Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Genetické programování a celulární automaty

Diplomová práce

Autor: Iva Dibitanzlová
Studijní obor: aplikovaná informatika

Vedoucí práce: prof. RNDr. Josef Hynek, MBA, Ph.D.

Hradec Králové duben 2016

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracovala samostatně a s použitím uvedené literatury.

V Hradci Králové dne Iva Dibitanzlová

Poděkování

Děkuji vedoucímu této diplomové práce, prof. RNDr. Josefu Hynkovi, MBA, Ph.D., za cenné rady a připomínky a za poskytnutí odborné literatury k danému tématu.

Anotace

Cílem této diplomové práce je shrnutí principů genetického programování, popsání celulárních automatů a následné využití získaných poznatků k řešení problému klasifikace většiny pro celulární automaty prostřednictvím genetického programování. Práce je rozdělena na teoretickou a praktickou část. V první části je popsáno genetické programování, celulární automaty a problém klasifikace většiny. Druhá část se zaměřuje na řešení tohoto problému pomocí vlastního programu implementovaného v jazyce Java, který využívá genetické programování.

Annotation

Title: Genetic programming and cellular automata

The objectives of this diploma thesis is to summarize the main principles of genetic programming and cellular automata and to use them on discovery of a cellular automata rule for the majority classification problem with genetic programming. This thesis consists of theoretical and practical part. The first part describes genetic programing, cellular automata and the majority classification problem. The second part is dedicated to solving this problem with a program implemented in Java language that uses genetic programming.

Obsah

1 Úvod

V dnešní době se každý den setkáváme s počítači, které jsou naprogramovány k provádění přesně definovaných úkolů. Programy bývají tím složitější, čím náročnější je splnění daného úkolu. Existuje však i opačný přístup. V této práci budou předvedeny programy, jejichž chování je naopak velmi jednoduché, ale které přitom dokáží řešit i problémy, které jsou natolik komplexní, že je nelze řešit pomocí tradičních programů. Druh jednoduchých programů, který bude popsán v následujících kapitolách a který je jádrem přiložené aplikace, se označuje jako *celulární automaty*. Celulární automaty tvoří posloupnost buněk, které v každém kroku mění stav na základě původního stavu jak sebe sama, tak také na základě stavu buněk ve svém nejbližším okolí. Změna je určena jednoduchým pravidlem, které je pro všechny buňky stejné. Podstatou celulárních automatů je jejich schopnost produkovat celkové chování, které může být jak pravidelné a jednoduché, tak vysoce komplexní nebo chaotické, a to stále za použití jediného pravidla. Z tohoto chování vyplývá, že při nalezení vhodného pravidla mohou celulární automaty nacházet řešení mnohých komplexních problémů.

Protože pro člověka je příliš složité programovat celulární automaty tak, aby dosáhl jejich konkrétního globálního chování, v rámci této práce je jejich tvorba prováděna prostřednictvím *genetického programování*. Genetické programování je jedním z mnoha přístupů k oblasti umělé inteligence, která se snaží napodobit inteligentní chování prostřednictvím strojů. Vychází z analogie s evoluční biologií, především z Darwinovy teorie o přežití nejsilnějších organismů. Namísto populace živých organismů se v této technice pracuje s populací počítačových programů, které se v každé nové generaci vyvíjejí a zdokonalují za účelem najít takový program, který je pro řešení stanoveného problému optimální. Tímto hledaným programem může být rovněž i pravidlo pro celulární automaty.

V první, teoretické části práce jsou představeny evoluční výpočetní techniky, ze kterých genetické programování vychází (především technika genetických algoritmů). Dále se teoretická část zaměřuje na podrobnosti genetického programování a jeho rozdíly oproti genetickým algoritmům. Poté jsou popsány celulární automaty, včetně klasifikace jejich chování a příkladů oblastí, v níž lze poznatky získané jejich studiem prakticky využít. Následuje popis problému klasifikace většiny, jehož řešení prostřednictvím vlastní implementace genetického programování je hlavním cílem této práce. Dosažené výsledky jsou prezentovány v závěrečných kapitolách.

2 Genetické programování

V systémech strojového učení se řešení předkládaných problémů nachází v prostoru hledání, který obsahuje všechna potenciální řešení. K jeho prohledávání lze využít řadu metod, z nichž Banzhaf [1] uvádí tři základní: hledání naslepo (*blind search*), metodu výstupu na vrchol (*hill climbing*) a paprskové hledání (*beam search*). Hledání naslepo vybírá řešení z prostoru hledání nezávisle na struktuře problému a výsledcích předchozích kroků hledání a používá se pouze v případech, kdy je počet potenciálních řešení dostatečně nízký. Metoda výstupu na vrchol začíná na libovolně zvoleném řešení z prostoru hledání, toto řešení transformuje a výsledek si ponechá pro další zpracování tehdy, pokud došlo ke zlepšení; jinak jej znovu transformuje, popř. vybere jiné počáteční řešení. Tuto metodu používají například některé algoritmy neuronových sítí.

Genetické programování, podobně jako další evoluční výpočetní techniky zmiňované dále, používá metodu paprskového hledání, která je určitým kompromisem mezi dvěma výše uvedenými metodami. Tato metoda nepracuje pouze s jedním řešením z prostoru hledání, ale s *populací* těchto řešení. V paprskovém hledání se používají metriky pro hodnocení řešení, pomocí kterých se vybírají řešení nejvhodnější pro další transformace [1].

Princip genetického programování vychází z myšlenky Charlese Darwina o přežití a reprodukci nejsilnějších [2]. Na rozdíl od tradičního programování, kdy programátor řeší daný problém ručním psaním jediného programu, genetické programování automaticky vytváří náhodnou populaci programů a následně je ohodnocuje (určuje jejich *zdatnost*). Na základě jejich zdatnosti, prokázané při řešení problému, se vybírají programy z populace, aby vytvořily další generaci. Ta vznikne pomocí *genetických operací* nad těmito programy (*jedinci*), například pomocí křížení nebo mutace. Proces tvorby nových generací pokračuje, dokud není nalezeno řešení daného problému, nebo dokud není splněna ukončovací podmínka (obvykle vytvoření maximálního počtu generací).

2.1 Evoluční výpočetní techniky

Genetické programování je jednou z oblastí evolučních výpočetních technik (*Evolutionary Computation*). V této kapitole jsou stručně představeny nejznámější z nich. Nejprve bude popsán základní koncept evolučních výpočetních technik, který vychází z biologie a z teorie samoorganizace.

Do evolučních výpočetních technik patří následující oblasti (toto členění je převzato z [3] a [4]):

- genetické algoritmy,
- genetické programování,
- evoluční programování,
- evoluční strategie,
- optimalizace hejnem částic (*Particle Swarm Optimization*).
- diferenciální evoluce,
- kulturní evoluce,
- koevoluce.

Výčet těchto technik není úplný a existují i jiné přístupy k jejich členění (například je možné považovat genetické programování za druh genetických algoritmů).

Prvních pět výše uvedených oblastí evolučních výpočetních technik bude podrobněji popsáno v následujících kapitolách. *Diferenciální evoluce* je technika podobná genetickým algoritmům, která se liší v použitém mechanismu reprodukce. *Kulturní evoluce* modeluje evoluci kultury populace a způsob, jakým kultura ovlivňuje genetickou a fenotypovou evoluci jedinců. *Koevoluce* umožňuje jedincům v populaci vývoj na základě společné kooperace nebo soutěže, čímž jedinci získávají vlastnosti nutné k přežití [4].

Oblast evolučních výpočetních technik vychází z biologické genetiky, zejména z konceptu chromozómů. **Chromozómy** jsou struktury uvnitř buněk žijících organismů, které slouží k předávání genetické informace. Každý živočišný druh má charakteristický počet chromozómů, například u člověka jde o 46 chromozómů, rozdělených na 23 homologních (shodných) párů u žen, nebo 22 homologních a 1 heterologní pár u mužů. V každém páru je jeden odvozen od otce a druhý od matky. V evolučních výpočetních technikách jsou vytvářené vzory nebo řetězce analogické k chromozómům v biologických systémech a výraz *chromozóm* se například v genetických algoritmech užívá zcela běžně. Umělé chromozómy jsou však navrženy podle biologických pouze přibližně. Zatímco biologické chromozómy obsahují lineární vlákna DNA, nukleové kyseliny, tvořící komplexní strukturu dvojité šroubovice, umělé chromozómy tvoří obvykle pouze řetězce binárních nebo reálných číselných hodnot. Na rozdíl od umělých chromozómů se biologické liší délkou, přestože jeden konkrétní chromozóm míva u všech organismů stejnou délku. Další rozdíl se nachází ve způsobu reprodukce. V biologii se chromozómy duplikují během buněčného dělení a během reprodukce vajíčko a spermie přispívají jedním chromozómem k sestavení každého homologního páru. V evolučních výpočetních technikách se proces analogický k duplikaci nazývá

reprodukce a syntéza nových chromozómů na základě rodičovských se nazývá **křížení** nebo *rekombinace.* Na rozdíl od lidské reprodukce, kde oba rodičovské chromozómy ovlivňují výsledný chromozóm potomka každý z 50 %, bývá při umělém křížení tento podíl náhodný [3].

Dalším pojmem převzatým z biologie je **genotyp**, který v genetice popisuje kolekci chromozómů, která přesně specifikuje daný organismus. V evolučních výpočetních technikách jsou pojmy *chromozóm* a *genotyp*, případně také *struktura,* často zaměnitelné [3]. Výraz *genotyp* používám ve své implementaci genetického programování jako označení pro jedince v populaci, složeného z **genů**. V genetickém programování jsou za geny považovány uzly a listy stromového grafu, představujícího jedince. V biologii jsou z genů sestaveny chromozómy. Každý gen je identifikován svým umístěním a svou funkcí (například gen určující barvu vlasů). Geny jsou specifické segmenty chromozómů s přidělenými konkrétními funkcemi. Konkrétní hodnoty, kterých mohou geny nabývat, se nazývají **alely** (například alela genu barvy vlasů představuje konkrétní barvu). V případě evolučních výpočetních systémů jsou chromozómové vzory nebo řetězce sestaveny z parametrů nebo vlastností, které mohou nabývat předem určeného rozsahu hodnot. Danou vlastnost nebo parametr tedy tvoří pevně určená pozice v umělém chromozómu (řetězci). Chromozóm pak představuje sadu parametrů nebo vlastností [3].

Samoorganizace je jedním z klíčových konceptů v oblasti umělé inteligence a evoluce. Poprvé tento pojem použil v roce 1945 W. Ross Ashby [4], který jako příklad samoorganizace uváděl nervový systém, který si v novém prostředí vyvíjí takovou vnitřní organizaci, která je tomuto prostředí přizpůsobena. Podle Ashbyho má samoorganizace dva možné způsoby vzniku. Prvním způsobem je vznik ze systému, jehož jednotlivé části jsou na počátku oddělené a neovlivněné stavem ostatních částí a které se poté změní za účelem vytváření propojení. Druhý způsob vychází z již propojených systémových komponent, které se začnou organizovat produktivním nebo smysluplným způsobem.

Dnes existuje mnoho definic samoorganizace, z nichž většina sdílí následující prvky [3]:

- samoorganizované systémy obvykle vykazují spontánní uspořádání,
- na samoorganizaci může být nahlíženo jako na neustálé snahy systému o organizaci sebe sama do stále více komplexnějších struktur, proti kterým působí síly popsané v druhém zákonu termodynamiky,
- celkový stav samoorganizovaného systému je emergentní vlastností tohoto systému,

- propojené systémové komponenty se organizují produktivním nebo smysluplným způsobem v závislosti na lokálních informacích a globální dynamika se vynořuje na základě lokálních pravidel,
- komplexní systémy se mohou samoorganizovat,
- samoorganizační proces se pohybuje blízko okraje chaosu (*edge of chaos*).

Jedním z mnoha příkladů samoorganizace jsou vzory generované celulárními automaty, specifikovanými jednoduchými matematickými funkcemi. Celulární automaty jsou detailně popsány v kapitole 3.

Koncept samoorganizace významně ovlivnil pohled na evoluci a umělou inteligenci [3]. Evoluce v evolučních výpočetních technikách však vychází především z myšlenky o přežití nejsilnějších a přirozeném výběru, kterou popsal Charles Darwin ve svém díle *On the Origin of Species by Means of Natural Selection* v roce 1859. Tato myšlenka je založena na různorodosti jednotlivých druhů v populaci, jež se projevuje odlišnými strukturami a chováním (a kterou způsobují variace chromozómů, přestože chromozómy ještě v Darwinově době nebyly známé). Důležité je zde rovněž působení vnějšího prostředí. Variace ve struktuře a chování se odrážejí v míře přežití a reprodukce. Jedinci, kteří jsou lépe přizpůsobeni k životu v daném prostředí, přežívají a reprodukují se větší mírou než ti, kteří jsou přizpůsobení hůře (kteří jsou méně *zdatní* ve svém prostředí) [6].

Darwinovu myšlenku rozšiřuje tzv. neodarwinovský pohled na evoluci, podle kterého je kompozice chromozómů určena jeho rodiči (minimálně u zvířat a lidí). Dále zavádí myšlenku náhodné mutace, která poskytuje potřebný atribut diverzity. [3].

2.1.1 Genetické algoritmy

Koncept genetických algoritmů začal vznikat v 50. letech 20. století, kdy biologové pomocí výpočetní techniky simulovali biologické genetické systémy. **A. S. Fraser**, pracující v oblasti výzkumu epistáze (epistáze je situace, kdy je aktivita jednoho genu maskována efekty jiného genu), reprezentoval každý ze tří parametrů epistázové funkce jako 5 bitů v 15bitovém řetězci a výběr rodičovských řetězců založil na výběru těch řetězců, jejichž hodnoty parametrů produkovaly funkční hodnoty v rozmezí -1 a 1. Tato metodologie připomínala postupy současných genetických algoritmů, byla však užívána pouze v oblasti biologických, nikoli umělých systémů [3].

