaku nezávisle na rozdieloch fitness jedincov a na druhej strane jedince s menšou fitness hodnotou tiež dostávajú (na rozdiel napríklad od ruletovej selekcie) nezanedbateľne veľkú šancu na vybratie, čo zvyšuje obranu proti predčasnej konvergencii.
- *selekcia s orezávaním (truncation selection)* - elitistická selekcia počas ktorého populácia je utriedená podľa fitness, slabšie jedince sú orezané (podľa nejakej prahovej hodnoty) a nedostanú žiadnu šancu na prežitie, ostatní sú vybratí.
- *elitizmus (elitism)* - nie je to samostatná metóda, zabezpečuje iba to, aby najlepší jedinec (príp. množina najlepších jedincov) bol v každom prípade prenesený bez zmien do ďalšej generácie, zvyšok je už vybratý iným mechanizmom. Väčšina selekčných metód to totiž negarantuje a v každom kroku máme šancu na stratenie najlepšieho jedinca.
- *selekcia podľa pareto dominancie* - používa sa pri viacúčelovej optimalizácie (viď 2.2.3), jedince dostanú ohodnotenie na základe pareto dominancie (*pareto rank*) a podľa nej sa robí selekcia. Vo všeobecnosti je elitistická, najlepšie jedince (ktoré nie sú dominované inými jedincami) sú nechané v populácii alebo sú pamätané v archíve aby sa zachovala rozmanitosť populácie - kvôli tomu odložené jedince často ani nie sú naďalej vybratí na reprodukciu.

- **kríženie** - alebo rekombinácia je operácia, počas ktorého sa kombinujú dva jedince (rodičia) podľa pevne daných pravidiel a výsledkom toho je nový jedinec (potomok). Operácia je stochastická, dvojice rodičov sú náhodne vybrané, tak ako aj ich časti na kríženie. Je to najčastejšie binárna operácia, hoci kríženie viac než dvoch rodičov je matematicky realizovateľné.

- **mutácia** - ďalšia operácia ktorá sa aplikuje len na jedného jedinca a výsledkom je jeden nový, modifikovaný jedinec. Taktiež stochastická operácia.

- **nahradenie** - mechanizmus, ktorý sa spúšťa na konci každej generácie a rozhoduje sa ktoré jedince môžu dostať do ďalšej generácie. Veľkosť populácie

počas celej evolúcie je často konštantná, a vyberanie vhodných jedincov na rozdiel od ostatných komponentov algoritmu je deterministické. Obyčajne ostanú individuá s najlepšou fitness hodnotou, ale často aj vek je rozhodovací faktor. Často použité metódy nahradenia (charakteristické pre evolučné stratégie - podoblasť EA):

- *elitistické* $(\mu + \lambda)$ - v každej generácii sú vybratí μ rodičia na kríženie, ktorí vyprodukujú λ potomkov. Do ďalšej generácie sa dostane μ najlepších jedincov z celej populácie
- *neelitistické* (μ, λ) - v každej generácii sú vybratí μ rodičia a vyprodukujú λ potomkov. Do ďalšej generácie sa dostane μ najlepších potomkov

Evolučné algoritmy sú teda optimalizačné algoritmy a ich úlohou je nájsť optimum (maximum alebo minimum) fitness funkcie f - teda optimálneho genotypa:

$$g(opt) = \arg\max_{g \in G^*} f(g) \tag{2.1}$$

kde G^* je množina všetkých možných genotypov pre daný problém.

2.2.2 Reprezentácia

Voľba vhodnej reprezentácie genotypov je dôležitým krokom pri tvorbe evolučných algoritmov a značne ovplyvňuje efektívnosť prehľadávania priestoru. Ak dve individuá sú podobné na úrovni genotypu, mali by byť podobné aj na úrovni fenotypu. Inak povedané, malé zmeny v genotype by nemali spôsobiť príliš veľké zmeny vo fenotype, inak prehľadávací priestor sa stane menej a menej spojitým a operátor mutácie dokáže vytvárať diery v pokrytí okolia daného jedinca. Efektívna reprezentácia teda môže redukovať prehľadávací priestor a zvýšiť adaptáciu. Medzi často zvolené reprezentácie patria:

- binárne reťazce (genetické algoritmy)
- syntaktické stromy (genetické programovanie)
- reálne vektory (evolučné stratégie)
- konečné stavové automaty (evolučné programovanie)

Typy reprezentácií môžeme rozdeliť do troch hlavných kategórií:

- **priama reprezentácia**

- **nepriama reprezentácia**
- **celulárna reprezentácia**

Priama reprezentácia je najjednoduchší typ reprezentácie, kde genotyp jedinca je zároveň aj fenotypom alebo sú črty fenotypu jedna k jednej priamo zobrazené v genotype - tým pádom testovanie fitness potrebuje len jednoduché príp. nepotrebuje žiadne trasformácie, napr. dvojrozmerné bitové pole na kódovanie čierno-bieleho obrazu alebo neurónové siete.

V prípade nepriamej reprezentácie (alebo generatívnej reprezentácie) genotyp kóduje inštrukcie ako z neho skonštruovať výsledný fenotyp, pričom transformácia je zvyčajne vykonaná pomocou stavového automatu alebo formálnej gramatiky. Viac biologicky inšpirovaná ako priama reprezentácia, na podobnom princípe fungujú pravidlá prekladu DNA na proteíny a v konečnom dôsledku na embriá.
Príklad takejto reprezentácie sú *L-systémy* (alebo *Lindenmayerové systémy*) [21]. L-systémy sú paralelné prepisovacie systémy podobné ako Chomského gramatiky. Skladajú sa z inicializačného reťazca symbolov a množiny odvodzovacích pravidiel, ktoré sú aplikované na jednotlivé symboly reťazca na vytvorenie nových reťazcov. Napr. pre symboly A, B, štartovací reťazec *'A'* a prepisovacie pravidlá $(A \mapsto AB), (B \mapsto A)$ dostaneme:
$n = 0 : A$
$n = 1 : AB$
$n = 2 : ABA$
$n = 3 : ABAAB$
$n = 4 : ABAABABA$
$\ldots$

Celulárna reprezentácia používa celulárne automaty na genetickej úrovni a tým pádom v istom zmysle kombinuje vlastnosti predchádzajúcich dvoch reprezentácií: topológia automatu môže priamo reprezentovať výsledný obrázok (napr. každá bunka jeden pixel) na druhej strane farbu príp. iné vlastnosti buniek môžeme získať implicitne.

2.2.3 Viacúčelová optimalizácia

Optimalizačné algoritmy sú často použité na optimalizáciu viacerých funkcií (príp. funkcií ktoré mapujú do viacrozmerného priestoru), vďaka ktorému sa prehľadávací

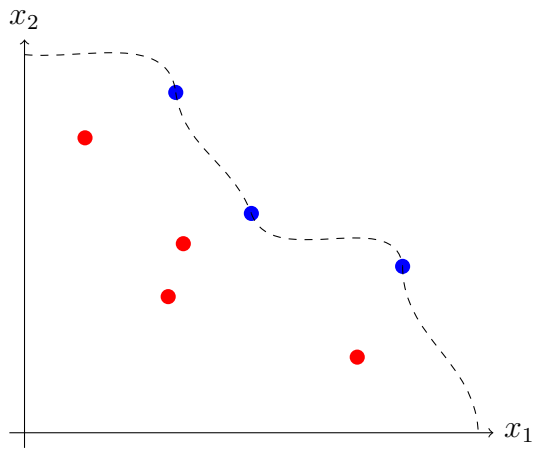
priestor stane čiastočne usporiadaným a zvyčajne namiesto jedného optima máme množinu optimálnych kompromisov. Vo všeobecnosti problém môžeme zadefinovať ako funkciu f, ktorá mapuje vektor premenných (tzv. *rozhodovací vektor*) na vektor účelových hodnôt (*účelový vektor*):

$$(y_1, y_2, \ldots, y_n) = f(x_1, x_2, \ldots, x_n) \tag{2.2}$$

Môžeme pritom predpokladať, že každú zložku účelového vektora maximalizujeme (keďže $\max_{x \in X} f(x) \Leftrightarrow \min_{x \in X} -f(x)$).
V tomto prípade riešenie môže byť lepšie aj horšie ako nejaké iné riešenie (t.j. je lepšie príp. horšie vzhľadom na každý účel), ale môžu byť aj neporovnateľné. Ak riešenie X je lepšie ako riešenie Y, hovorí sa, že X dominuje Y. Optimálne riešenie je teda také, ktoré nie je dominované žiadnym iným riešením - volá sa to *pareto optimálne*, a množina pareto optimálnych riešení tvorí zoznam optimálnych kompromisov. Obvykle na konci chceme vybrať len jedno z týchto riešení, napr. tak, že skalarizujeme účelový vektor:

$$\max_{x \in X} \sum_{i=1}^{n} w_i f_i(x) \tag{2.3}$$

kde w_i je váha i-tého účelu.



Obr. 2.6: Pareto fronta dvojrozmernej funkcie (ktorú maximalizujeme). Modré body sú pareto optimálne a červené dominované.

V prípade evolučných algoritmov [33] viacúčelovosť prinesie niekoľko zaujímavých problémov: ako pridelovať fitness hodnoty jedincom a ako zabezpečiť rozmanitosť populácie.

Jeden z možných spôsobov pridelenia fitness je horeuvedená skalarizácia (prípadne s premenlivými váhami kvôli nájdeniu rôznych možných kompromisov). Iné postupy robia selekciu na základe jedného zvoleného kritéria - buď náhodne, alebo pridelením pravdepodobnosti k jednotlivým kritériám.
Rozmanitosť môže byť zabezpečená začlenením informácie o hustote do selekčného procesu: pravdepodobnosť vybratia jedinca na selekciu klesá so zvyšujúcou sa hustotou jedincov v jeho okolí.

2.3 CMAES

CMAES (Covariance Matrix Adaptation Evolution Strategy) [19] je evolučný algoritmus na spojitú optimalizáciu, ktorý sa používa na optimalizáciu nelineárnych, nekonvexných funkcií. Základná idea algoritmu je podobná ako v prípade analýzy hlavných komponentov (PCA), ale množina prehľadávacích bodov v tomto prípade nie je vopred daná ale iteratívne vzorkovaná. Následne pomocou kovariančnej matice prehľadávacích bodov sa optimalizuje prehľadávací priestor. Keďže jedince sú reálne vektory, algoritmus patrí do kategórie evolučných stratégií.

2.3.1 Vzorkovanie novej generácie

Nové prehľadávacie body sú vzorkované pomocou viacrozmernej normálnej distribúcie s kovariančnou maticou C a strednými hodnotami m (2.4). Takéto vzrokovanie pre evolučný algoritmus znamená aplikácie mutácie: všetky jedince z populácie sú mutované náhodným gaussovským vektorom x_i.

$$x_i \sim \mathcal{N}_i(m, \sigma^2 C) = m + \sigma \mathcal{N}_i(0, C) \text{ for } i = 1, \ldots, \mu \tag{2.4}$$

kde
$x_i, m \in \mathbb{R}^n, \sigma \in \mathbb{R}^+, C \in \mathbb{R}^{n \times n}$
x_i je gaussovský vektor,
m je vektor stredných hodnôt a reprezentuje aktuálneho najlepšieho jedinca,
σ je veľkosť kroku (ovplyvňuje veľkosť prehľadávacieho priestoru),
C je kovariančná matica (ovplyvňuje smery prehľadávacieho priestoru)

2.3.2 Parametre a operátory

Parametre algoritmu:

- veľkosť populácie λ

- počet vybratých rodičov na selekciu μ
- váha jednotlivých rodičov $w_{i=1\ldots\mu}$
- kontrola kroku c_σ, d_σ
- počet parametrov jedincov N

Inicializácia: $g = 0, C = I, p_\sigma = 0$, pričom stred distribúcie m a krok σ závisí od problému.

$$\lambda = 4 + \lfloor 3 \ln N \rfloor \tag{2.5}$$

$$\mu = \left\lfloor \frac{\lambda}{2} \right\rfloor \tag{2.6}$$

CMAES používa elitistickú selekciu, vyberie μ najlepších jedincov z populácie (rodičov) a ich váhovanou rekombináciou aktualizuje stred distribúcie.

$$\langle y \rangle_w = \sum_{i=1}^{\mu} w_i y_{i:\lambda} \tag{2.7}$$

$$m^{g+1} = m^g + \delta \langle y \rangle = \sum_{i=1}^{\mu} w_i x_{i:\lambda} \tag{2.8}$$

$$w_i > \cdots > w_\mu > 0, \sum_{i=1}^{\mu} w_i = 1 \tag{2.9}$$

Váhy sú pridelené v závislosti od fitness lineárne (2.10) alebo logaritmicky (2.11) klesajúco, príp. aj zložitejšími spôsobmi (2.12).

$$w_i = \mu - i \tag{2.10}$$

$$w_i = \log \mu + 1 - \log i + 1 \tag{2.11}$$

$$w_i = \frac{w_i'}{\sum_{j=1}^{\mu} w_j'}, \; w_i' = \ln(\mu' + 0.5) - \ln i \quad \text{for } 1, \ldots, \mu \tag{2.12}$$

Kontrola kroku:

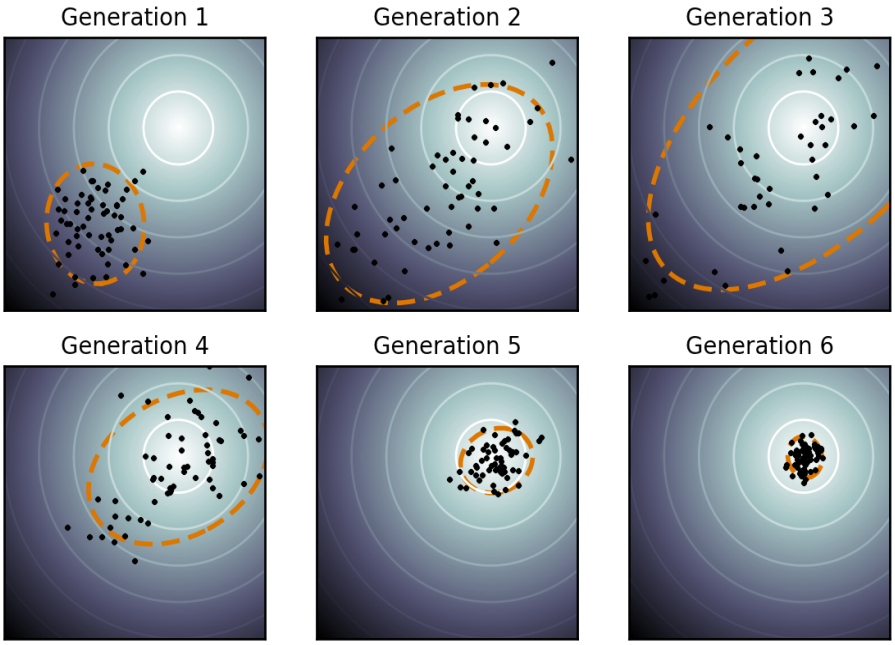
$$c_\sigma = \frac{\mu_{\text{eff}} + 2}{N + \mu_{\text{eff}} + 5}, \mu_{\text{eff}} = \left(\sum_{i=1}^{\mu} w_i^2 \right)^{-1} \tag{2.13}$$

$$d_\sigma = 1 + 2 \max \left(0, \sqrt{\frac{\mu_{\text{eff}} - 1}{N + 1}} - 1 \right) + c_\sigma \tag{2.14}$$

$$p_\sigma = (1 - c_\sigma)p_\sigma + \sqrt{c_\delta(2 - c_\sigma)\mu_{\text{eff}}}C^{-\frac{1}{2}}\langle y \rangle \tag{2.15}$$

$$\sigma = \sigma \cdot \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|p_\sigma\|}{E\|\mathcal{N}(0, I)\|} - 1\right)\right) \tag{2.16}$$

S konštantnou veľkosťou kroku by algoritmus s veľkou pravdepodobnosťou stagnoval v okolí globálneho optima.



Obr. 2.7: Adaptácia v CMAES, zdroj: [11]

2.3.3 Adaptácia kovariančnej matice

Kovariančná matica C náhodného vektora x je definovaná ako:

$$C = E\left[(x - E[x])(x - E[x])^T\right] \tag{2.17}$$

Pomocou vybratých μ rodičov nová kovariančná matica sa dá aproximovať nasledovným vzorcom:

$$C^{g+1} = (1 - c_{\text{cov}})^{g+1}C^g + c_{\text{cov}}\sum_{i=1}^{\mu} w_i y_{i:\lambda}^{g+1}\left(y_{i:\lambda}^{g+1}\right)^T \tag{2.18}$$

c_{cov} je rýchlosť učenia ($c_{\text{cov}}, (1 - c_{\text{cov}}) \in \langle 0, 1 \rangle$) a riadi, že do akej veľkej miere pozeráme spätne (jej váha exponenciálne klesá s rastúcim číslom generácií).

Ďalšia aproximácia novej kovariančnej matice je možná pomocou evolučnej cesty 2.15:

$$C^{g+1} = (1 - c_{\text{cov}})^{g+1} C^g + c_{\text{cov}} \left(p_c^{g+1}\right) \left(p_c^{g+1}\right)^T \tag{2.19}$$

Finálnu aproximáciu dostaneme kombináciou predošlých dvoch:

$$\begin{aligned} C^{g+1} = (1 - c_{\text{cov}})^{g+1} C^g &+ \frac{c_{\text{cov}}}{\mu_{\text{cov}}} \left(p_c^{g+1}\right) \left(p_c^{g+1}\right)^T \\ &+ c_{\text{cov}} \left(1 - \frac{1}{\mu_{\text{cov}}}\right) \sum_{i=1}^{\mu} w_i y_{i:\lambda}^{g+1} \left(y_{i:\lambda}^{g+1}\right)^T \end{aligned} \tag{2.20}$$

kde μ_{cov} slúži na nastavenie váh zložiek.

2.4 Neurónové siete

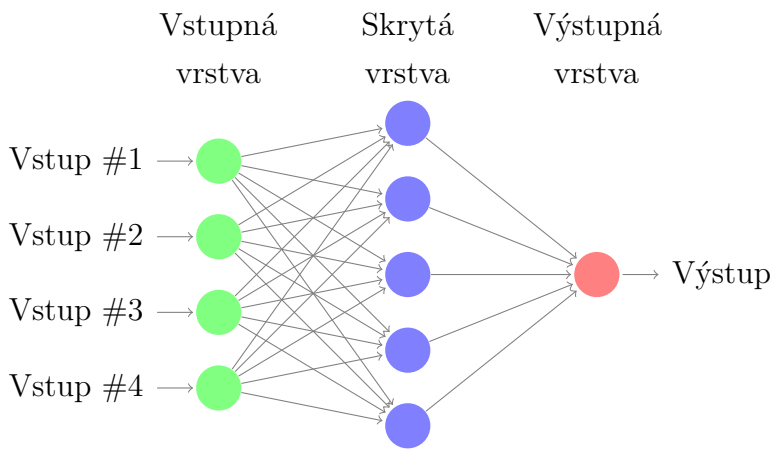
Umelé neurónové siete (*artificial neural networks*, ANN) sú biologicky motivované výpočtové modely založené na neurofyzikálnych poznatkoch (inšpirované biologickým nervovým systémom), použité predovšetkým na inteligentné spracovanie informácie. Skladajú sa z množiny jednoduchých výpočtových jednotiek - umelých neurónov - prepojené hranami (*synaptickými váhami*), ktoré spolu vytvárajú súvislú sieť (orientovaný graf).
Neurónové siete môžu byť rozdelené do rôznych kategórií podľa topológie (napr. rekurentné siete, siete s echo stavmi, samoorganizujúce sa mapy, konvolučné siete, atď...), z pohľadu tejto práce sú dôležité dopredné siete (feedforward networks) - t.j. siete v ktorých sa signál šíri len jedným smerom: dopredu.

