VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

# CELULÁRNÍ AUTOMAT V DYNAMICKÉM PROSTŘEDÍ

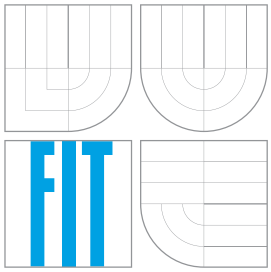
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAROSLAV BENDL

BRNO 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

# CELULÁRNÍ AUTOMAT V DYNAMICKÉM PROSTŘEDÍ

CELLULAR AUTOMATON IN DYNAMICAL ENVIRONMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAROSLAV BENDL
AUTHOR

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

BRNO 2009

# Abstrakt

Tato bakalářská práce se zabývá metodou evoluce celulárního automatu schopného sebeopravy po poškození vlivem externího prostředí. Popisovaná metoda je založená na algoritmu celulárního programování a využívá i principů biologického developmentu. V rámci této práce jsou provedeny experimenty vedoucí k ověření regeneračních schopností automatu vyvinutého za pomoci tohoto postupu.



# Abstract

This bachelor thesis focuses on the method of evolution of cellular automaton capable of self-repair after being damaged by external environment. The described method is based on cellular programming algorithm and uses principles of biological development. Experiments leading to verification of regenerative ability for cellular automaton evolved by this approach are presented in this work.



# Klíčová slova

celulární automat, celulární programování, evoluční algoritmus, sebeoprava

# Keywords

cellular automaton, cellular programming, evolutionary algorithm, self-repair


# Citace

Jaroslav Bendl: Celulární automat v dynamickém prostředí, bakalářská práce, Brno, FIT VUT v Brně, 2009

# Celulární automat v dynamickém prostředí

## Prohlášení

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

.....
Jaroslav Bendl
19. května 2009

## Poděkování

Děkuji svému vedoucímu práce ing. Michalu Bidlovi, Ph.D. za poskytnutí cenných rad a nápadů, které mi pomohli tuto práci vytvořit.

# Obsah

# Kapitola 1

# Úvod

Tato práce se zabývá metodou evoluce celulárního automatu schopného sebeopravy po poškození vlivem externího prostředí. Popisovaná metoda je založená na algoritmu celulárního programování a využívá i principů biologického developmentu.

Text práce je rozdělen následovně: Kapitola 2 se zabývá charakteristikou evolučních výpočetních technik a detailněji se věnuje genetickým algoritmům. Kapitola 3 popisuje principy celulárních automatů a jejich využití jako modelů developmentu. Kapitola 4 je zaměřena na vysvětlení metody sloužící ke konstrukci celulárního automatu schopného sebeopravy. V kapitole 5 jsou pak uvedeny experimenty s takto vyvinutým celulárním automatem. Kapitola 6 obsahuje závěr a shrnutí celé práce.

# Kapitola 2

# Evoluční algoritmy

Evoluční algoritmy patří mezi netradiční výpočetní postupy inspirované přírodními vývojovými procesy. Mají charakter stochastického algoritmu a nejčastěji se využívají pro hledání optimálního řešení v rozsáhlém stavovém prostoru s velkým množstvím přípustných řešení, která není možné v rozumném čase všechna otestovat. Zatímco u slepého stochastického algoritmu je stavový prostor prohledáván bez jakékoliv návaznosti na optimalizovanou funkci, evoluční algoritmy přidávají techniky usměrňující generování nového řešení blíže k řešení optimálnímu.

## 2.1 Základní charakteristiky evolučních algoritmů

Princip evolučních algoritmů je založen na teorii přirozeného výběru, podle něhož mají největší šanci na přežití takoví jedinci, kteří se dokáží co nejlépe přizpůsobit podmínkám prostředí. Reprodukcí těchto jedinců pak s velkou pravděpodobností vznikne jedinec s lepšími vlastnostmi pro dané prostředí, než jaké měli jeho rodiče. Tyto poznatky dobře vystihuje následující citát: "V přírodní evoluci je základní úlohou biologického druhu vyhledávání výhodných adaptací vůči složitému a dynamicky se měnícímu prostředí. 'Znalost', která charakterizuje každý biologický druh, byla získána vývojem a je shrnuta v chromozomech každého jedince." [2]

Běžné optimalizační techniky jsou založené na vhodně zvoleném počátečním odhadu řešení, které je posléze iterativně vylepšováno. Evoluční algoritmy ale nepracují pouze s jediným řešením, nýbrž s jejich množinami označovanými jako *populace*. Počáteční populace je obvykle složena z náhodně vybraných jedinců [7].

Vývoj probíhá v diskrétních časových krocích označovaných jako *generace*. V každém kroku vývoje dochází k obměně populace, čímž je zajištěno postupné odstranění nevhodných řešení. Rozhodnutí o setrvání či odstranění jedince z populace se provádí na základě míry kvality daného jedince, tzv. *fitness hodnoty*, která je získána pomocí *fitness funkce* a vyjadřuje schopnost jedince přežít a reprodukovat se v daných podmínkách.

Mezi základní typy evolučních výpočetních technik patří genetický algoritmus, genetické programování, evoluční strategie a evoluční programování. V následující sekci bude popsán podrobněji genetický algoritmus, na jehož principu pracuje celulární programování, které využijeme pro evoluci celulárního automatu.

```
evoluční_krok = 0
inicializace P(c)
vyhodnocení P(c)
while(not zastavovací_pravidlo) {
    evoluční_krok++
    selekce P(c)
    křízení P(c)
    mutace P(c)
    vyhodnocení P(c)
}
```

Obrázek 2.1: Pseudokód genetického algoritmu.

## 2.2 Genetický algoritmus

Základní principy genetického algoritmu zformuloval John Holland v polovině 70. let 20. století [5]. Podrobnějšímu studiu této evoluční techniky, z pohledu snažícího se chápat genetické algoritmy jako metodu aplikovatelnou na širokou škálu úloh, se však věnoval až David Goldberg na konci 80. let [4].

V rámci genetického algoritmu je jedinec populace, reprezentující jedno kandidátní řešení, chápan jako řetězec symbolů konečné délky, který je ve shodě s terminologií obecné genetiky označován jako *chromozom*. Chromozom sestává z posloupnosti symbolů vybraných zpravidla z jediné abecedy. Konkrétní symbol v chromozomu se pak nazývá *gen* [7].

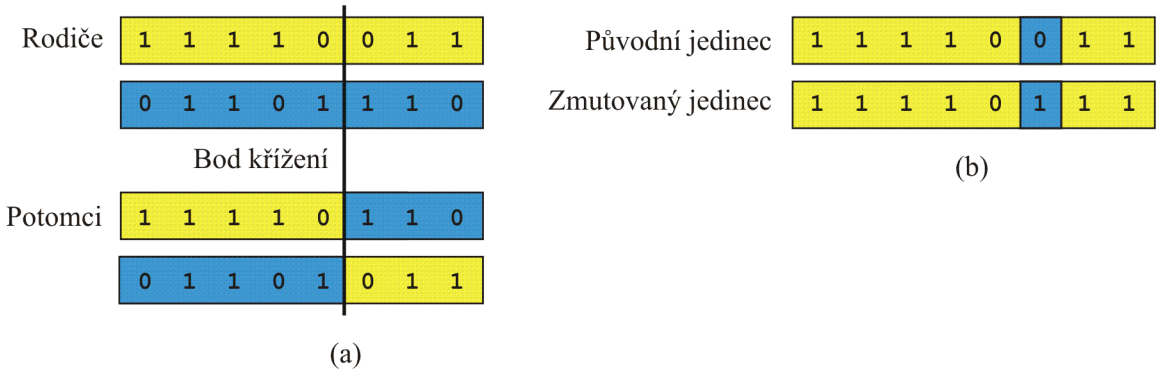
Obecný tvar genetického algoritmu je zobrazen na obrázku 2.1. Nejprve je vynulován vývojový čas a v kroku *inicializace* vytvořena počáteční populace sestavena zpravidla z náhodně vybraných jedinců. Každému jedinci je pak pomocí fitness funkce (operace *vyhodnocení*) přiřazena odpovídající fitness hodnota. Následující cyklus simuluje vývojový čas, přičemž jednomu kroku cyklu odpovídá jeden krok evolučního vývoje.

*Selekce* napodobuje zápas o přežití a vybírá z předchozí populace jedince do nové populace. Způsob výběru jedinců záleží na vybrané selekční strategii, obvykle však mají větší šanci na zařazení do nové populace jedinci s vyšší hodnotou fitness. V současné době je pravděpodobně nejpoužívanější strategií *turnajová selekce*. Její princip spočívá v náhodném výběru malé, často pouze dvoučlenné, skupiny jedinců, mezi nimiž se uspořádá turnaj spočívající ve vzájemném porovnání hodnot fitness. Nejlépe ohodnocený prvek v rámci dané skupiny se pak stává prvkem nové populace a turnaj takto pokračuje až do získání požadovaného počtu jedinců.

Ačkoliv je selekce účinný prostředek vedoucí ke zlepšení kvality populace, ve své podstatě jde o pouhé kopírování vybraných jedinců z jedné množiny do druhé. Noví jedinci se v evolučním procesu tvoří pomocí *rekombinačních operátorů*, mezi které patří zejména *křížení* a *mutace* (viz obrázek 2.2).

Mutace je prostředek sloužící k udržení genetické variace. Rozšiřuje prohledávaný prostor o řešení, která nelze získat křížením a předchází tak možnosti uváznutí v lokálním extrému. Příkladem mutace chromozomu reprezentovaného binárním řetězcem je invertování (překlopení) hodnoty bitu. Takové mutaci se podrobí každý bit řetězce s určitou (typicky velmi malou) pravděpodobností. Obvykle je pravděpodobnost mutace $p_m$ v rozsahu pouze 0.05-0.15, aby nebyla genetická informace příliš často narušována.

Křížení slouží k vytvoření zcela nových chromozomů. Ty jsou vytvořeny ze dvou již existujících chromozomů označovaných jako rodiče, které si vymění úseky kódu vymezené jedním nebo více body křížení. Obvykle se používá jednobodové křížení, přičemž pozice tohoto bodu je určena náhodně. Popsanou výměnou genetické informace mezi dvěma rodiči vzniknou dva potomci. Pravděpodobnost výběru chromozomu do procesu křížení $p_x$ je obvykle v rozsahu 0.6-1.




Obrázek 2.2: Genetické operátory: (a) jednobodové křížení, (b) mutace.

Aplikací výše popsaných genetických operátorů je získána populace potomků. Pro stanovení množiny považované v dalším evolučním kroku za novou populaci je nutné vybrat vývojovou strategii, určující, jakým způsobem zkombinovat množinu potomků s množinou rodičů. *Generační vývojová strategie* provede kompletní náhradu populace rodičů za populaci potomků. Naproti tomu při *postupné vývojové strategii* jsou rekombinační operátory aplikovány pouze na část původní populace a rodiče koexistují se svými potomky [7].

# Kapitola 3

# Celulární automaty

Historie celulárních automatů sahá do 40. let 20. století. V té době pracoval John von Neumann na konceptu známém pod názvem kinematický model, který směřoval k vytvoření stroje – robota schopného vytvářet své vlastní kopie. Problémem však byla nejasná vnitřní stavba součástek, z nichž měl být robot sestaven. Řešení von Neumannovi nabídl jeho spolupracovník Stanislaw Ulam, který, inspirován růstem krystalů, navrhl prostředí tvořené dvourozměrnou mřížkou, kde každé políčko reprezentuje jednu buňku. Každá buňka pak obsahuje konečný automat založený na pravidlech shodných s ostatními buňkami. Tento model přenesl úlohu do roviny čisté logiky a na jejím základě předělal John von Neumann svůj kinematický model na první buňkový – celulární automat [7].

## 3.1 Základní charakteristiky celulárních automatů

Celulární automat je matematický model diskrétní v hodnotách, prostoru a čase. Tento systém sestává z pravidelné struktury buněk v N-rozměrném prostoru, přičemž každá buňka se může v daném okamžiku nacházet v daném stavu z konečné množiny stavů. Často jde pouze o dva stavy – 0 reprezentující mrtvou buňku a 1 reprezentující živou buňku. Obvykle se předpokládá nekonečný počet buněk automatu, avšak v praktických realizacích je prostor automatu omezen a buňky za hranicí celulární struktury se buď pokládají identicky za nulové, nebo jsou cyklicky spojeny s buňkami z opačné strany. V případě jednorozměrného automatu tak vytvářejí smyčku, v případě dvourozměrného automatu anuloid [6].