V roce 1975 **John H. Holland** ve svém díle *Adaptation in Natural and Artificial Systems* poskytl rámcový pohled na přirozené i umělé adaptivní systémy a předvedl

způsob, jakým mohou být evoluční procesy aplikovány v umělých systémech, jenž označil jako genetické algoritmy [6].

V přírodě se po celé délce molekuly DNA vyskytují 4 nukleotidové báze – adenin, cytosin, guanin a thymin. Sekvence těchto nukleotidových bází utvářejí řetězec chromozómu. Podřetězce obsahující tisíce a více bází za použití genetického kódu vedou ke vzniku proteinů a enzymů, které tvoří strukturu a řídí chování biologických buněk. Tyto struktury a chování umožňují jedinci provádět ve svém prostředí úkoly, vedoucí k jeho přežití, a určitou mírou se reprodukovat. Chromozómy potomka obsahují řetězce nukleotidových bází jeho rodičů, čímž je zajištěno, že řetězce bází, které vedou k mimořádně zdatným jedincům, budou větší mírou přeneseny do budoucích generací. Rovněž se u chromozómů může omezeně vyskytovat mutace. *Genetické algoritmy* se touto biologickou strukturou a chováním inspirují. Jsou to vysoce paralelní matematické algoritmy, které transformují sadu (*populaci*) matematických objektů do nové generace na základě jim přidělené hodnoty *zdatnosti*, přičemž k tvorbě této nové generace využívají operace vycházející z darwinovského principu reprodukce a přežití nejzdatnějších a operace vycházející z přirozených genetických operací (především ze sexuální rekombinace). Matematické objekty, které představují jednotlivé jedince, jsou v případě genetických algoritmů obvykle řetězce znaků o pevně stanovené délce, analogické k řetězcům nukleotidových bází v chromozómech [6].

Genetický algoritmus je doménově nezávislým algoritmem, který problém řeší prohledáváním prostoru kandidátních řešení. Každý existující bod v prostoru hledání je nejprve zakódován do reprezentace vhodné pro použití genetického algoritmu. Dále je před spuštěním algoritmu nutné určit způsob měření zdatnosti, řídící parametry, podmínku ukončení a metodu určení výsledku. Reprezentační schéma určuje transformaci, která mapuje body v prostoru hledání na konkrétní řetězec znaků o pevně stanovené délce (nebo na jinou datovou strukturu) a k ní inverzní transformaci, která mapuje každý řetězec znaků nebo strukturu na bod v prostoru hledání řešení problému. V případě, že jsou jedinci reprezentováni řetězcem znaků, je nutné stanovit délku L tohoto řetězce a abecedu o velikosti K. Nejčastěji se užívá binární abeceda o délce 2 a znacích „0“ a „1“. Pro reprezentaci lze využít i jiné datové struktury, například řetězce znaků s proměnlivou délkou. Zvolené reprezentační schéma musí být schopné pojmout řešení problému a tudíž musí zahrnovat všechny body v prostoru hledání, které mohou být výsledným řešením [2].

Genetický algoritmus je řízen ohodnocovací funkcí, jež je problémově specifická a jejímž úkolem bývá nejčastěji přiřazení numerické hodnoty každému jedinci v populaci na základě jeho zdatnosti. Obecně ohodnocovací funkce musí umět vyhodnotit

zdatnost jakéhokoli jedince, který je za běhu genetického algoritmu vytvořen. Jejím účelem je výběr jedinců pro vytvoření nové generace, který bude v souladu s darwinovským principem o přežití a reprodukci nejzdatnějších. Ti jsou na základě přidělených zdatností vybíráni prostřednictvím různých metod výběru (viz kapitola 2.2.4 *Selekční mechanismy*). Existují různé typy těchto metod, ale všechny v sobě zahrnují následující principy [2]:

- zdatnější jedinci budou vybráni spíše než méně zdatní,
- konkrétní jedinec může být zvolen vícekrát,
- výběr je stochastický.

Kvůli stochastickému výběru není zaručeno, že bude pro tvoření příští generace vždy vybrán nejzdatnější jedinec v populaci. Rovněž není zaručeno, že se tvorby příští generace nebude účastnit nejméně zdatný jedinec. Selekce upřednostňuje zdatnější jedince, ale zároveň umožňuje postup i všem ostatním. Přestože tento postup nevede k okamžitému získání nejvyšší možné celkové zdatnosti, provádí se s myšlenkou, že v příštích generacích povede k nalezení takových bodů v prostoru hledání, jejichž zdatnost bude ještě vyšší. Netriviální problémy vždy vyžadují nalezení vysoce hodnocených bodů v prostoru hledání, které nelze najít pouze metodou výstupu na vrchol (*hill climbing*) [2].

Genetický algoritmus je možné řídit pomocí parametrů *velikost populace* (*M*) a *maximální počet generací* (*G*). Populace může obsahovat desítky až milióny, ale i více, jedinců. Počet jedinců nemusí být ve všech generacích shodný, ale může se měnit (příklad viz kapitola 2.2.6 *Sekundární operace*). Dále lze určit například pravděpodobnost provádění jednotlivých genetických operací. Nezbytná je také specifikace metody pro určení výsledku a specifikace ukončovací podmínky, která určuje, kdy ukončit běh algoritmu a zobrazit výsledek. Ukončovací podmínkou může být nalezení dostatečně úspěšného řešení, ale současně také vypršení časového limitu nebo dosažení maximálního povoleného počtu generací. Určení, zda je řešení dostatečně úspěšné, záleží na konkrétním problému a na cíli uživatele. Zobrazení výsledku může probíhat jak na konci běhu genetického algoritmu, tak i v jeho průběhu po každém vytvoření nové generace a nalezení nejzdatnějšího jedince v této generaci [2].

2.1.2 Evoluční programování

Evoluční programování vyvinul **Larry J. Fogel** jako techniku, která využívá princip výběru nejzdatnějších jedinců, ale jedinou operací pro úpravu jejich struktury je mutace. Křížení se v této technice vůbec nevyužívá. Larry J. Fogel a jeho kolegové pra-

covali s konečnými automaty a strojovou inteligencí. V této oblasti dokázali pomocí evolučního programování řešit problémy, které byly pro genetické algoritmy příliš obtížné. Fogel popsal evoluční programování jako techniku založenou na zcela jiném přístupu než genetické algoritmy, která může být pro řešení některých problémů vhodnější, protože přirozená selekce neovlivňuje jednotlivé komponenty izolovaně, ale ovlivňuje veškerá chování v organismu v závislosti na jeho interakcích s prostředím. Každý jedinec v populaci totiž představuje odlišný druh, přičemž každý tento druh soupeří s ostatními o volný prostor v daném prostředí. Fogel považuje evoluční programování za implementaci principu „přežití nejschopnějších“, nikoli „přežití nejzdatnějších“ [3].

2.1.3 Evoluční strategie

Ingo Rechenberg a Hans-Paul Schwefel jako první simulovali různé verze přístupu, který byl později označen jako *evoluční strategie*. Experimentovali s mutací, aby nalezli optimální konfiguraci řady otočných desek ve větrném tunelu a v trubici s protékající tekutinou, protože techniky klesání podle gradientu (*gradient descent*) nedokázaly vyřešit soustavy rovnic pro snížení odporu vzduchu. Používali mutaci na dosud nejlepších nalezených řešení problému tak, aby prozkoumali blízká okolí těchto řešení v prostoru hledání [3].

2.1.4 Optimalizace hejnem částic

Optimalizace hejnem částic vychází ze tří různých oblastí – z umělého života obecně (především z teorie rojení (*swarming theory*), například z utváření hejn ptáků nebo ryb), z evolučních výpočetních technik (konkrétně genetických algoritmů a evolučních strategií) a ze sociální psychologie [3].

Ze sociální psychologie využívá optimalizace hejnem částic několik různých paradigmat. Prvním je teorie dynamického sociálního dopadu (*Dynamic Social Impact Theory*), která uvádí, že chování jedinců lze vysvětlit samoorganizačními vlastnostmi jejich sociálního systému, že si shluky jedinců vyvíjejí podobná přesvědčení a že se subpopulace rozcházejí jedna od druhé (polarizují se). Hlavními charakteristikami teorie sociálního dopadu jsou konsolidace, shlukování, korelace a pokračující diverzita. Konsolidace zde představuje proces, který redukuje rozdílnost v názorech, když jsou jedinci vystaveni názorům většiny. Shlukování zajišťuje, že se chování jedinců začne podobat chování jejich sousedů v sociálním prostoru. Korelace znamená, že chování, které zpočátku bylo nezávislé, vykazuje tendenci postupně se měnit na závis-

lé. Pokračující diverzita říká, že shlukování chrání menšinu před úplnou konsolidací. Ve výsledku jedinci ovlivňují jeden druhého, čímž se sobě začínají navzájem podobat. Přesvědčení jednotlivců korelují v závislosti na populační oblasti. Tato teorie se ukazuje jako konzistentní s výzkumem v oblastech sociální psychologie, ekonomie a antropologie. Druhým paradigmatem ze sociální psychologie, ze kterého vychází optimalizace hejnem částic, je Axelrodův kulturní model [3], který reprezentuje populaci jedinců jako řetězce symbolů (nebo vlastností). Pravděpodobnost interakce mezi dvěma jedinci je funkcí jejich podobnosti. Zvýšení podobnosti jedinců je výsledkem jejich interakce. Pozorovanou dynamikou je polarizace, která znamená, že se homogenní subpopulace liší jedna od druhé. Třetím použitým paradigmatem je Kennedyho adaptivní kulturní model, v němž vzájemná podobnost jedinců nezvyšuje pravděpodobnost jejich interakce a v němž jsou hranice mezi kulturními oblastmi tvořeny naopak vzájemnou rozdílností. K interakci mezi jedinci dochází tehdy, když jsou jejich zdatnosti odlišné. Jedinci hledající řešení se učí ze zkušeností svých blízkých sousedů. Dále je patrný fenomén, při kterém lze jednotlivce považovat za části (jedinci, kteří často interagují s ostatními, se začnou navzájem podobat). Kultura ovlivňuje výkon jedinců, kteří ji tvoří (jedinci získávají výhodu napodobováním svých sousedů). Kennedy zkoumal možnost, zda v tomto modelování sociálních systémů mohou hrát roli evoluční výpočty [3].

2.2 Principy genetického programování

V této kapitole je nejprve popsána návaznost genetického programování na genetické algoritmy. Poté je představen hlavní algoritmus genetického programování a struktura vytvářených programů. Dále jsou uvedeny některé selekční mechanismy a genetické operace, které se v této evoluční výpočetní technice využívají.

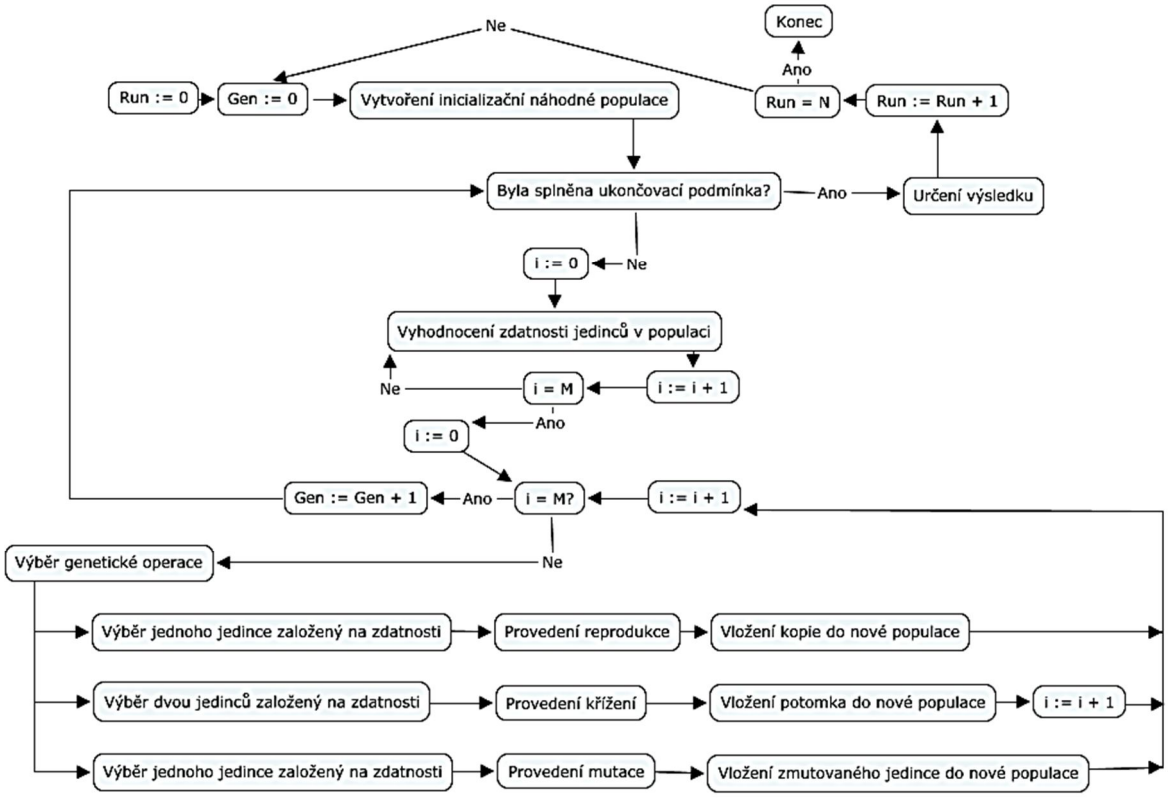
Genetické programování vychází z evoluční výpočetní techniky zvané genetické algoritmy. K počítačovému řešení problémů jej poprvé použil J. R. Koza v roce 1987 a tuto techniku pak podrobně popsal v roce 1988 [2]. Základní prvky genetického programování Koza poprvé představil v článku *Hierarchical Genetic Algorithms Operating on Populations of Computer Programs* [7] a později je upřesnil v knize *Genetic Programming: On the Programming of Computers by Means of Natural Selection* [6].