2.4.1 Perceptrón

Najjednoduchší typ dopredných neurónových sietí je *perceptrón*, ktorý obsahuje jediný neurón a mapuje vstup siete x na výstup y, kde $x \in \mathbb{R}^n, y \in \mathbb{R}$. Mapovanie znamená vykonanie lineárnej kombinácie x s váhami w medzi zložkami vstupu a neurónom siete - t.j. $w \in \mathbb{R}^n$ - a následné preženie výsledku cez tzv. *aktivačnú funkciu* (*activation function*) f:

$$y = f\left(\sum_{i=1}^{n} w_i x_i\right) \tag{2.21}$$

, t.j. mapovanie $F : \mathbb{R}^n \mapsto \mathbb{R}$.
Pri vhodnom nastavaní váh perceptrón dokáže klasifikovať nejakú dátovú množinu lineárne aj nelineárne v závislosti od aktivačnej funkcie. Bežne použité aktivačné



Obr. 2.8: Viacvrstvový perceptrón s jednou skrytou vrstvou a jedným výstupom

funkcie sú unipolárna alebo bipolárna funkcia na lineárnu klasifikáciu (sú dobré na nájdenie n-rozmernej separovacej nadroviny) príp. *hyperbolický tangent* alebo *sigmoid* (2.22) na nelineárnu.

$$S(x) = \frac{1}{1 + e^{-x}} \tag{2.22}$$

Trénovanie perceptrónu znamená nájdenie vhodných hodnôt váh, aby model klasifikoval správne čo najviac príkladov pre daný klasifikačný problém.

2.4.2 Viacvrstvový perceptrón

Viacvrstvový perceptrón (multi layer perceptron - MLP) je generalizácia jednoduchého perceptrónu na mapovanie do viacrozmerného priestoru. Skladá sa z viacerých neurónov ktoré sú rozdelené do viacerých vrstiev, pričom každý neurón i-tej vrstvy je spojený s každým neurónom $(i+1)$-ej vrstvy - t.j. tieto váhy môžeme predstaviť ako maticu $\mathbb{R}^{n\times m}$, ak i-tá vrstva sa skladá z n a $(i+1)$. z m neurónov. Prvá vrstva obsahuje vstupy siete, posledná výstupy, pričom medzi nimi môže existovať ľubovoľný počet skrytých vrstiev.
Výpočet hodnoty k-tého neurónu na n-tej vrstve, ak $(n-1)$. vrstva obsahuje m neurónov:

$$h_{n,k} = f\left(\sum_{i=1}^{m} w_{n-1,kj} h_{n-1,j}\right) \tag{2.23}$$

Viacvrstvový perceptrón je univerzálny aproximátor funkcií [23], t.j. pre ľubovoľnú množinu dát $A = \{x^1, x^2, \ldots, x^k\}, x^i \in \mathbb{R}^n$ pre $i = 1, \ldots, k$; pre $\epsilon > 0$ a pre ľubovoľnú spojitú funkciu $F : \mathbb{R}^n \mapsto \mathbb{R}^m$ definovaná nad množinu A existuje funkcia $G\left(x^p\right) = f\left(\sum_{k=1}^{q} w_k f\left(\sum_{j=1}^{n} v_{kj} x_j^p\right)\right)$ taká, že $\sum_p |F(x^p) - G(x^p)| < \epsilon$.

Hovoríme, že G aproximuje F nad A s presnosťou ϵ. G môžeme interpretovať ako 2-vrstvový perceptrón s m výstupom.

2.4.3 Spôsoby učenia

Učenie neurónovej siete znamená spustenie trénovacieho algoritmu na nájdenie hodnôt pre synaptické váhy, ktoré s čo najväčšou presnosťou aproximujú hľadanú funkciu (predpokladáme, že optimálna funkcia existuje). Metódy učenia môžu byť priradené do 3 paradigiem:

- učenie s učiteľom - máme k dispozícii označkované dáta (t.j. výstup ku každému vstupu) a cieľom je nájsť čo najpresnejšie mapovanie
- učenie bez učiteľa - máme k dispozícii neoznačkované dáta a cieľom je nájsť v nich nejakú štruktúru alebo štatistické pravidelnosti
- učenie odmenou a trestom - nachádza sa niekde medzi učením s učiteľom a učením bez učiteľa, nie sú explicitne dané hľadané výstupy, ale sieť dostane spätnú väzbu o kvalite riešenia

2.5 Voronoiov diagram

Voronoiov diagram [7] je známa geometrická štruktúra, ktorá opisuje rozdelenie priestoru X na regióny na základe nejakej množiny bodov $\{p_i\}$, nazvané ako *generátory* alebo *jadrá*, pričom ku každému generátoru patrí presne jeden región $\mathcal{V}(p_i)$ (alebo *Voronoiová bunka)*. Rozdelenie prebieha na základe jednoduchého pravidla: bod $x \in X$ patrí do regiónu $\mathcal{V}(y)$, pokiaľ vzdialenosť x od y je ostro menšia ako vzdialenosť x od ľubovoľného bodu z množiny $\{p_i\} \setminus \{y\}$. Platí teda:

$$\mathcal{V}(p_i) = \{x \in X \mid d(x, p_i) < d(x, p_j) \text{ pre všetky } j \neq i\} \tag{2.24}$$

Nás teraz zaujíma euklidovský priestor, teda funkcia d bude v našom prípade euklidovská vzdialenosť:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{2.25}$$

Pre dvojice bodov $p, q \in X$ môžeme zadefinovať rozdelovaciu priamku (*bisektor*), ktorá je kolmá na úsečku $\overline{pq}$ a pretína $\overline{pq}$ práve v polovici. Taká priamka rozdeľuje X na dva polpriestory: nech polpriestor, ktorý obsahuje p sa volá $h(p, q)$ a polpriestor,

ktorý obsahuje q $h(q,p)$. Zjavne platí, že ak $r \in h(p,q)$, potom $d(r,p) < d(r,q)$. Vidíme teda, že:

$$\mathcal{V}(p_i) = \cap_{1 \leqslant j \leqslant n, j \neq i} h(p_i, p_j) \tag{2.26}$$

Keďže každý región dostaneme ako prienik $n-1$ polpriestorov, výsledok v konečnom euklidovskom priestore bude v každom prípade konvexný polytóp.

2.5.1 Konštrukcia Voronoiovho diagramu

Priamočiary algoritmus na výpočet Voronoiovho diagramu by sme dostali aplikovaním vzorca 2.26. Nájdenie prieniku polpriestorov $h(p_i, p_j)$ ak $i \neq j$ pre n generujúcich bodov je realizovateľné pomocou lineárneho programovania v čase $O(n \log n)$, čo priamo vedie k výslednému algoritmu so zložitosťou $O(n^2 \log n)$. Ukazuje sa, že dá sa to aj efektívnejšie: inkrementálny algorimus založený na postupnom pridávaní bodov do $\mathcal{V}(p_i)$, t.j.

$$\mathcal{V}(p_{i+1}) = \mathcal{V}(p_i) \cap p_{i+1} \tag{2.27}$$

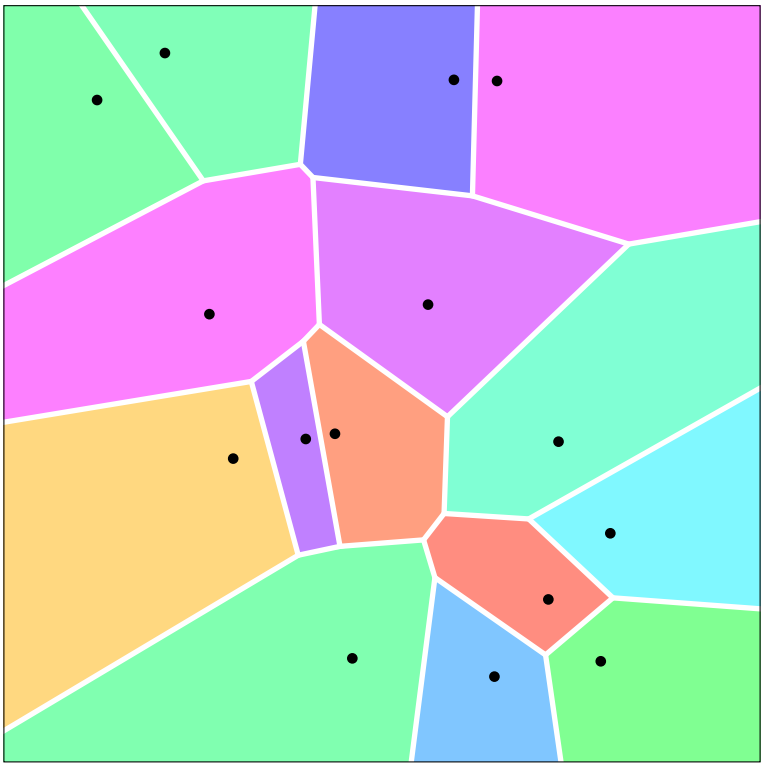
pričom základná idea spočíva v aktualizácii vygenerovanej štruktúry v predošlej iterácii, pridaním nového bodu do už existujúcej bunky a vytvorením jeho hranice useknutím okolitých hrán. Efektívne prechádzanie susedstva je zabezpečené *half-edge* štruktúrou reprezentácie. Keďže pri každej novej bunke strávi lineárne veľa času v závislosti od spracovaných bodov, algoritmus má kvadratickú zložitosť.
Pokročilejšie algoritmy založené na *rozdeľuj a panuj* výpočte hrán [1] alebo na vyskladaní parabol popri *sweep* čiare (Fortunov algoritmus [18]) pracujú v čase $\Theta(n \log n)$, a tie sú navyše asymptoticky optimálne, pretože problém usporiadania n celých čísel je redukovateľný na skonštruovanie $\mathcal{V}(p_i)$ [7] a tým pádom vyžaduje $\Omega(n \log n)$ logických krokov.

Algoritmus Inkrementálny Voronoi

Vstup: množina generujúcich bodov $P = \{p_1, \ldots, p_n\}$

Výstup: $\mathcal{V}(p_i)$ ako half-edge štruktúra

```
skonštruuj simplex Ω zahŕňajúce P a V(Ω)
for p in P do
    nájdi V(p_i) obsahujúci p
    b ← bisektor p̄p_i
    w_1, w_2 ← b ∩ V(p_i), w_2 vľavo od w_1
    w_j ← w_1
    repeat
        nájdi V(p_j) incidentné s V(p_{j-1}), na ich spoločnej hrane leží w_j
        b ← bisektor p̄p_j
        w_j, w_x ← b ∩ V(p_j), w_j vľavo od b
        V(p_{j-1}) ← V(p_j)
    until w_j = w_i
    V(p) ← sled w_1, ... w_j
```



Obr. 2.9: Grafické znázornenie Voronoiovho diagramu pozostávajúceho z 15 generátorov. Voronoiové bunky sú vyznačené farebne.

KAPITOLA 3

Východiská

Centrálna idea, ktorá tvorí jadro našej práce bola predstavená v dizertačnej práci Alexandra Deverta [15] a následne vylepšená v [16], je venovaná tzv. *umelej ontogenéze (artificial ontogeny)* - smer v rámci evolučného dizajnu, ktorý uprednostňuje budovanie vývojového procesu (alebo programu) namiesto evolvovania konkrétnych jedincov reprezentujúcich hotové riešenia (t.j. priama reprezentácia). Motivácia za týmto rozhodnutím je taká, že keď dôraz je kladený na nájdenie systému, ktorý dynamicky buduje riešenia namiesto nejakej statickej reprezentácie, ľahšie a efektívnejšie sa skúmajú niektoré vlastnosti kódovania, ako napríklad *modularita* (dobre definovaná lokalizácia nejakého špecifického elementu - funkcionálna alebo štrukturálna), *hierarchia komponentov* (rekurzívna kompozícia štruktúry) alebo *škálovateľnosť*. Umelé ontogenézy môžu byť rozdelené do dvoch skupín v závislosti od spôsobu budovania vývojového procesu na vytvorenie riešenia zakódovaného v genotype:

- *explicitná* - priama evolúcia procesu ako programu, napr. vytvorenie syntaktického stromu pomocou genetického programovania
- *implicitná* - evolúcia množiny jednoduchých pravidiel, ktoré sú iteratívne aplikované na každý element riešenia, prípadne riešenia počas tejto iterácie dokážu prepísať svoju štruktúru a časom rásť, napr. pomocou kódovania bezkontextových gramatík, L-systémov alebo cez viac biologicky motivovanú bunkovú reprezentáciu (napodobňujúca spôsoby správania buniek v živých organizmoch)

Devert vo svojej práci skúmal bunkovú reprezentáciu cez celulárne automaty na riešenie *problému vlajky (Flag problem)*. V nasledujúcich sekciách popíšeme hlavné myšlienky jeho práce.

3.1 Problém vlajky

Problém vlajky - známy aj ako *problém Holandskej vlajky (Dutch national flag problem)* - je optimalizačný problém použitý ako *benchmark* v rôznych oblastiach informatiky, vrátane evolučného dizajnu. Jeho pôvodnú verziu navrhol Edsger Dijkstra, úlohou je preusporiadanie farebných elementov do takého poradia, aby vytvorili Holandskú vlajku, t.j. tri farebné pásy pod sebou: červený, biely, modrý. Viac abstraktne: preusporiadať elementy poľa do troch partícií podľa ich farieb, za účelom nájdenia triedacieho algoritmu, ktorý sa dokáže efektívne vysporiadať s veľkým počtom opakujúcich sa elementov.
Jeho celulárna varianta znie nasledovne: úlohou je nájsť prechodové pravidlo pre dvojdimenzionálny celulárny automat, pomocou ktorého automat dokonverguje do stabilného stavu (trieda 1 podľa Wolframovej kategorizácie), pričom ak stavy interpretujeme ako farby, automat vykresí nejaký cieľový vzor (v tomto špecifickom prípade Holandskú vlajku).
Rozhodnutie pri použití tohto problému má svoje opodstatnenie: zabezpečuje priestor na študovanie a experimentáciu so spomenutými vlastnosťami umelých ontogenéz, pričom je dostatočne jednoduchý na pochopenie a vizualizáciu.

3.2 Organizmus

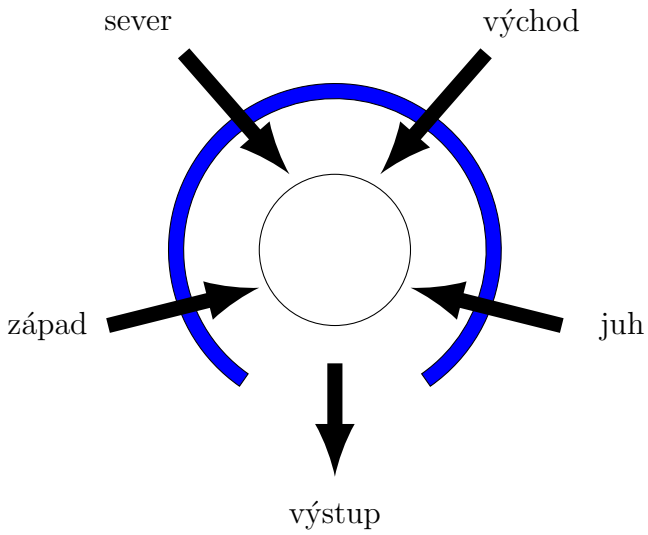
Samotný automat alebo *organizmus* - vzhľadom na silnú biologickú inšpiráciu sa občas preberá aj samotná terminológia - mal klasickú štruktúru, t.j. dvojdimenzionálna mriežka štvorcového tvaru (teda s rovnakou šírkou a výškou) identických buniek (štvorce jednotkovej dĺžky). Bunky komunikovali so susedstvom podľa von Neumannovho princípu, pričom komunikácia prebiehala prostredníctvom tzv. *chemikálií (chemicals)* reprezentovaných ako reálne vektory: bunka posielala chemikálie susedným bunkám a od každého z nich dostala spätne chemikálie - teda tvorili zároveň aj stav danej bunky, ktorý aktualizovali na základe vlastných a susedných chemikálií. Chemikálie boli rozdelené do dvoch skupín:

- *interné chemikálie* - použité len interne jedine na výpočet nasledujúceho stavu danej bunky
- *externé chemikálie* - použité v rámci komunikácie, externé chemikálie sú poslané susedom

Bunky o svojich pozíciach v rámci automatu nič nevedeli, teda podmienky emergentného výpočtu boli splnené: globálna informácia sa šírila jedine cez lokálne interakcie jednotiek systému.
Vhodnosť organizmu zadefinoval ako vzdialenosť medzi cieľovým vzorom a vygenerovaným obrázkom na konci vývoja simulácie automatu, účelová funkcia pre organizmus o na minimalizáciu bola nasledovná:

$$f_{\text{color}}(o) = \sqrt{\frac{\sum_{x=0}^{width} \sum_{y=0}^{height} (I_o(x,y) - I_{target}(x,y))^2}{width \cdot height}} \tag{3.1}$$

kde $I_o(x,y)$ vráti farbu organizmom vygenerovaného obrázku na pozícii (x,y) - reálne číslo z intervalu $[0,1]$, $I_{\text{target}}(x,y)$ farbu cieľového vzoru na pozícii (x,y) a v menovateli je normalizačný koeficient.



Obr. 3.1: Vstup/výstup jednej bunky

3.3 Zastavenie vývoja

Použitý celulárny automat fungoval podľa definície, t.j. nebol diskrétny len v priestore ale aj v čase, a pri každom tiku časovača bunky synchrónne aktualizovali svoje chemikálie. Kritická otázka bola teda riešenie problému zastavenia, ktoré zároveň tvorí jeden z najdôležitejších aspektov Devertovej práce.
Primočiary a často použitý postup je zastavenie vývoja po fixnom počte vývojových krokov (iterácií). Zásadný problém tejto metódy je zjavný: správne odhadnutie tohto arbitrárneho čísla je problematické a vyžaduje dodatočnú experimentáciu; navyše je v spore s predpokladom systému aby bol škálovateľný, pretože väčšie

automaty spravidla vyžadujú väčší počet iterácií na konvergenciu do stabilného stavu. Použitie rovnakého modelu pri fixnom počte krokov pre rôzne organizmy by bolo nerealizovateľné.
Ďalšia možnosť je automatizované zistenie potrebných iterácií cez evolučný algoritmus, zahrnutím tejto informácie do genotypu.
Devertové riešenie problému bolo nové, viac škálovateľné a menej náhodné ako predošlé postupy: použitím dynamickej stability alebo *energie* systému. Počas behu v každom kroku sa vyhodnotí úroveň energie automatu na základe aktivít buniek a dochádza k zastaveniu iterácie ak energia je stabilná. Neformálne, aktivita bunky je definovaná ako funkcia zmien interných a externých chemikálií a energia organizmu je spoločná variácia aktivít buniek. Ak variácia počas nejakého časového intervalu je permanentne dostatočne malá (je pod nejakou prahovou hodnotou), predpokladáme, že automat sa dostal do stabilného stavu.
Viac formálne, energia v čase t je definovaná nasledovne:

$$e(t) = \sqrt{\sum_{c=1}^{N_{\text{cells}}} \|state(c,t)\|^2} \tag{3.2}$$

kde N_{cells} je počet buniek automatu, $state(c,t)$ je zreťazený vektor interných a externých chemikálií bunky c v čase t, $\|\cdot\|$ reprezentuje euklidovskú normu.
Vidíme teda, že aktivita bunky c je daná vzťahom:

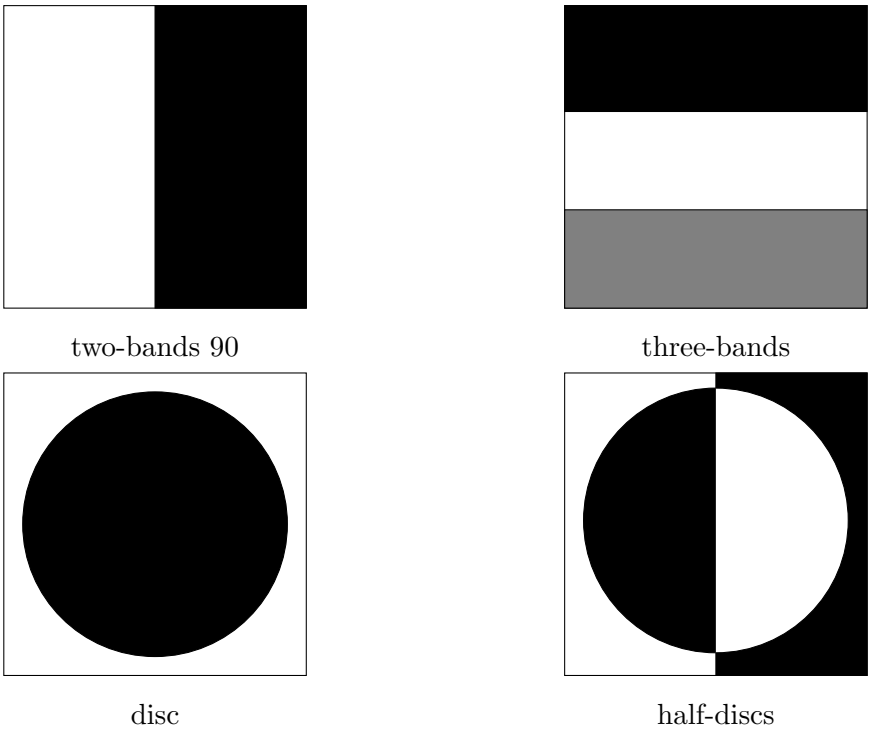
$$a(c) = \|state(c,t)\|^2 = \|u\|^2 + \|v\|^2 \tag{3.3}$$

kde u je vektor externých a v je vektor interných chemikálií.
Priemerná energia $\overline{e}(t)$ a štandardná odchýlka energie $\mathcal{V}ar(t)$ v čase t použitím pozorovaného časového okna veľkosti k:

$$\overline{e}(t) = \frac{1}{k} \sum_{i=t-k}^{t} e(i) \tag{3.4}$$

$$\mathcal{V}ar(t) = \sqrt{\frac{1}{k} \sum_{i=t-k}^{t} (e(i) - \overline{e}(t))^2} \tag{3.5}$$

Vývoj sa zastaví v čase t, pre ktorú platí, že $\mathcal{V}ar(t) < \epsilon$, pre nejakú (dostatočne malú) konštantu ϵ.