Stav buněk se mění v diskrétních časových krocích podle lokální přechodové funkce (lokálního přechodového pravidla). Lokální přechodová funkce určuje nový stav buňky podle aktuálního stavu buňky a podle stavu buněk ve svém okolí (viz obrázek 3.1).

Okolí jednorozměrného celulárního automatu sestává z buněk po obou stranách od vyšetřované buňky a z této buňky samotné, přičemž sledovaný počet buněk z každé strany se označuje jako poloměr. Do okolí dvourozměrného celulárního automatu se zařazují čtyři přilehlé buňky (von Neumannovo okolí), nebo se přidávají i buňky sousedící s vyšetřovanou buňkou pouze v rozích (Moorovo okolí). Typy okolí jsou graficky znázorněny na obrázku 3.2.

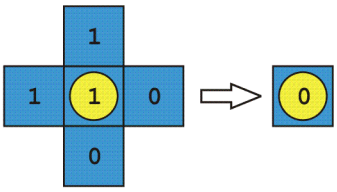
Z předcházejících odstavců vyplývá, že CA je definován těmito vlastnostmi: strukturou buněk, stavy buněk, uvažovaným okolím buněk, lokální přechodovou funkcí a počáteční konfigurací.

Důležité rysy celulárních automatů jsou [7]:

- **paralelismus** (výpočet nových hodnot buněk všech prvků probíhá současně, na běžných sériových počítačích se musí tento postup simulovat),
- **lokalita** (nový stav buňky závisí jen na jeho původním stavu a na původních stavech prvků z jeho okolí),
- **homogenita** (pro všechny buňky platí stejná lokální přechodová funkce).

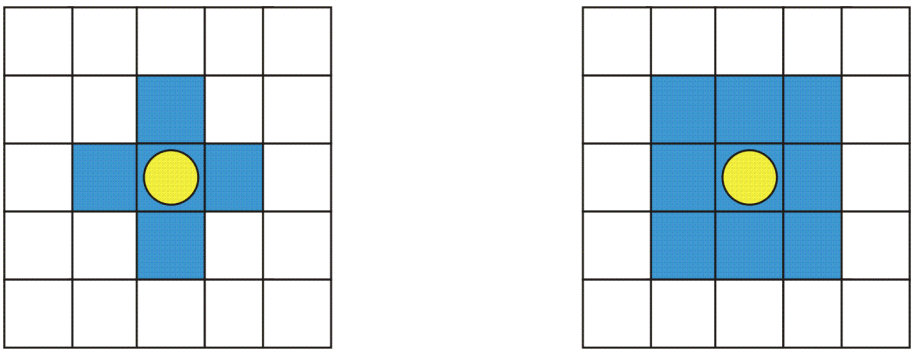
| Okolí | Výstup | Okolí | Výstup | Okolí | Výstup |
|---|---|---|---|---|---|
| 00000 | 1 | 01011 | 0 | 10110 | 1 |
| 00001 | 0 | 01100 | 0 | 10111 | 0 |
| 00010 | 0 | 01101 | 0 | 11000 | 0 |
| 00011 | 0 | 01110 | 1 | 11001 | 1 |
| 00100 | 1 | 01111 | 1 | 11010 | 1 |
| 00101 | 1 | 10000 | 0 | 11011 | 0 |
| 00110 | 0 | 10001 | 1 | 11100 | 0 |
| 00111 | 1 | 10010 | 0 | 11101 | 0 |
| 01000 | 1 | 10011 | 0 | 11110 | 0 |
| 01001 | 0 | 10100 | 1 | 11111 | 0 |
| 01010 | 1 | 10101 | 0 | | |

(a)




(b)

Obrázek 3.1: (a) Tabulka přechodových pravidel tvořících lokální přechodovou funkci celulárního automatu uvažujícího von Neumannovo sousedství. Sloupec *Okolí* označuje aktuální stav vyšetřované buňky a sousedních buněk, sloupec *Výstup* určuje nový stav buňky po provedení přechodu. Pořadí zápisu stavů buněk ve sloupci *Okolí* je následující: aktuální buňka, horní soused, levý soused, dolní soused, pravý soused. (b) Ukázka přechodu jedné buňky automatu podle odpovídajícího pravidla lokální přechodové funkce z tabulky.



(a) (b)

Obrázek 3.2: Typy okolí celulárního automatu: (a) von Neumannovo okolí, (b) Moorovo okolí.

## 3.2 Celulární automaty z hlediska počtu lokálních přechodových funkcí

V předcházejícím textu bylo uvedeno, že buňky mění svůj stav podle lokální přechodové funkce. Ta může být buď identická pro celou mřížku, anebo může být v různých buňkách odlišná. V prvním případě hovoříme o uniformním CA, v druhém o neuniformním CA. Speciálním případem neuniformního CA je kvaziuniformní CA, kde ve většině buněk převažuje jedno, nebo několik pravidel.

Vyhledávací prostor neuniformních CA je díky použití více pravidel výrazně větší než u jejich uniformních protějšků. S tím souvisí i jejich výpočetní síla – zatímco k dosažení výpočetní síly srovnatelné s Turingovým strojem je potřeba v dvoudimenzionálním prostoru alespoň dvoustavový CA s 9-okolím, v případě dvoustavového neuniformního CA postačuje 5-okolí [8].

## 3.3 Celulární programování

Pojmem celulární programování je označován relativně nový přístup k evoluci celulárních automatů, který vynalezl v 90. letech Moshe Sipper [8]. Obecný tvar algoritmu celulárního programování je uveden na obrázku 3.4.

Celulární programování je založené na genetickém algoritmu, avšak narozdíl od něj nepracuje s celou populací kandidátních řešení – uniformních celulárních automatů, nýbrž si vystačí pouze s jediným kandidátem – neuniformním celulárním automatem. Buňky takového automatu jsou zpočátku inicializovány náhodně zvolenými lokálními přechodovými funkcemi. Ty jsou generovány s odlišnými hodnotami parametru udávajícího poměr přechodových pravidel, jejichž výstupem je neklidový stav (živá buňka), vůči pravidlům, jejichž výstupem je klidový stav (mrtvá buňka). Pravidla jsou zakódována jako řetězce obsahující výstupní stavy pro všechny možné kombinace stavů z uvažovaného okolí.

Jelikož je k dispozici pouze jediný celulární automat, nemá z hlediska evoluce smysl hodnotit jeho celkovou úspěšnost. Namísto toho se ohodnocují lokální přechodové funkce jednotlivých buněk, a to na základě schopnosti dostat buňku během stanoveného počtu přechodů z různých počátečních stavů do požadovaného cílového stavu. Počet náhodně generovaných konfigurací, na kterých je úspěšnost lokálních přechodových funkcí v každém evolučním kroku testována, se v obecném tvaru algoritmu celulárního programování značí proměnnou $pocet\_konfiguraci$.

Z výše uvedených poznatků je zřejmé, že evoluce probíhá pro každou buňku zvlášť, přičemž v každém evolučním kroku se provádí porovnání pouze s buňkami z definovaného okolí. Existuje-li v něm právě jedna lokální přechodová funkce s vyšší hodnotou fitness, pak se tato funkce převezme, zatímco existuje-li jich více, náhodně jsou dvě z nich vybrány a je na ně aplikován operátor křížení. Takto vzniklá lokální přechodová funkce pak nahradí původní přechodovou funkci na pozici vyšetřované buňky. Na jednotlivé bity takových lokálních přechodových funkcí je pak s určitou, typicky velmi malou pravděpodobností aplikován operátor mutace.

## 3.4 Využití celulárních automatů jako modelů vývinu

Umělý development (umělý vývin) je další technika inspirovaná přírodními biologickými jevy. Princip je založen na procesu ontogeneze, která se zabývá vývojem jedince ze zygoty

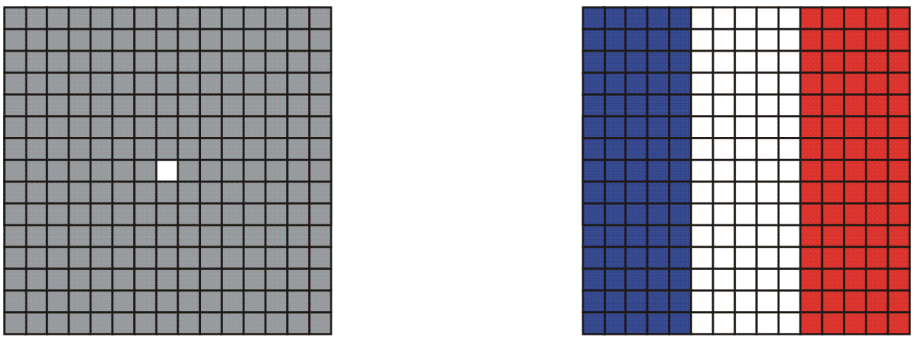
do podoby vyspělého vícebuněčného organizmu. Jelikož se ale jedná o velmi složitý proces, bývají modely vývinu využívané v informatice značně zjednodušeny a obvykle jsou simulovány jen některé jeho části.

Development využívá nepřímé mapování genotypu na fenotyp. Ve shodě s terminologií obecné genetiky lze považovat genotyp za genetickou informaci zobrazenou do konkrétního jedince – fenotypu, představujícího řešení dané úlohy. Genetická informace je přitom uložená v chromozomu a tvoří ji prvky vyhledávacího prostoru. Předmětem evoluce u nepřímého mapování je postup konstrukce cílového objektu ze zárodečné buňky, což je hlavní rozdíl oproti přímému mapování, ve kterém se evoluci podrobí genotypy přímo vázané na fenotypy.

Celulární automaty patří mezi nejznámější modely developmentu. Předpis vývoje je v nich vyjádřen lokální přechodovou funkcí. Počáteční konfigurace automatu pak reprezentuje zygotu a jednotlivé kroky vývoje napodobují tvorbu cílového objektu.

### Problém "francouzská vlajka"

Úloha generování vzoru francouzské vlajky byla poprvé představena embryologem Lewisem Wolpertem na konci 60. let 20. století jako problém sloužící ke studiu vývinu zygoty do podoby vyspělého vícebuněčného organizmu schopného regulace svého růstu [1]. Jednotlivé barvy vlajky zde přitom představují různé typy buněk. Takto definovaná úloha se později stala jakýmsi standardem pro testování vlastních metod v oblasti umělého developmentu [3]. Při použití celulárních automatů by zadání problému mohlo vypadat podobně, jak je zobrazeno na obrázku 3.3.



(a) (b)

Obrázek 3.3: Úloha "francouzská vlajka": (a) počáteční konfigurace – zygota, (b) cílová konfigurace – vyspělý vícebuněčný organizmus

```
// Inicializace pravidel
for(int i = 0; i < počet_buněk; i++) {
    inicializuj pravidla buňky i
    fitness[i] = 0
}

// Evoluce pravidel
čítač_konfigurací = 0
while(not zastavovací_pravidlo) {
    vygeneruj náhodnou počáteční konfiguraci
    proveď běh automatu pro vygenerovanou počáteční konfiguraci
    for(int i = 0; i < počet_buněk; i++) {
        if(buňka i je v požadovaném cílovém stavu)
            fitness[i]++;
    čítač_konfigurací++
    if((čítač_konfigurací % počet_konfigurací) == 0) {
        for(int i = 0; i < počet_buněk; i++) {
            n = zjistiPočetLépeOhodnocenýchSousedů()
            if(n== 0)
                ponech pravidlo aktuální buňky i nezměněné
            elseif(n == 1)
                nahraď pravidlo aktuální buňky i pravidlem lépe
                ohodnocené sousední buňky a proveď mutaci tohoto pravidla
            elseif(n == 2)
                nahraď pravidlo aktuální buňky i pravidlem vzniklým
                křížením dvou lépe ohodnocených sousedních buněk
                a proveď mutaci tohoto pravidla
            elseif(n > 2)
                nahraď pravidlo aktuální buňky i pravidlem vzniklým
                křížením dvou náhodně vybraných lépe ohodnocených buněk
                a proveď mutaci tohoto pravidla
            fitness[i] = 0
        }
    }
}
```

Obrázek 3.4: Pseudokód celulárního programování (převzato z [8]).