Jeho algoritmus se od GA algoritmu neliší. Rozdíl spočívá ve strukturách, které genetické programování generuje. Zatímco jiné evoluční výpočetní techniky pracují s řetězci nebo vektory binárních nebo reálných hodnot, genetické programování vyvíjí hierarchické spustitelné struktury (obecně počítačové programy), které reprezentuje pomocí stromové struktury. Dalším rozdílem oproti genetickým algoritmům, které

obecně používají pevně danou délku řetězce, je vývoj programů lišících se navzájem svou velikostí, tvarem a složitostí. V genetických algoritmech probíhá měření zdatnosti převodem řetězce na hodnoty parametrů a následným použitím těchto hodnot při řešení problému, přičemž se vhodnou metodou vypočítá zdatnost daného řešení. V genetickém programování je zdatnost jedince zjišťována spuštěním jeho programu [3].

2.2.1 Genetický algoritmus

Populace programů je vytvořena a upravována pomocí genetického algoritmu. Schéma algoritmu, které se neliší od schématu algoritmu používaného v GA, znázorňuje obrázek 1.



Obrázek 1: Schéma genetického algoritmu (převzato a upraveno z [2])

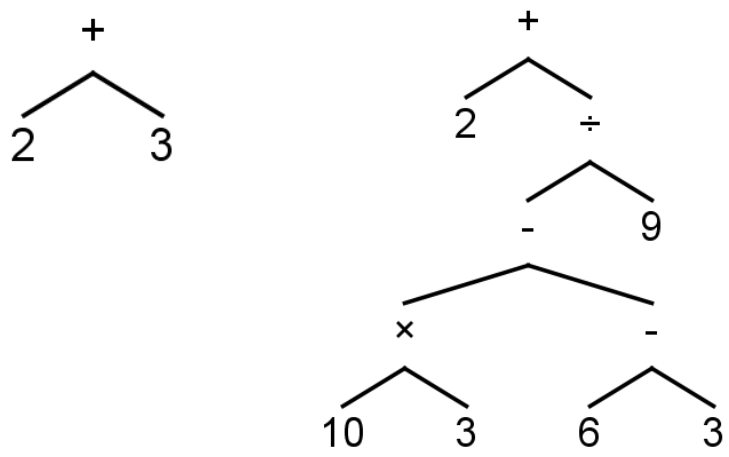
Na počátku vytvoří tento algoritmus inicializační populaci. Jedince pro tuto populaci vytvoří náhodně za použití dostupných prvků (více v kapitole 2.2.2 *Složení programů*). V této fázi je možné přidat i dodatečná omezení, například maximální hloubku vytvářených stromů.

Poté je následující postup opakován pro každou generaci. Každý jedinec je otestován při řešení zadaného problému a dle jeho úspěšnosti je mu přiřazena *zdatnost*. Tento údaj je později využit pro výběr tak, aby jedinci s vyšší zdatností měli vyšší pravděpodobnost postupu do další generace, ale zároveň tak, aby měli určitou šanci, byť nízkou, i méně zdatní jedinci. Jakmile jsou všichni jedinci ohodnoceni, využijí se genetické operátory (nejčastěji reprodukce, křížení a mutace) pro vytvoření další generace. Genetické operátory pracují s jedinci zvolenými selekčním mechanismem. Reprodukce pouze zkopíruje jedince do další generace v nezměněné podobě. Křížení, nejdůležitější operátor, vybírá na každém jedinci jeden uzel (včetně podstromu tímto uzlem začínajícím), který si jedinci vzájemně vymění a oba postoupí do nové generace. Mutace, používaná s nízkou pravděpodobností, náhodně vybírá uzel a náhodně jej mění; případně může na jeho místě vygenerovat novou větev stromu jako v případě tvorby inicializační populace.

2.2.2 Složení programů

Program se skládá z prvků z množiny funkcí a množiny terminálů, které jsou zadány programátorem před spuštěním genetického algoritmu. Terminály mohou nabývat podobu konstant různých typů (např. „5“, „3.14“, „true“ apod.), ale také se může jednat o proměnné nebo o funkce s nulovým počtem argumentů. Funkce jsou operace, které tyto terminály zpracovávají a vracejí vypočítaný výsledek, z čehož vyplývá, že funkce musí mít vždy minimálně jeden argument [8].

V genetickém programování je každý program uspořádán do struktury stromového grafu. V tomto grafu uzly představují funkce s argumenty a listy představují terminály – funkce bez argumentů. Kořenem grafu je vždy funkce, která jako své argumenty přijímá další funkce nebo terminály. Příkladem jednoduchého programu může být funkce sčítání, umístěná jako kořen, a její argumenty 2 a 3, umístěné jako podlisty kořene (viz obrázek 2), která bude vracet jako výsledek hodnotu 5.



Obrázek 2: Strom vlevo s funkcí *sčítání* a terminály *2* a *3*, který po svém vyhodnocení vrací hodnotu *5*. Strom vpravo vrací tutéž hodnotu. Vyhodnocování probíhá od listů v nejvyšší hloubce směrem ke kořenu.

Funkce se mohou v průběhu genetického algoritmu zanořovat jedna pod druhou, čímž mohou vytvářet stromy velké hloubky. Protože příliš rozsáhlý strom s velkým množstvím uzlů pravděpodobně nebude příliš efektivní při dalším křížení (při němž náhodně vybíráme některý z uzlů a v případě výběru uzlu ve velké hloubce se chování algoritmu obvykle příliš nezmění), je vhodné přidat parametr maximální hloubky. Tento parametr se využívá při tvorbě počáteční populace a při použití některých genetických operátorů (především reprodukci a mutaci s generováním nových větví stromového grafu).

2.2.3 Měření zdatnosti

John Koza v [6] uvádí 4 různé způsoby pro měření zdatnosti jednotlivých programů v populaci. Pokaždé se jedná o hodnocení číselnou hodnotou.

Čistá zdatnost (*raw fitness*) označuje základní způsob měření zdatnosti, jehož výpočet záleží na řešeném problému. Obvykle je jedinec tím zdatnější, čím vyšší má čistou zdatnost; ale pokud je cílem například minimalizace chyby a jako čistou zdatnost zjišťujeme její velikost, je tomu naopak. V případě celulárních automatů a problému klasifikace většiny se jedná o počet počátečních konfigurací, při kterých bylo dosaženo správného výsledku (podrobnosti viz kapitola 3.4 *Problém klasifikace většiny*).

Standardizovaná zdatnost (*standardized fitness*) vychází z čisté zdatnosti a jedná se o úpravu čisté zdatnosti takovou, aby nižší hodnota znamenala lepší výsledek. Při minimalizaci chyby se rovná čisté zdatnosti; v ostatních případech je dána vzorcem

$$s(i,t) = r_{max} - r(i,t)$$

kde *s(i, t)* je standardizovaná zdatnost jedince *i* v čase *t*, r_{max} je maximální dosažitelná čistou zdatnost a *r(i, t)* je čistá zdatnost. Například v problému klasifikace většiny je *s(i, t)* = 1000 – *r(i, t)* při použití 1000 počátečních konfigurací.

Přizpůsobená zdatnost (*adjusted fitness*) je vyjádřena vzorcem

$$a(i,t) = \frac{1}{1 + s(i,t)}$$

a používá se při výpočtu normalizované zdatnosti.

Normalizovaná zdatnost (*normalized fitness*) je definovaná jako

$$n(i,t) = \frac{a(i,t)}{\sum_{k=1}^{M} a(k,t)}$$

kde *M* je velikost populace. Výsledná hodnota leží v intervalu $\langle 0, 1 \rangle$. Zdatnější jedinci mají tuto hodnotu vyšší než méně zdatní. Součet všech normalizovaných zdatností v populaci je vždy roven 1. Tento způsob měření zdatnosti využívají některé selekční mechanismy.

2.2.4 Selekční mechanismy

V genetickém programování jsou upřednostňovány programy, které mají vyšší zdatnost při řešení problému, před těmi, jejichž zdatnost je nižší. Přesto je pro zachování různorodosti v populaci důležité, aby se i málo zdatné programy dokázaly v omezené míře dostat do další generace. S tímto problémem se různými způsoby vypořádávají všechny selekční mechanismy.

Efektivním způsobem výběru, který zároveň upřednostňuje zdatné jedince a umožňuje postup do další generace i méně zdatným, je **turnajová selekce**. Pro výběr jednoho jedince je z populace náhodně vybrán předem určený počet jedinců. Porovná se jejich zdatnost (čistá či standardizovaná) a vybrán je ten jedinec, jehož zdatnost je nejvyšší (nebo v případě standardizované zdatnosti nejnižší).

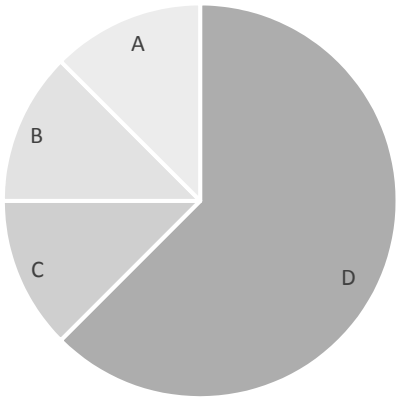
Při použití **výběru úměrného zdatnosti** má každý jedinec pravděpodobnost rovnou jeho normalizované zdatnosti (viz kapitola 2.2.3 *Měření zdatnosti*), že bude zvolen pro genetickou operaci a postup do další generace. Normalizovaná zdatnost hodnotí zdatné jedince lépe než méně zdatné a pohybuje se v rozmezí $\langle 0, 1 \rangle$.

Při **výběru podle pořadí** se využívá čistá zdatnost. Všichni jedinci v populaci se podle zdatnosti seřadí a očíslují od 1 do *n* (kde *n* je velikost populace) tak, že nejzdat-

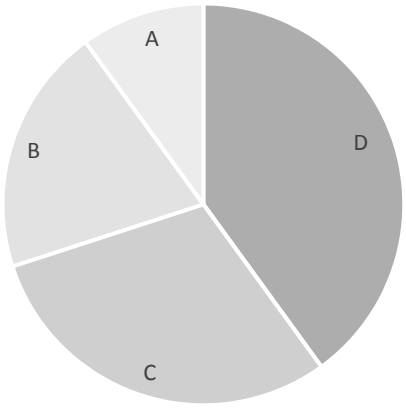
nějšímu jedinci je přiřazeno číslo *n* a nejméně zdatnému číslo 1. Poté je vytvořena nová množina jedinců, z nichž bude proveden výběr, a do této množiny je každý jedinec přidán tolikrát, jaké bylo jeho pořadí (jeho přiřazené číslo od 1 do *n*).

V porovnání s výběrem úměrným zdatnosti dochází při výběru podle pořadí k vyrovnávání selekčního tlaku v případech, kdy se v populaci vyskytují jedinci s výrazně vyšší zdatností, než jakou má většina ostatních jedinců. Ve výběru podle pořadí se šance zvolení rozloží rovnoměrněji. Z důvodu rovnoměrnějšího rozložení šancí pro výběr i v případě výskytu vysoce zdatného jedince v populaci slabých jedinců může tento selekční mechanismus zabránit předčasné konvergenci populace. Naopak v případech, kdy mají všichni jedinci přibližně stejnou hodnotu zdatnosti, přiřazuje tato metoda vyšší pravděpodobnost výběru těm jedincům, jejichž zdatnost je jen nepatrně vyšší oproti ostatním.

Obě metody výběru porovnávají grafy 1 a 2, které znázorňují případ populace čtyř jedinců (označených jako *A, B, C, D*), z nichž jedinec *D* má čistou zdatnost 5 a ostatní 1. Jak je patrné z grafu 1, který znázorňuje pravděpodobnost výběru jedinců na základě jejich čisté zdatnosti, výběr úměrný zdatnosti vytváří velkou pravděpodobnost výběru zdatných jedinců na úkor méně zdatných. Při použití výběru podle pořadí dojde ke snížení selekčního tlaku zdatných jedinců ve prospěch méně zdatných.