Obr. 3.2: Použité cieľové vzory v experimentoch

3.4 Riadiaci prvok

Ako sme už povedali, stav bunky bol reprezetovaný dvojicou reálnych vektorov (u, v), kde u je vektor externých chemikálií veľkosti M, a v je vektor interných chemikálií veľkosti N. Jednotlivé zložky nadobúdali spojitú hodnotu z intervalu $[-1, 1]$ a boli inicializované na 0. Vstupom prechodovej funkcie pre danú bunku c bola koncentrácia interných **aj** externých chemikálií: lokálne interné chemikálie c a externé chemikálie zo susedstva. Výstupom boli nové hodnoty interných aj externých chemikálií pre c. Prechodová funkcia vyzerala nasledovne:

$$\begin{aligned}\Delta_{i,j} = f_u(&u_{i+1,j}(t), u_{i-1,j}(t), u_{i,j}(t),\\ &v_{i,j}(t), u_{i,j-1}(t), u_{i,j+1}(t))\end{aligned} \tag{3.6}$$

$$u_{i,j}(t+1) = \tanh(u_{i,j}(t) + \Delta_{i,j}(t)) \tag{3.7}$$

$$\begin{aligned}v_{i,j}(t+1) = f_v(&u_{i+1,j}(t), u_{i-1,j}(t), u_{i,j}(t),\\ &v_{i,j}(t), u_{i,j-1}(t), u_{i,j+1}(t))\end{aligned} \tag{3.8}$$

Obe funkcie f_u a f_v môžu byť zadefinované ako dopredné viacvrstvové perceptróny (MLP) s jednou skrytou vrstvou a použitím hyperbolického tangentu na aktiváciu

neurónov. Pokiaľ sieť má H skrytých neurónov a každá vrstva jeden *bias* neurón, prechodová funkcia má $(4M + N + 1) \cdot H + (H + 1) \cdot (M + N)$ parametrov.
Uvedené pravidlo samozrejme nesedí pre okrajové prípady na hraniciach, keď bunka nemá presne štyroch susedov - chýbajúce hodnoty boli vtedy doplnené nulami.
Na konci ešte potrebujeme výpočet výslednej farby zo získaných chemikálií. Ako farby boli použité odtiene šedej, ich úroveň zakódovaná na intervale $[0, 1]$ a počítaná nasledujúcim vzorcom:

$$\phi_{\text{color}}(u, v) = \frac{1}{2}(1 + \tanh(s_u \cdot u + s_v \cdot v + s_{bias})) \tag{3.9}$$

Vzorec je znova interpretovateľný ako dopredný perceptrón, tentoraz bez skrytej vrstvy, kde s_u, s_v a s_{bias} sú označenia pre synaptické váhy. Počet parametrov má teda $M + N + 1$.
Nájdenie hodnôt váh bolo realizované cez evolučný algoritmus CMA-ES.
Výhoda použitia tohto modelu spočíva v skutočnosti, že je potrebné ladiť len 3 parametre: M, N a H, ktoré zároveň určujú aj jeho komplexitu.
V jeho práci boli preskúmané aj iné typy riadiacich prvkov:

- NEAT (NeuroEvolution of Augmenting Topologies) - evolučný algoritmus, ktorý optimalizuje neurónovú sieť nielen na úrovni váh, ale aj jej štruktúru.
- neurónová sieť s echo stavmi - sieť realizujúca tzv. rezervoárny výpočet vďaka svojej rekurentnej skrytej vrstvy. Tiež bola použitá s kombináciou CMA-ES.

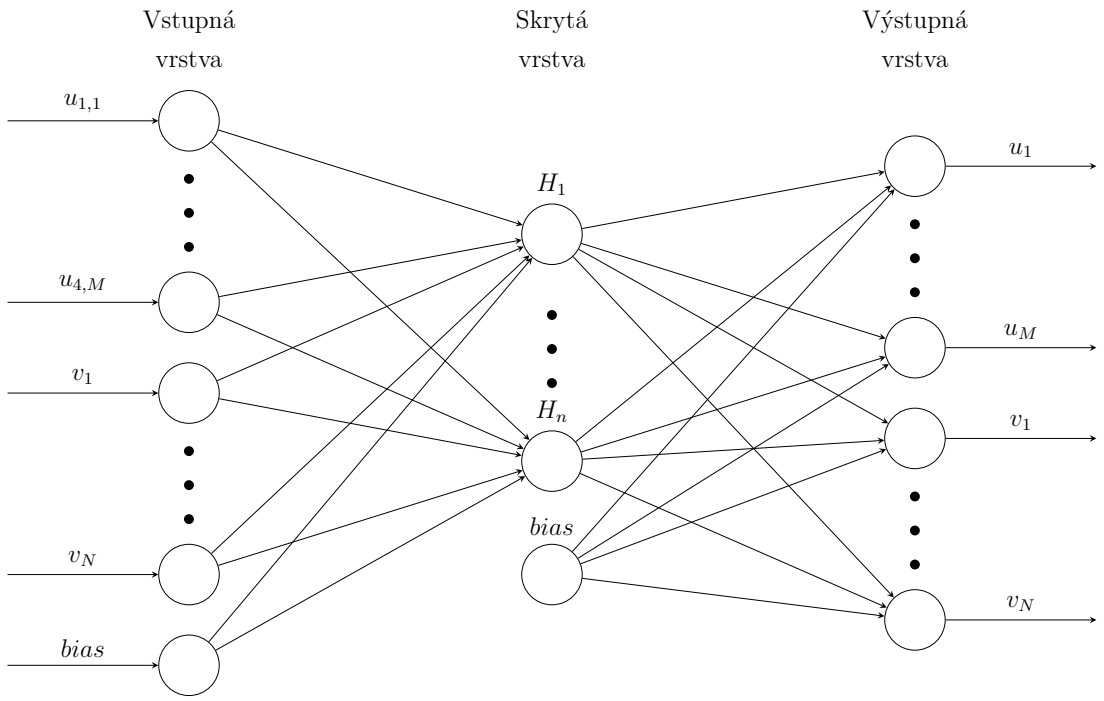
Oba riadiace elementy dosiahli pri tejto úlohe slabšie výsledky ako jednoduchý MLP, takže ďalej sa s nimi nebudeme zaoberať.

3.5 Reakčný-diffúzny prístup

Prechodová funkcia 3.6 celulárneho automatu mala ešte jednu dôležitú vlastnosť: namiesto toho, aby ako vstup priamo vzal externé chemikálie susedstva, prichádzajúci vektor rozmazal pomocou gaussovského filtra - s polomerom 1, keďže je použité len priame susedstvo. Matica takéhoto filtra má tvar:

$$\begin{pmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{pmatrix} \tag{3.10}$$

Priame použitie týchto koeficientov v tomto prípade by nebolo realizovateľné, pretože sa nepoužívala Moorovo susedstvo. Kvôli tomu Devert zostrojil nasledujúci



Obr. 3.3: Vizualizácia topológie neurónovej siete na kódovanie prechodovej funkcie celulárneho automatu. Vstupom sú interné chemikálie bunky a externé jej susedstva, výstupom sú interné a externé chemikálie bunky.

vzorec na filtrovanie:

$$\begin{aligned} externalChemicals\,[i] = (neighbourChemicals_1[i] + \\ 2 \cdot cellChemicals[i] + neighbourChemicals_2[i])\,/\,4 \end{aligned} \tag{3.11}$$

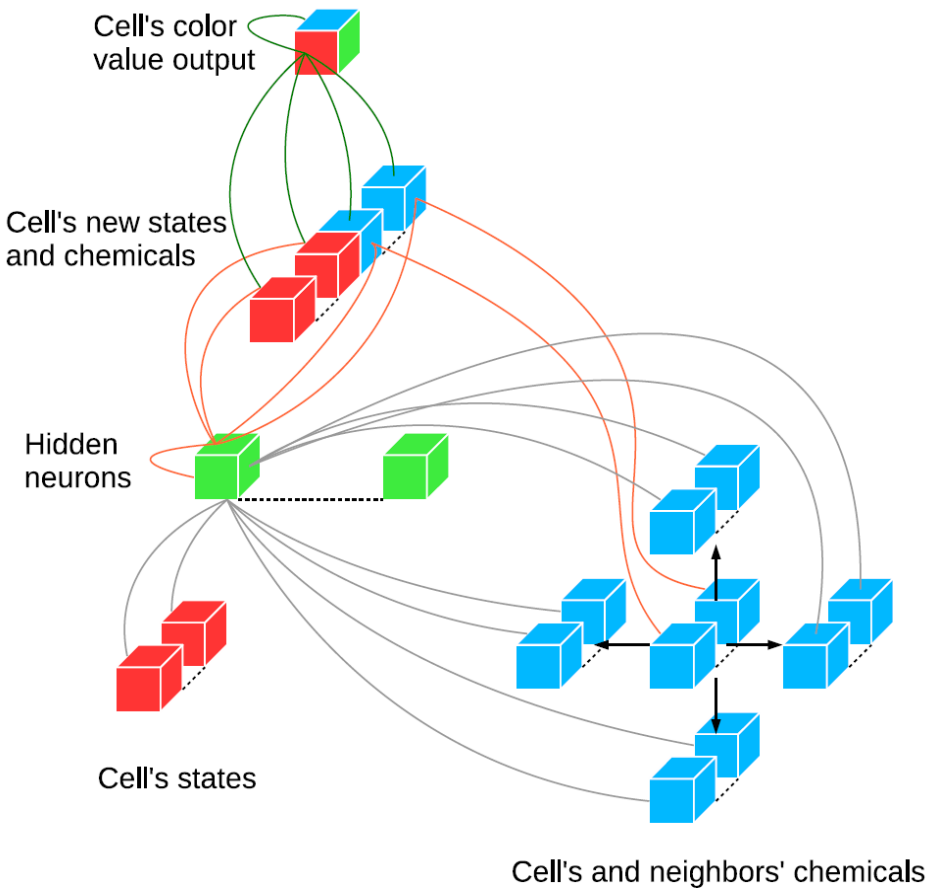
Vzorec sa aplikuje v dvoch fázach: $neighbourChemicals_1$ je vektor externých chemikálií od ľavého suseda v horizontálnej fáze a vektor externých chemikálií od dolného suseda vo vertikálnej fáze. $neighbourChemicals_2$ analogicky označuje chemikálie buď od pravého alebo horného suseda. Z toho vidíme, že matica filtrácie sa modifikuje nasledujúcim spôsobom:

$$\begin{pmatrix} 0 & \frac{1}{4} & 0 \\ \frac{1}{4} & 1 & \frac{1}{4} \\ 0 & \frac{1}{4} & 0 \end{pmatrix} \tag{3.12}$$

Teda prideľuje koeficienty len vo vyžadovaných smeroch, ale poruší pravidlo gaussovskej matice, nesumuje sa do jednotky.

Je to simulácia tzv. *reakčného-difúzneho modelu (reaction-diffusion model)* navrhnutého Alanom Turingom [29] - matematický model, ktorý simuluje zmeny v substanciách na základe lokálnych chemických reakcií a ich následné roznesenie v

priestore (diffúzia). V tomto prípade prechodová funkcia mimikuje rolu chemickej reakcie, ale samozrejme ide len o hrubú aproximáciu, pretože namiesto spojitých substrátov v automate pracujeme v diskrétnom 2D priestore.
Motivácia za týmto mechanizmom je získanie čo najvšeobecnejšieho kandidáta, ktorý dokáže riešiť danú úlohu, pričom nespolieha príliš na štruktúru cieľového vzoru a tým pádom je potenciálne použiteľný aj na riešenie iných problémov.



Obr. 3.4: Schéma výpočtu chemikálií pre jednu bunku pomocou MLP - prevzaté od [20]

3.6 Ďalšie práce

Devertov výskum bol rozvíjaný v rámci dvoch diplomoviek.
Miroslav Beka [4] reprodukoval jeho experimenty, vytvoril prostredie na pokročilú vizualizáciu automatu, ktoré bolo navyše viac modulárne a ľahšie rozšíriteľné ako pôvodná práca - teda používateľsky viac prívetivý nástroj na ďalšiu experimentáciu, aj keď predovšetkým vďaka zvoleným technológiám aj značne pomalší.

Zaujímavejším príspevkom bola diplomová práca Jany Hlaváčikovej [20] priamo nadväzujúca na prácu Deverta. Experimentovala s celulárnymi automatmi s nepravidelnou štruktúrou, konkrétnejšie namiesto pravidelnej štvorcovej mriežky bunky vedeli nadobudnúť tvar štvorca ľubovoľnej veľkosti, čo prinieslo niekoľko potrebných modifikácií do pôvodnej práce:

- komunikácia medzi bunkami sa stala zložitejšou, nakoľko jedna bunka už nemusela nutne mať presne štyroch susedov (z každej strany jedného) ale minimálne štyroch (minimálne jedného z každej strany) pričom horný limit bol obmedzený len veľkosťou priestoru (automatu). Táto skutočnosť ovplyvnila proces aktualizácie stavu jednotlivých buniek, konkrétne prevzatie externých chemikálií a vytvorenie z nich vstup pre neurónovú sieť. Ako následok, premenné $neighbourChemicals_*$ vo vzorci 3.11 nereprezentovali vektor externých chemikálií z danej strany, ale váhovaný vektor všetkých susedov z danej strany vypočítané nasledujúcim spôsobom:

$$\begin{aligned} &neighbourChemicals_*[i] = \\ &\sum_{j=1}^{n} chemicals_j[i] \cdot sharedLength_j / sideLength \end{aligned} \tag{3.13}$$

  kde n je počet susedov bunky z danej strany, $chemicals_j$ je vektor externých chemikálií j-tého suseda, $sharedLength_j$ je dĺžka spoločnej hrany bunky s j-tým susedom a $sideLength$ je dĺžka jednej strany bunky.

- bolo potrebné zaviesť mechanizmus na zmenu štruktúry buniek a zároveň aj automatu počas výpočtu, čo dosiahla modifikáciou riadiacieho prvku, ktorý zo stavu bunky zistil okrem výslednej farby aj *rastúcu akciu (growing action)*, veľmi podobným spôsobom:

$$\phi_{\text{action}}(u, v) = \frac{1}{2}(1 + \tanh(s_u \cdot u + s_v \cdot v + s_{bias})) \tag{3.14}$$

  Akcia mohla nadobudnúť hodnotu z trojice

  1. *nerob nič*
  2. *rozpadni sa* na menšie bunky
  3. *spoj sa* s inou bunkou

  Mapovanie hodnoty funkcie 3.14 už prebiehalo priamočiaro: obor hodnôt funkcie sa rozdelil na tri podintervaly rovnakej veľkosti a v závislosti od toho, že do ktorého intervalu padla vygenerovaná hodnota, bola vybratá akcia.

Rozpadnutie bunky znamenalo vytvorenie štyroch štvorcov rovnakej veľkosti, pričom každý z nich obsadil presne štvrtinu priestoru pôvodnej bunky a každý z nich zdedil aj jej stav - okrem prípadu ak bunka na rozpadnutie mala jednotkovú dĺžku.
Pri akcii spojenia si bunka vypýtala od štyroch svojich susedov ich akciu a pokiaľ každý z nich sa chcel spojiť, zo štyroch sa vytvorila jedna nová bunka - za predpokladu, že každá z nich mala rovnakú veľkosť a výsledok bola nová štvorcová bunka. Stav novej bol daný aritmetickým priemerom stavu komponentov.

- zaviedla účelovú funkciu, ktorá hodnotila štruktúru automatu:

$$f_{\text{structure}}() = \frac{\sum_{i=1}^{n} \min(black_i, white_i) + \frac{n}{width}}{\min\left(\sum_{i=1}^{n} black_i, \sum_{i=1}^{n} white_i\right) + height} \tag{3.15}$$

kde n je počet buniek, $black_i$ resp. $white_i$ je počet čiernych resp. bielych pixlov v i-tej bunke s ohľadom na cieľový vzor. Funkcia teda penalizuje plochu bunky, ktorá sa nachádza v inej komponente cieľového vzoru ako samostatná bunka, pričom komponenty rozlišuje podľa ich farby - predpokladá použitie monochromatických cieľových vzorov, t.j. buď je biela alebo čierna. Komponent bunky je ten komponent vzoru (alebo postupnosť komponentov rovnakej farby), s ktorým (ktorými) sa podiela na najväčšej ploche spomedzi všetkých komponentov.
Nájdenie riešenia prebiehalo buď *sekvenčne*, alebo *paralelne*. Sekvenčná optimalizácia znamenala nájdenie vhodnej štrukúry bez optimalizácie farby a následné spustenie optimalizácie farby bez toho, aby sa štruktúra ďalej menila. Pri paralelnej optimalizácii bol vyhodnotený fitness podľa oboch kritérií a zoznam týchto dvojíc bol čiastočne usporiadaný podľa pareto dominancie.

KAPITOLA 4

Formulácia hypotéz

Ciele tejto diplomovej práce majú implementačný aj výskumný charakter a dajú sa zosumarizovať v dvoch hlavných bodoch:

1. vytvorenie dostatočne všeobecného rámcového prostredia, ktoré realizuje model evolučného dizajnu na celulárnych automatoch podľa [15] popísaný v predošlej kapitole, pričom zabezpečuje prispôsobiteľnosť tohto modelu, ako aj ľahkú modifikovateľnosť a experimentálne vyhodnotenie pokusov. Z implementačného hľadiska teda buduje podobný výstup ako [4], i keď v našom prípade dôraz je viac kladený na rýchlosť, efektivitu, flexibilitu a modulárnosť systému.

2. pokus o nájdenie vylepšených metód z hľadiska celulárnej reprezentácie v rámci evolučného dizajnu s dôrazom na využitie neregulárnosti v štruktúre automatu, ako aj vysporiadanie sa s ňou pri aplikovaní alebo modifikácie ideí predchádzajúcich prác v oblasti - ešte ďalej implementácia a overenie úspešnosti týchto metód.

Táto kapitola sa zaoberá druhým z týchto dvoch bodov a zhrnie všetky naše modifikácie a potenciálne vylepšenia, pričom nasledujúca sa zaoberá implementačnými detailami.