# Kapitola 4

# Konstrukce celulárního automatu se schopností sebeopravy

Biologické organizmy mají pozoruhodnou schopnost přizpůsobit se vlivům prostředí, do kterého jsou zasazeny. Adaptace vůči měnícím se podmínkám je pro ně důležitá z hlediska samotného přežití. Pokud uvažujeme celulární automat jako model biologií inspirovaného vývinu, pak je vhodné takový model doplnit o schopnost sebeopravy, která zajistí jeho existenci v dynamicky se měnícím prostředí. Pro účely této práce předpokládejme, že toto prostředí může ovlivňovat stav některých buněk celulárního automatu, tj. zanášet do právě vyvinuté konfigurace automatu určitou formu poškození. V této kapitole bude popsána uvažovaná forma externí informace, simulující vliv prostředí, a navržen celulární automat schopný se s takovým prostředím patřičně vypořádat. Pro demonstraci této schopnosti je zvolena úloha vývinu definovaného vzoru pomocí dvourozměrného celulárního automatu.

## 4.1 Evoluce celulárního automatu

V této práci je evoluce celulárního automatu založena na algoritmu celulárního programovaní popsaného v kapitole 3.3. Tento přístup se inspiruje genetickým algoritmem a předmětem evoluce jsou v něm lokální přechodové funkce jednotlivých buněk, které představují předpis vývoje celulárního automatu.

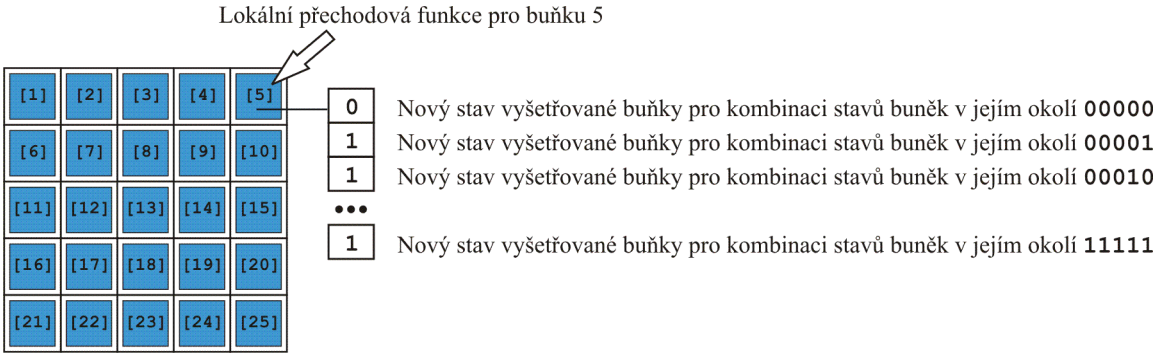
Proces tvorby lokálních přechodových funkcí se dá rozdělit do dvou etap. Zatímco v první etapě jsou hledána pravidla pro vývoj z počáteční do cílové konfigurace (tzv. základní pravidla), v druhé etapě se tato pravidla modifikují takovým způsobem, aby byla schopna opravit automat po jeho poškození. K tomuto procesu je vhodné poznamenat, že žádná dodatečná úprava pravidel v druhé etapě jejich vývoje nemůže poškodit schopnost získanou v první etapě, tedy dosažení cílové konfigurace z definované počáteční konfigurace. Konkrétní přechodová pravidla využitá při takovém vývinu jsou totiž "zamknutá" vůči jakékoliv pozdější modifikaci. Sloučení obou zmíněných etap je sice principielně možné, ale v praxi se ukázalo jako efektivnější ponechat je rozdělené. Důvodem je rozdílné využití algoritmu celulárního programování – zatímco při hledání základních pravidel uvažujeme pouze jedinou počáteční konfiguraci představující zygotu, při evoluci pravidel pro sebeopravu využijeme celou množinu počátečních konfigurací tvořenou poškozenými cílovými vzory.

Všem buňkám celulárního automatu je po provedení každého evolučního kroku přiřazena fitness hodnota pomocí fitness funkce. Fitness funkce zkoumá, zda buňka dosáhne po provedení stanoveného počtu přechodů požadovaného cílového stavu. To se zjišťuje pro

každou uvažovanou počáteční konfiguraci zvlášť (v případě evoluce základních pravidel je taková počáteční konfigurace samozřejmě pouze jedna) a při každé shodě se fitness hodnota dané buňky zvýší o 1. Nakonec se fitness hodnota buňky normalizuje do intervalu $\langle 0; 1\rangle$, a to tak, že se spočítaná hodnota podělí maximální možnou fitness hodnotou, která odpovídá počtu počátečních konfigurací. Pokud chceme určit fitness hodnotu pro celý automat, nikoliv samostatně pro jednotlivé buňky, pak fitness hodnoty jednotlivých buněk zprůměrujeme.

### 4.1.1 Reprezentace celulárního automatu

Jak již bylo řečeno dříve, v rámci této práce budeme uvažovat dvourozměrný neuniformní celulární automat. Takový automat je tvořen množinou lokálních přechodových funkcí přiřazující každé buňce vlastní přechodová pravidla. Struktura kandidátního řešení úlohy, tvořeného takovou množinou lokálních přechodových funkcí, je uvedena na obrázku 4.1.




Obrázek 4.1: Struktura chromozomu pro dvourozměrný neuniformní celulární automat o 25 buňkách. Z obrázku je patrná úsporná notace zápisu lokální přechodové funkce.

Počet přechodových pravidel, tvořících lokální přechodovou funkci, závisí na počtu možných kombinací stavů buněk v definovaném okolí a je dán výrazem $pocet\_stavu^{velikost\_okoli}$. Například pro binární celulární automat využívající von Neumannovo sousedství obsahuje každá buňka $2^5 = 32$ přechodových pravidel. Z uvedeného vztahu je patrné, že s rostoucím počtem možných stavů se podstatně zvětšuje počet přechodových pravidel, a tedy i velikost struktury kandidátního řešení, v důsledku čehož dochází k významnému nárůstu časové složitosti jeho výpočtu.

K výsledným podobám lokálních přechodových funkcí se dospěje evolucí využívající rekombinační operátory, jejichž implementace je detailně popsána v jedné z dalších částí této kapitoly. Při evoluci může dojít k modifikaci jakéhokoliv přechodového pravidla, s jedinou výjimkou pro konfiguraci tvořenou výhradně neživými buňkami – v takovém případě bude buňka v dalším kroku vždy opět neživá. Pokud by tomu bylo jinak, došlo by k porušení konceptu biologického developmentu, kterým se mnou zvolené řešení inspiruje. Pro úplnost nechť je uvedeno, že za neživé buňky jsou považovány buňky nacházející se ve stavu s označením “0”. Buňky v jiném stavu jsou pak pokládány za živé.

#### Velikost buněčného okolí

K dosažení výpočetní síly srovnatelné s Turingovým strojem postačuje v případě dvourozměrného neuniformního celulárního automatu von Neumannovo okolí – použití Moorova

okolí je tedy z tohoto hlediska zbytečné. Jeho aplikací by navíc došlo k exponenciálnímu nárůstu počtu stavů prohledávacího prostoru. To lze ukázat například na vzoru zmíněné francouzské vlajky uvažující 4 stavy (pro červenou, bílou, modrou barvu a pro stav reprezentující mrtvou buňku), který při použití 5-okolí pracuje s $4^5 = 1\,024$ přechodovými pravidly, zatímco při použití 9-okolí jich je $4^9 = 262\,144$. Výrazně nižší časová i prostorová náročnost jsou důvody, proč v experimentech pracuji výhradně s von Neumannovým okolím.

### 4.1.2 Implementace algoritmu celulárního programování

Experimenty provedené v rámci této práce prokázaly, že algoritmus celulárního programování využívající rekombinačními operátory, popsané v jedné z dalších částí této kapitoly, není schopen vytvořit správná přechodová pravidla pro určité konfigurace buněk. Obvykle jde o takové konfigurace, ve kterých má centrální buňka dosáhnout jiného stavu než ostatní buňky z jejího okolí. V takovém případě totiž křížením nikdy nevznikne korektní přechodové pravidlo a rozsah mutace je pro patřičnou úpravu příliš malý. K řešení tohoto problému jsou pro účely této práce do algoritmu celulárního programování zaneseny tyto modifikace:

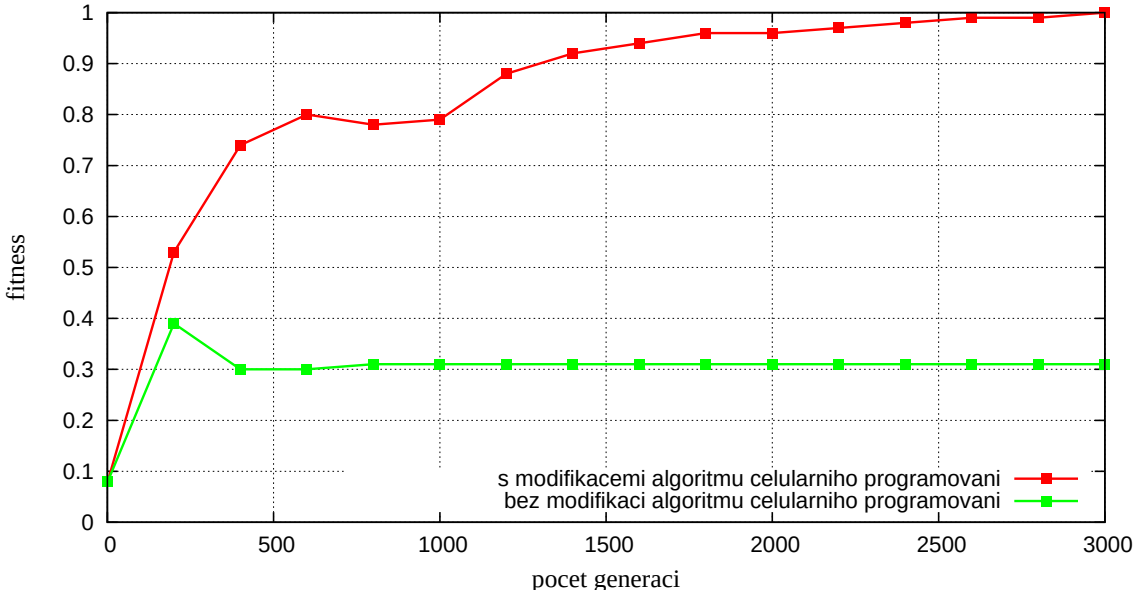
- V případě, že lze u přechodové funkce buňky pozorovat pozitivní vývoj její hodnoty fitness (tzn., že je vyšší než v předchozím kroku), pak je provedena pouze mutace této přechodové funkce a nezjišťuje se, zda v jejím okolí existují jedinci s vyšší fitness hodnotou, ze kterých by za normálních okolností byla sestavena nová přechodová funkce.

- V případě, že přechodová funkce buňky nedosahuje stanovené minimální hodnoty fitness v určitém počtu po sobě jdoucích kroků, pak dojde k jejímu "resetu". To znamená, že jsou znovu nastaveny nové stavy vyšetřované buňky pro všechny možné kombinace stavů buněk v definovaném okolí. Výběr nového stavu z množiny přípustných stavů pak probíhá zcela náhodně.
  Minimální hodnotu fitness stanovujeme podle charakteru řešené úlohy, obvykle přitom vybíráme z rozmezí intervalu $\langle 0; 0.5\rangle$.

- V případě, že průměrná hodnota fitness všech přechodových funkcí v automatu nedosahuje stanovené minimální hodnoty, pak jsou všechna lokální přechodové funkce "resetovány" způsobem popsaným v předchozím bodě.
  Minimální hodnotu fitness stanovujeme rovněž podobně jako v případě resetu jediné buňky, vybíráme tedy hodnotu z rozmezí intervalu $\langle 0; 0.5\rangle$.

- V případě, že přechodová funkce buňky dosahuje maximální hodnoty fitness ve stanoveném počtu po sobě jdoucích kroků, pak dojde k jejímu "skrytí". To znamená, že nebude uvažována svými sousedy při sestavování jejich nových lokálních přechodových funkcí (při operaci křížení).
  Počet kroků, po kterých má dojít ke skrytí, nastavujeme obvykle na relativně nízkou hodnotu – v našich experimentech uvažujeme 15 kroků.

Před aplikací principů skrývání či resetování buněk je vhodné poskytnout evoluci určitý čas k vývoji vlastního řešení. Z toho důvodu se principy těchto úprav aplikují až po uplynutí určitého počtu evolučních kroků vyjádřeného parametrem *délka inicializační fáze*. Za přiměřenou hodnotu přitom považujeme zpravidla 100-1000 evolučních kroků, přičemž konkrétní číslo volíme v závislosti na obtížnosti řešené úlohy. Význam výše uvedených modifikací algoritmu celulárního programování je patrný z grafu na obrázku 4.2.