Graf 1: Výběr úměrný zdatnosti

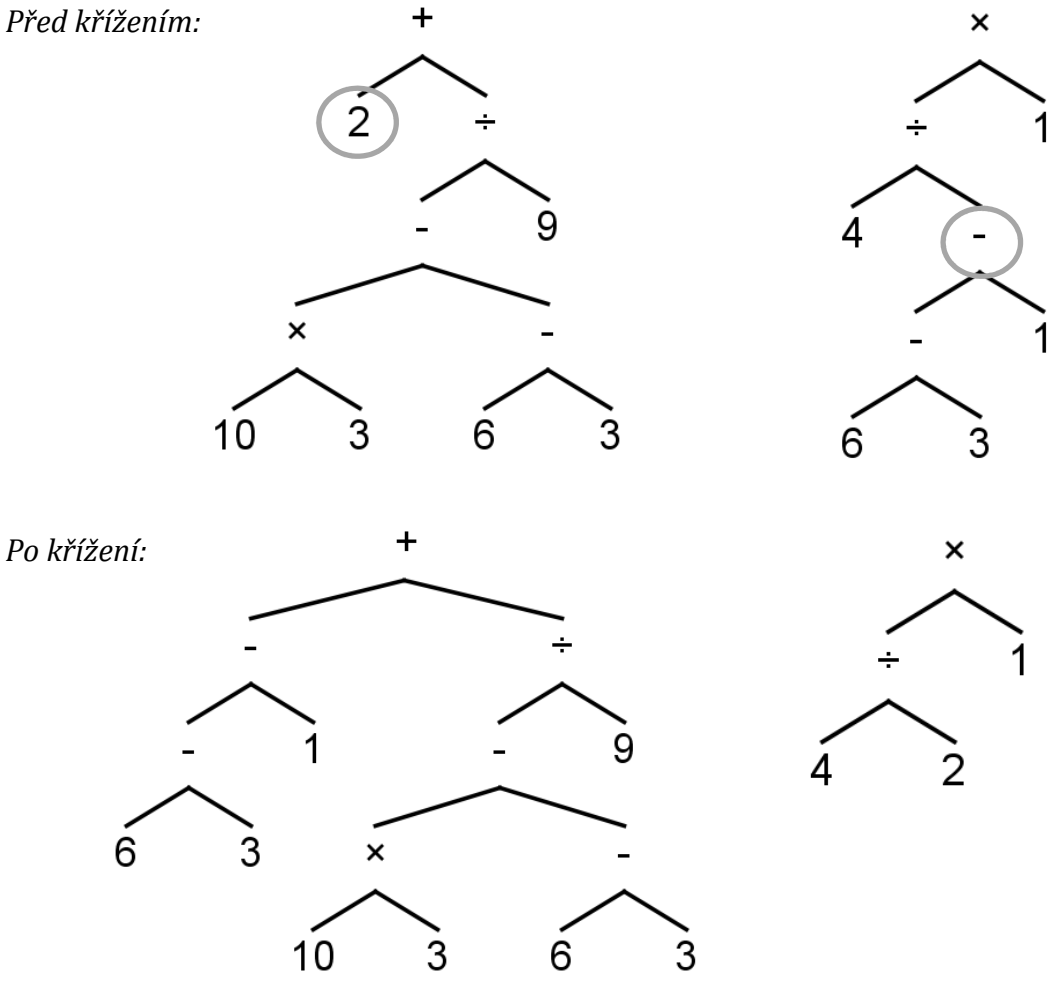


Graf 2: Výběr podle pořadí

2.2.5 Primární genetické operace

Každá nová generace vzniká z předchozí pomocí výběru jedinců zvoleným selekčním mechanismem a jejich úpravy pomocí některých z dále uvedených genetických operací. Nejvýznamnější operací je *křížení*, ale v menší míře bývají uplatňovány i reprodukce, mutace a další operace.

Křížení je nejpřínosnější metodou pro vytvoření rozmanitosti v populaci, proto je obvykle používáno k vytvoření největší části jedinců v nové generaci. Při této metodě se selekčním mechanismem vybírají dva jedinci, na nichž je potom náhodně vybrán jeden uzel. Tento uzel může být jak funkcí, tak i terminálem; je-li funkcí, pracuje se dále s celým podstromem, který daným uzlem začíná. Následně si oba jedinci mezi sebou tento uzel (včetně případného podstromu) vymění a tím vzniknou noví dva odlišní jedinci, kteří pokračují do příští generace (viz obrázek 3).

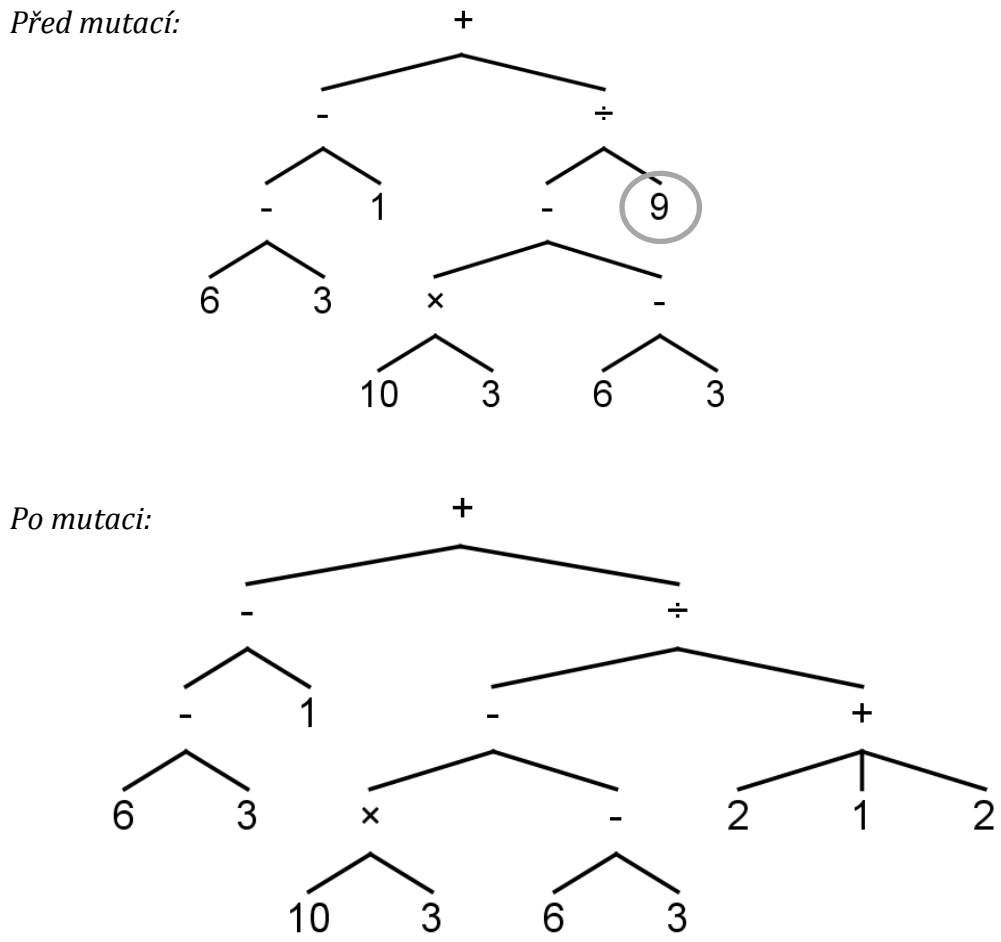


Obrázek 3: Křížení. Na každém rodičovském jedinci je vybrán uzel pro křížení, který se spolu se všemi jeho poduzly vloží na místo vybraného uzlu na druhém stromu.

Křížení, v němž jsou všechny uzly vybírány se shodnou pravděpodobností, může vést k pouhé záměně koncových listů. Tato záměna nepřináší dostatečnou změnu struktury algoritmů, neboť ta zůstává zachována a mění se pouze terminály. Tento problém lze řešit omezením, aby se v určeném množství případů křížení na jedincích volily pouze vnitřní uzly, obsahující funkce [9].

Reprodukce je označení pro přesunutí jedince do nové generace v nezměněné podobě. Tato operace neslouží k vytvoření nových struktur a nepřináší do populace žádné zlepšení, ale umožňuje zachovat nejzdatnější jedince. S reprodukcí souvisí **elitismus**, který zaručuje, že určitý počet nejzdatnějších jedinců v populaci bude vždy beze změn zkopírován do následující generace. V případě implementace podle algoritmu uvedeného na obrázku 1 jsou reprodukce a elitismus dvě odlišné genetické operace, ale v implementacích neprovádějících reprodukci mohou tyto pojmy splývat (elitismus pak nemusí automaticky kopírovat do dalších generací *nejzdatnější* jedince, ale jedince vybrané některou selekční metodou).

Mutace slouží k náhodné změně uzlů jedince. Může být použita jak jako primární genetický operátor, pomocí kterého bude tvořena část jedinců v nové generaci, tak jako sekundární operace, která bude působit s určitou (obvykle velmi nízkou) pravděpodobností na všechny uzly každého jedince v nové generaci, vytvořené pomocí ostatních genetických operátorů, jakými jsou například křížení a reprodukce. Pomocí mutace lze nahradit mutovaný uzel prvkem z množiny terminálů nebo z množiny funkcí; v případě množiny funkcí je potom potřeba vytvořit znovu její podlisty (argumenty funkce) a rekurzivně i jejich podlisty. Pokud tudíž dojde k výběru z množiny funkcí, vytváří mutace na místě uzlu novou větev stromu stejným způsobem jako při vytváření náhodných jedinců pro inicializační populaci. Hloubku vytvářených větví stromu je vhodné omezit (například parametrem *maximální inicializační hloubka*), aby nemohlo dojít k nekonečnému zanořování funkcí. Mutaci znázorňuje obrázek 4. Mutace na vybraném uzlu nemusí probíhat pouze náhodným vygenerováním nové větve, ale může být aplikována řízeně vzhledem ke konkrétní podobě uzlu a jeho poduzlů [10]. John Koza v [6] mutaci označuje jako sekundární operaci, nikoli za primární genetickou operaci pro tvorbu nových generací. Příklady dalších dodatečných genetických operací jsou uvedeny v následující kapitole.



Obrázek 4: Tvoření jedinců prostřednictvím mutace probíhá náhodným výběrem uzlu stromu a vygenerováním náhodné nové větve v tomto místě.

2.2.6 Sekundární operace

Mezi sekundární operace bývá někdy zařazována mutace, a to z důvodu nízkého procentuálního využití v porovnání s operací křížení. Mutace se v genetickém programování používá jen ve velmi omezené míře na rozdíl od evolučního programování, které naopak vytváří nové generace právě pomocí mutace a vůbec nepracuje se křížením [3].

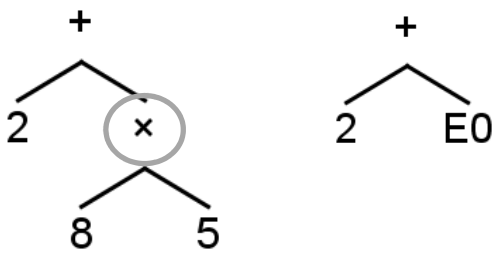
John Koza v [6] zmiňuje kromě mutace čtyři další možné sekundární operace - permutaci, enkapsulaci, decimaci a editaci.

Permutace je zobecněním inverzní operace uplatňované v genetických algoritmech, při které dochází k přeřazení znaků ležících mezi dvěma zvolenými body v řetězci, a to obrácením jejich pořadí. Inverze navzájem přibližuje a oddaluje konkrétní alely. Při použití inverzní operace na jedincích s relativně vysokou zdatností může tato operace pomoci k vytvoření blízkých genetických vazeb mezi alelami, které přispívají k vyšší zdatnosti jedince. Tyto soustavy alel budou v budoucích generací prav-

děpodobně zachovány, protože budou méně podléhat nebezpečí narušení operací křížení. V oblasti genetických algoritmů mají alely význam určený jejich konkrétní pozicí v řetězci. Z toho důvodu musejí být při uplatnění inverzní operace alely označeny tak, aby při dekódování řetězce mohl být alelám přiřazen zamýšlený význam. *Permutace* je zobecněním této operace a je aplikovatelná na struktury vytvářené v genetickém programování (viz obrázek 5). Pracuje se stromovou strukturou pouze jednoho rodičovského jedince. Operace je zahájena náhodným zvolením jedné z funkcí jedince. Jestliže zvolená funkce přebírá k argumentů, pak je permutace zvolena náhodně z $k!$ možných permutací. Poté jsou argumenty funkce náhodně permutovány. V případě komutativní funkce nemá tato operace vliv na výsledek. Rozdíl oproti inverzní operaci v genetických algoritmech spočívá v možnosti výskytu jakékoli z $k!$ možných permutací, zatímco inverzní operace způsobuje výskyt pouze jedné z těchto $k!$ permutací, a to permutaci obrácením pořadí [6].

Obrázek 5: Permutace. Při změně pořadí pomocí permutace dojde ke změně výsledné hodnoty.

Enkapsulace spočívá v automatické identifikaci užitečného podstromu, jeho pojmenování a následnému použití v jiných stromech. Tato operace je často velmi užitečná; klíčovou otázkou v oblasti umělé inteligence a strojového učení je, jak rozšířit techniky úspěšné při řešení dílčích problémů na řešení problémů větších, přičemž jednou z možností je dekomponovat problém na hierarchii menších. Důležitý zde je *automatizovaný* rozklad na dílčí problémy, který je jedním z hlavních cílů umělé inteligence a strojového učení. Enkapsulaci ilustruje obrázek 6.



Obrázek 6: Enkapsulace. Zvolený uzel se stává definicí nové funkce a je nahrazen jejím názvem – E0.

Enkapsulace pracuje vždy na jednom stromovém grafu, na němž náhodně zvolí jeden z uzlů. Podstrom začínající tímto vybraným uzlem se pak stane definicí nového podstromu. Enkapsulace ze stromu odstraní tento podstrom a definuje novou funkci,

umožňující odkazování na tento podstrom. Tato funkce nepřijímá žádné argumenty (mohla by se tudíž označovat také jako *terminál*). Její tělo tvoří podstrom, který byl vybrán z původního stromu. Funkce bývají pojmenovány *E0, E1,* ... , *En* s číslováním podle pořadí vzniku. Takto definovanou funkci pak enkapsulace vloží na místo původního podstromu, chování původního jedince tak zůstane beze změny. Dojde také k rozšíření množiny funkcí o tuto novou funkci, aby mohla do nové generace přejít nejen prostřednictvím jedince, na němž byla enkapsulace provedena, ale také prostřednictvím tvorby nových podstromů pomocí operace mutace zmíněné výše. Výhodou enkapsulace je ochrana podstromu před možným narušením jeho struktury při křížení nebo mutaci. Tato technika připomíná princip ADF, ale existují zde významné rozdíly (viz kapitola 2.2.7 *Automaticky definované funkce*) [6].