4.1 Geometria buniek

Vylepšenie práce Hlaváčikovej [20] spravením malého kroku v smere neregulárnej celulárnej reprezentácie čisto demonštruje dôležitosť voľby geometrickej štruktúry buniek z pohľadu rýchlosti konvergencie. Vhodná reprezentácia na úrovni buniek

zredukuje ich počet v organizme a tým pádom značne zjednodušuje celý automat. V ideálnom prípade ho zjednodušuje nielen štruktúrovo ale aj v komplexite prepisovacieho pravidla - aj keď to príde s cenou zavedenia netriviálneho problému do evolučného procesu v podobe nájdenia vhodných pravidiel na iteratívne prerobenie topológie automatu, spolu s modifikáciou účelovej funkcie na ohodnotenie tejto činnosti.
Toto zjednodušenie môže mať pozitívny vplyv aj na škálovateľnosť systému, vytvorenie určitých vzorov v priestore inej veľkosti ako tá, ktorá bola použitá počas evolúcie môže vyžadovať rovnaké pravidlá, hlavne ak vzor je dostatočne regulárny a geometria umožňuje vyskladanie cieľového vzoru z malého počtu elementov, ktorých počet navyše ostane nezmenený aj po škálovaní. Táto vlastnosť modelu v predošlých prácach nebola preskúmaná a budeme jej venovať pozornosť.
Práca overuje vhodnoť dvoch reprezentácií na celulárnej úrovni, ktoré majú potenciál dosiahnuť lepšie výsledky ako predošlé pokusy: bunky *obdĺžníkového tvaru* a bunky ako *regióny vo Voronoiovom diagrame.*

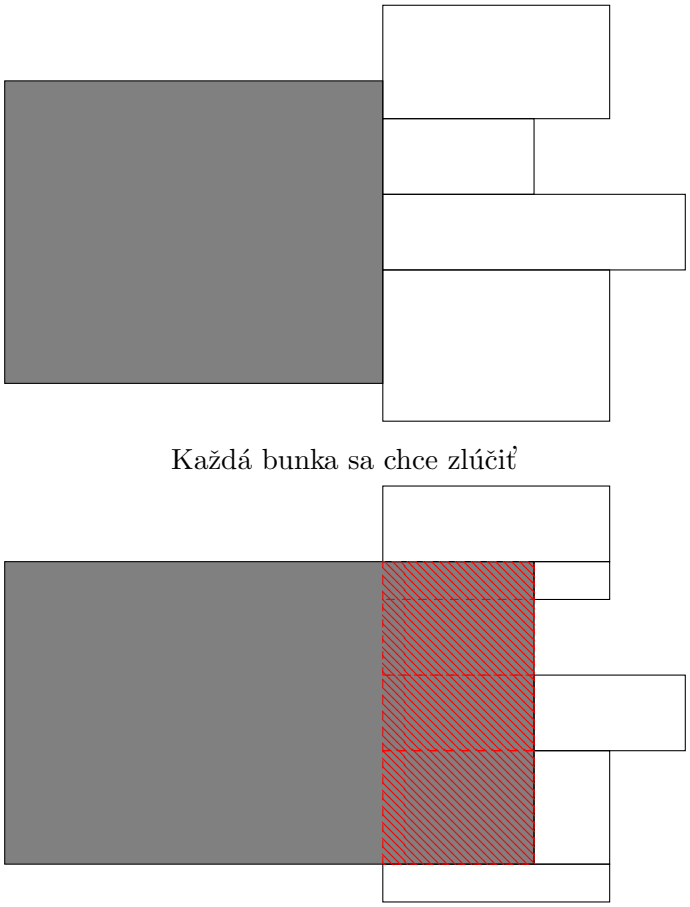
4.1.1 Obdĺžníková reprezentácia

Odstránenie obmedzenia kladeného na bunky ohľadom dimenzionality pri neregulárnej štvorcovej sieti zvyšuje flexibilitu operátorov na zlučovanie a delenie, a umožňuje prechodovému pravidlu vyskladanie rovnakého obrazu z menších počtov elementov. Logika na výpočet stavu pomocou váhovaných externých chemikálií (3.13) je v tomto prípade aplikovateľná bez zmien, jediným rozdielom je potrebná zmena mechanizmu zmeny štruktúry - keďže výsledný tvar bunky po spájaní už nemusí byť nutne štvorcový.
Nami navrhnuté postupy sú nasledovné:

- rozdelenie bunky môže byť realizované podobne ako pri štvorcoch, teda rozdelenie na štyri podčasti rovnakej veľkosti (alebo skoro rovnakej, ak dĺžka niektorej strany je nepárna). Drobnú zmenu predstavuje skutočnosť, že môže nastať prípad, keď iba jedna strana obdĺžníka je jednotková: vtedy sa rozdelí na dve podčasti podľa dlhšej strany a druhú dimenziu nechá jednotkovú. Ak obe strany majú dĺžku 1, nič sa nestane.

- spájanie buniek je realizované v každom okamihu len v jednom smere: bunka, ktorá sa chce spojiť si vypýta stav akcie susedných buniek a spájania nastane len v prípade, ak každá bunka v danom smere sa o to hlási. Keďže výsledkom má byť znova obdĺžník, algoritmus zistí dĺžku posunu hranice pôvodnej bunky

ako dimenziu najkratšieho suseda vo zvolenom smere a zo všetkých ostatných vyreže časť rovnakej veľkosti - zo zvyšku sa vytvorí nová bunka. Špeciálny prípad tvoria bunky na okrajoch, keďže tie môžu mať podčasť, ktorá nie je obsiahnutá spájajúcim sa obdĺžníkom a z tejto časti sa vytvorí ďalšia bunka - t.j. rozpadne sa na tri podčasti. Grafické znázornenie tejto udalosti je na obrázku 4.1.



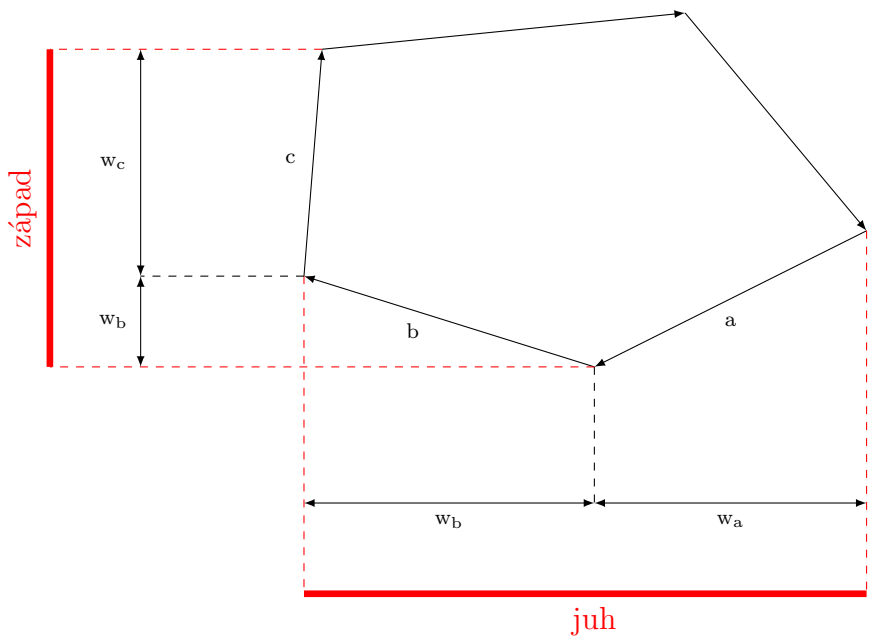
Obr. 4.1: Ukážka spájanie buniek v obdĺžníkovom celulárnom automate prvým spôsobom

- alternatívny (viac striktný) algoritmus na spájanie buniek je podobný ako pri štvorcoch: zlúčenie 4 susediacich ak majú rovnaké rozmery, doplnené s možnosťou zlúčenia s jedným susedom len v jednom smere, pokiaľ 4 sa nepodarilo zlúčiť - samozrejme za predpokladu, že každý účastník musí súhlasiť. Je to teda skoro-inverzná operácia rozdelenia.

4.1.2 Voronoiovská reprezentácia

Voronoiovská reprezentácia na rozdiel od predošlej úplne zahodí klasickú mriežkovú štruktúru a interpretuje celulárny automat ako Voronoiov diagram, pričom každý región diagramu predstavuje jednu bunku v automate. Priestorové rozloženie automatu je teda jednoznačne dané množinou generujúcich bodov, ktoré tvoria jadro pravidiel rastúcich akcií:

- pri rozpade bunky sú vytvorené štyri nové generátory v smere štyroch svetových strán, každý presne v polovici úsečky ktorá spojí bod pôvodného generátora a najvzdialenejší bod patriaci do regiónu v danom smere. Pôvodný generátor, a zároveň príslušná bunka zanikne.
- zlúčenie buniek je realizované po dvojiciach, pričom oba regióny musia s tým súhlasiť a musia mať jednu spoločnú hranu. Výsledný generátor sa potom vytvorí uprostred cesty medzi spájajúcimi sa bodmi.



Obr. 4.2: Náčrt južného a západného susedstva vo Voronoiovom diagrame a príspevku jednotlivých hrán

Okrem topologickej zmeny, Voronoiové bunky vyžadovali vlastný prístup na ošetrenie susedstiev, nakoľko v tomto prípade nie je úplne jednoznačné, že ktorý sused prispieva z danej strany - pokiaľ chceme zachovať metodiku pôvodnej práce a pokračovať použitím von Neumannovho susedstva. Keďže ku každému susedu

patrí presne jedna ohraničujúca hrana polygónu, naše riešenie spočíva vo využití *sklonu* hrany pri predpoklade orientácie hrán v smere hodinových ručičiek okolo bunky, dané vzťahom:

$$m = \frac{\Delta y}{\Delta x} = \tan\theta \tag{4.1}$$

kde θ reprezentuje uhol zvieraný úsečkou a x-ovou osou. Ak spoločná hrana má pozitívny sklon, bunka patrí do ľavého susedstva, v prípade negatívneho sklonu patrí do pravého. Ďalej, uhol na intervale $\left[\frac{\pi}{2}, \frac{3\pi}{2}\right]$ (horizontálna orientácia hrany) znamená dolného, v opačnom prípade horného suseda. Následkom toho jedna hrana (a zároveň jeden sused) môže byť súčasťou aj dvoch susedstiev, pričom jej váha je určená *dĺžkou projekcie* na príslušnú os - v prípade ľavého a pravého susedstva x-ová, inak y-ová (4.2), pričom túto hodnotu normalizuje projekcia celého susedstva. Viac formálne:

$$neighbourChemicals_*[i] = \frac{\sum_{j=1}^{n} chemicals_j[i]\, p(e_j, a)}{\sum_{k=1}^{n} p(e_k, a)} \tag{4.2}$$

kde n je počet susedov v danom smere, e_m je spoločná hrana s m-tým susedom a $p(u, a)$ je dĺžka projekcie u na príslušnú os a (skalárna projekcia), teda:

$$p(u, a) = |u| \cos\theta = u \cdot \hat{a} \tag{4.3}$$

kde θ je uhol medzi u a a, $\hat{a}$ je jednotkový vektor v smere a, operátor $\cdot$ tu indikuje skalárny súčin vektorov.

4.2 Viacúčelovosť

Potreba vyjadrenia vhodnosti jedinca pomocou viacerých účelových funkcií už pribudla aj v [20] po zavedení neregulárnosti, ale vtedy problém bol riešený kombináciou hodnôt jednotlivých účelov, pričom z pohľadu evolučného algoritmu stále išlo a jednoúčelovú optimalizáciu a ostal bez modifikácie. V našej práci viacúčelovosť je priamo riešená na úrovni optimalizátora pomocou nasledujúcich účelových funkcií:

- vzdialenosť farby jednotlivých pixlov vygenerovaného (*gen*) a cieľového (*target*) obrázku

$$f_1(gen) = \frac{\sum_{x=0}^{width} \sum_{y=0}^{height} \sum_{r,g,b} |col_{gen}(x, y) - col_{target}(x, y)|}{3 \cdot 255 \cdot width \cdot height} \tag{4.4}$$

- počet iterácií automatu, kým dokonvergoval do stabilného stavu (podľa 3.5)

$$f_2(CA) = \frac{N_{\text{iters}}(CA)}{limit} \tag{4.5}$$

- počet buniek automatu

$$f_3(CA) = \frac{N_{\text{cell}}(CA)}{width \cdot height} \tag{4.6}$$

Každá funkcia je minimalizačná s pridelenou váhou a najlepší jedinec je vybratý z populácie P s najmenšou hodnotou podľa

$$\min_{x \in P} \sum_{i=1}^{n} w_i f_i(x) \tag{4.7}$$

Na optimalizáciu sme používali implementáciu viacúčelovej varianty CMA-ES v knižnici *Shark Machine Learning*. Zaujímavou vlastnosťou algoritmu je jeho elitistická selekcia podľa fitness jedincov a hustoty populácie [14]: jedince sú ohodnotené podľa pareto dominancie do jednotlivých tried, v prvej triede sú všetky jedince ktoré nie sú dominované žiadnym iným jedincom, v druhej všetky ktoré nie sú dominované žiadnym iným jedincom okrem prvej triedy atď... Následne sú vybratí rodičia pomocou binárnej turnajovej selekcie a rozhoduje sa pareto trieda jedinca (menšie číslo je lepšie) - v prípade zhody je vybratý jedinec s menšou hustotou okolia. O hustote sa rozhoduje veľkosť najväčšieho kvádra okolo daného jedinca, ktorý neobsiahne žiadneho iného jedinca (čím väčší je kváder, tým menšia je hodnota hustoty okolia).

4.3 Ďalšie modifikácie

- možnosť generovania farebných obrázkov pomocou rozšíreného riadiacieho prvku o dva dodatočné neuróny na výstupnej vrstve, ktoré spolu s pôvodnou predstavujú *rgb* hodnotu farby

- rozhodnutie o vybratí rastúcej akcie predošlá práca riešila rozdelením oboru hodnôt aktivačnej funkcie neurónovej siete na rovnako veľké podintervaly a vybrala akciu v závislosti od toho, že do ktorého z nich padla vygenerovaná hodnota (3.14).

  Považujeme tento postup nevhodný v tom, že nie je dostatočne flexibilný, v závislosti od daného problému sa môže stať, že by bolo lepšie priradiť väčšiu váhu spájacej akcii ako rozdelovacej, alebo naopak, uprednostňovať

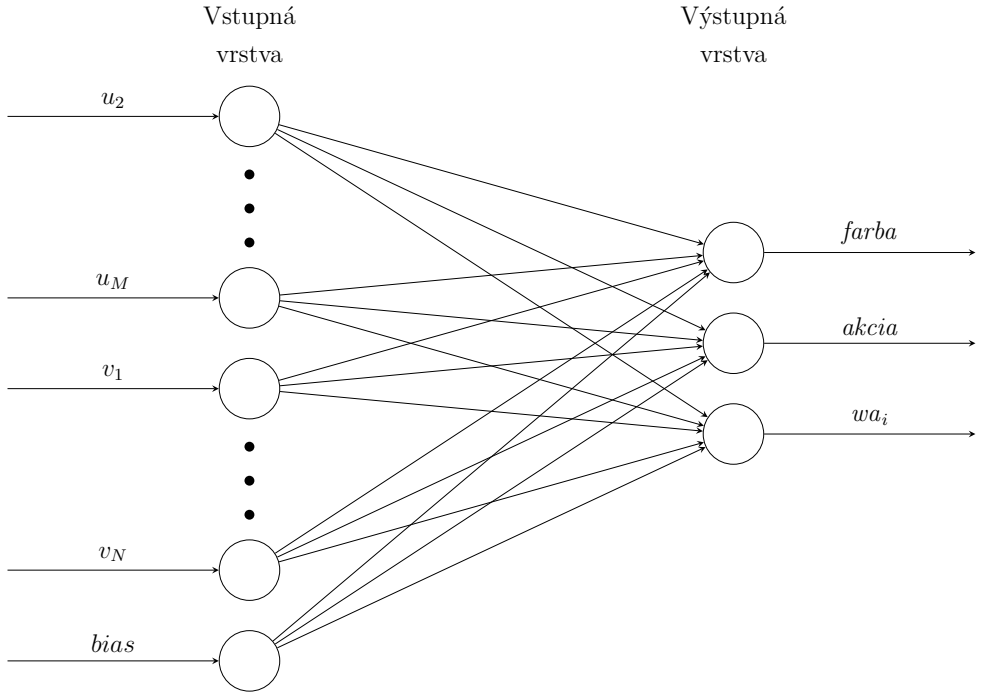
aktualizáciu, ktorá zriedkavo mení štrukúru. Navyše, napriek rovnako veľkým intervalom metóda nie je spravodlivá, pretože pravidlá na zlúčenie majú zvyčajne viac predpokladov (viac buniek sa musí o to rovnako rozhodnúť, pričom rozpadnutie závisí len od jednej) a teda menšiu šancu na vykonanie. Keďže manuálne nájdenie týchto hraníc by bolo ťažkopádne a môže to závisieť aj od konkrétneho problému, rozhodli sme si tento proces zabudovať do genotypu: každá akcia (všetky tri) na výstupnej vrstve dostane svoj neurón na rozhodnutie o váhe danej akcie - nech sa to volá wa_i (vypočítaná rovnako ako 3.14). Podľa týchto akcií sa vytvorí diskrétna pravdepodobnostná distribúcia, ktorá každej akcii priradí pravdepodobnosť nastatia úmernú k wa_i. Pomocou týchto pravdepobností sa rozdelí interval na tri podintervaly a klasifikuje sa vygenerovaná hodnota akcie podľa vypočítaných hraníc - nech veľkosť podintervalu pre danú akciu je ra_i.

$$ra_i = \frac{range \cdot wa_i}{\sum_j wa_j} \tag{4.8}$$

, kde *range* je veľkosť intervalu - v našom prípade 2, kedže používame hyperbolický tangent na aktiváciu neurónov.
Tento postup je viac dynamický v zmysle, že hranice sa môžu meniť aj počas iterácie automatu.

- drobná zmena koeficientov matice gaussovského filtra kvôli konzistencii

$$\begin{pmatrix} 0 & \frac{1}{8} & 0 \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ 0 & \frac{1}{8} & 0 \end{pmatrix} \tag{4.9}$$



Obr. 4.3: Perceptrón počítajúci farbu a akciu bunky

KAPITOLA 5

Softvér na realizáciu experimentov

Motivácia za vytvorením nového rámcového prostredia na vykonanie experimentov vychádzala zo skutočnosti, že v oboch predošlých implementáciách chýbali niektoré, nami želané vlastnosti, pričom ich dodatočné dopĺňanie v oboch prípadoch by sa stretlo s prekážkami: Devertová implementácia nebola dostatočne pripravená na rozšíriteľnosť modelu v zmysle predošlej kapitoly, a Bekov kód (aj keď sa už predošlý problém snažil riešiť), okrem nekompatibility s našími ideami v niektorých prípadoch, mal nedostatky z technického hľadiska - v prvom rade vďaka technologickému riešeniu, a keďže experimentácia s evolučnými algoritmami je vo všeobecnosti silne výpočtovo náročná príležitosť, radšej sme sa vybrali inou cestou. Naprogramovanie vlastnej implementácie teda bolo lákavé - vytvoriť systém, ktorý kombinuje pozitíva oboch predchodcov:

- efektívnosť a rýchlosť prvej
- modulárnosť a ľahkú rozšíriteľnosť druhej

5.1 Technologické pozadie

Druhý bod je samozrejme nezávislý od použitej technológie, skôr od vhodného návrhu systému - UML diagram tried je zachytený na 5.1.
Pri realizácii prvého je užitočné mať čo najviac nízkoúrovňový programovací jazyk, ktorý ešte má podporu na vysokoúrovňové koncepty potrebné pre druhý bod (hierarchia tried, dedičnosť, atď...). Rozhodli sme sa teda pri C++ vo viacvláknovom prostredí, nakoľko ťažisková časť vyhodnocovania jedincov populácie je simulácia automatu pomocou vygenerovanej prechodovej funkcie, a aktualizácia stavu buniek je skvelá možnosť na paralelizáciu.

Kód je teda napísaný v C++ (použitím syntaxe C++11) - okrem malej pomocnej aplikácie na generovanie cieľových vzorov v C# - a je dostupný (spolu s návodom na kompiláciu a spustenie v priloženom *readme.txt*) na repozitári https://github.com/dobsapetike/Cellular-Pattern-Evolution.
Bol vyvíjaný pod platformom *Windows* (ale kód je platformovo nezávislý), použitím knižníc *Shark Machine Learning* (implementácia viacúčelovej CMA-ES), *FreeImage* (generovanie obrázkov) a *TinyXML* (parsovanie vstupných súborov) - ďalej sú použité nástroje *gnuplot* na kreslenie grafu vývoja najlepšieho jedinca počas evolúcie a *ffmpeg* na generovanie videa simulácie.