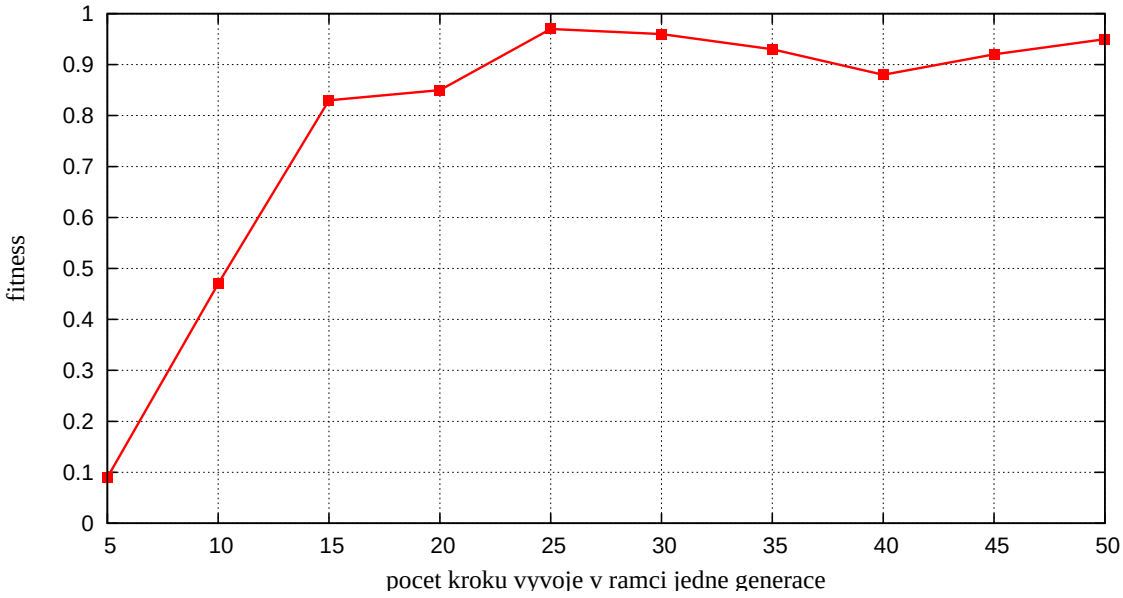

Obrázek 4.2: Graf znázorňující závislost hodnoty fitness nejlepšího řešení v populaci na počtu generací. Experiment je proveden na úloze "francouzská vlajka" s rozměrem mřížky 15 x 15 buněk.
Z grafu je patrný rozdíl v úspěšnosti pravidel získaných s využitím algoritmu celulárního programování standardního tvaru oproti pravidlům získaným s využitím algoritmu celulárního programování, v němž byly implementovány popsané modifikace. Objektivita experimentu je podložena deseti nezávislými běhy, jejichž výsledky byly zprůměrovány.

Obecně se často předpokládá, že čím déle evoluce probíhá, tím lepší budou výsledná přechodová pravidla jednotlivých buněk. Tento předpoklad však nemusí být zcela pravdivý, neboť vlivem použité metody, vybírající pro křížení výhradně nejlépe ohodnocené sousedy, dochází často k homogenizaci populace, přičemž ani mutace pak není schopna populaci nějakým zásadním způsobem obohatit. Je tedy vhodné počet generací evolučního algoritmu přiměřeně omezit a zároveň samostatně ukládat doposud nejlepší nalezené řešení.

Kromě implementace výše popsaných úprav algoritmu celulárního programování je rovněž velmi důležité správné nastavení jeho parametrů. Z toho důvodu se nyní pokusím vysvětlit význam těchto parametrů a stanovit jim vhodné hodnoty, případně určit metodiku jejich výpočtu.

### Počet kroků vývoje celulárního automatu v rámci jedné generace

V každé generaci evoluce celulárního automatu dochází k ohodnocení úspěšnosti přechodových funkcí jednotlivých buněk. Tato úspěšnost je získána na základě schopnosti přechodové funkce dostat buňku během stanoveného počtu kroků do požadovaného cílového stavu. Počet těchto kroků, po kterých dojde k testování stavu buňky, je závislý na tvaru a rozměrech uvažované celulární struktury. Na základě provedených experimentů, jehož výsledky jsou shrnutý v grafu na obrázku 4.3, jsem se rozhodl počet kroků vývoje pro čtvercovou celulární strukturu určovat pomocí vzorce $pocet\_kroku = \sqrt{pocet\_bunek} \cdot 2$.




Obrázek 4.3: Graf znázorňující vliv počtu kroků vývoje celulárního automatu v rámci jedné generace na úspěšnost vývinu požadovaného cílového vzoru. Experiment je proveden na úloze "francouzská vlajka" s rozměrem mřížky 15 x 15 buněk.
V grafu jsou vyneseny fitness hodnoty nejlepších řešení nalezených během prvního tisíce generací. Objektivita experimentu je podpořena pěti nezávislými běhy pro každou vyšetřovanou hodnotu počtu kroků vývoje, přičemž výsledky těchto běhů byly zprůměrovány.

### Ukončovací podmínka

Proces evoluce je samozřejmě možné ukončit při nalezení bezchybného, stoprocentně fungujícího řešení. U obtížnějších úloh však taková situace buď vůbec nenastane, nebo by její dosažení bylo časově příliš náročné, proto je vhodné evoluci nějakým dalším způsobem omezit. Za takové omezení se často považuje určení maximálního počtu generací evolučního algoritmu, po jehož dosažení je za řešení úlohy prohlášeno nejlepší kandidátní řešení získané v průběhu vývoje. Maximální počet generací obvykle stanovujeme na základě času, který jsme pro řešení úlohy vyhradili, přičemž platí, že s rostoucím počtem iterací se zvyšuje i šance na nalezení lepšího řešení. Tento růst však není lineární, neboť při evoluci často dochází k homogenizaci populace, přičemž ani mutace pak není schopna populaci nějakým zásadním způsobem obohatit.

### Parametry sebeopravy

Již na začátku této kapitoly bylo uvedeno, že trénování schopnosti sebeopravy probíhá na množině poškozených cílových konfigurací. Velikost této množiny má přitom zásadní vliv na úspěšnost zotavení automatu po jeho poškození. Čím více konfigurací bude testováno, tím větší regenerační schopnosti automat získá, ale zároveň dojde i k nárůstu doby výpočtu jeho přechodových funkcí.

Množina poškozených cílových konfigurací je generována automaticky a velikost poškození se určí pro každou konfiguraci náhodně z intervalu shora omezeného parametrem *maximální uvažované poškození*. Tento parametr představuje hodnotu z intervalu $\langle 0; 1 \rangle$,

přičemž je vhodné volit číslo bližší spíše jeho dolnímu okraji, neboť vliv prostředí v jednom kroku zpravidla není tak velký, aby poškodil významnější část mřížky.

### 4.1.3 Implementace rekombinačních operátorů

Rekombinační operátory byly obecně popsány v rámci kapitoly 2.2 věnující se genetickému algoritmu. V kontextu této práce se využívají zejména operátory křížení a mutace, které se aplikují na přechodové funkce jednotlivých buněk automatu.

Křížení je operace kombinující úseky složené z přechodových pravidel lokálních přechodových funkcí. Tento proces je proveden tehdy, nacházejí-li se v definovaném okolí vyšetřované buňky alespoň dva jedinci s vyšší hodnotou fitness. Výsledná lokální přechodová funkce, vzniklá jejich kombinací v náhodném bodě křížení, pak nahradí původní lokální přechodovou funkci vyšetřované buňky.

Mutace je operace spočívající v relativně malé modifikaci původního jedince. Tato modifikace je aplikována s určitou pravděpodobností na každé přechodové pravidlo lokální přechodové funkce vyšetřované buňky, přičemž je-li výsledek náhodného experimentu s pravděpodobností úspěchu $p_m$ nad tímto pravidlem pozitivní, dojde k jeho úpravě. Ta spočívá v náhodné změně nového stavu vyšetřované buňky pro kombinaci stavů buněk v uvažovaném okolí danou vybraným přechodovým pravidlem. V kontextu této práce se mutace používá v následujících případech:

- Po vytvoření nové lokální přechodové funkce pomocí operátoru křížení (vyšetřovaná buňka má ve svém okolí dva či více lépe ohodnocených sousedů).
- Po převzetí lokální přechodové funkce od sousední buňky (vyšetřovaná buňka má ve svém okolí jediného lépe ohodnoceného souseda).
- Při stagnující hodnotě fitness lokální přechodové funkce vyšetřované buňky, aby bylo stimulováno opuštění případného lokálního extrému.

Pravděpodobnost mutace $p_m \in \langle 0; 1\rangle$ obvykle stanovujeme blízko spodnímu okraji vymezujícího intervalu, protože jejím častým používáním by byl proces evoluce příliš často narušován a vytratily by se z něj prvky systematičnosti, což by mělo negativní dopad na úspěšnost nalezeného řešení.

## 4.2 Specifikace vlivu prostředí

Jak již bylo řešeno v úvodu kapitoly, schopnost adaptace vůči podmínkám prostředí je pro biologické organizmy zcela zásadní. V celulárních automatech, snažících se tuto vlastnost napodobit, bývá obvykle projev prostředí vyjádřen těmito způsoby: [8]

1. Poškozením lokálních přechodových pravidel určité části buněk. Takové poškození představuje změnu jednoho či několika přechodových pravidel určujících nové stavy buňky pro možné konfigurace stavů buněk v uvažovaném okolí.
2. Záměnou lokální přechodové funkce mezi dvěma buňkami. Taková záměna bývá zpravidla trvalá.
3. Změnou aktuálního stavu určité části buněk na jiný, zpravidla náhodně zvolený stav.

Pakliže je uvažován jakýkoliv z uvedených projevů prostředí, získá automat nedeterministické chování. To znamená, že i přes znalost lokálních přechodových pravidel jednotlivých buněk není možné jednoznačně určit jeho konfiguraci v dalším kroku, neboť se s určitou pravděpodobností může vyskytnout takové působení vlivu prostředí, které předpokládanou konfiguraci pozmění.

V rámci této práce se uvažuje pouze třetí z uvedených projevů prostředí, tedy změna stavu určité části buněk na jiný, zpravidla náhodně zvolený stav. Taková změna přitom může být aplikována pouze na živou buňku (tj. na buňku nenacházející se ve stavu 0). Experimenty jsou pak zaměřeny především na sledování schopnosti sebeopravy automatu.

# Kapitola 5

# Experimenty

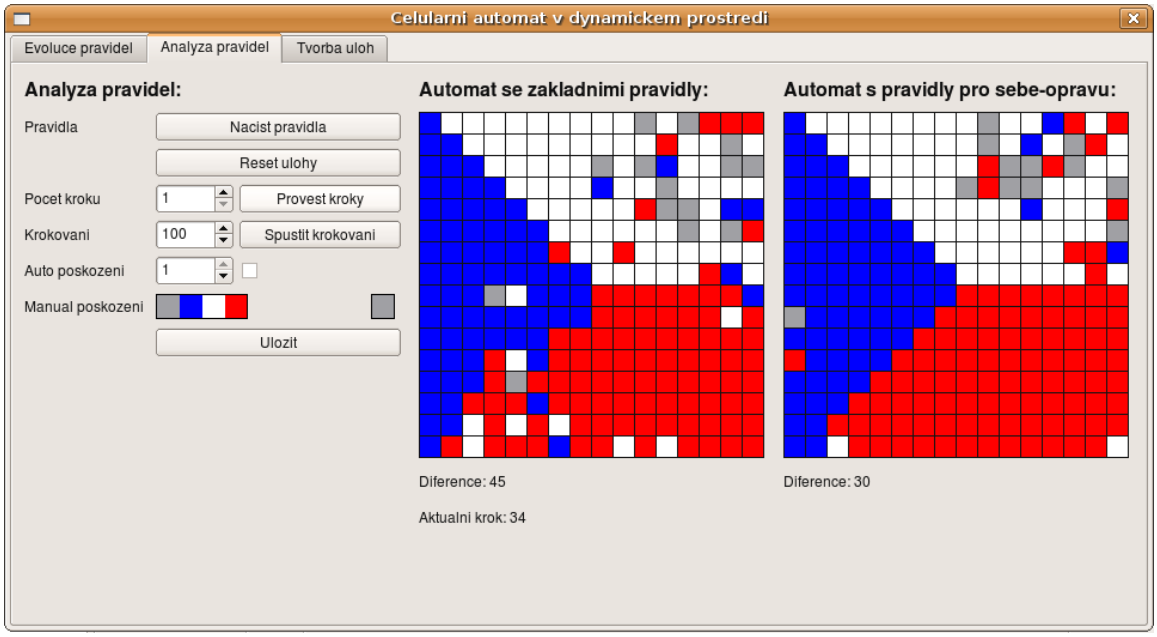
Algoritmus celulárního programování s vhodně nastavenými parametry a úpravami popsanými v předchozí kapitole je schopen nalézt přechodové funkce pro dosažení řady netriviálních cílových konfigurací. V praxi však s rostoucím počtem buněk a s přibývajícími stavy dochází k významnému nárůstu složitosti výpočtu přechodových pravidel, takže není-li poskytnut k řešení úlohy dostatečný čas, nemusí se podařit ho nalézt.