Decimace se používá, neboť pro některé problémy mohou být hodnoty zdatnosti v inicializační populaci zkresleny tak, že se v ní nachází velké procento jedinců velmi málo zdatných, což může být způsobeno penalizací jedinců na základě jejich časové náročnosti (například v problémech časově optimálního řízení nebo v problémech zahrnujících iterativní cykly). V takových případech je k výpočtu těchto málo zdatných jedinců zapotřebí velkého množství výpočetního času. Další nevýhodou takové populace je, že několik jedinců se zdatností jen nepatrně vyšší začne v populaci ihned dominovat a rozmanitost populace se začne rychle snižovat. V genetickém programování je operace křížení schopná populační různorodost rychle navracet zpět, ale protože je výběr rodičovských jedinců založen na zdatnosti, soustředí se operace křížení pouze na několik málo jedinců v populaci s mírně vyššími hodnotami zdatnosti. *Decimace* pomáhá tento problém řešit. Řídí se dvěma parametry: procentem populace a podmínkou, která určuje, kdy bude operace vyvolána. Obvykle se zde stanovuje pořadové číslo generace. Například parametry *10 %* a *10. generace* znamenají, že po vytvoření *10. generace* a ohodnocení jejích jedinců dojde k vymazání všech jedinců kromě 10 %. V případě užití decimace je velikost inicializační populace zvýšena oproti velikosti určené příslušným parametrem genetického algoritmu. V uvedeném příkladu tedy dojde k vytvoření desetinásobku zadaného počtu jedinců, ale ještě před vytvořením jedenácté generace je tento počet opět snížen na původně stanovené množství. Výběr jedinců při této operaci je založený na jejich zdatnosti. Opakovaný výběr není umožněn, aby se mezi zbývajícími jedinci zachovala diverzita [6].

Editace umožňuje v průběhu genetického programování úpravu a zjednodušování konkrétních podstromů obsažených ve vytvářených stromech. Tato operace na každém jedinci uplatňuje *editační pravidla*. Ta se dělí na pravidla nezávislá na doméně a na pravidla doménově specifická. Doménově nezávislým pravidlem je například případ, kdy je funkce, která nemá žádné vedlejší efekty, není kontextově závislá a přebírá jako argumenty pouze konstanty, celá nahrazena výsledkem jejího výpočtu. Například funkce *+(2, 1)* bude nahrazena hodnotou *3* (viz obrázek 7).

Dále může editace nahrazovat podstromy v programu podle doménově specific-

Obrázek 7: Editace. Strom funkce +(2,1) byl nahrazen hodnotou 3.

kých pravidel, která jsou předem stanovena uživatelem před spuštěním genetického algoritmu. Rekurzivní aplikace editačních pravidel však způsobuje vysokou časovou náročnost. V genetických algoritmech se nepoužívá editace ani žádný její ekvivalent, protože zde je struktura jedinců již optimálně zakódována a má jednotnou strukturální složitost. Editace se využívá v genetickém programování ze dvou různých důvodů. Prvním je úprava stromu do zjednodušené a tudíž i pro uživatele lépe čitelné podoby. V takovém případě stačí editaci aplikovat jen na programy, jejichž výstup je reprezentován uživateli. Druhým důvodem pro použití editace je zjednodušení výstupu funkcí beze ztráty dosažených výsledků a vylepšení celkového výkonu genetického programování. Pak je editace použita za běhu genetického algoritmu a je aplikována na každém jedinci v populaci. Editace je řízena parametrem *frekvence*, který určuje, zda bude operace aplikována na každou generaci, nebo bude k editaci docházet pouze s určitou frekvencí. Editace může zvyšovat výkon genetického programování tím, že redukuje zranitelnost složených podstromů vůči rozkladu způsobeném křížením, ale zároveň může tento výkon naopak snižovat tím, že předčasně redukuje rozmanitost struktur [6].

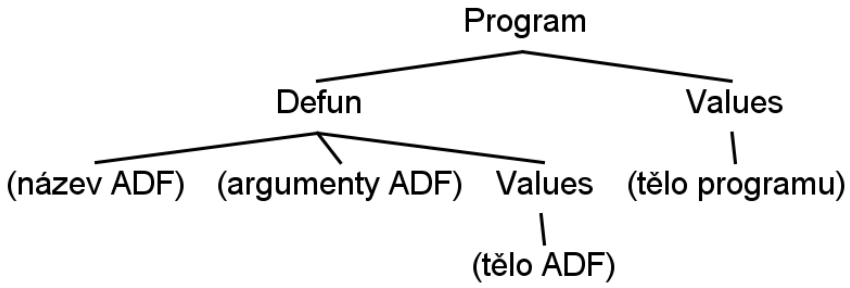
2.2.7 Automaticky definované funkce

John Koza v [6] přišel s konceptem automaticky definovaných funkcí (ADF). Ten vychází z poznatku, že je užitečné definovat funkce tak, aby bylo možné provádět konkrétní výpočet s použitím různých kombinací argumentů. Takto definované funk-

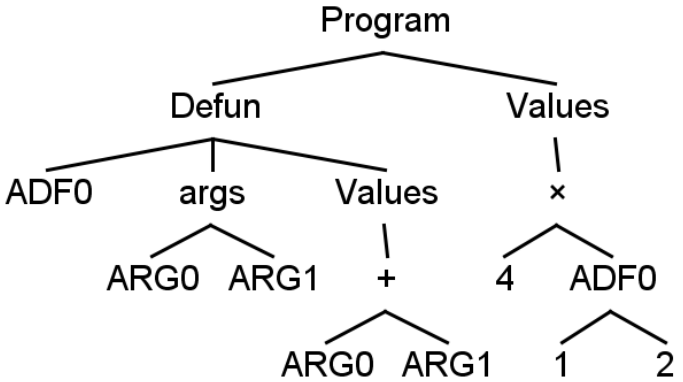
ce také zlepšují srozumitelnost a čitelnost programů, protože zvýrazňují často používané výpočty. Důležitější však je, že používání definovaných funkcí dekomponuje problém do hierarchie, v níž jsou tyto funkce jednotlivými částmi. Spolu se vzrůstající velikostí a složitostí problému se stávají definované funkce stále důležitějším nástrojem.

Automaticky definované funkce představují doplňkovou metodu v genetickém programování, pomocí které lze snižovat výpočetní složitost problému pomocí sestavení vhodných stavebních bloků pro generované programy. Tyto bloky budou součástí množiny funkcí a tak jako ostatní funkce v této množině budou i ony jako argumenty přijímat libovolné podstromy programu. Vyznačovat se však budou tím, že nebudou do této množiny zadány programátorem před spuštěním genetického algoritmu, ale budou vytvořeny automaticky v jeho průběhu [6].

Použití automaticky definovaných funkcí vyžaduje úpravu způsobu, jakým jsou jednotlivé programy v populaci zaznamenávány, viz obrázky 8 a 9.



Obrázek 8: Schéma každého programu v populaci při použití jedné automaticky definované funkce (převzato a upraveno z [11])



Obrázek 9: Triviální příklad použití automaticky definovaných funkcí. Zde použitá ADF sčítá převzaté argumenty. Výsledný program vrací hodnotu 12.

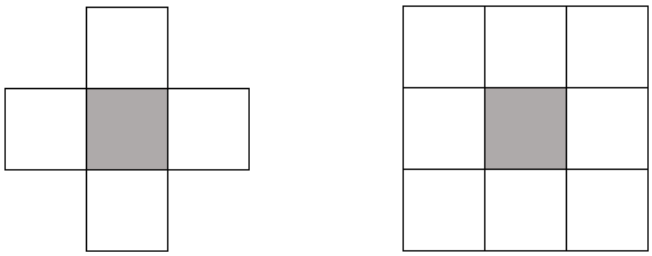
Pro každou automaticky definovanou funkci, která je zadána v množině funkcí, následuje pod kořenem stromu programu samostatná větev, která obsahuje jako uzly název dané funkce, seznam jejích argumentů a vlastní tělo funkce (které může obsahovat mnoho dalších uzlů vnořených na různých úrovních). Tělo hlavního programu popisuje uzel (a vnořené uzly z něho vycházející), který se nachází na stejné úrovni jako uzly automaticky definovaných funkcí. Tělo hlavního programu může obsahovat všechny prvky z množin terminálů a funkcí (včetně ADF). Pokud je vložena ADF, její formální parametry jsou nahrazeny hodnotami argumentů (předaných v podobě uzlů zanořených pod uzel ADF). Použití formálních parametrů je jednou z hlavních předností automaticky definovaných funkcí. Vývoj nových generací programů s ADF je podobný vývoji klasických programů v genetickém programování, ovšem s upravenými genetickými operátory. Například operace křížení musí být modifikována tak, aby v případě výběru uzlu pro křížení v oblasti definice ADF vybírala na druhém jedinci opět uzel z definice ADF a nikoli z těla hlavního programu, protože tím by došlo k umístění uzlů s formálními parametry do oblasti, která má obsahovat pouze hodnoty terminálů, a naopak, hodnoty terminálů by se objevily mezi formálními parametry v oblasti definice automaticky definované funkce [11].

Potenciálním problémem automaticky definovaných funkcí, který je popsán v [12], je náhodný výběr uzlů. Přestože výběr programů, které se účastní reprodukčních operací, je založen na jejich zdatnosti, u uzlů tomu tak není a jsou vybírány náhodně. To může způsobit ztrátu užitečných změn, ke kterým došlo v průběhu vývoje, nebo neefektivnost z důvodu výběru uzlů, které neovlivňují konečný výsledek. Z těchto důvodů byla v [12] navržena technika ARL (*Adaptive Representation Learning*), která zakládá výběr uzlů při vytváření a modifikaci automaticky definovaných funkcí na informovaném nebo adaptivním způsobu hodnocení uzlů. Algoritmus ARL se snaží automaticky vyhledávat užitečné subrutiny a adaptovat je pomocí heuristického pravidla, které říká, že užitečný kód může být zobecněn a úspěšně aplikován v obecnějším kontextu. Algoritmus byl úspěšně implementován a výsledky porovnány s výsledky standardního genetického programování i GP s automaticky definovanými funkcemi. ARL dosahoval obvykle (v závislosti na typu řešené úlohy) významně vyšších zdatností a to již v raných generacích.

3 Celulární automaty

Celulární automat je diskrétní matematický model, používaný ke studiu systémové samoorganizace. Představuje matematickou idealizaci fyzikálních systémů s diskrétním prostorem a časem a s fyzikálními veličinami sestávajícími z konečného počtu diskrétních hodnot. Skládá se z pravidelné *mřížky* (nebo, v případě jednorozměrného automatu, z *pole*), v níž se na každé pozici (v každé *buňce*) nachází právě jedna diskrétní hodnota. *Stav* celulárního automatu je jednoznačně určen hodnotami proměnných v těchto buňkách.

Celulární automat se z počátečního stavu dále vyvíjí v diskrétních časových krocích, přičemž hodnota proměnné v buňce je ovlivňována hodnotami buněk v jejím *okolí* (nebo též *sousedství*) z předcházejícího kroku. Obvykle se za okolí považuje buňka samotná a její přímo sousedící buňky. V případě, že se okolí buňky nachází za hranicemi mřížky, se za něj obvykle považují buňky z protějšího konce mřížky. Definice okolí závisí na dimenzi celulárních automatů. Například v případě dvourozměrných celulárních automatů se typicky používá *von Neumannovo sousedství* nebo *Moorovo sousedství* (viz obrázek 10). Tato sousedství obsahují buňky přímo sousedící zleva, zprava, shora a zdola a v případě Moorova sousedství také všechny buňky v rozích. Moorovo sousedství je využito např. ve Hře života (viz kapitola 3.2) [13].



Obrázek 10: Vlevo von Neumannovo sousedství, vpravo Moorovo sousedství. Aktualizována bude šedě zvýrazněná buňka.

Hodnoty proměnných se mění synchronně v závislosti na stavu okolí v předešlém časovém kroku podle *lokálních pravidel*. Pravidla jsou sadou všech dostupných stavů okolí a hodnot, kterých buňka v příštím kroku nabude v případě konkrétního stavu okolí [14]. Pravidla uvažovaná v rámci této práce jsou deterministická, ale mohou být rovněž i stochastická [15].