5.2 Štruktúra programu

Implementácia je štrukturovaná do nasledujúcich hlavných častí (modulov), komunikujúcich so zvyškom systému cez spoločný *interface*:

- *controller* - implementácia riadiacieho prvku (3.4), jeho úlohou je poskytnutie parametrov kódovania prechodovej funkcie evolučnému algoritmu vo forme reálneho vektora a realizácia výpočtu nasledujúceho stavu bunky. V našom prípade ide o dopredný perceptrón.
- *phenotype* - implementácia celulárneho automatu a všetky výpočty okolo nich, spolu s abstraktnou triedou pre špecifické bunky *lattice_cell.*
- *optimizer* - implementácia optimalizačného algoritmu, vyžaduje sa od neho aby bol krokovateľný a po každom kroku (generácii) poskytol pareto frontu potenciálnych riešení.
- *multiobjective_function* - vykoná vyhodnotenie jedinca podľa zadefinovaných účelových funkcií, odvodené od triedy *objective_function.*
- *stop_criterion* - terminačná podmienka simulácie automatu.
- *lattice* - zapuzdruje fenotyp a genotyp, zabezpečí simuláciu celulárnych automatov.
- *task* - manažuje uskutočnenie jedného experimentu.

Konkrétne definície týchto interface-ov, spolu s triedami ktoré ich implementujú sú taktiež uvedené v diagrame 5.1.

5.3 Konfiguračné nastavenia

Každá časť systému má svoj konfiguračný súbor vo formáte *.xml*, pričom načítanie špecifických nastavení sú nechané na prekrytie (*override*) - t.j. rôzne implementácie sú ľahko parametrizovateľné cez ten istý súbor. V nasledujúcich sekciách predstavíme jednotlivé konfiguračné súbory.

5.3.1 Konfigurácia experimentu

Hlavná konfigurácia v ktorom sa nachádza rozpis zoznamu všetkých plánovaných experimentov a cesty k ostatným konfiguračným súborom. Jeho relatívna cesta vzhľadom na umiestnenie spustiteľného súboru je jediný parameter aplikácie (ak nie je poskytnutá, používa sa predvolená cesta *config/confExperiment.xml*). Jeho štruktúra:

```
<Experiments>
  <Experiment>
    <Name>ExperimentName</Name>
    <Id>1</Id>
    <RunCount>5</RunCount>
    <Controller>Config/confController.xml</Controller>
    <Lattice>Config/confLattice.xml</Lattice>
    <Objective>Config/confObjective.xml</Objective>
    <Optimizer>Config/confOptimizer.xml</Optimizer>
    <StopCriterion>Config/confStopCriterion.xml</StopCriterion>
    <VideoFPS>30</VideoFPS>
    <GenerateVideo>1</GenerateVideo>
    <SimulateOnly>0</SimulateOnly>
    <ObservedRuns>5</ObservedRuns>
    <Plot>1</Plot>
  </Experiment>
  ...
  <Experiment>
        ...
  </Experiment>
</Experiments>
```

- *Name, Id* - identifikácia experimentu, obe musia byť unikátne

- *Controller, Lattice, Objective, Optimizer, StopCriterion* - cesty k ostatným konfiguračným súborom
- *RunCount* - počet spustení toho istého experimentu - pri vygenerovaných súboroch (obrázky, logy, ...) sú odlíšené v názve s číslom behu v zátvorke
- *VideoFPS* - počet snímok za sekundu vo výslednom videu
- *GenerateVideo* - prepínač (0,1) na generovanie videa
- *SimulateOnly* - prepínač (0,1) na spustenie výlučne simulácie automatu s nastaveniami definovanými v ostatných súboroch, bez optimalizácie parametrov
- *ObservedRuns* - v istých generáciách sa spustí tzv. *observed run* s najlepším jedincom populácie, počas ktorého sa logujú rôzne štatistiky (ako vývoj fitness podľa farby, počet rastúcich akcií, vývoj energie, atď. ...), tu sa dá nastaviť počet takýchto simulácií (maximum je počet dovolených generácií). Takéto behy sú potom vykonané periodicky po rovnakých intervaloch.
- *Plot* - prepínač (0,1) na kreslenie grafu o priebehu vývoja fitness najlepšieho jedinca (vyžaduje nainštalovaný *gnuplot*)

5.3.2 Konfigurácia riadiacieho prvku

```
<Controller>
  <Type>FeedForwardANNController</Type>
  ...
</Controller>
```

Riadiaci prvok má jediný všeobecný parameter: typ, všetky ostatné závisia od tohto typu. MLP má nasledujúce:

- *HiddenNeuronCount* - počet skrytých neurónov
- *ColorType* - typ použitej farby, *grayscale* alebo *rgb*
- *FixedIntervals* - prepínač (0, 1) na použitie premenlivých hraníc na zistenie rastúcej akcie 4.3
- *Params* - váhy siete ako zoznam reálnych čísel oddelených medzerami. Ak nie sú poskytnuté (prázdny tag), tak sú použité pseudonáhodné čísla.

5.3.3 Konfigurácia automatu

```
<LatticeSettings>
  <LatticeWidth>32</LatticeWidth>
  <LatticeHeight>32</LatticeHeight>
  <CellType>Voronoi</CellType>
  <PhenotypeSettings></PhenotypeSettings>
  <AllowStructureChange>1</AllowStructureChange>
  <TargetPattern>Target/target.targetpattern</TargetPattern>
  <InitPattern></InitPattern>
  <State>
    <InternalCount>10</InternalCount>
    <ExternalCount>8</ExternalCount>
    <InitialValues>0</InitialValues>
  </State>
</LatticeSettings>
```

- *LatticeWidth, LatticeHeight* - dimenzie automatu
- *CellType* - geometria buniek, možnosti: *Regular, IrregularSquare, IrregulaRectangle, IrregularRectangle_S, Voronoi*
- *PhenotypeSettings* - dodatočné parametre fenotypu (ak potrebné)
- *AllowStructureChange* - prepínač (0,1) na dovolenie topologických zmien (použitie rastúcich akcií)
- *InitPattern* - iniciálny stav štruktúry, v prípade štvorcov a obdĺžníkov je to zoznam štvoríc celých čísel `"x y šírka výška"` v prípade Voronoi je to zoznam súradníc generátorov `"x y"` - všetky oddelené medzerami
- *State/{InternalCount, ExternalCount, InitialValues}* - počet interných a externých chemikálií a ich počiatočná hodnota
- *TargetPattern* - cesta k súboru obsahujúceho cieľový vzor. Má formát *svg* a jeden hlavný element `svg` so šírkou, výškou (musia sa zhodovať s hodnotami z konfigu) a farbou pozadia, obsahujúci zoznam polygónov, ďalej každý polygón obsahuje zoznam bodov (atribút *points*) vo formáte `"x y"` oddelené čiarami a farbu (atribút *fill*) v formáte `rgb(r,g,b)`

5.3.4 Konfigurácia optimalizátora

```
<OptimizerSettings>
  <Algorithm>MOCMAES</Algorithm>
  <Params> ... </Params>
</OptimizerSettings>
```

Obsah `Params` závisí od algoritmu - zatiaľ jediný je CMA-ES s parametrami:

- *InitStepSize* - začiatočná veľkosť kroku 2.4
- *ParentSelectionCount* - veľkosť populácie
- *EvolutionStepCount* - počet generácií evolúcie

5.3.5 Konfigurácia účelových funkcií

```
<ObjectiveFunctions>
  <Function Name="ColorDist" Importance="100"/>
    ...
</ObjectiveFunctions>
```

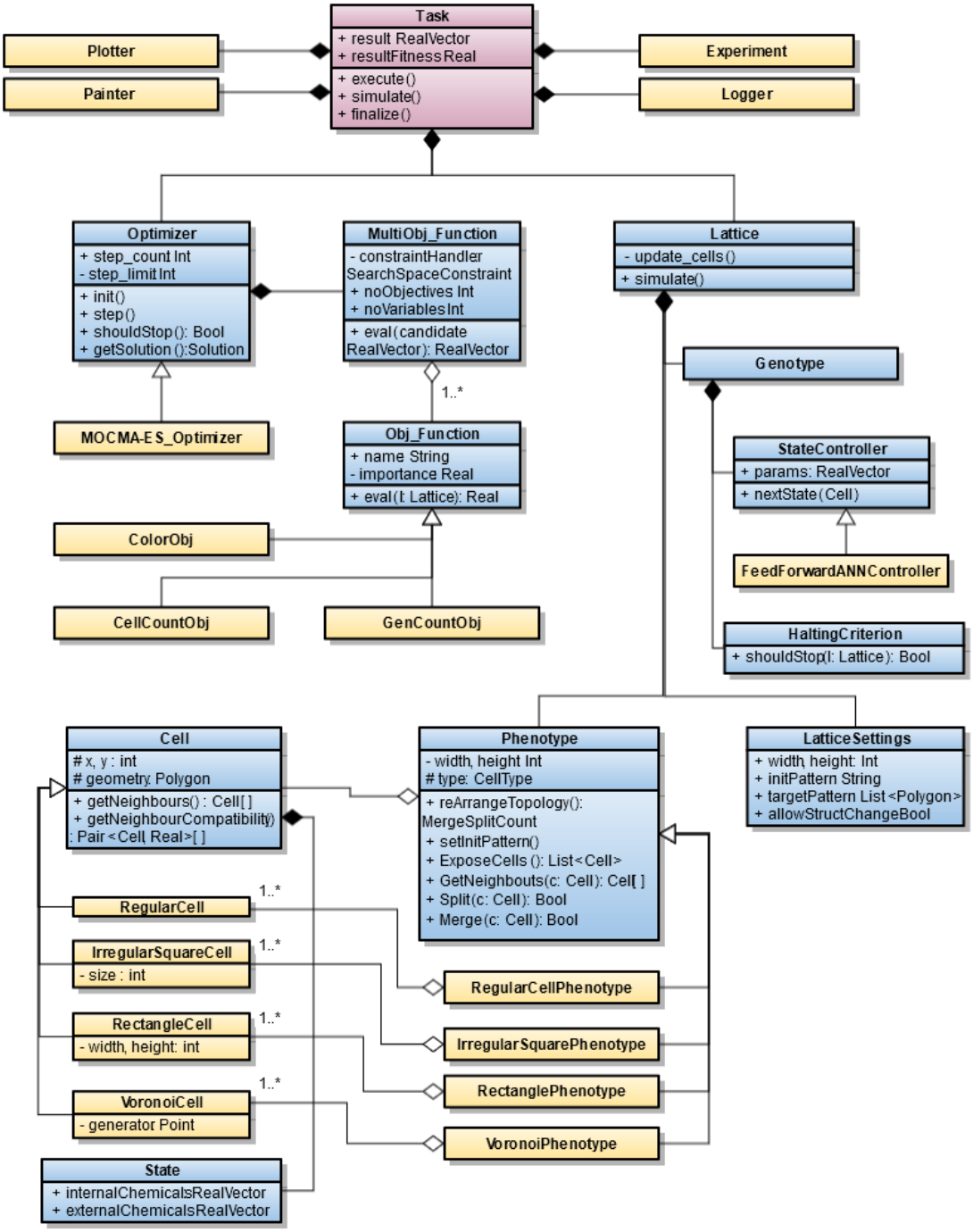
Typ účelovej funkcie je identifikovaný podľa názvu: *ColorDist, CellCount* alebo *GenCount*, pričom váha sa nastavuje cez *Importance* - príspevok je vypočítaný ako $fitness \cdot importance \cdot 10^{-2}$. Váha nie je zhora ohraničená.

5.3.6 Konfigurácia kritéria ukončenia iterácie automatu

```
<StopCriterion>
  <Criterion>Energy</Criterion>
  ...
</StopCriterion>
```

Typ kritéria (tag `Criterion`) je buď *EvalNums* (počet iterácií) alebo *Energy* (podľa 3.3) s parametrami:

- *StepLimit* - maximálny počet iterácií (v oboch prípadoch)
- *WindowSize* - veľkosť okna do minulosti, t.j. počet hodnôt podľa ktorých sa rozhoduje o stabilite energie
- *Threshold* - ϵ v 3.5



Obr. 5.1: UML diagram tried

KAPITOLA 6

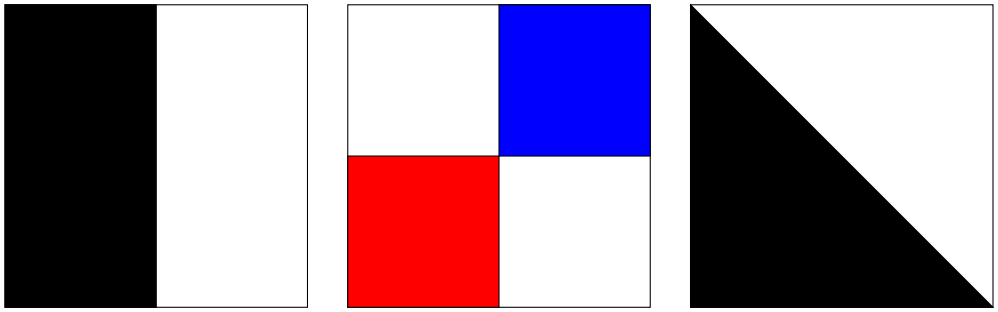
Experimenty a výsledky

V tejto kapitole opíšeme vykonané experimenty, ich parametre a detaily, a následne predstavíme a prediskutujeme získané výsledky. Najskôr uvedieme zoznam použitých skratiek, ktoré kvôli prehľadnosti používame pri označeniach v grafoch a tabuľkách:

- **COL** - optimalizácia podľa farby
- **CELL** - optimalizácia podľa počtu buniek
- **GEN** - optimalizácia podľa počtu iterácií automatu
- **ALLOBJ** - optimalizácia podľa všetkých kritérií
- **REGULAR** - automat s regulárnou štruktúrou
- **SQR** - automat s neregulárnymi štvorcami
- **RECT** - automat s neregulárnymi obdĺžníkmi používajúci nestriktné pravidlá
- **RECT_S** - automat s neregulárnymi obdĺžníkmi používajúci striktné pravidlá
- **VOR_C** - automat s Voronoiovým diagramom inicializovaný s jedným, centrálnym bodom
- **VOR_#** - automat s Voronoiovým diagramom inicializovaný s mriežkou bodov
- **FIX** - fixné hranice pri výpočte topologickej akcie
- **VAR** - premenlivé hranice pri výpočte topologickej akcie
- **TB{90,45}** - cieľový vzor Two-Bands{90,45}

6.1 Experimenty

Vykonali sme tri sady experimentov, každú sadu s iným cieľovým vzorom a v priestore veľkosti 32×32. *Two-Bands 90* je dvojica zvislých pruhov rovnakej veľkosti, ľavý čierny a pravý biely - štandardný vzor, ktorý bol použitý vo všetkých predošlých prácach; *Chess* je šachovnica skladajúca zo 4 štvorcov rovnakej veľkosti a namiesto čiernych políčok sme mali modré resp. červené, vzor zameraný na vyskladanie použitím `rgb` zložiek; *Two-Bands 45* je prvý vzor otočený o 45 stupňov za účelom zavedenia nepravých uhlov v tvaroch.
Konfiguračné parametre a nastavania pre jednotlivé sady sú detailnejšie rozpísané v ďalších sekciách.



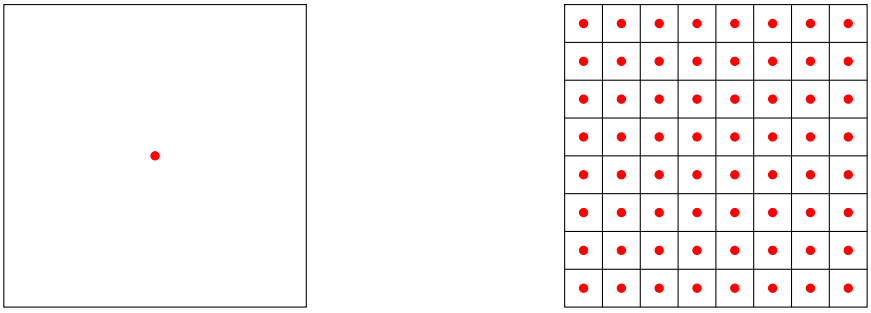
TwoBands 90 Chess TwoBands 45

Obr. 6.1: Použité cieľové vzory v experimentoch

6.2 Inicializácia

Dôležitým parametrom experimentov je inicializácia štruktúry automatu: problém, ktorý pri regulárnych bunkách sa vôbec nepribudne, pri neregulárnych môže byť celkom kriticky z pohľadu konvergencie, keďže rôzne počiatočné topológie môžu vyžadovať rôzne a rôzne zložité prechodové funkcie.
V našich experimentoch neregulárne štvorcové a obdĺžníkové automaty boli inicializované ako regulárne, skladajúce sa z $w \times h$ jednotkových buniek, kde w a h sú rozmery priestoru. Pri Voronoiových diagramoch sme skúšali dve inicializačné podmienky: s jednou bunkou na začiatku (s jediným generátorom uprostred priestoru) a s mriežkou buniek, pričom každá bunka diagramu mala štvorcový tvar veľkosti 4×4.
Počiatočný stav buniek je vektor chemikálií rovnakých hodnôt, pričom táto hodnota

je nakonfigurovateľný parameter experimentu - my sme používali nulové vektory.



Obr. 6.2: Inicializačné tvary Voronoiovej reprezentácie

6.3 Two-Bands 90

Two-Bands 90 je štandardný prípad, ktorý bol použitý aj v predošlých prácach a ktorý tvorí jadro našich experimentov, keďže väčšina behov bola spustená na tomto vzore. Boli použité externé chemikálie veľkosti 2 a jediný interný, neurónová sieť s 8 skrytými neurónmi, populácia veľkosti 100 s maximálnym počtom generácií 250, počet iterácií automatu bol zhora ohraničený na 150 a odchýlka energie mala prahovú hodnotu 10^{-8}. Každá verzia bola spustená 12 krát, samotné experimenty boli zamerané na 4 rôzne aspekty modelu:

- viacúčelovosť - skúšali sme použitie všetkých kombinácií účelových funkcií (ohodnotenie farby zrejme tam musí byť vždy) na nájdenie riešenia pre SQR. Na základe konečných tvarov (obr. 6.4) a konvergencie (obr. 6.3) vidíme, že farbu najlepšie optimalizuje verzia s jednou účelovou funkciou na farbu, napriek tomu v ďalších častiach sme použili jedine plnú viacúčelovú verziu, pretože v tomto prípade každé riešenie sa skončilo v stabilnom stave, pričom pri jednoúčelovom polovica automatov sa skončila na hornom limite a optimálnu štruktúru sa ani jedenkrát nepodarilo vygenerovať.

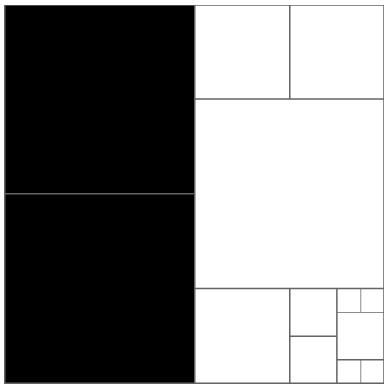
- vyhodnotenie premenlivej hranice topologických zmien - rovnako sme spustili behy na neregulárnych štvorcoch na porovnanie efektívnosti, a keďže použitie premenlivej hranice sa ukázalo byť prudko vhodnejší prístup na evolúciu (obr. 6.5, 6.7, 6.8), v ďalších behoch sme fixné hranice zanedbali.

- vyhodnotenie rôznych celulárnych reprezentácií - každý algoritmus dokázal vyriešiť úlohu efektívne a rýchlo (obr. 6.10, 6.12, 6.15), pri Voronoiových diagramoch s jedným bodom môžeme vidieť jemnú nepresnosť pri umiestnení

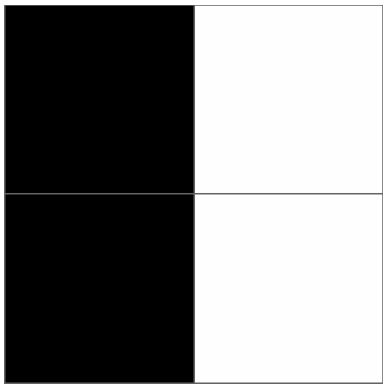
generátorov - keďže v tomto prípade hlavným operátorom topologickej zmeny je rozpadnutie, vďaka fixne danej vzdialenosti nájdenie presnej polohy môže vyžadovať viac času, alebo úplne iný spôsob delenia. Automat s regulárnou štruktúrou sme spustili len zo zaujímavosti s veľkosťou 8×8, pri takej zjednodušenej úlohe je jeho konvergencia porovnateľná s ostatnými.