V předcházející kapitole již bylo řečeno, že v rámci této práce se za projev prostředí považuje náhodná změna stavu, která je s určitou pravděpodobností aplikována na každou živou buňku (tj. buňku nenacházející se ve stavu 0) částečně nebo i zcela vyvinutého automatu. Tato kapitola se zabývá experimenty dvojího druhu:

- Sledování chování celulárního automatu při trvalém působení projevů prostředí. V takovém experimentu hledáme velikost poškození, při kterém ještě nedochází k výrazné destrukci vyvinutého cílového vzoru. Uvažovaná tolerance buněk, které mohou být v jiném než cílovém stavu, je přitom menší nebo rovna 10%.
- Sledování regeneračních schopností automatu po závažném poškození. Takové poškození se přitom pokládá za jednorázový projev prostředí a v dalších krocích již není uvažováno.

Výše zmíněné experimenty jsou prováděny na dvou celulárních automatech. Zatímco první z nich je produktem první etapy evoluce automatu, při které jsou hledána pravidla pro vývoj z počáteční do cílové konfigurace, druhý z nich je produktem druhé etapy evoluce automatu, ve které jsou dříve získaná pravidla modifikována takovým způsobem, aby byla schopna opravit poškozenou konfiguraci. Protože určitou schopnost regenerace poškozeného automatu mají i pravidla získaná v první etapě, lze tímto způsobem efektivně vyhodnotit přínos a úspěšnost druhé etapy zaměřené na trénování schopnosti sebeopravy.

Chování obou automatů je ve výsledné aplikaci znázorněno ve dvou dvourozměrných mřížkách (viz obrázek 5.1). Automatické krokování jejich vývoje se stanoveným časovým intervalem je možné nastavit pomocí příslušných prvků grafického rozhraní, stejně jako velikost poškození buněk v každé generaci. Poškození je ale možné provést také manuálním nastavením prostřednictvím štětce a palety barev. Regenerační schopnosti se vyhodnocují pomocí ukazatele *diference* mezi aktuální a cílovou konfigurací, která udává počet buněk se vzájemně rozdílným stavem.




Obrázek 5.1: Ukázka grafického prostředí aplikace

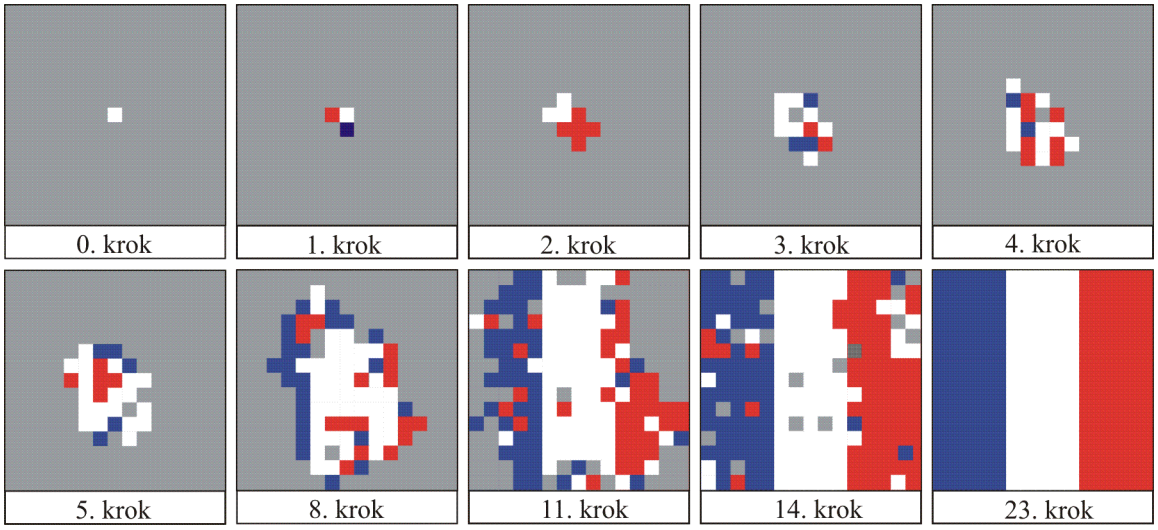
## 5.1 Výběr úlohy a nastavení parametrů

Experimenty realizované v rámci této kapitoly jsou založené na úloze francouzské vlajky popsané v kapitole 3.4. Tato úloha počítá s mřížkou o rozměrech 15 x 15 buněk, přičemž tyto buňky mohou nabývat jednoho ze čtyř stavů. Tři stavy jsou přitom vyhrazeny pro živé buňky (červená, bílá a modrá barva), čtvrtý stav je vyhrazen pro mrtvou buňku (šedá barva).

Přechodová pravidla zajišťující požadované chování automatu, které bude předmětem pokusů v dalších částech této kapitoly, jsou získána dříve popsaným algoritmem založeným na celulárním programování. Parametry tohoto algoritmu jsem zvolil podle doporučení a metodiky popsané v kapitole 4.1, přičemž konkrétní hodnoty jsou vypsány v tabulce 5.1. Ukázka vývoje automatu podle takto získaných pravidel je pak zobrazena na obrázku 5.2.

| **Parametry algoritmu** | |
|---|---|
| Okolí | von Neumannovo |
| Počet kroků vývoje celulárního automatu v rámci generace | 30 |
| Maximální počet iterací celulárního programování | 100 000 |
| Pravděpodobnost mutace | 0.01 |
| Sebe-oprava: Počet počátečních konfigurací | 20 000 |
| Sebe-oprava: Maximální poškození vyvinutého vzoru | 0.5 |
| Zamykání buněk | 15 |
| Reset buňky | 0.4 |
| Reset mřížky | 0.2 |
| Délka inicializační fáze | 100 |

Tabulka 5.1: Parametry algoritmu pro evoluci přechodových pravidel celulárního automatu.




Obrázek 5.2: Ukázka vývinu celulárního automatu z počáteční konfigurace do podoby cílového vzoru.

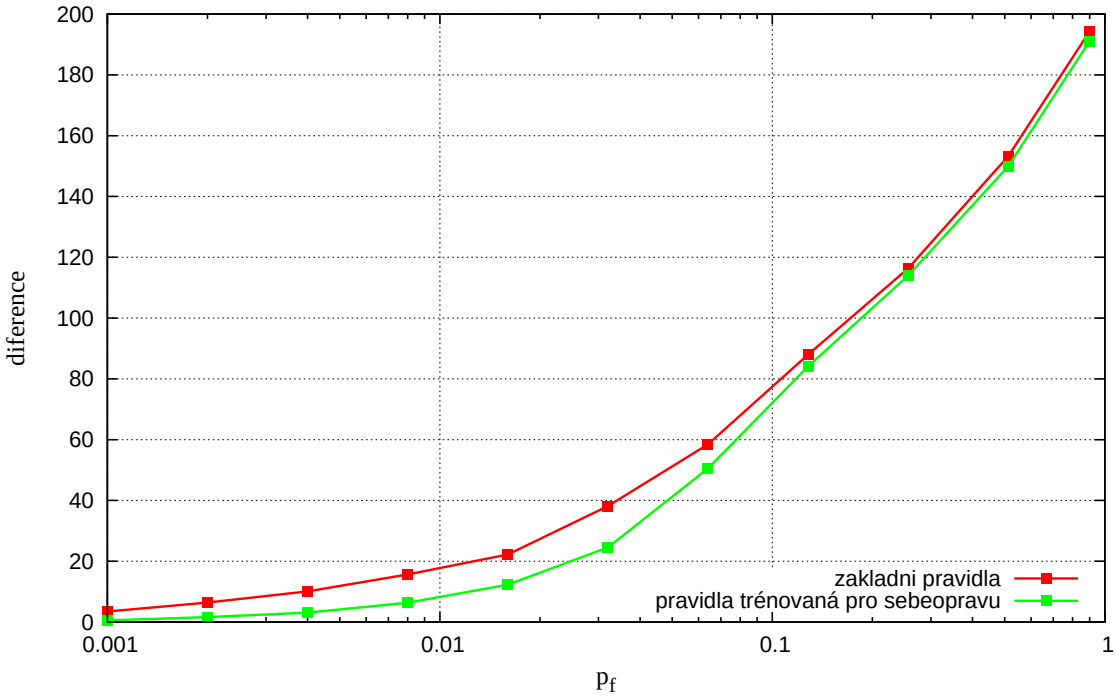
## 5.2 Chování automatu při trvalém působením projevů prostředí

V této podkapitole je zkoumána odolnost celulárního automatu vůči projevům prostředí. Za takový projev přitom považujeme náhodnou změnu stavu, která je aplikována na každou buňku s pravděpodobností $p_f \in \langle 0; 1 \rangle$. Cílem experimentů je zjištění závislosti hodnoty fitness na pravděpodobnosti poškození a určení oblasti hodnot, při kterých nedochází k výrazné destrukci cílového vzoru (tzv. fault tolerant zone). Zároveň experimenty slouží jako prostředek k porovnání úspěšnosti běhu celulárního automatu s pravidly trénovanými pouze pro vývin z počáteční do cílové konfigurace vůči pravidlům trénovaným i pro sebeopravu. Provedená měření jsou zaznamenána v tabulce 5.2 a graficky znázorněna na obrázku 5.3.

| $\mathbf{p}_f$ | 0.001 | 0.002 | 0.004 | 0.008 | 0.016 | 0.032 | 0.064 | 0.128 | 0.256 | 0.512 | 0.9 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **(a)** | 3.4 | 6.4 | 10.1 | 15.6 | 22.2 | 38.1 | 58.4 | 87.9 | 116.3 | 146.9 | 194.3 |
| **(b)** | 0.5 | 1.6 | 3.1 | 6.3 | 12.3 | 24.5 | 52.5 | 84.1 | 113.9 | 153.2 | 191.1 |

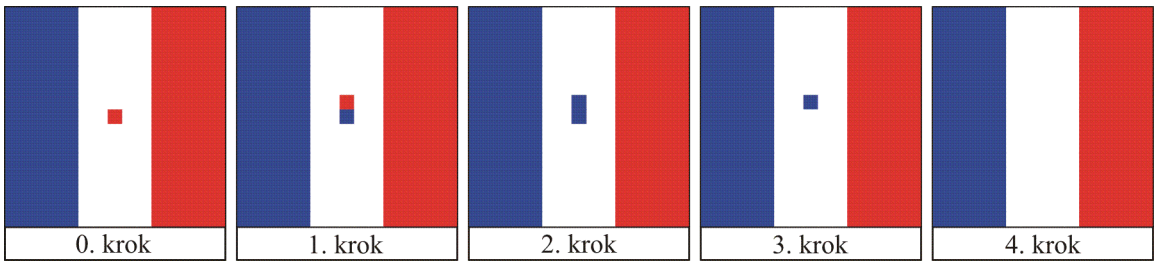
Tabulka 5.2: Zjištěné diference mezi aktuální konfigurací a požadovanou cílovou konfigurací pro vyšetřované hodnoty pravděpodobnosti poškození po sto provedených krocích vývoje automatu. Pro každou hodnotu poškození bylo provedeno dvacet měření, jejichž výsledky byly zprůměrovány. (a) základní pravidla. (b) pravidla trénovaná pro sebeopravu.