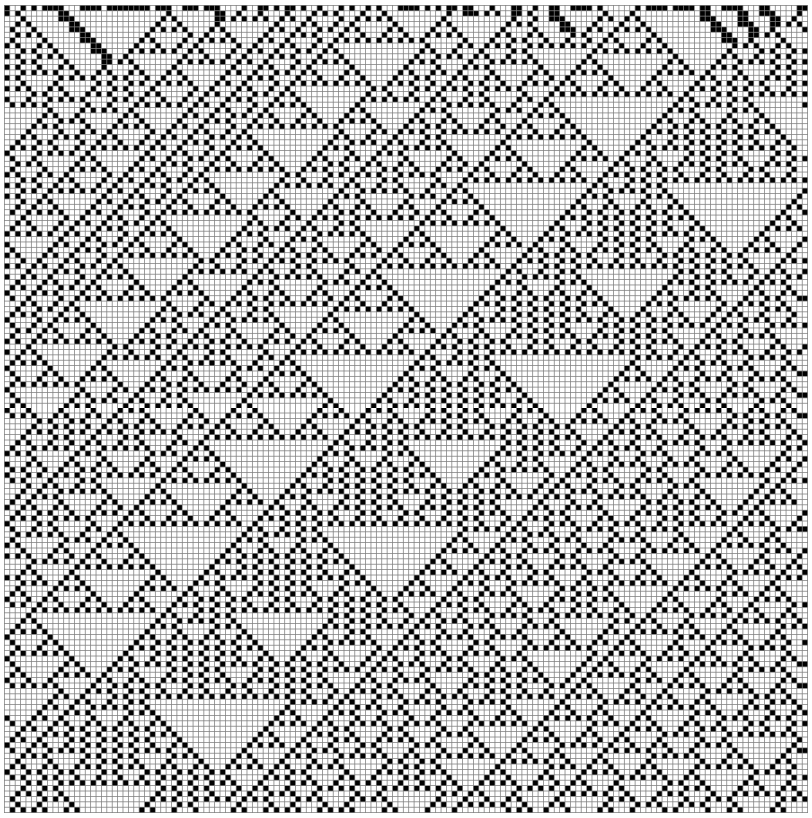
Koncept celulárních automatů poprvé představili ve 40. letech 20. století Stanislaw Ulam a John von Neumann. Koncept, pojatý jako idealizaci biologických systémů, pojmenovali *celulární prostory* a vytvořili jej za konkrétním účelem – pro modelování biologické reprodukce. Poté byl mnohokrát aplikován znovu pro řešení dalších

problémů a byl označován například jako *teselační automaty, homogenní struktury, celulární struktury, teselační struktury* nebo *iterativní pole* [14]. Mimo oblast akademického zájmu se rozšířil až v 70. letech díky Conwayově Hře života (viz kapitola 3.2). Stephen Wolfram se v 80. letech zabýval studií elementárních celulárních automatů - jednorozměrných celulárních automatů se dvěma přípustnými hodnotami v proměnné každé buňky (tzn. se základem 2) a s okolím definovaným jako aktuální buňka a libovolný počet buněk s ní přímo sousedících zleva a zprava. V roce 2002 ve své práci *A New Kind of Science* ukázal, že lze celulární automaty využít pro řešení problémů z mnoha oblastí vědy, například z problematiky počítačových procesorů nebo šifrování [14]. Celulární automaty představují často využívané paradigma pro modelování nejen proto, že umožňují postupnou specifikaci zákonů řídících dynamiku daného systému, ale také díky skutečnosti, že jde o distribuované nástroje podléhající výpočetní paralelizaci [16].

Celulární automaty mají následující základní vlastnosti [15]:

- všechny buňky jsou aktualizovány podle stejné sady pravidel,
- všechny buňky jsou aktualizovány současně,
- pravidla mají lokální charakter.

Elementární celulární automat, který je využit v rámci této práce při řešení problému klasifikace většiny, se skládá z řady polí (buněk), kde každé pole představuje právě jeden celulární automat. Stav automatu, tedy aktuální hodnota proměnné v buňce, je určen jeho barvou, a to buď černou, nebo bílou. Automaty mají definované okolí, tj. počet přímých sousedů vpravo a vlevo, jejichž stav předurčuje, jak se změní stav prostředního automatu v příštím kroku. Odtud vyplývá jedna z důležitých vlastností tohoto systému, a tou jsou omezené informace každého automatu. V každém kroku je podle lokálních pravidel vyhodnocen následující stav automatu a celá řada je zobrazena pod poslední řadou. Díky tomuto postupu lze vidět v globálním chování vzory, jejichž ukázku zachycuje obrázek 11 a které budou podrobněji popsány v kapitole 3.1 *Vzorce chování*. [17]



Obrázek 11: Ukázka globálního chování celulárního automatu.

Každé pravidlo lze zapsat pomocí tabulky (viz obrázek 12), která přiřazuje každé možné kombinaci stavů právě jednu výslednou hodnotu proměnné v aktuální (středové) buňce. Na každém řádku tabulky udává posloupnost buněk vlevo možnou podobu okolí včetně středové buňky, která po aplikaci pravidla změní barvu tak, jak je v tabulce uvedeno vpravo. Například pravidlo na obrázku 12 mění středovou buňku na bílou kromě případů, kdy je celé okolí černé nebo bílé; pak je středová buňka obarvena černě. Tento způsob zápisu bude používán pro popsání všech dále uváděných pravidel.

Pro dvoustavové automaty s okolím o velikosti 3 existuje 2^3 kombinací stavů tří vedle sebe stojících polí (jak je patrné z obrázku 12) a lze pro ně najít celkem 2^8 různých pravidel. Tato pravidla byla očíslována a jsou známá pod názvy jako *pravidlo 30*, *pravidlo 90* a podobně [18].

Obrázek 12: Pravidlo CA pro okolí o velikosti 3.

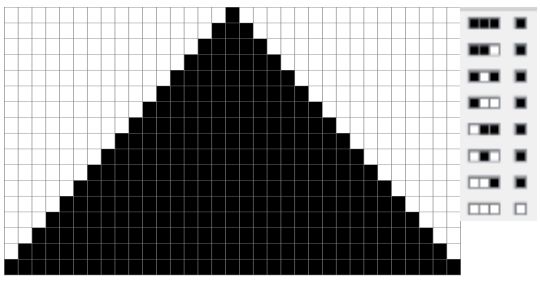
Zápisem s_i^t se rozumí stav buňky s indexem i v čase t. Stav s_i^t buňky i společně se stavy buněk k této připojených se nazývá *sousedství* η_i^t. Formálně se pravidlo celulárních automatů označuje jako *pravidlo přechodu* $\phi(\eta_i^t)$, které určuje nadcházející stav $s_i^{(t+1)}$ pro každou buňku i jako funkci s parametrem η_i^t [13].

Výše uvedené celulární automaty bývají označovány jako *uniformní* celulární automaty, protože všechny buňky se řídí stejným pravidlem přechodu. Existuje však i speciální případ celulárních automatů, kdy je každá buňka řízena odlišným pravidlem; pak se tyto automaty označují jako *neuniformní* [19].

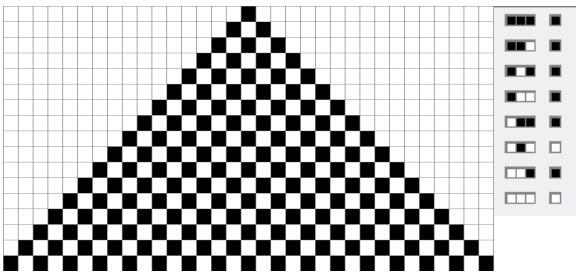
3.1 Vzorce chování

V závislosti na počáteční konfiguraci a na použitém pravidle můžeme vidět v globálním chování automatů určité vzorce. Tato kapitola blíže představí druhy těchto vzorců. Ve všech případech se vychází z počáteční konfigurace jednoho černého bodu uprostřed řady bílých polí a z okolí o velikosti 3.

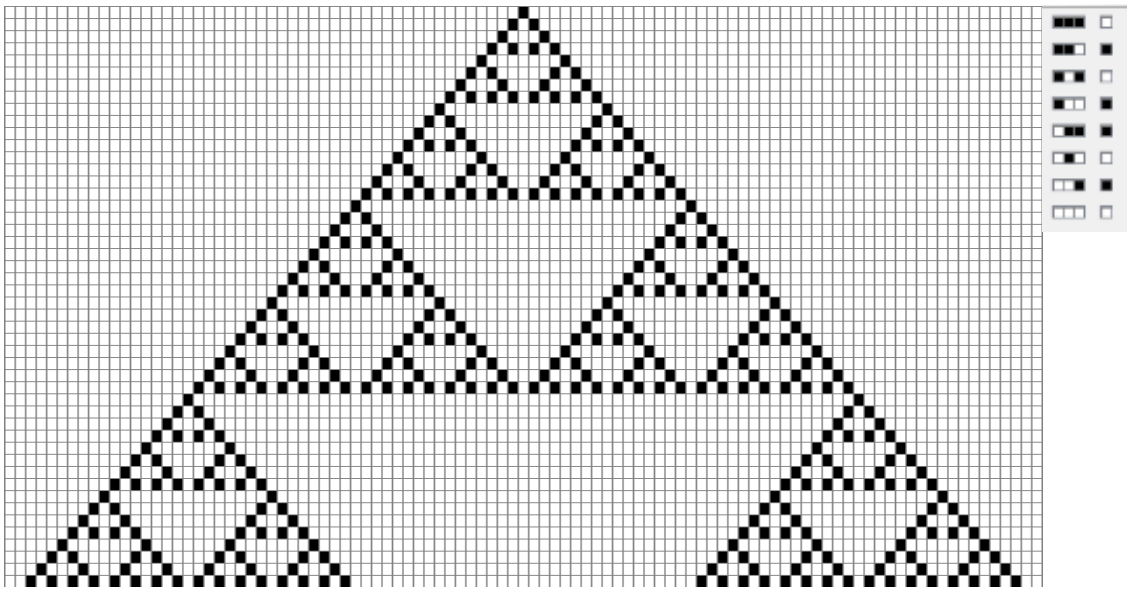
Obrázek 13 znázorňuje vznik jednoduchého rozšiřujícího se vzoru, jednotně vyplněného černou barvou. Ovšem ke změně tohoto chování stačí již malá změna v pravidle, viz obrázek 14, kde vzniká šachovnicový vzor. Přestože odsud vidíme, jak velký vliv na konečné chování může mít jediný černý bod v počáteční konfiguraci, tato chování jsou stále velmi jednoduchá.



Obrázek 13: Jednoduchý expandující vzor

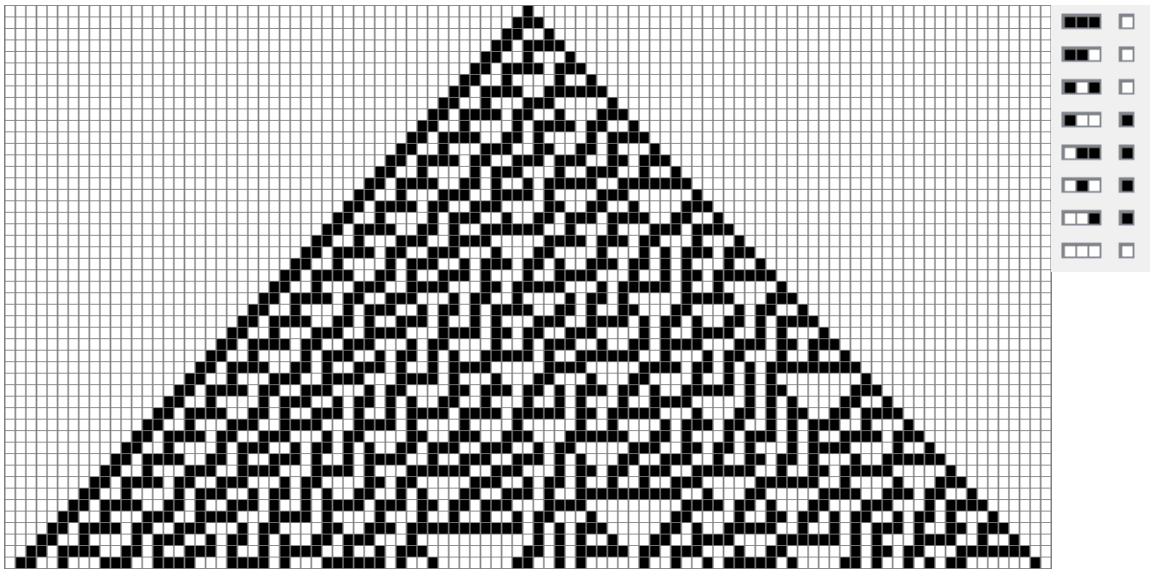


Obrázek 14: Změna vzoru z obrázku 13, dosažená změnou v pouze jediné položce pravidla (3. zdola)