- škálovateľnosť - výslednú neurónovú sieť (t.j. prechodové pravidlo) sme následne použili na simuláciu rovnakého automatu na veľkosti 8×8 a 64×64, pričom okrem Voronoiových diagramov (obr. 6.14) v každom prípade bol vygenerovaný presne rovnaký tvar. Voronoi s jednou bunkou bol úspešný len pri veľkosti 64×64, verzia s mriežkou v oboch prípadoch bola ďaleko od TB90. V tabuľke tiež vidíme, že akcia rozpadu pri obdlžníkoch a štvorcoch nedostane žiadne slovo, zrejme vďaka inicializačnej podmienky.

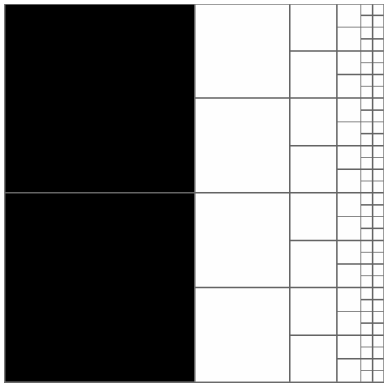
Jednotlivé behy so štvorcami a oblžníkmi trvali v priemere 8 hodín, s Voronoiovými reprezentáciami v priemere 2.



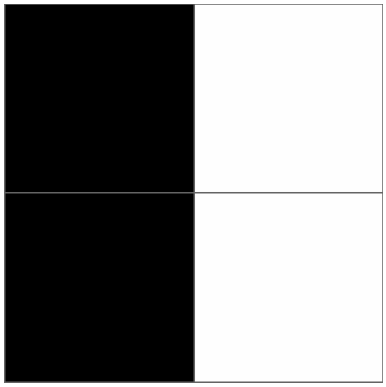
Farba



Farba a počet buniek

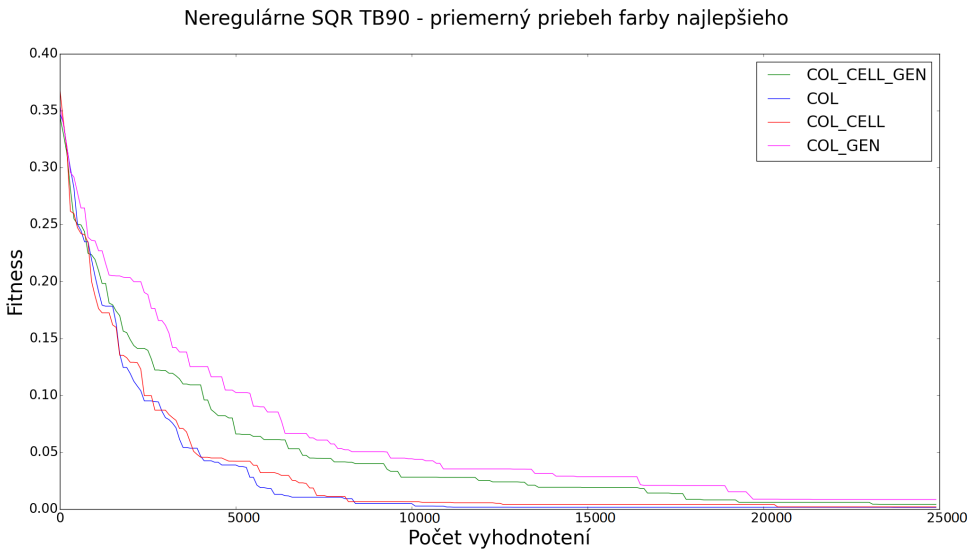


Farba a počet iterácií

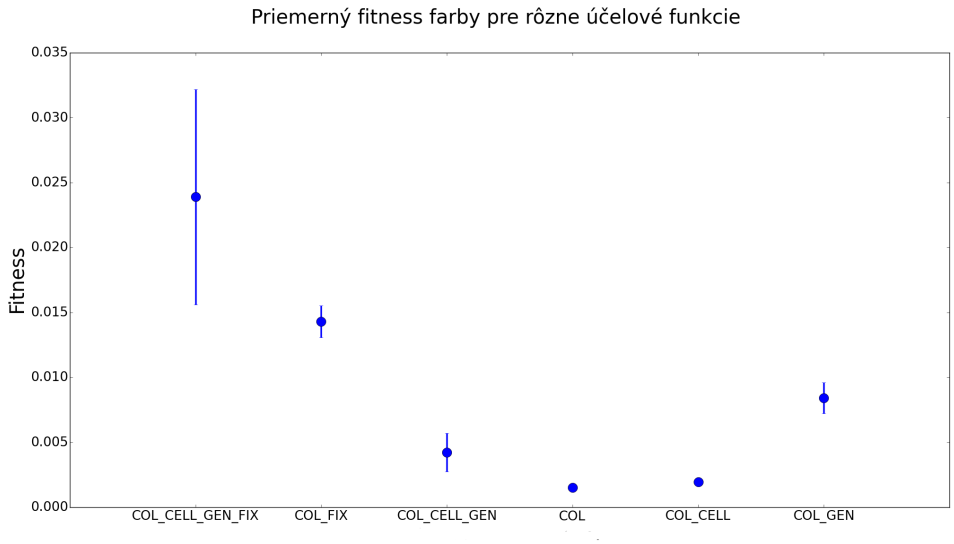


Farba, počet buniek a počet iterácií

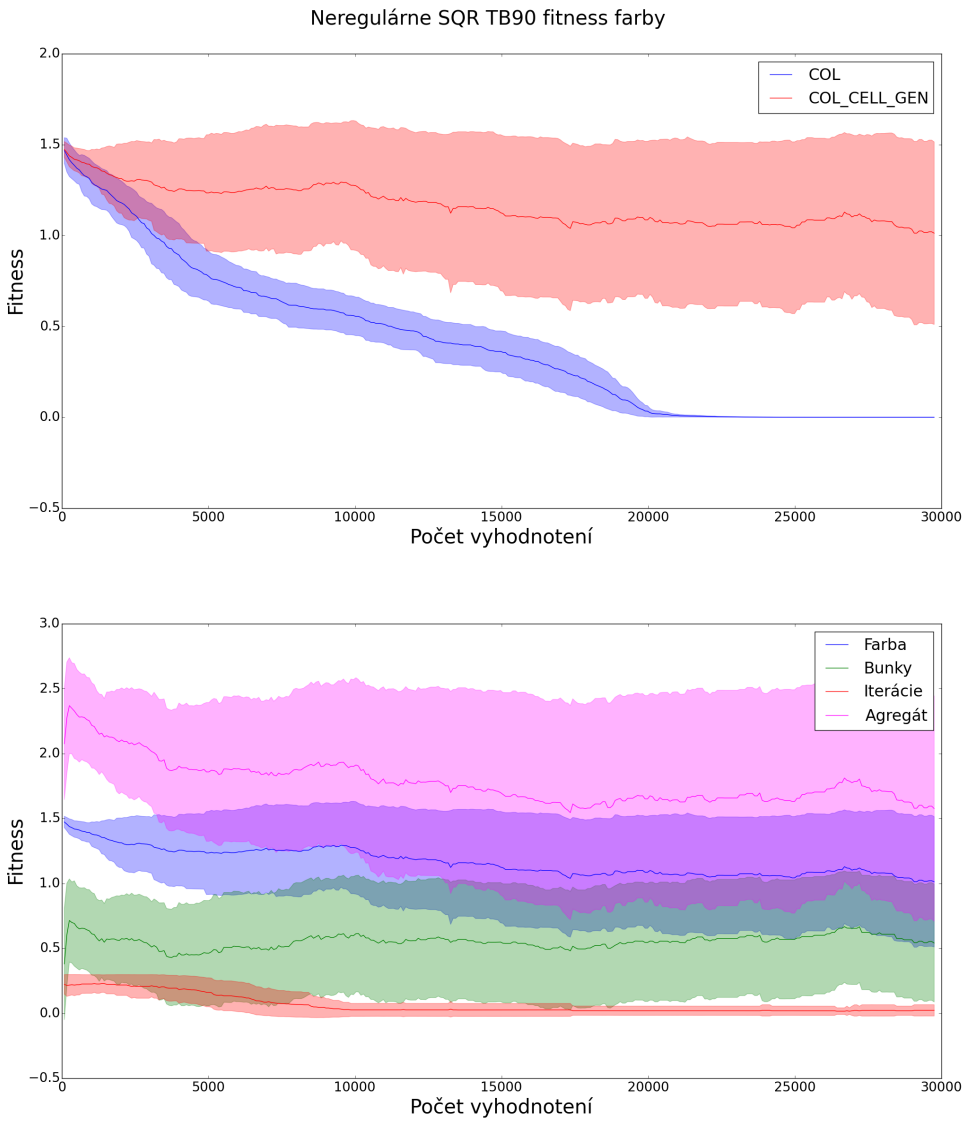
Obr. 6.3: TB90 - výsledné tvary pri použití rôznych účelových funkcií



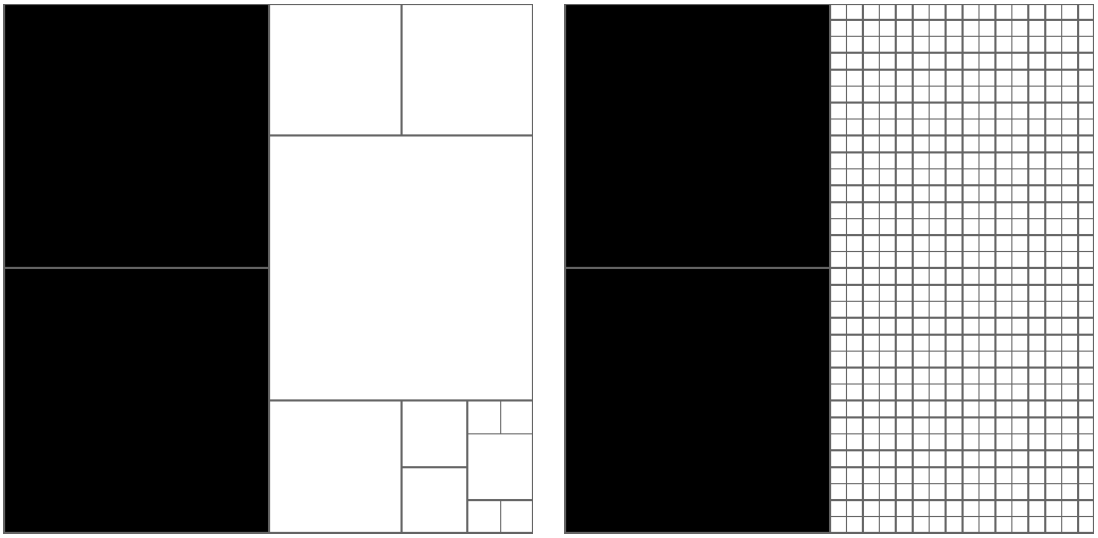
Obr. 6.4: Konvergencia priemerného fitness najlepšieho jedinca populácie (podľa farby) pre rôzne účelové funkcie



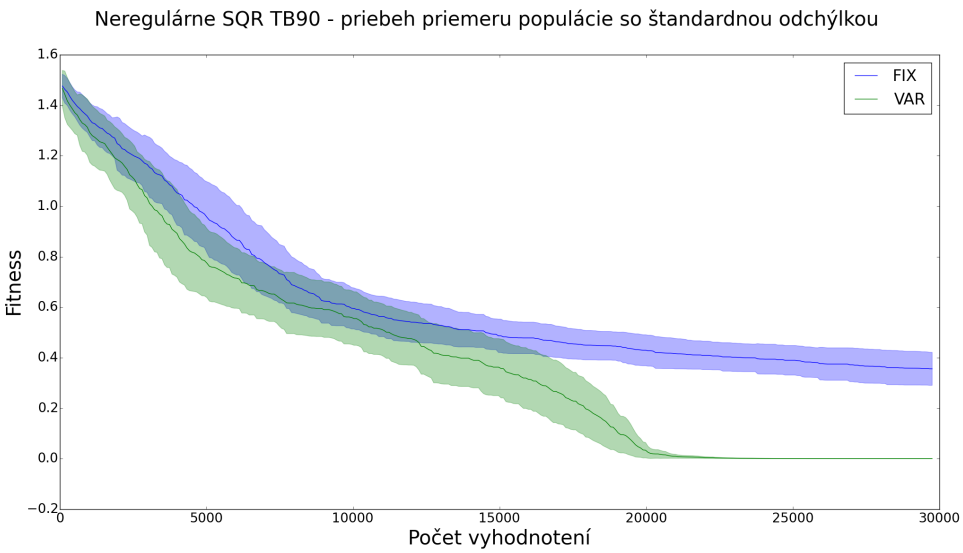
Obr. 6.5: Priemerný fitness farby výsledku pre rôzne účelové funkcie, chybová úsečka reprezentuje jednu štandardnú chybu (TB90)



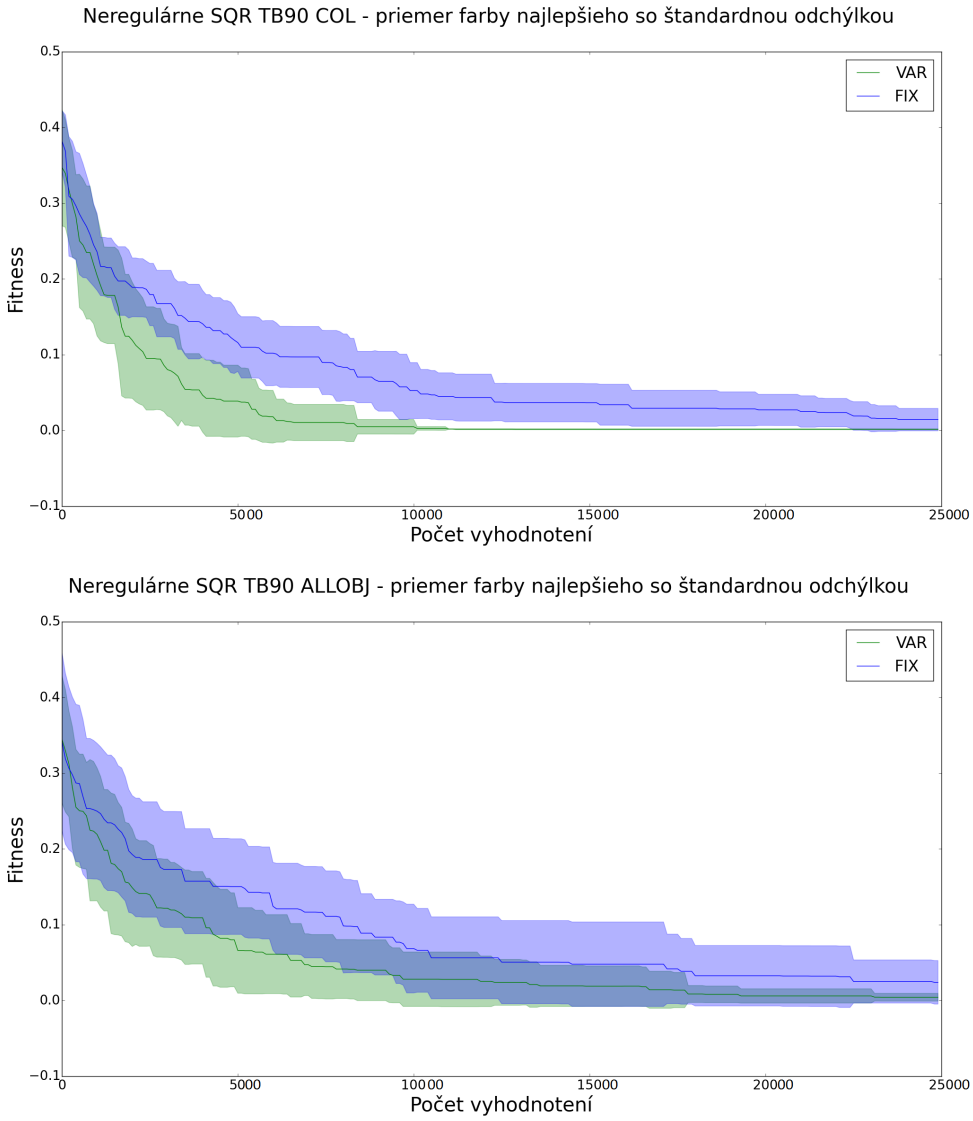
Obr. 6.9: Zaujímavé pozorovnie vplyvu viacúčelovosti pri použití rôznych počtov účelových funkcií: ak sa optimalizuje úloha len podľa jedného účelu, tak po nájdení optimálneho riešenia sa celá populácia je postupne pritiahnutá do tohto minima, pričom pri viacerých je priemer viac-menej konštantný počas celej evolúcie pre každý účel, a zvýši sa štandardná odchýlka od tohto priemeru keď je priestor na zlepšenie. Je to samozrejme priamym dôsledkom použitia pareto frontu dominujúcich sa riešení podľa rôznych účelov. Na hornom grafe je porovnanie priebehu priemeru a odchýlky toho istého účelu pre jednu- a viacúčelovú optimalizácia a na dolnom sú priebehy jednotlivých účelov počas úlohy TB90.



Obr. 6.6: Dva jedince pareto frontu reprezentujúce dva rovnako vhodné riešenia z pohľadu farby, pričom ľavý viac minimalizuje počet buniek a vpravý trvanie dosiahnutia stabilného stavu



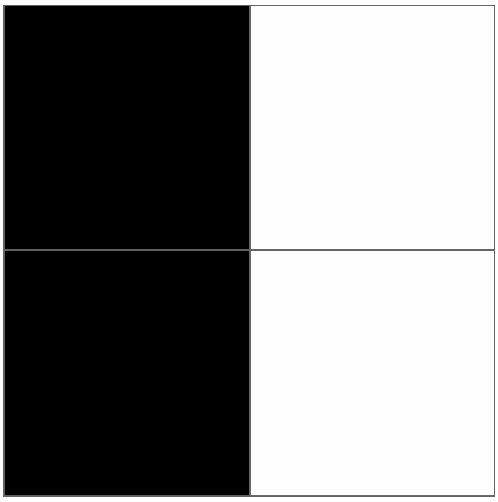
Obr. 6.7: Porovnanie použitia fixných a premenlivých hraníc pre topologické akcie. Na grafe vidíme priemer populácie najlepšieho behu.



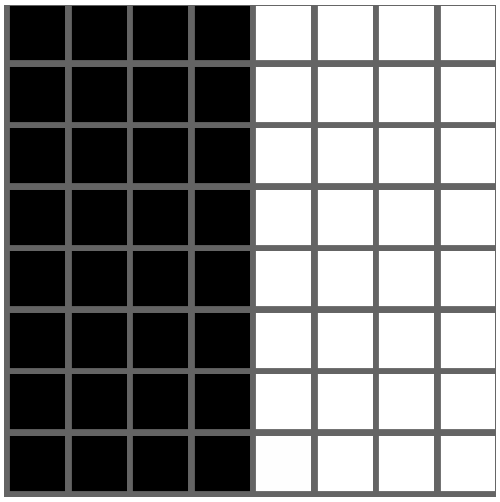
Obr. 6.8: Porovnanie použitia fixných a premenlivých hraníc pre topologické akcie. Na grafoch je zachytení priebeh priemerného fitness najlepšieho populácie.