Z výsledků měření je patrné, že pozitivní efekt pravidel trénovaných pro sebeopravu se naplno projevuje zejména v oblasti hodnot pravděpodobností poškození, při kterých nedochází k výrazné destrukci vyvinutého vzoru. Do této oblasti spadají takové velikosti poškození, které zapříčiní změnu stavu pouze menšího množství buněk, přičemž v rámci této práce se za takové kritické množství považuje 10% jejich celkového počtu (to odpovídá diferenci 25). V této zóně je pak diference mezi aktuální konfigurací a požadovanou cílovou konfigurací v případě použití pravidel trénovaných pro sebeopravu i několikanásobně menší než při použití základních pravidel. Souběžně se stoupající mírou poškození




Obrázek 5.3: Graf znázorňující průměrnou diferenci mezi aktuální konfigurací a požadovanou cílovou konfigurací v závislosti na pravděpodobnosti poškození vyvinutého vzoru po sto provedených krocích vývoje automatu. Pro každou hodnotu poškození bylo provedeno dvacet měření, jejichž výsledky byly zprůměrovány.

však tento efekt pomalu mizí, což je způsobeno tím, že regenerace poškozeného stavu probíhá zpravidla v několika po sobě jdoucích krocích, při kterých jsou často modifikovány i stavy okolních buněk. S vyšší pravděpodobnosti poškození pak vzrůstá riziko narušení tohoto procesu další změnou stavu vyšetřované buňky či buňky v jejím okolí, čímž je znehodnocen použitý "scénář" regenerace a musí být vybrán jiný, což celkovou dobu tohoto procesu prodlouží. Při soustavném poškozování této oblasti tedy mají šanci proběhnout jen krátké, často pouze jednokrokové, scénáře obnovy. Pro větší názornost je ukázka fiktivního vícekrokového procesu obnovy zobrazena na obrázku 5.4.

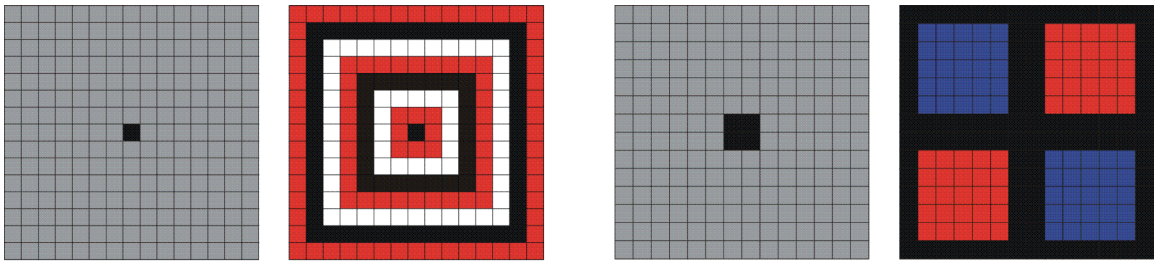



Obrázek 5.4: Ukázka vícekrokového procesu regenerace buňky na pozici (7, 7) zasažené náhodnou změnou stavu.

## 5.3 Regenerace automatu po jednorázovém poškození

V této podkapitole je zkoumána schopnost regenerace automatu po relativně velkém poškození vyvinutého vzoru vyjádřeném změnou stavů značného počtu buněk mřížky. Takové poškození přitom proběhne jen v jednom kroku a v dalších krocích je sledováno zotavení automatu. Cílem experimentů je zjištění, v jaké míře jsou pravidla vytvořená dříve popsaným algoritmem schopna takovou regeneraci provést, případně jaký čas jim to zabere. Kromě toho opět sledujeme rozdíl ve schopnostech základních pravidel a pravidel trénovaných i pro sebeopravu.

Experimenty v rámci této podkapitoly provedeme na dříve představené úloze "francouzská vlajka" (kapitola 3.4) a dále na úlohách "pruhy" a "dlaždice", které jsou zobrazeny na obrázku 5.5.



(a) (b)

Obrázek 5.5: Úlohy vývinu využité v rámci experimentů v této kapitole (dvojice počáteční konfigurace – cílová konfigurace): (a) pruhy, (b) dlaždice.

### Poškození definovanými vzory

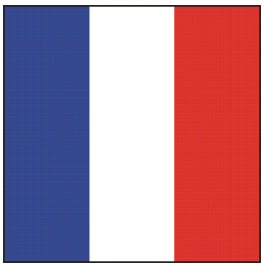
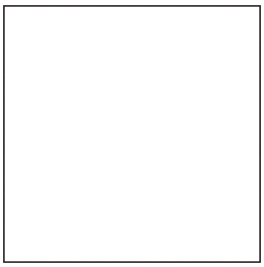
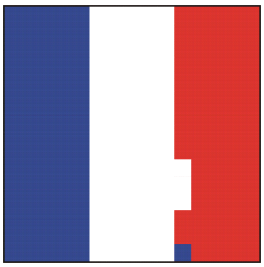
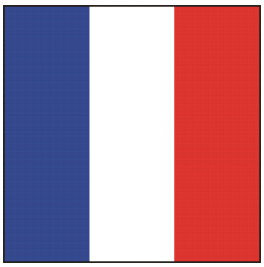
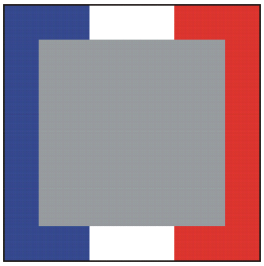
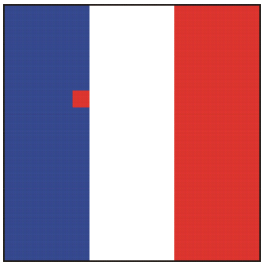
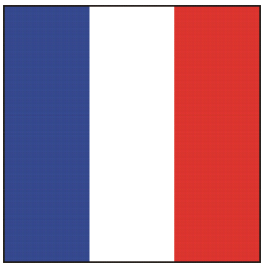
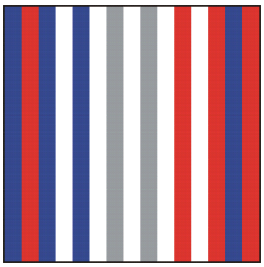
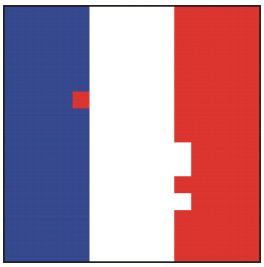
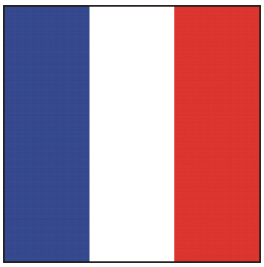
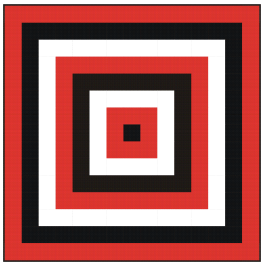
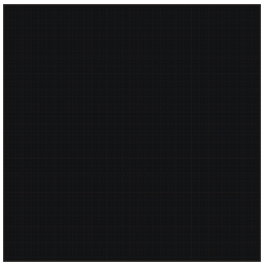
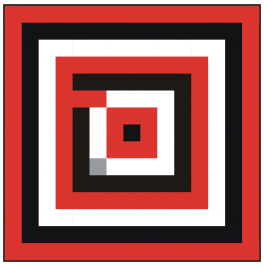
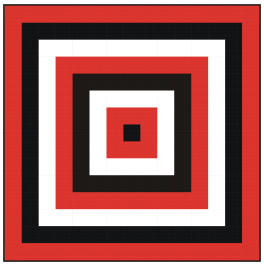
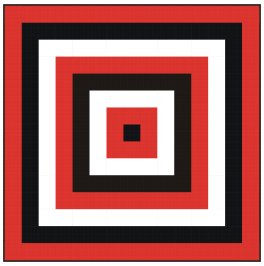
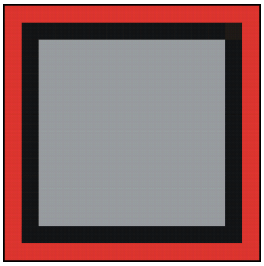
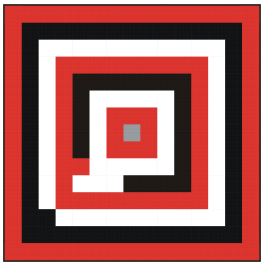
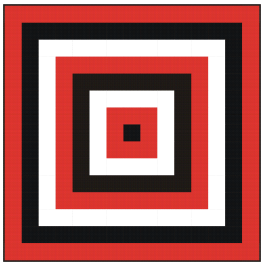
Tato sada experimentů slouží ke zkoumání regeneračních schopností automatu po jeho poškození pevně stanoveným vzorem. Takové poškození se přitom aplikuje na plně vyvinutý automat, jehož konfigurace odpovídá tvaru cílového vzoru. Jednotlivé pokusy na všech uvažovaných úlohách jsou zobrazené na obrázku 5.6. V experimentech je sledována doba regenerace, vyjádřená počtem kroků, a její úspěšnost, vyjádřená diferencí určující počet buněk, ve kterých se zregenerovaný vzor liší od cílového vzoru. Toto vyhodnocení se provádí pro automat řízený základními pravidly i pro automat řízený pravidly pro sebeopravu.

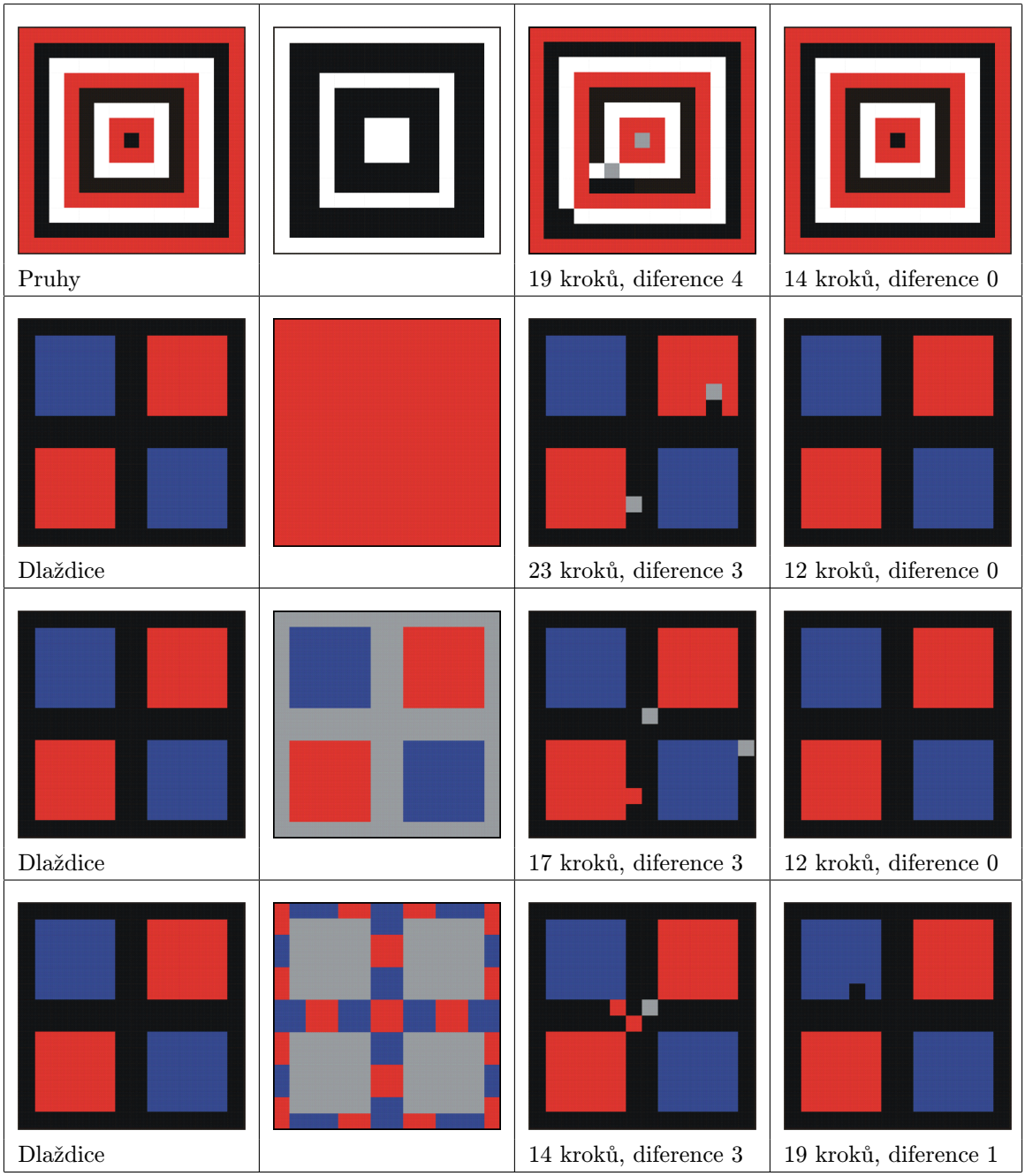
Z výsledků experimentů je patrné, že oba typy pravidel vykazují značnou regenerační sílu. Bez výjimky ve všech případech pak byly úspěšnější pravidla trénovaná pro sebeopravu, která vykazovala také výrazně kratší dobu regenerace.