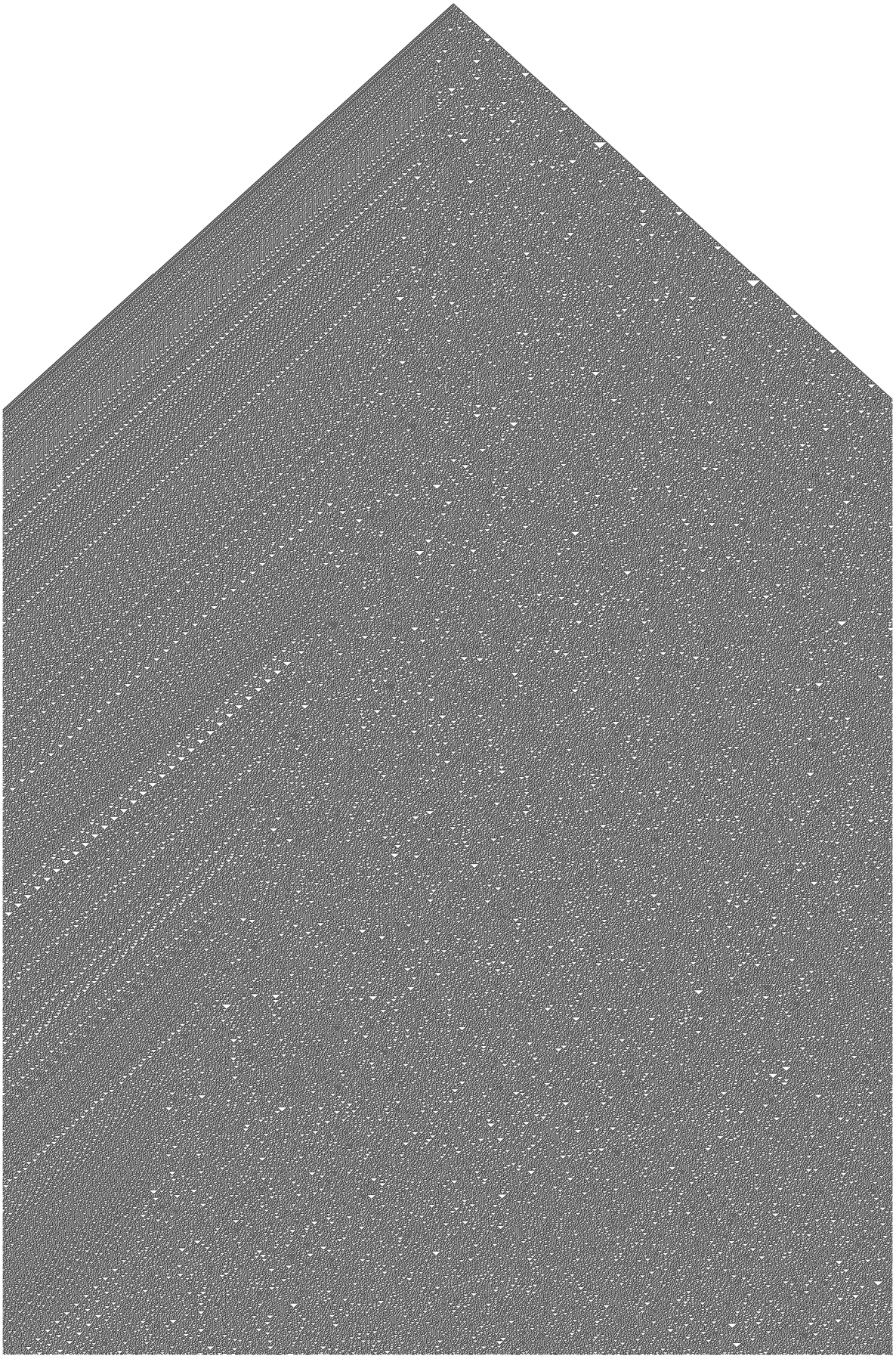
Obrázek 15: Pravidelný vzor vytvořený aplikací pravidla 90

Obrázek 15 zachycuje výsledek po použití pravidla 90. V celkovém vzoru se nachází pravidelný trojúhelníkový vzorec, který je rekurzivně opakován uvnitř každého menšího vzorce. Ovšem výsledná chování celulárních automatů vůbec nemusejí být pravidelná a předvídatelná. Důkazem velmi nepravidelného a komplexního vzoru je výsledek pravidla 30 (obrázek 16).



Obrázek 16: Komplexní vzor vygenerovaný pomocí pravidla 30

Mohlo by se zdát, že by pravidelnost mohla nastat po provedení většího počtu kroků, ale ani experimenty uvedené v [17] s milionem časových kroků nezaznamenaly pravidelné chování (viz obrázek 17 s ukázkou 2 500 kroků); přestože v některých částech obrazce jsou místy viditelné určité pravidelnosti, celkové chování je zcela nepředvídatelné a chaotické.

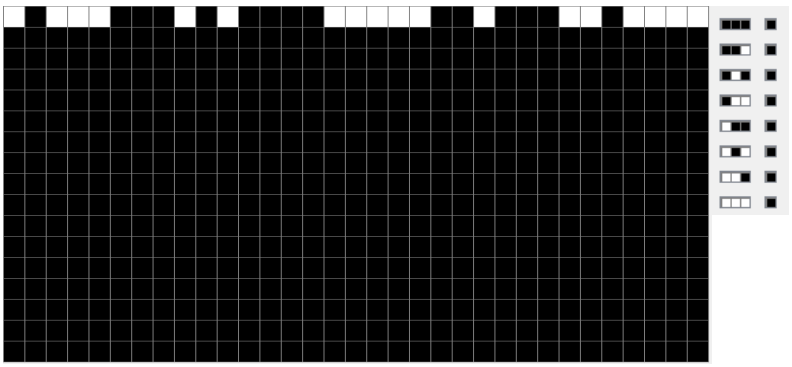


Obrázek 17: Chování pravidla 30 po 2500 časových krocích. Převzato z [1].

Vztah mezi obecným dynamickým chováním celulárních automatů a jejich výpočetními možnosti se stal předmětem mnoha vědeckých výzkumů. Je součástí obsáhlejší otázky vztahů mezi teorií dynamických systémů a výpočetní teorií. Stephen Wolfram pohlížel na celulární automaty jako na diskrétní dynamické systémy rozšířené o dimenzi prostoru a navrhl kvalitativní klasifikaci jejich chování analogickou s klasifikací v teorii dynamických systémů [13].

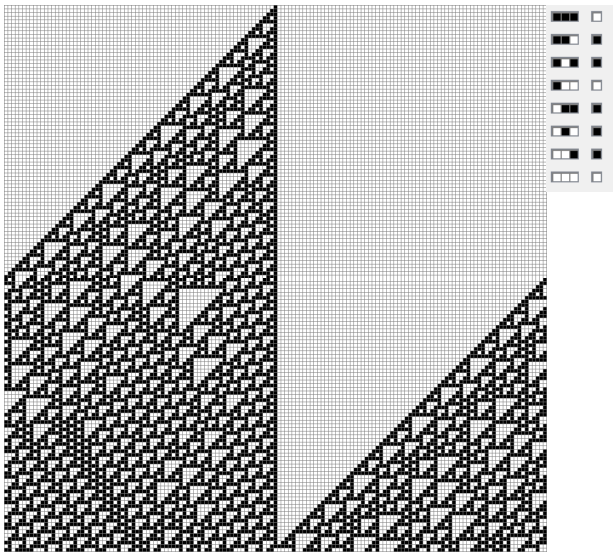
Wolfram rozděluje chování celulárních automatů do následujících čtyř tříd [13]:

- *Třída 1*: Téměř všechny počáteční konfigurace se po určitém počtu kroků ustálí na jediné konkrétní konfiguraci, například na konfiguraci samých jedniček, nebo samých nul, viz obrázek 18.



Obrázek 18: Příklad pravidla třídy 1.

- *Třída 2*: Téměř všechny počáteční konfigurace se po určitém počtu kroků ustálí v některém fixním bodě nebo v některé periodicky se opakující sérii konfigurací, ale jejich konkrétní podoba je závislá na počáteční konfiguraci. V konečných mřížkách je jen konečný počet k^Nmožných konfigurací (kde k je počet možných stavů a N je počet buněk v řadě), proto všechna pravidla vedou k periodickému chování; chování třídy 2 nevedou k takovému druhu periodického chování, ale opakují chování v cyklech s periodou kratší než k^N. Ukázka na obrázku 15 zobrazuje pravidelné chování pravidla 90.
- *Třída 3*: téměř všechny počáteční konfigurace se po určitém počtu kroků ustálí na chování, které je chaotické (nepředvídatelné), ukázkou je pravidlo 30 na obrázku 16.
- *Třída 4*: některé počáteční konfigurace ústí v komplexní lokalizované struktury, které mohou mít různě dlouhou životnost. Příkladem je pravidlo 110 na obrázku 19.

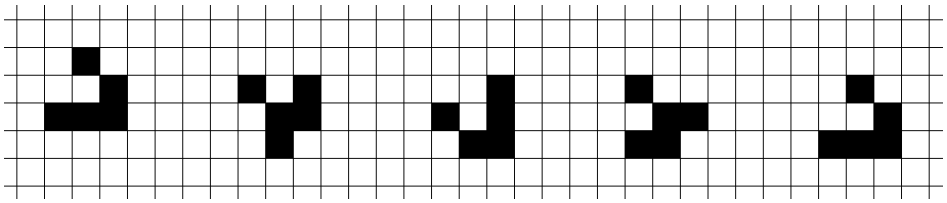


Obrázek 19: Pravidlo 110 a ukázka chování celulárních automatů třídy 4.

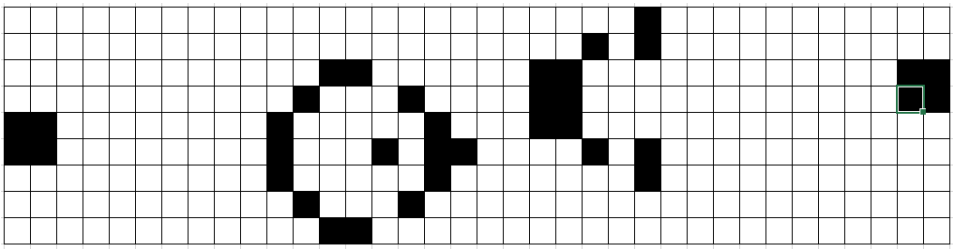
3.2 Hra života

Celulární automat označovaný jako Hra života navrhl John Conway v 60. letech 20. století. Jedná se o dvourozměrný automat se dvěma stavy: stav 1 odpovídá černé buňce (označované jako *živé*), stav 0 odpovídá bílé (*neživé*) buňce. Používá se zde Moorovo sousedství (obsahující 8 okolních buněk přímo sousedících s výchozí, středovou buňkou, viz výše obrázek 10) a následujícím pravidlem: pokud je $s_i^t = 1$, pak je $\phi(\eta_i^t) = 1$ právě tehdy, když jsou 2 nebo 3 buňky v sousedství také ve stavu 1; pokud ne, je $\phi(\eta_i^t) = 0$. Pokud je naopak $s_i^t = 0$, pak je $\phi(\eta_i^t) = 1$ právě tehdy, když se právě 3 buňky v sousedství nacházejí ve stavu 1; jinak je stále $\phi(\eta_i^t) = 0$. Toto pravidlo přechodu často vede k velmi komplikovaným a zajímavým vzorům v prostoru buněk. Některé vznikající struktury byly pojmenovány, například *kluzáky* (*glidery*), viz obrázek 20, jsou propagující lokalizované struktury, z nichž lze utvářet komplexní vzory. Stacionární struktura označovaná jako *kluzákové dělo* (*glider gun*), viz obrázek 21, tyto kluzáky produkuje, a to vždy jeden po každých 30ti časových krocích [13].

Přestože je Hra života založena na jednoduchých pravidlech, bylo prokázáno, že dokáže provádět univerzální výpočty. Klíčovými prvky při vytváření univerzálního počítače jsou kluzáky a kluzáková děla. Namísto simulace univerzálního Turingova stroje se v tomto případě sestavují základní logické funkce z interakcí mezi proudy kluzáků přicházejících z kluzákových děl. Příkladem použití mohou být případy, kdy se dva kluzáky při vzájemné kolizi navzájem vyruší a zmizí. Toto chování lze využít k sestavení logické funkce NOT: proud kluzáků představuje vstupní proud bitů, přičemž kluzáky znamenají hodnotu 1 a mezery mezi nimi znamenají hodnotu 0. Tento proud koliduje s dalším proudem vygenerovaných kluzáků, který je na něj kolmý. Protože



Obrázek 20: Kluzák v průběhu pěti iterací. V páté iteraci se kluzák vrací do původního tvaru, ale dojde k jeho posunutí o jednu buňku směrem doprava a o jednu buňku směrem dolů.



Obrázek 21: Kluzákové dělo. Dvě struktury uprostřed se pohybují k sobě a od sebe a při vzájemném střetnutí vytvoří strukturu kluzáku (viz výše).

při tomto sestavení se dva kluzáky navzájem vyruší, jedinou možností pro jejich zachování je střetnutí se s mezerou ve druhém proudu. Výsledný proud, který přitom vzniká, je výsledkem logické funkce NOT aplikované na vstupní proud. Obdobně se vytvářejí také funkce AND a OR. Dále konstrukce univerzálního počítače zahrnuje způsoby kopírování jednotlivých proudů, změnu jejich pozic, zdržení, ukládání informací v podobě cirkulujících proudů a implementace pomocného úložiště pomocí stacionárních buněčných struktur, které jsou řízeny pomocí kluzáků. Tyto techniky umožňují vytvoření obvodů z proudů kluzáků, které dokáží provádět univerzální výpočty, což znamená, že tato jednoduchá architektura dokáže v principu provádět stejné výpočty jako běžný počítač. V praxi se tyto univerzální počítače CA k výpočtům nepoužívají, protože pouhé nastavení počáteční konfigurace takové, aby bylo dosaženo požadovaných výsledků, by bylo extrémně náročné; a dále pak proto, že by tyto výpočty trvaly velmi dlouho v porovnání s běžně používanými výpočetními zařízeními [13].

3.3 Význam a aplikace celulárních automatů

Celulární automaty a obdobné systémy založené na stejných principech se v současné době využívají jak při komplexních výpočtech, u kterých je vyžadován vysoký stupeň efektivnosti a robustnosti, tak také jako nástroj pro tvorbu modelů, pomocí kterých lze například zkoumat chování přirozeně se vyskytujících systémů složených z jednoduchých komponent se schopností lokální komunikace a s emergentními kolektivními vlastnostmi [13].

Jednou z konkrétních oblastí, v níž lze využít výpočetní potenciál celulárních automatů v praxi, je kryptografie a kryptoanalýza. Například pravidlo 30 (viz kapitola 3.1 *Vzorce chování*, obrázek 16) lze využít v kryptografii při blokovém šifrování. Dalším příkladem je použití dvourozměrného celulárního automatu pro vytvoření generátoru náhodných čísel. Nejprve je pomocí techniky celulárního programování (viz níže) vyvinuta mřížka celulárních automatů, která je dále převedena na náhodné číslo následujícím způsobem: celulární automat je spuštěn celkem čtyřikrát, přičemž v každém časovém kroku produkuje každá buňka sekvenci 4 bitů, které dohromady utvářejí hexadecimální číslici. Tyto číslice jsou poté rozmístěny vedle sebe tak, aby vznikla sekvence $x \times y$ náhodných čísel, kde x a y jsou rozměry mřížky. Proces lze poté opakovat. Například mřížka 8×8 produkuje 64 hexadecimálních náhodných číslic během každých 4 časových kroků [20].