Typ	spojenie			rozdelenie		
	8×8	32×32	64×64	8×8	32×32	64×64
SQR	21	341	1365	0	0	0
RECT_S	20	340	1364	0	0	0
VOR_C	2	14	14	2	5	5
VOR_#	3	62	254	0	0	0

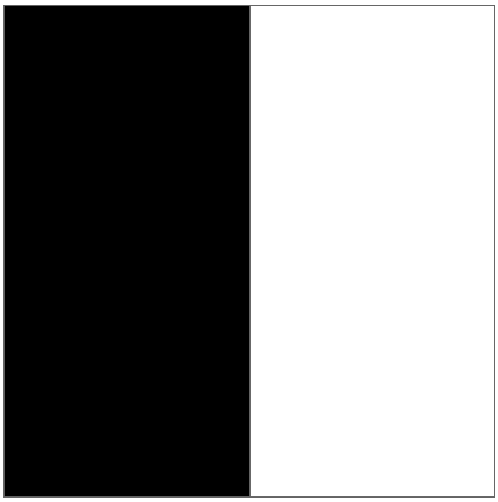
Tabuľka 6.1: TB90 - početnosť akcií topologických zmien počas simulácie automatu v priestoroch rôznej veľkosti



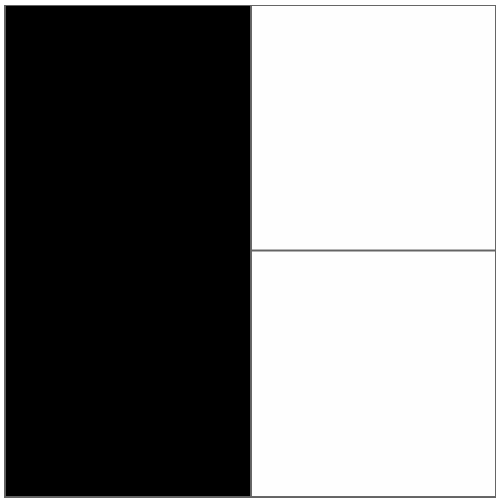
Neregulárne štvorce



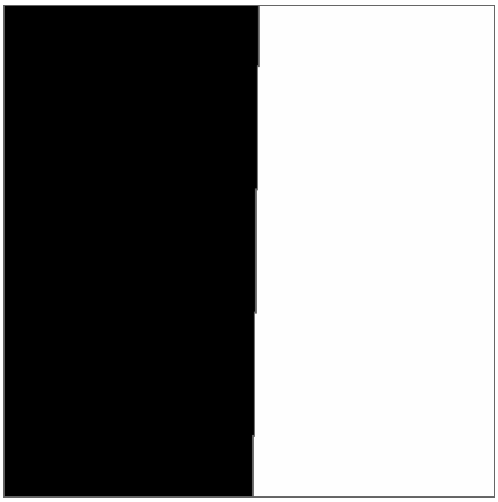
Regulárne štvorce



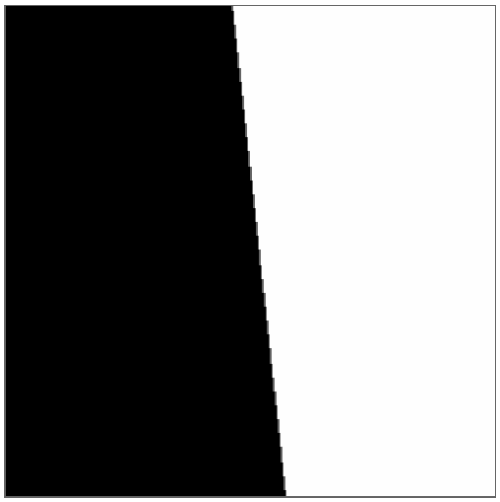
Obdĺžníky



Obdĺžníky podľa striktných pravidiel

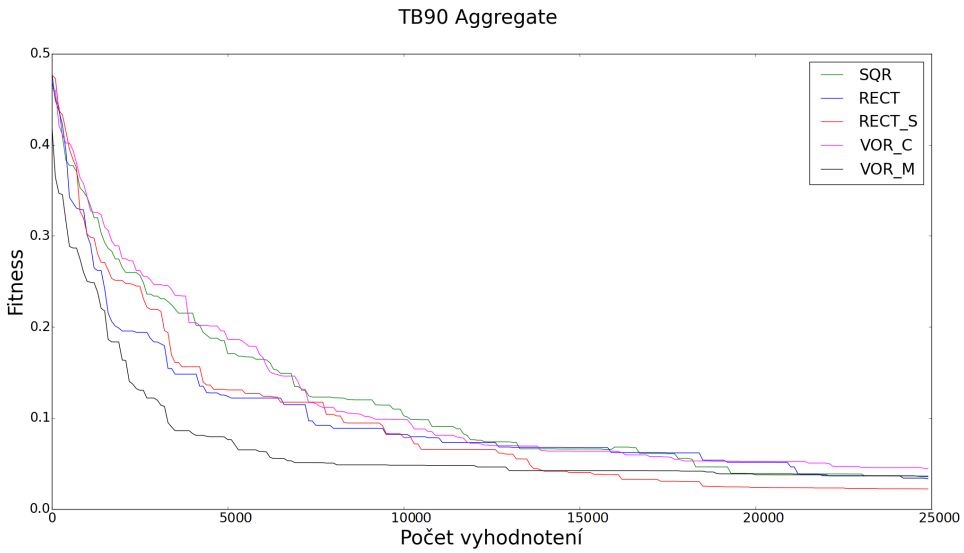


Voronoi s mriežkou

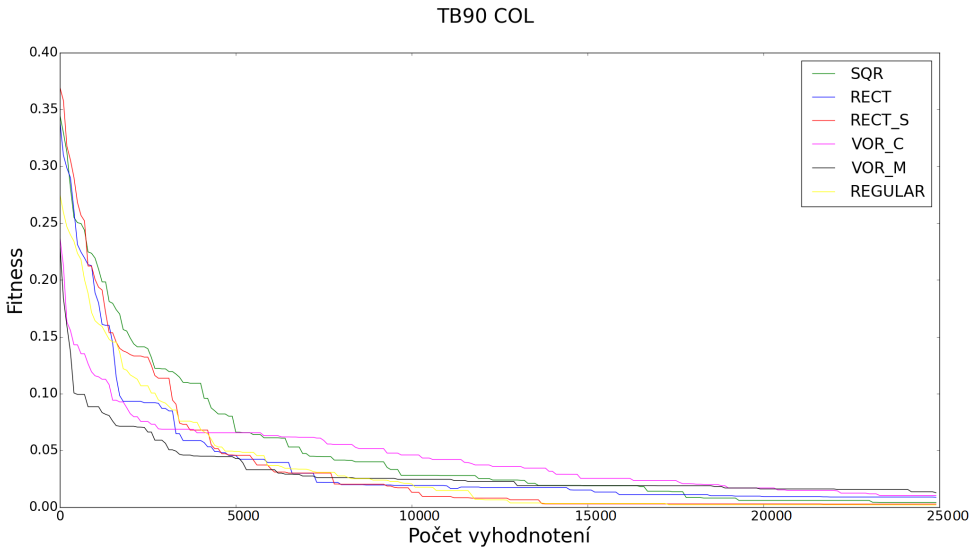


Voronoi s jedným bodom

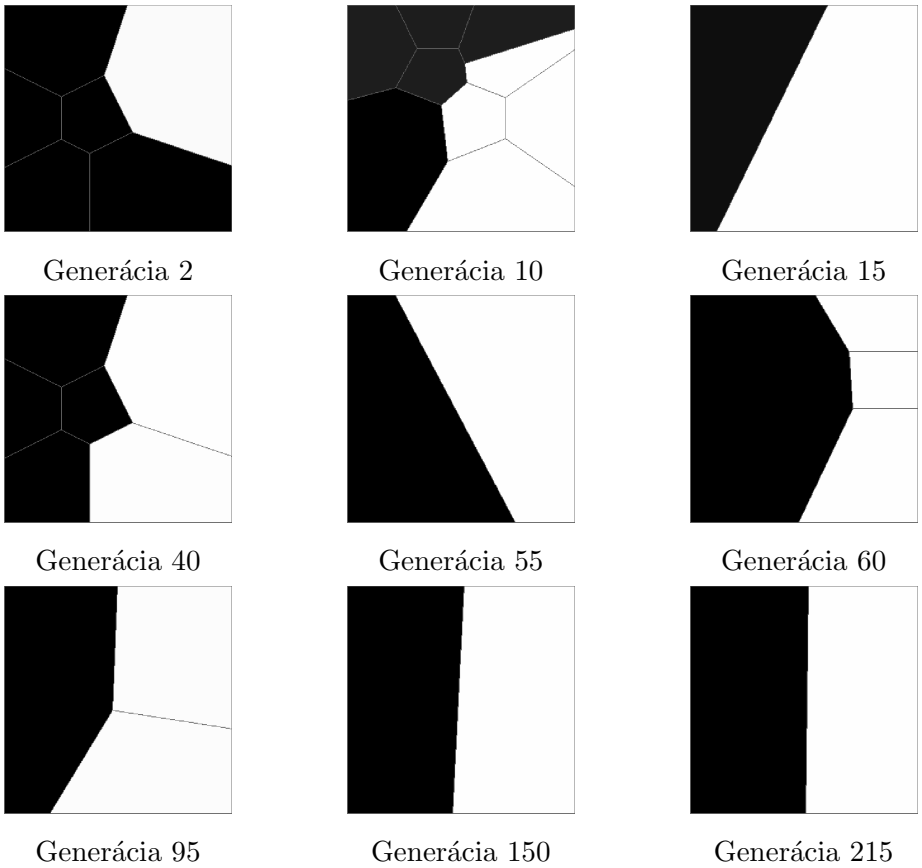
Obr. 6.10: TB90 - najlepšie riešenia pre jednotlivé algoritmy



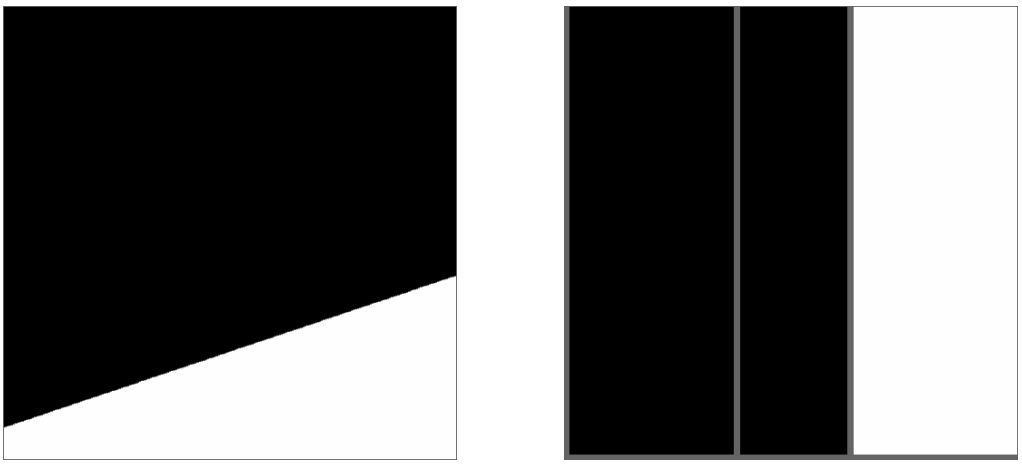
Obr. 6.11: Konvergencia priemerného fitness najlepšieho jedinca populácie (podľa aggregovaného fitness) pre jednotlivé algoritmy



Obr. 6.12: Konvergencia priemerného fitness najlepšieho jedinca populácie (podľa fitness farby) pre jednotlivé algoritmy

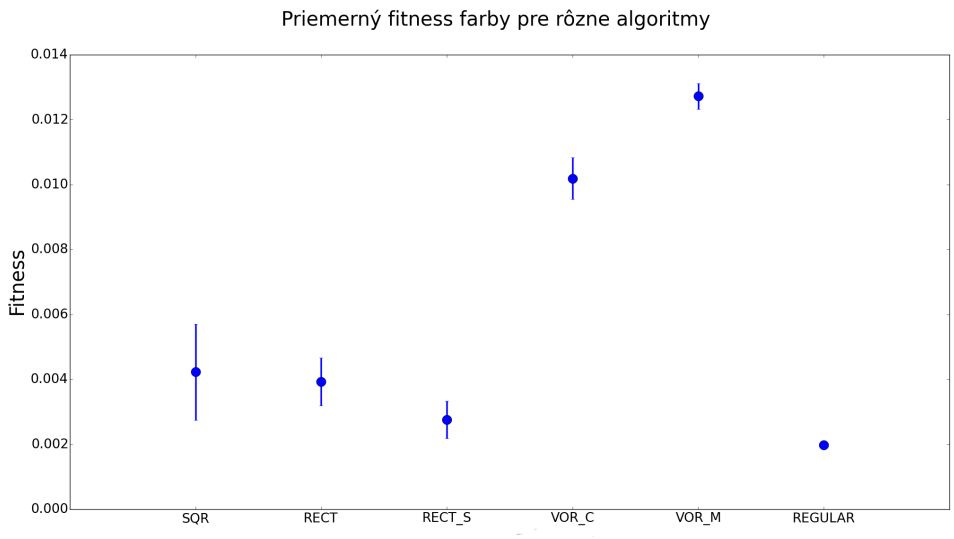


Obr. 6.13: TB90 - Vývoj najlepšieho jedinca pre Voronoi s mriežkou počas evolúcie



Voronoi s mriežkou pri veľkosti 64×64 Voronoi s jedným bodom pri veľkosti 8×8

Obr. 6.14: TB90 - Nekorektné vzory po spustení na automate s iným rozmerom

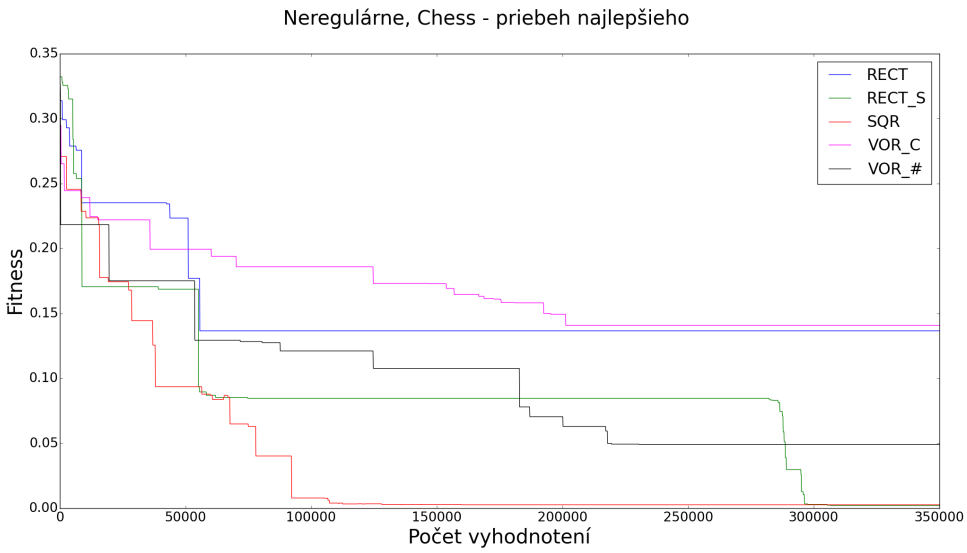


Obr. 6.15: Priemerný fitness farby výsledku, chybová úsečka reprezentuje jednu štandardnú chybu

6.4 Chess

Farebná šachovnica sa síce skladá len zo štyroch rovnako veľkých štvorcov, využitie trojice hodnôt na reprezentáciu farby značne komplikuje úlohu a vzhľadom na túto skutočnosť sme vybrali parametre modelu: boli použité externé chemikálie veľkosti 4 a interné veľkosti 3 (neurónová sieť teda mala vstup veľkosti 20), 12 neurónov na skrytej vrstve, populácia veľkosti 100 s maximálnym počtom generácií 3500, počet iterácií automatu bol zhora ohraničený na 350 a odchýlka energie mala prahovú hodnotu 10^{-8}. Výsledky sú z jedného behu.
Konvergencia najlepších jedincov pre jednotlivé algoritmy je načrtnutá na 6.16 a výsledné obrázky na 6.17. Korektné riešenie našli len modely používajúce štvorce a viac obmedzené pravidlá na obĺžníkoch. Pomerne biedny výsledok druhej verzie s obdĺžníkmi indikuje extrémnu citlivosť algoritmu na komplexitu modelu: vďaka voľnejším pravidlám a z nich vyplývajúcej veľkej množstve rôznych možností čas konvergencie sa rýchlo rastie so stúpajúcim prehľadávacím priestorom. Zjavne množina možných prechodových funkcií pre striktnejší algoritmus je podmnožina všetkých možností pre voľnejší, teda potenciál by mal na vygenerovanie lepšieho kandidáta. Rovnako aj automaty s Voronoiovým diagramov by vyžadovali viac času na nájdenie pravidiel, ktoré by viedli k presnejšej aproximácie, i keď riešenia v týchto prípadoch už viac podobajú na cieľový vzor a pravdepodobne by vyžadovali len jemné doladenie váh.

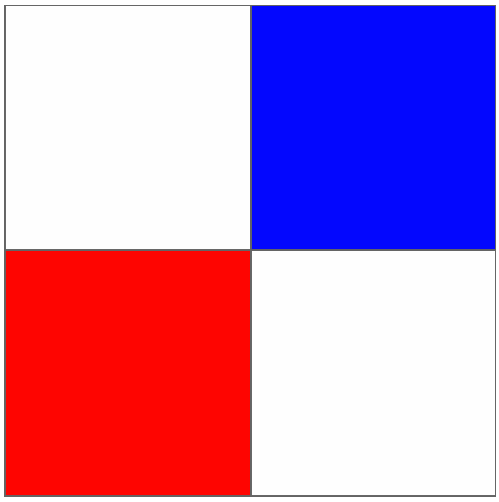
Testy škálovateľnosti dávali rovnaké výsledky ako v predošlej: štvorcové a obdĺžníkové tvary vedeli vygenerovať ten istý tvar, pričom Voronoi s mriežkou nie a Voronoi s jedným bodom len pre 64×64. SQR a RECT_S mali v každom behu rovnaké počty akcií (tabuľka 6.3), nájdená funkcia teda musí byť veľmi podobná.