### Poškození procentuálně definované části mřížky

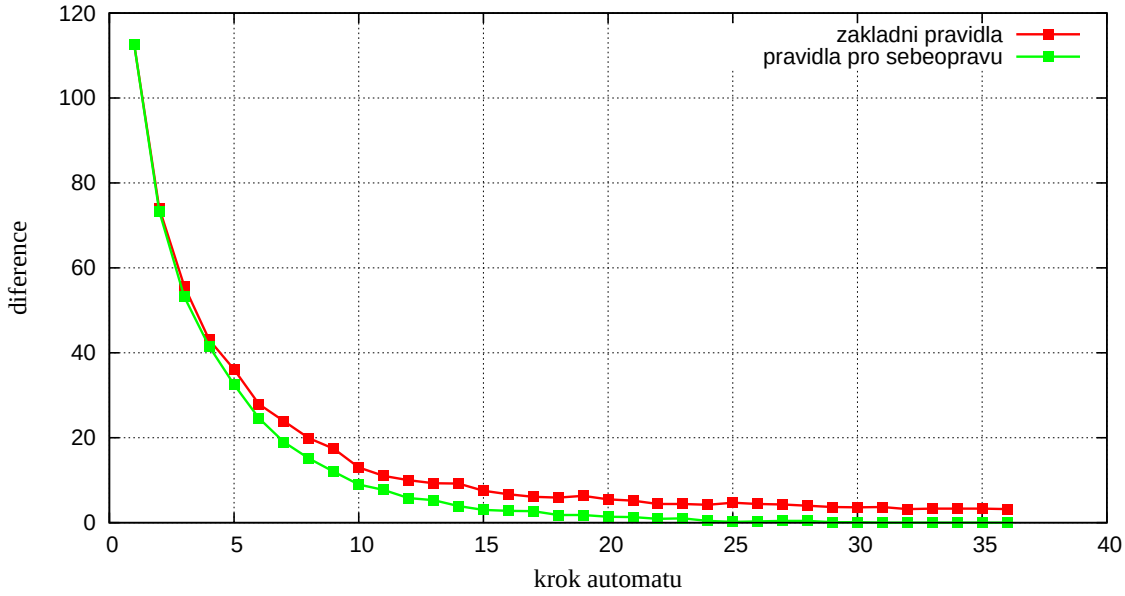
Tato sada experimentů slouží rovněž ke zkoumání regeneračních schopností automatu. Na rozdíl od předchozích pokusů však není poškození vyjádřeno pevně definovaným vzorem, nýbrž procentem udávajícím, jaké množství buněk změní svůj stav. Výběr těchto buněk je přitom náhodný.

Provedené experimenty jsou shrnuty v grafech 5.7 a 5.8. Jejich výsledky potvrzují dříve zjištěné skutečnosti, že pravidla pro sebeopravu jsou úspěšnější než základní pravidla, a že při jejich využití je doba regenerace automatu kratší.

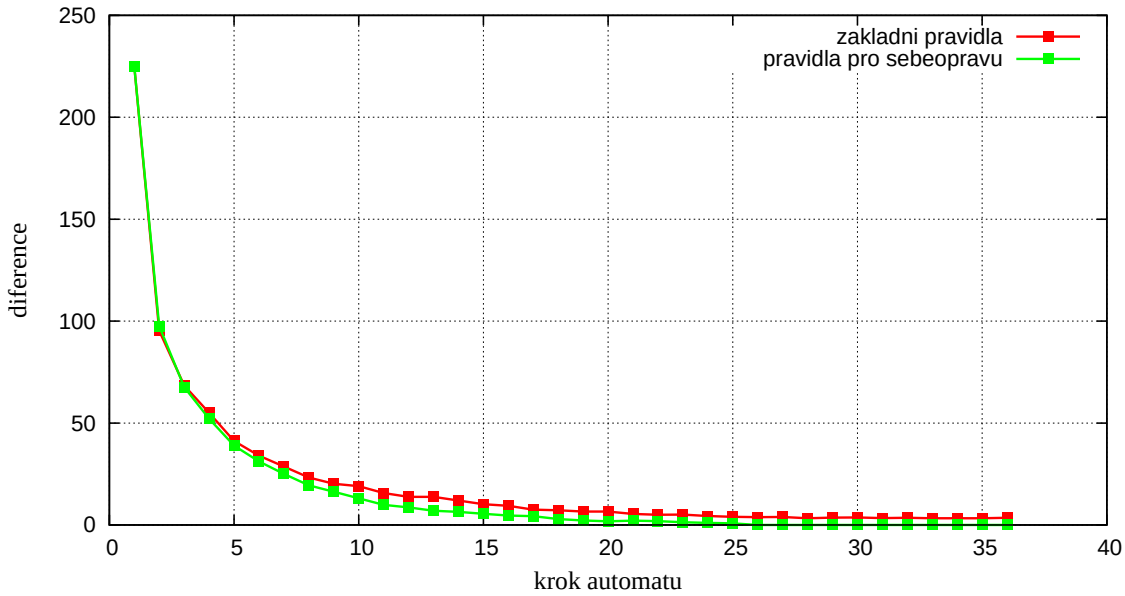
| Úloha | Poškození | Regenerace 1 | Regenerace 2 |
|---|---|---|---|
|  Francouzská vlajka |  |  15 kroků, diference 4 |  15 kroků, diference 0 |
|  Francouzská vlajka |  |  31 kroků, diference 1 |  13 kroků, diference 0 |
|  Francouzská vlajka |  |  27 kroků, diference 4 |  13 kroků, diference 0 |
|  Pruhy |  |  16 kroků, diference 3 |  15 kroků, diference 0 |
|  Pruhy |  |  20 kroků, diference 6 |  18 kroků, diference 0 |




Obrázek 5.6: Zadání a výsledky provedených experimentů. Poškození vyvinutého vzoru určeného sloupcem *Úloha* proběhne podle předlohy ze sloupce *Poškození*. Sloupec *Regenerace 1* pak udává konfiguraci automatu po procesu zotavení řízeného základními pravidly. Sloupec *Regenerace 2* má stejný význam, avšak zotavení je řízeno pravidly trénovanými pro sebeopravu. V těchto posledních dvou sloupcích je také informace o počtu kroků, které byly k zotavení potřeba a diferenci zregenerované konfigurace od cílové konfigurace.




Obrázek 5.7: Graf znázorňující závislost diference mezi aktuální konfigurací a požadovanou cílovou konfigurací na kroku automatu po 50% poškození. Objektivita výsledků je podložena deseti nezávislými běhy pro každou z vyšetřovaných úloh, přičemž výsledky běhů byly zprůměrovány.




Obrázek 5.8: Graf znázorňující závislost diference mezi aktuální konfigurací a požadovanou cílovou konfigurací na kroku automatu po 100% poškození. Objektivita výsledků je podložena deseti nezávislými běhy pro každou z vyšetřovaných úloh, přičemž výsledky běhů byly zprůměrovány.

## 5.4 Diskuze

V rámci této kapitoly byla prověřena schopnost vývinu a sebeopravy automatu s přechodovými funkcemi získanými pomocí dříve představeného algoritmu založeného na celulárním programování. Výsledky provedených experimentů ukazují, že tento algoritmus je schopen nalézt přijatelné řešení u řady netriviálních úloh.

Řešení je v relativně krátkém čase nalezeno u dvourozměrných celulárních automatů s 2, 3 a 4 stavy do rozměrů mřížky $20 \times 20$ buněk. Souběžně se stoupajícím počtem použitých stavů však dochází k výraznému nárůstu složitosti problému, což se negativně projevuje na časové náročnosti. Pokusy s větším počtem stavů proto nedosahovaly uspokojivých výsledků. Nalezení přijatelného řešení je přitom silně závislé na počáteční volbě parametrů.

Experimenty týkající se poškození automatu prokázaly, že pravidla trénovaná speciálně pro sebeopravu mají větší schopnost regenerace než pravidla v základním tvaru, a že délka tohoto procesu je u nich kratší. Při práci automatu vystaveného permanentnímu vlivu prostředí existuje silná závislost pozitivního efektu pravidel trénovaných pro sebeopravu na velikosti poškození v každém kroku vývoje. V pásmu poškození do 10% vykazují totiž speciálně trénovaná pravidla i několikanásobně menší diferenci mezi aktuální konfigurací a požadovanou cílovou konfigurací než základní pravidla.

# Kapitola 6

# Závěr

V rámci této práce byla představena metoda evoluce přechodových funkcí vedoucí k vytvoření celulárního automatu se schopností sebeopravy. Tato metoda vychází z algoritmu celulárního programování a využívá principů biologického developmentu.

Z výsledků pokusů je patrné, že algoritmus je schopen nalézt uspokojivé řešení v řadě netriviálních úloh. Přechodové funkce byly vypočteny v relativně krátkém čase pro úlohy až s čtyřmi stavy a rozměrem mřížky 20 x 20 buněk. S větším počtem stavů však dojde k značnému rozšíření prohledávacího prostoru a nalezení přijatelného řešení proto trvá výrazně delší dobu. Co se týče procesu regenerace, pak bylo demonstrováno, že přechodové funkce trénované speciálně pro sebeopravu jsou v porovnání s přechodovými funkcemi nepodstupující tento trénink úspěšnější – dokáží zregenerovat více buněk a celý proces jim zabere méně času.

Řešený problém poskytuje dostatek prostoru pro další výzkum. Kupříkladu by bylo vhodné implementovat automatický výpočet parametrů algoritmu podle charakteru dané úlohy, přičemž je zřejmé, že taková kalkulace by musela být podložena vzorci kombinujícími více faktorů, na kterých je daný parametr závislý. Po takové úpravě by jistě bylo možné řešit obtížnější úlohy s větší úspěšností.

# Literatura


[1] Chavoyaa, A.; Duthen, Y.: A cell pattern generation model based on an extended artificial regulatory network. *BioSystems*, 2008: s. 95–101, ISSN 0303-2647.

[2] Davis, L.: *Genetic Algorithms and Simulated Annealing*. Morgan Kaufmann Pub, 1987, ISBN 978-0934613446.

[3] Devert, A.; Bredeche, N.; Schoenauer, M.: Robust Multi-Cellular Developmental Desig. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, ACM, 2007, ISBN 978-1-59593-697-4, s. 982–989.

[4] Goldberg, D. E.: *Genetic Algorithms in Search,Optimization Machine Learning*. Addison-Wesley Professional, 1989, ISBN 978-0201157673.

[5] Holland, J. H.: *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975, ISBN 978-0-262-58111-0.

[6] Kvasnička, V.: *Evolučné algoritmy*. STU Bratislava, 2000, ISBN 80-227-1377-5.

[7] Mařík, V.; Štěpánková, O.; Lažanský, J.: *Umělá Inteligence (3)*. Academica, 2001, ISBN 80-200-0472-6.

[8] Sipper, M.: *Evolution of Parallel Cellular Machines: The Cellular Programming Approach*. Springer, 1997, ISBN 3-540-62613-1.

# Dodatek A

# Obsah CD

Přiložené CD má následující strukturu:

| | |
|---|---|
| `/doc/` | programová dokumentace |
| `/program/install.sh` | instalační skript |
| `/program/readme.txt` | popis instalace programu |
| `/program/src.zip` | zdrojové soubory programu |
| `/text/bp.pdf` | text bakalářské práce |
| `/text/src/` | zdrojové soubory textu bakalářské práce (LaTeX) |

# Dodatek B

# Návod k použití

Program je rozdělen do následujících tří částí:

- **Tvorba úloh** – slouží k vytvoření nové úlohy (dvojice počáteční konfigurace – cílová konfigurace).
- **Evoluce pravidel** – slouží k evoluci pravidel pro vývin z počáteční do cílové konfigurace.
- **Analýza pravidel** – slouží k analýze a testování funkčnosti vyvinutých pravidel.