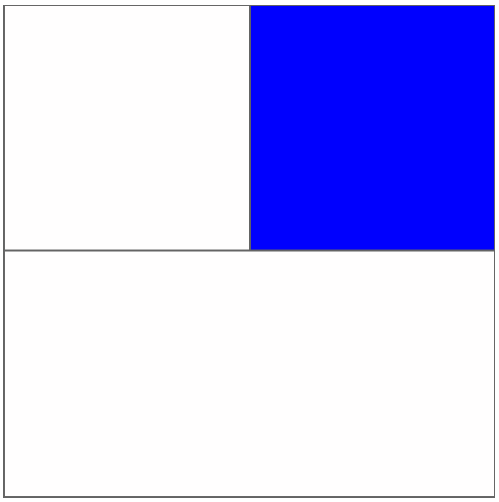
Obr. 6.16: Konvergencia priemerného fitness najlepšieho jedinca populácie (podľa aggregovaného fitness) pre jednotlivé algoritmy

Typ	spojenie			rozdelenie		
	8×8	32×32	64×64	8×8	32×32	64×64
SQR	20	340	1364	0	0	0
RECT	44	533	1876	0	3	13
RECT_S	20	340	1364	0	0	0
VOR_C	2	5	5	4	5	3
VOR_#	0	58	61	0	0	0

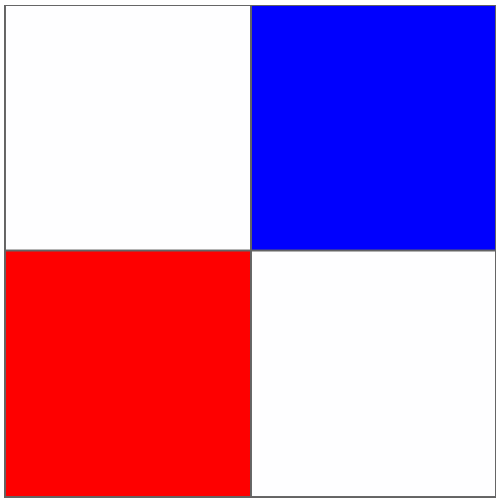
Tabuľka 6.2: Chess - početnosť akcií topologických zmien počas simulácie automatu v priestoroch rôznej veľkosti



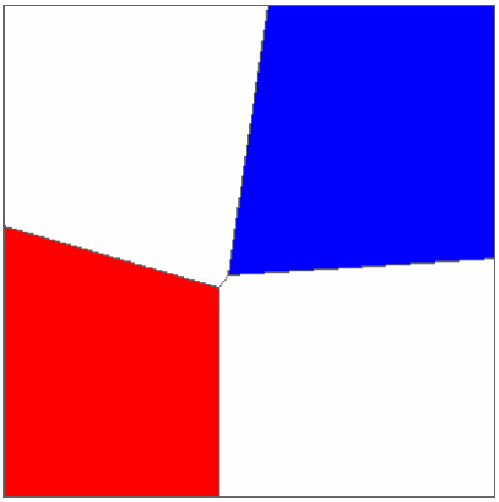
Neregulárne štvorce



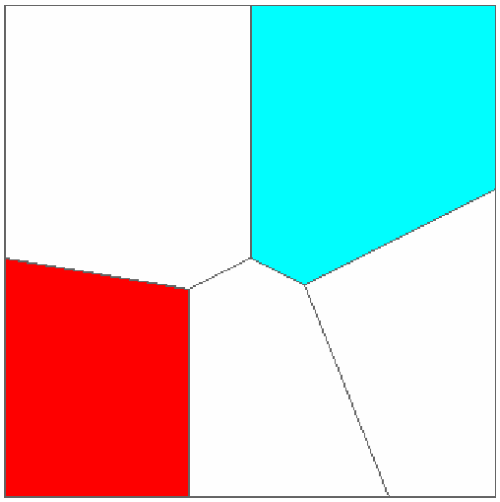
Obdĺžníky



Obdĺžníky podľa striktných pravidiel



Voronoi s mriežkou

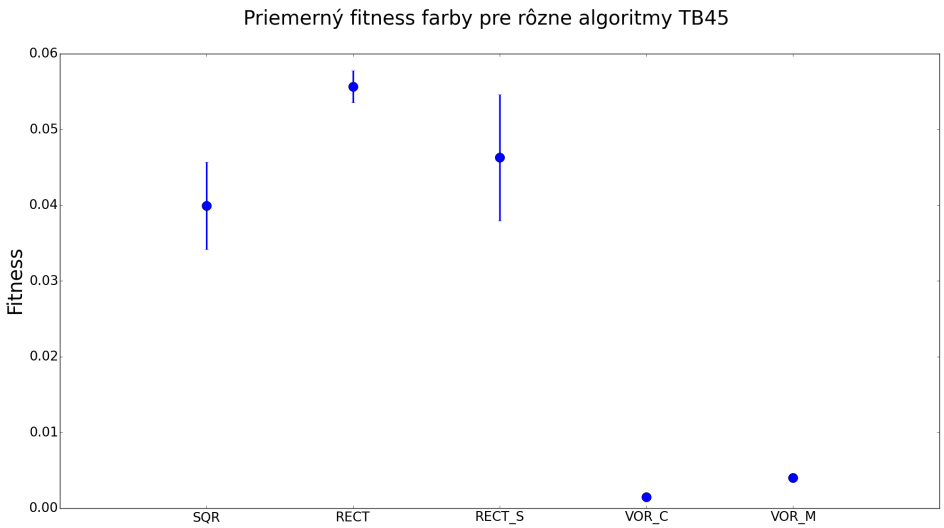


Voronoi s jedným bodom

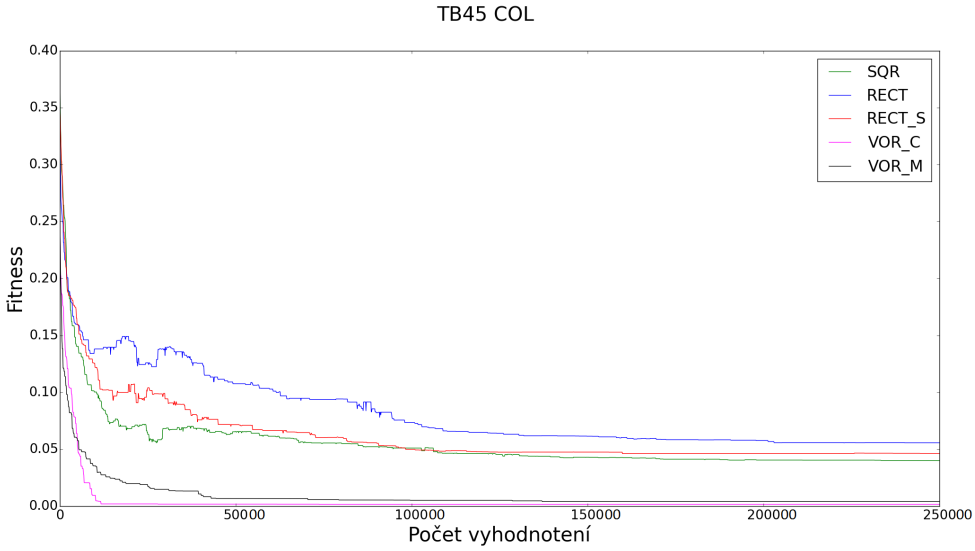
Obr. 6.17: Chess - najlepšie riešenia pre jednotlivé algoritmy

6.5 Two-Bands 45

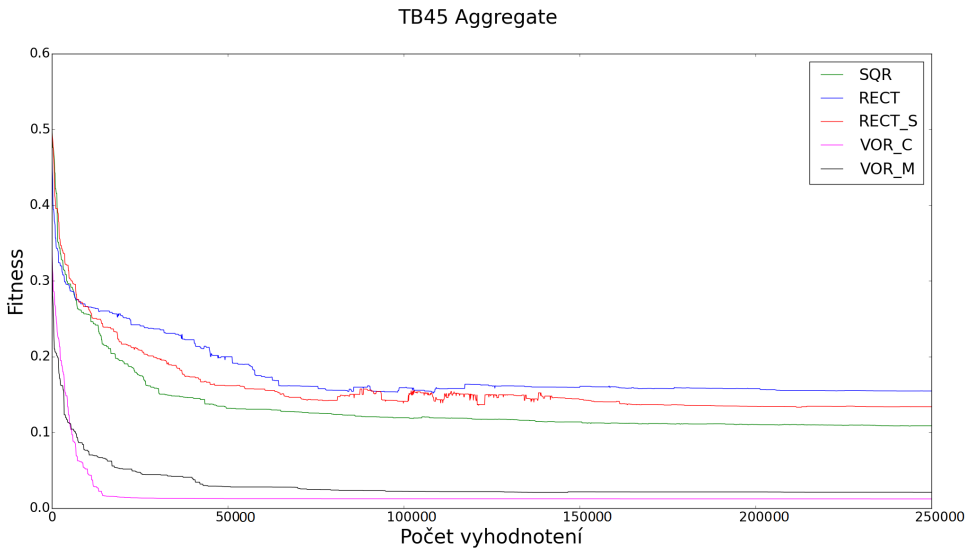
Pri otočenom Two-Bands boli použité skoro rovnaké parametre ako pri pôvodnom, t.j. externé chemikálie veľkosti 2 a 1 interný, 8 neurónov na skrytej vrstve, počet iterácií automatu bol zhora ohraničený na 150 a odchýlka energie mala prahovú hodnotu 10^{-8}, populácia veľkosti 100, ale vďaka nepravidelným tvarom sme zvýšili počet generácií na 2500. Každá verzia bola spustená 12 krát.
Ako vidíme na výsledkoch (obr. 6.18, 6.20, 6.22), štvorcové a obdĺžníkové tvary majú svoje rezervy, ale predstavujú celkom presnú aproximáciu cieľa, pričom Voronoi verzie sú úplne presné - čo neprekvapuje vzhľadom na použité pravidlo rozdelenia, Voronoi s jedným generátorom má priamočiaru možnosť rozpadu s hľadaným rozložením, kvôli tomu je aj konvergencia taká rýchla. Škálované behy v tomto prípade vždy generovali rovnaké výsledky.
Jednotlivé behy so štvorcami a oblžníkmi trvali v priemere 12 hodín, s Voronoiovými reprezentáciami v priemere 3.



Obr. 6.18: Priemerný fitness farby výsledku pre rôzne algoritmy, chybová úsečka reprezentuje jednu štandardnú chybu



Obr. 6.19: Konvergencia priemerného fitness najlepšieho jedinca populácie (podľa fitness farby) pre jednotlivé algoritmy

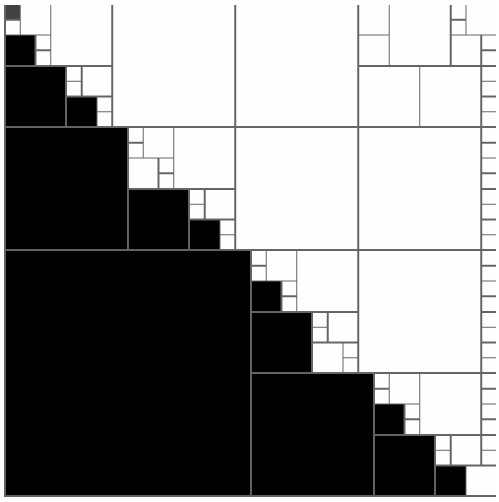


Obr. 6.20: Konvergencia priemerného fitness najlepšieho jedinca populácie (podľa aggregovaného fitness) pre jednotlivé algoritmy

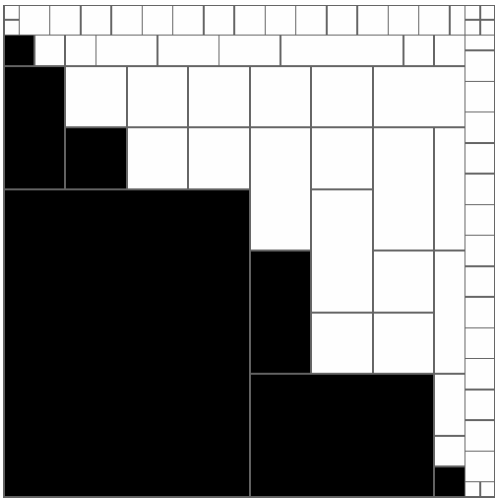
Typ	spojenie			rozdelenie		
	8×8	32×32	64×64	8×8	32×32	64×64
SQR	22	349	1380	8	23	56
RECT_S	13	308	1348	0	0	0
VOR_C	4	4	4	3	3	3
VOR_#	24	30	63	1	1	1

Tabuľka 6.3: TB45 - početnosť akcií topologických zmien

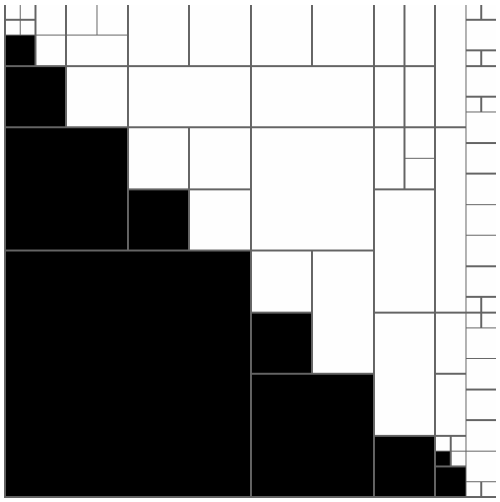
Obr. 6.21: Realizácia experimentov. Hromadné behy sme spúštali v počítačovej miestnosti FMFI UK na 48 počítačoch.



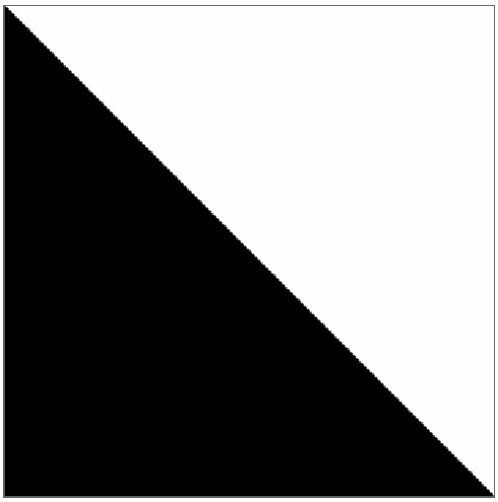
Neregulárne štvorce



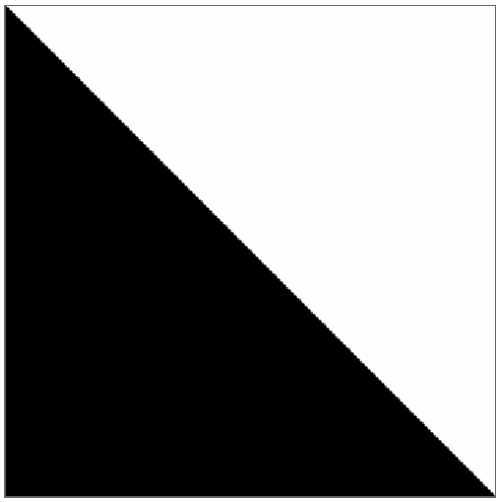
Obdĺžníky



Obdĺžníky podľa striktných pravidiel



Voronoi s mriežkou



Voronoi s jedným bodom

Obr. 6.22: TB45 - najlepšie riešenia pre jednotlivé algoritmy

KAPITOLA 7

Záver

Cieľom tejto implementačno-výskumno-experimentálnej diplomovej práce bolo implementovať rámcové prostredie na realizáciu experimentov na evolúciu prechodového pravidla celulárnych automatov, kódované ako dopredný perceptrón, za účelom nájdenia vhodnej celulárnej reprezentácie pre problém vlajky (a pre rôzne jeho modifikácie) v rámci evolučného dizajnu, konkrétnejšie pre umelú ontogenézu; ďalej navrhnúť, vykonať a vyhodnotiť experimenty použitím tohto prostredia.
V našej práci sme vychádzali z troch predošlých prác z tejto oblasti, poskytli sme prehľad o použitej metodike, načrtli sme potrebné teoretické pozadie na ich pochopenie a následne sme formulovali naše hypotézy na potenciálne vylepšenia ohľadom reprezentácie buniek a riadenia štruktúrových zmien automatu. Spomínané vylepšenia mali aj implementačný aj výskumný charakter. Z technického hľadiska sme sa snažili vytvoriť čo najefektívnejší nástroj na experimentáciu, pričom dôraz bol kladený na modularitu, rozšíriteľnosť a konfigurovateľnosť rámcového prostredia.
Z pohľadu experimentov medzi prínosy práce patria nové algoritmy na neregulárnu geometrickú reprezentáciu celulárnych automatov v rámci modelu používajúce obdlžníkové tvary a Voronoiové bunky, postupy na evolúciu prechodových pravidiel topologických zmien počas iterácie, ich experimentálne vyhodnotenie a zdokumentovanie.
Ciele diplomovej práce teda považujeme za splnené.
Predmetom ďalšieho výskumu v tomto smere môže byť rozšírenie zoznamu účelových funkcií (napr. pridelovať fitness podľa toho v akej miere pretínajú úsečky vygenerovaného vzoru cieľové tvary - ako indikátor štruktúrovej vhodnosti riešenia, ktorá by mohla vedieť k viacfázovej optimalizácie) alebo experimentácia s inými reprezentáciami buniek, ako štandardná trojuholníková mriežka, príp. fraktálová reprezentácia.

Literatúra

[1] **Aurenhammer F, Klein R** *Voronoi Diagrams (2000), Handbook of Computational Geometry, Elsevier Science Publishing, kapitola V, str. 201-290*

[2] **Back T** *Selective pressure in evolutionary algorithms: a characterization of selection mechanisms (1994), Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference*

[3] **Banks E R** *Information and Transmission in Cellular Automata (1971), Ph.D. Dissertation, Massachusetts Institute of Technology*

[4] **Beka M** *Evolution of growing and irregular cellular automata (2014), diplomová práca, FMFI UK, Bratislava*

[5] **Benett C** *Logical reversibility of computation (1973), IBM journal of Research and Development, strany 525–532, 1973.*

[6] **Berlekamp E, Conway J H, Guy R** *Winning Ways for Your Mathematical Plays, volume 2 (1982), Academic Press*

[7] **de Berg M, van Kreveld M, Overmars M, Schwarzkopf O** *Computational Geometry 3rd ed. (2008), Springer-Verlag TELOS Santa Clara, CA, USA, ISBN 3-540-65620-0*

[8] **Burks A** *Von Neumann's self-reproducing automata (1970), In Burks*

[9] **Clementi A, de Biase G A, Massini A** *Fast parallel arithmetic on cellular automata (1994), Complex Systems, 8(6), 435*

[10] **Codd E F** *Cellular automata (1968), New York: Academic Press*

[11] *CMEAS wikipedia online, https://en.wikipedia.org/wiki/CMA-ES*

[12] **Das R, Mitchell M, Crutchfield J** *A Genetic Algorithm Discovers Particle-Based Computation in Cellular Automata (1994), Parallel Problem Solving from Nature—PPSN III. Berlin Springer-Verlag*

[13] **Das R, Mitchell M, Crutchfield J** *Evolving Globally Synchronized Cellular Automata (1995), Proceedings of the Sixth International Conference on Genetic Algorithms*

[14] **Deb K, Pratap A, Agarwal S, Meyarivan T** *A Fast Elitist Multi-Objective Genetic Algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, 6:182-197, 2000*

[15] **Devert A** *Building processes optimization : Toward an artificial ontogeny based approach, Universit Paris-Sud, Francúzsko, 2009, Ph. D. dizertácia*

[16] **Devert A, Bredeche N, Schoenauer M** *Robustness and the Halting Problem for Multi-Cellular Artificial Ontogeny (2011), IEEE Transactions on Evolutionary Computation (Volume:15, Issue: 3)*

[17] **Fogelman-Soulie F, Robert Y, Tchuente M** *Automata Networks in Computer Science: Theory and Applications (1987), Manchester, UK : Manchester University Press*

[18] **Fortune S** *A sweepline algorithm for Voronoi diagrams (1987), Algorithmica 2, 153-174*

[19] **Hansen N** *The CMA Evolution Strategy: A Tutorial (2011)*

[20] **Hlavačiková J** *Cellular Embryogenic Representations in Evolutionary Design (2010), diplomová práca, FMFI UK, Bratislava*

[21] **Hornby G** *Generative representations for evolutionary design automation, Brandeis University, Waltham, MA, 2003, Ph.D. Dissertation*

[22] **Kiran Sree P, Ramesh Babu I** *Cellular Automata and Its Applications in Bioinformatics: A Review (2014), Global Perspectives on Artificial Intelligence (GPAI) Volume 2 Issue 2*

[23] **Kvasnička V, Beňušková Ľ, Pospíchal J, Farkš I, Tiňo P, Kráľ A** *Úvod do teórie neurónových sietí (2011), Iris: Bratislava*

[24] **Lohn J, Kraus W, Linden D, Colombano S** *Evolutionary optimization of yagi-uda antennas. Proc. ICES '01: From Biology to Hardware, strany 236–243, London, UK, 2001. Springer-Verlag*

[25] **Mitchell M** *Computation in Cellular Automata: A Selected Review (1998), Nonstandard Computation, pp. 95–140. Weinheim: VCH Verlagsgesellschaft, 1998.*

[26] **von Neumann J** *Theory of Self-Reproducing Automata (edited and completed by A.W. Burks) (1966), Urbana, IL: University of Illinois Press*

[27] **Popovici A, Popovici D** *Cellular Automata in Image Processing (2002), Fifteenth International Symposium on Mathematical Theory of Networks and Systems*

[28] **Sutner K** *Computational Classification of Cellular Automata (2012),Int. J. General Systems 41(6): 595-607 (2012)*

[29] **Turing A M** *The chemical basis of morphogenesis from Philosophical Transactions of the Royal Society of London (1952)*

[30] *Wolfram MathWorld - Elementary Cellular Automaton online, http://mathworld.wolfram.com/ElementaryCellularAutomaton.html*

[31] **Wolfram S** *A New Kind of Science (2002), Wolfram Media*

[32] **Zelinka I, Oplatková Z, Šeda M, Ošmera P, Včelař F** *Evoluční výpočetní techniky - Principy a aplikace (2009). Praha : BEN – technická literatura, 2009. 536 s. ISBN 978–80–7300–218–3*

[33] **Zitzler E** *Evolutionary Algorithms for Multiobjective Optimization, Evolutionary Methods for Design, Optimisation, and Control (EUROGEN 2001), strany 19–26, 2002. CIMNE*

Príloha

Priložené CD obsahuje:

- zdrojvé kódy aplikácie
- prácu vo formáte *pdf*