## B.1 Tvorba úloh

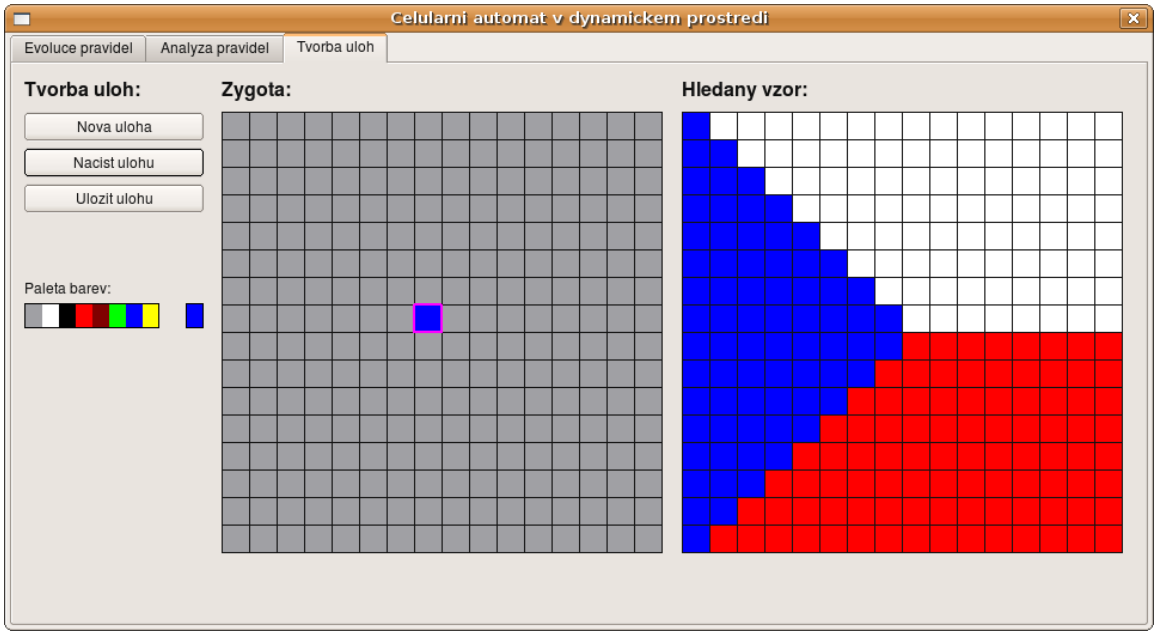
V záložce "Tvorba úloh" (viz obrázek B.1) lze vytvářet či editovat zadání úloh – dvojic počáteční konfigurace, cílová konfigurace. Jejich konkrétní podoba se vytváří pomocí štětce a palety barev. Ovládací prvky v této části mají následující význam:

- **Nová úloha** – vytvoření prázdné mřížky zadaných rozměrů pro počáteční konfiguraci – zygotu a cílovou konfiguraci – požadovaný vzor.
- **Načíst úlohu** – výběr souboru s již vytvořenou úlohou.
- **Uložit úlohu** – uložení vytvořené úlohy do souboru.

## B.2 Evoluce pravidel

V záložce "Evoluce pravidel" (viz obrázek B.2) lze ke konkrétní úloze vytvářet přechodová pravidla. Jejich kvalita je silně závislá na zadání korektních parametrů evoluce, přičemž metodika výpočtu těchto parametrů je blíže popsána v kapitole 4.1. V okně aplikace je kromě samotných textových polí pro zadání parametrů také miniaturní znázornění řešené úlohy, dvojice počáteční konfigurace – cílová konfigurace, a graf zobrazující aktuální průběh evoluce pravidel. Ovládací prvky v této části mají následující význam:

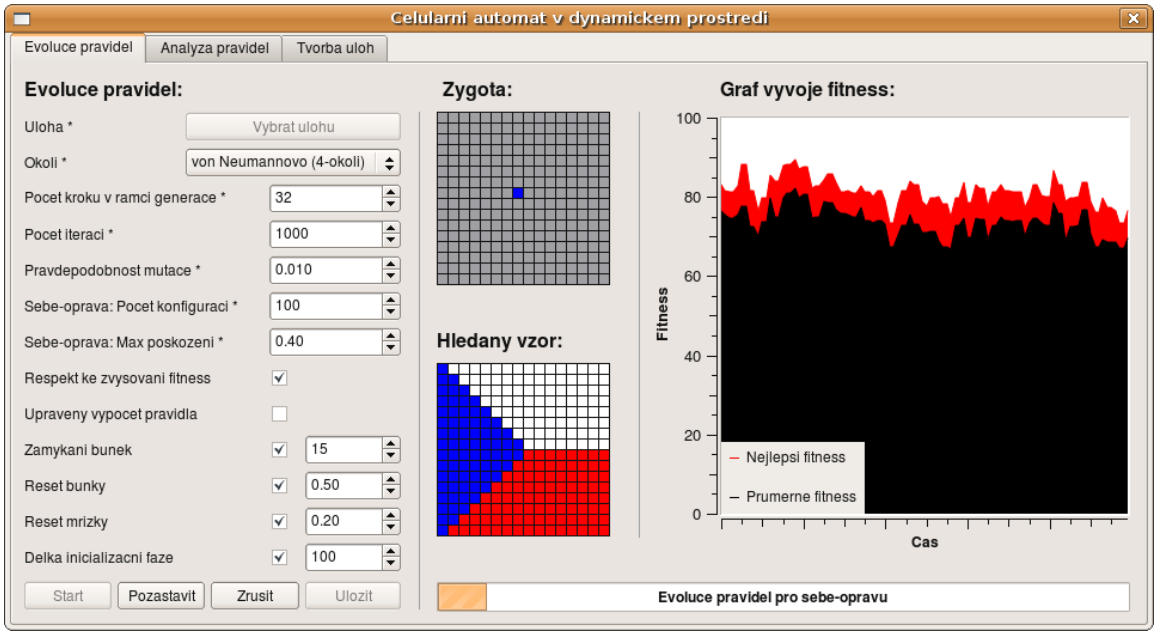
- **Úloha** – výběr souboru se zadáním konkrétní řešené úlohy.
- **Okolí** – výběr uvažovaného okolí: von Neumannovo / Moorovo.




Obrázek B.1: Ukázka aplikace – záložka "Tvorba úloh"

- **Počet kroků vývoje v rámci generace** – počet kroků vývoje celulárního automatu, po němž dochází k ohodnocení úspěšnosti pravidel.
- **Počet iterací** – celkový počet iterací algoritmu celulárního programování.
- **Pravděpodobnost mutace** – pravděpodobnost aplikace operátoru mutace.
- **Sebeoprava: počet konfigurací** – počet konfigurací, na kterých se trénuje schopnost sebeopravy.
- **Sebeoprava: max poškození** – procentuální vyjádření maximálního možného poškození uvažovaného při trénování pravidel pro sebeopravu, hodnota z intervalu $\langle 0; 1 \rangle$.
- **Respekt ke zvyšování fitness** – úprava spočívající v tom, že zvyšuje-li se fitness hodnota vyšetřované buňky ve dvou po sobě jdoucích krocích, nezjišťuje se, zda v jejím okolí existují jedinci s vyšší fitness hodnotou, ze kterých by za normálních okolností byla sestavena nová přechodová funkce.
- **Upravený výpočet pravidla** – úprava spočívající v tom, že při výpočtu nového pravidla pro buňku se uvažují sousedé z Moorova okolí (9-okolí), přestože pravidla jsou tvořena sousedy z von Neumannova okolí (5-okolí).
- **Zamykání buněk** – úprava spočívající v tom, že má-li buňka maximální hodnotu fitness v zadaném počtu po sobě jdoucích kroků, pak se její pravidlo přestane uvažovat sousedy pro výpočet nového pravidla (při operací křížení).
- **Reset buňky** – míra úspěšnosti buňky nutná k tomu, aby nešlo k resetu pravidel buňky.

- **Reset mřížky** – zprůměrovaná míra úspěšnosti všech buněk mřížky nutná k tomu, aby nedošlo k resetu pravidel všech buněk mřížky.
- **Délka inicializační fáze** – počet vývojových kroků, po kterých bude možné uplatňovat princip resetu buňky a resetu mřížky (jejich náhodného znovunastavení).

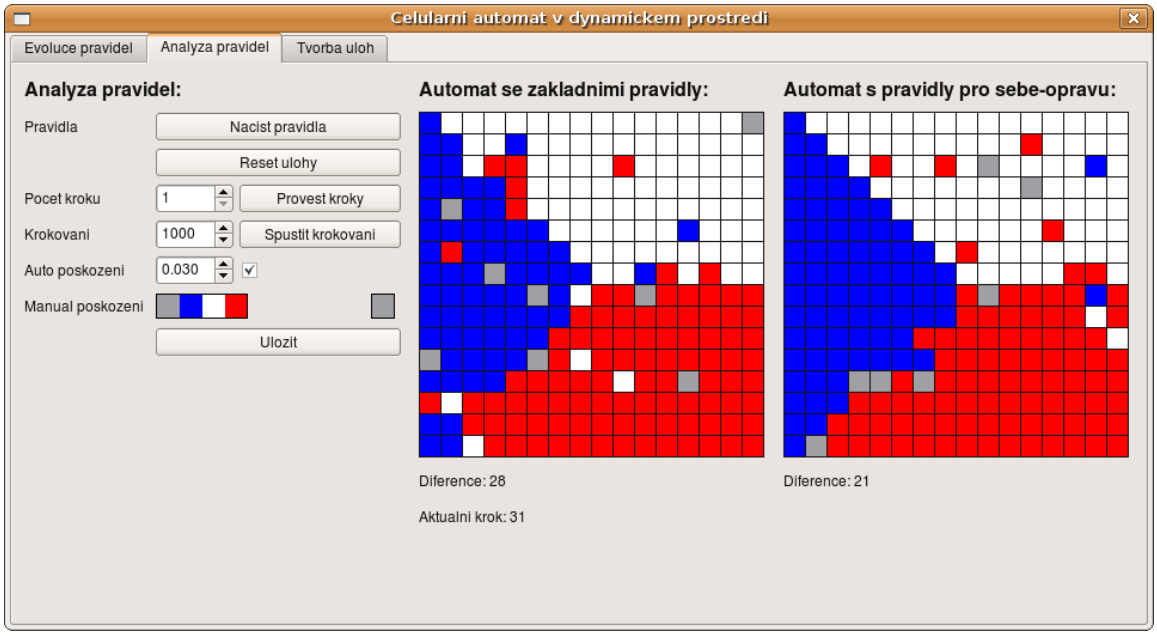



Obrázek B.2: Ukázka aplikace – záložka "Evoluce pravidel"

## B.3 Analýza pravidel

V záložce "Analýza pravidel" (viz obrázek B.3) lze ověřit kvalitu vyvinutého řešení. Je možné sledovat samotný vývin cílového vzoru, ale i jeho regeneraci po poškození. Poškození lze provést pomocí štětce a palety barev, nebo nastavením procentuální části plochy, jejíž buňky v následujícím kroku náhodně změní svůj stav. V okně aplikace jsou přitom dvě mřížky – jedna pro automat řízený základními pravidly, druhá pro automat řízený pravidly pro sebeopravu. Ovládací prvky v této části mají následující význam:

- **Načíst pravidla** – výběr souboru s pravidly, kterými bude vývin řízen.
- **Reset úlohy** – nastavení konfigurací obou mřížek do podoby zygoty.
- **Počet kroků** – počet kroků vývinu, který bude proveden po stisku tlačítka "provést kroky".
- **Krokování** – počet milisekund, po kterých bude automaticky proveden další krok vývinu při zapnutém automatickém krokování.
- **Automatické poškození** – velikost části mřížky, jejíž buňky v dalším kroku náhodně změní svůj stav, hodnota z intervalu $\langle 0; 1 \rangle$.




Obrázek B.3: Ukázka aplikace – záložka "Analýza pravidel"

## B.4 Možnosti spuštění programu

Program lze spustit s parametry příkazové řádky mající následující význam:

Synopsis: `./automaton [-u usage] [-i inputFile]`
`-u` Určuje využití programu, 1 = evoluce pravidel, 2 = analýza pravidel
`-i` Určuje cestu ke vstupnímu souboru se zadáním úlohy při `-u 1`, s pravidly při `-u 2`

Výchozí nastavení parametrů evoluce a analýzy pravidel je určeno konfiguračním souborem `configuration.txt` nacházejícím se v kořenové složce programu.