Celulární programování pracuje s neuniformními celulárními automaty, kde se pravidlo přechodu každé buňky zakóduje jako řetězec bitů (zakódování na řetězec bitů je využito i v aplikaci vytvořené v rámci této práce pro export uživatelsky zadaných pravidel). Celulární programování je založeno na podobných principech jako genetické programování. Vývoj pravidel v mřížce je řízen měřením lokální zdatnosti (které závisí na konkrétním řešeném problému). Používají se operátory známé z genetického programování, například selekce a křížení, které kříží pravidla mezi sousedícími buňkami. Cílem je vyvinout takovou mřížku pravidel pro neuniformní celulární automaty, která bude úspěšně řešit zadaný problém (například problém generování náhodných čísel uvedený výše) [20].

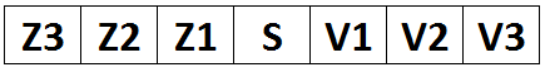
3.4 Problém klasifikace většiny

Problém klasifikace většiny je definován pro jednorozměrný dvoustavový celulární automat následovně: počáteční konfigurace je tvořena buňkami, kterým je přidělena černá nebo bílá barva tak, že obě barvy mají pravděpodobnost výskytu v buňce 50 %. Pokud je počet bílých buněk v počáteční konfiguraci větší než 1/2, celulární automat se musí po určitém konečném počtu časových kroků ustálit na vzoru tvořeném pouze bílými buňkami. Pokud je počet černých buněk v počáteční konfiguraci větší než 1/2, musí se celulární automat ustálit na vzoru tvořeném pouze černými buňkami. Pro shodný počet černých a bílých buněk není požadované chování definováno, a proto je tento výsledek předem vyloučen použitím lichého počtu buněk v počáteční konfiguraci [21].

Pokud by měl automat neomezené okolí, jednalo by se o triviální problém, ovšem právě omezení vnímatelného okolí pro každý automat je klíčovou vlastností celulár-

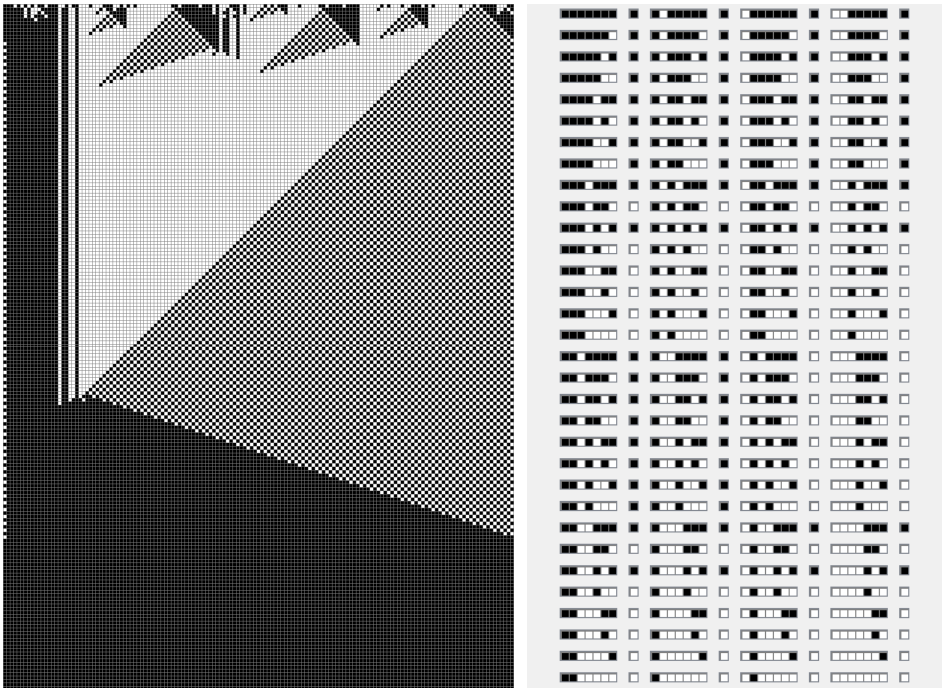
ních automatů. Nalezení pravidla, které by vedlo ke konkrétnímu globálnímu chování, by znamenalo prohledat prostor 2^{128} možných pravidel. Programování celulárních automatů je obtížné, především tehdy, pokud chceme dosáhnout takového chování, které vyžaduje globální komunikaci a globální integraci informací na dlouhou vzdálenost v celulárním prostoru [22]. Tato složitost vyplývá zejména z obtížného odhadování budoucího stavu prostoru. Přestože můžeme v některých případech částečně předvídat následující stav sledováním opakujících se vzorů (viz kapitola 3.1 *Vzorce chování*), je potřeba vzít v úvahu i rozmanitost počátečních konfigurací. Počáteční data pro tento problém jsou vždy jiná (nastavená náhodně) a protože výsledné chování automatů závisí kromě použitého pravidla na těchto počátečních datech, je potřeba každé nalezené pravidlo otestovat na co možná největším počtu náhodných počátečních konfigurací za účelem zjištění, v kolika procentech případů byl problém vyřešen správně. Přesto je toto procento pouze přibližným odhadem, neboť pro jeho přesné zjištění bychom museli pravidlo použít na všech možných počátečních stavech; máme-li 149 dvoubarevných polí v řadě, existuje 2^{149} možných konfigurací. Velké procento těchto konfigurací navíc má přibližně stejný počet černých a bílých polí [2].

Pro pojmenování jednotlivých buněk tvořících okolí o velikosti 7 je v rámci této práce využíváno jednotné označení (viz obrázek 22). Je-li buňka vyznačena bílou barvou, pak obsahuje logickou hodnotu *true* nebo numerickou hodnotu *1*. V opačném případě, je-li označena černou barvou, pak obsahuje logickou hodnotu *false* či numerickou hodnotu *0.*



Obrázek 22: Označení buněk tvořících okolí podle jejich vzdálenosti od středové buňky (S).

Dosud bylo provedeno několik pokusů o nalezení co nejúspěšnějšího řešení tohoto problému. V roce 1978 vytvořili Gacs, Kurdyumov a Levin dvoustavové pravidlo pro okolí o velikosti 7, a to při studiu spolehlivých výpočtů při náhodných odchylkách [23]. Toto pravidlo je nyní známé jako „pravidlo GKL“. Jeho zadání i aplikaci na náhodně vygenerované počáteční konfiguraci ilustruje obrázek 23.



Obrázek 23: Pravidlo GKL.

GKL pravidlo lze shrnout následovně [2]:

- pokud je $S = 0$ (tzn. hodnota buňky S je nulová), pak je nová hodnota S funkcí většiny aplikované na S, $Z1$ a $Z3$,
- pokud je $S = 1$, pak je nová hodnota S funkcí většiny aplikované na S, $V1$ a $V3$.

Funkcí většiny ve výše uvedené definici se rozumí funkce s n vstupy typu *boolean* a jedním výstupem, která vrací hodnotu *false*, pokud je $n/2$ nebo více argumentů hodnoty *false*; v opačném případě vrací hodnotu *true*.

Pravidlo GKL bylo při řešení problému klasifikace většiny úspěšné v 81,6 % případů. Jak uvádí [2], na toto pravidlo navázal Lawrence Davis v roce 1995, který jeho modifikací vytvořil pravidlo s přesností 81,8 %, a ve stejném roce také Rajarshi Das, který vytvořil pravidlo s přesností 82,178 %.

Das, Mitchell a Crutchfield v [24] použili pro hledání pravidel genetické algoritmy. Při použití této techniky byla pravidla celulárního automatu převedena na řetězce sestavené z výstupních bitů. Pravidla, vyvinutá genetickým algoritmem, nejčastěji spadala do jedné z následujících strategií:

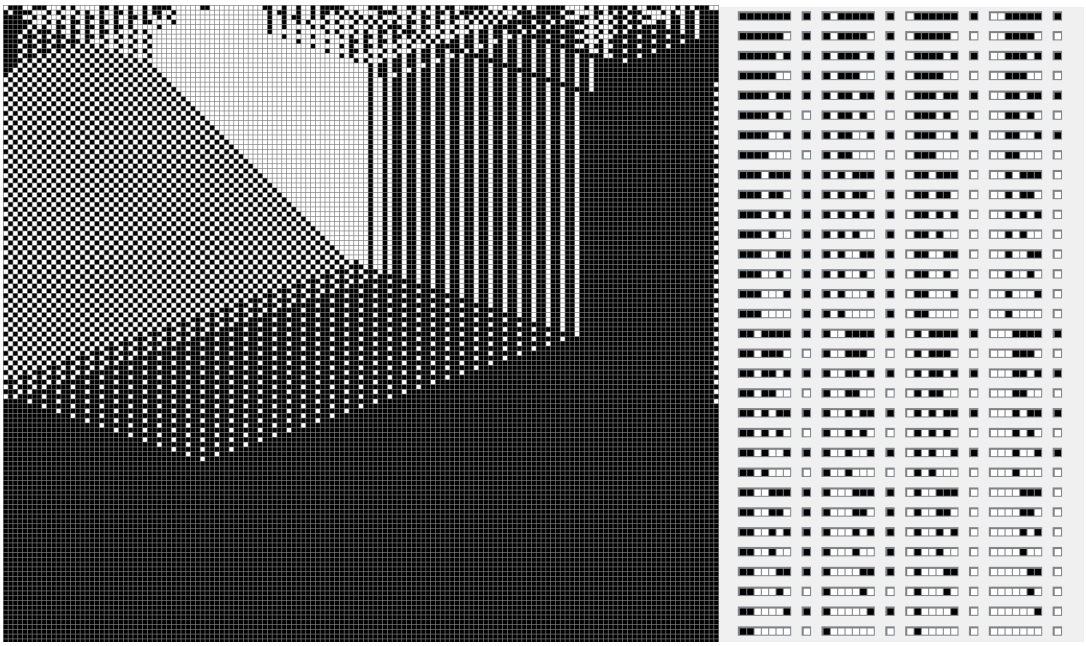
- dojde k ustálení na konfiguraci tvořené výhradně nulami, pokud v inicializační konfiguraci není dostatečně velký blok (přibližně $2r + 1$, kde r je počet sousedních buněk zleva nebo zprava – např. pro velikost okolí 7 je $r = 3$) tvořený

sousedními, nebo téměř sousedními, jedničkami; pokud ano, dojde k rozšíření tohoto bloku,

- dojde k ustálení na konfiguraci tvořené výhradně jedničkami, pokud v inicializační konfiguraci není dostatečně velký blok sousedních, nebo téměř sousedních, nul; jinak dojde k rozšíření tohoto bloku.

Technika genetických algoritmů však v několika případech dokázala nalézt pravidla se znatelně vyšší výkonností a sofistikovanějšími výpočetními vlastnostmi. Nejúspěšnější z těchto pravidel dosahovalo úspěšnosti 76,9 %. Většina provedených pokusů vedla k pravidlům s úspěšností pouze 65 – 70 % [24].

Další pravidlo vytvořil pomocí genetického programování John R. Koza v [2]. Toto pravidlo je při řešení problému klasifikace většiny úspěšné častěji než předchozí pravidla (jeho úspěšnost je 82,326 %). Jeho zadání i ukázku chování shrnuje obrázek 24.



Obrázek 24: Pravidlo vytvořené J. Kozou pomocí genetického programování.

4 Implementace problému klasifikace většiny

Tato kapitola popisuje postup implementace problému klasifikace většiny s využitím genetického programování a úspěšnost vytvořené aplikace při hledání řešení tohoto problému.

V jazyce Java jsem vytvořila aplikaci, jejímž úkolem je na základě principů genetického programování nalézt co nejlepší řešení daného problému (optimální řešení takto definovaného problému neexistuje, jak bylo dokázáno v [25]), tj. takové pravidlo, jehož aplikací na libovolnou počáteční řadu celulárních automatů dojde po určitém počtu kroků ke změně všech automatů v řadě buď na černé, nebo na bílé, v závislosti na barvě převládající v počáteční řadě. Nalezené pravidlo je aplikováno na každou buňku v řadě a na všechny řady, které následují, a to až do velikosti prostoru určené uživatelem.

Po zahájení činnosti genetický algoritmus nejprve vytvoří stromy (v programovém kódu nazývané *genotypy*, jejichž uzly jsou označovány jako *geny*) náhodně složené z prvků z množiny funkcí F a množiny terminálů T, které jsou zadány následovně:

$$T = \{S, V1, V2, V3, Z1, Z2, Z3\}$$

$$F = \{AND, OR, NAND, NOR, NOT, IF, XOR\}$$

Množinu terminálů tvoří všechny pozice, které utvářejí sousedství buňky. Terminál S označuje aktuální (prostřední) buňku. Terminál $V1$ symbolizuje buňku ležící východně (vpravo) od prostřední buňky, $V2$ buňku vpravo od buňky $V1$ A $V3$ buňku vpravo od buňky $V2$. $Z1$ označuje buňku vlevo od S, $Z2$ buňku vlevo od $Z1$ a $Z3$ buňku vlevo od $Z2$. Každá buňka obsahuje logickou hodnotu *true* (je-li označena bílou barvou), nebo *false* (je-li označena černě). S těmito hodnotami pracují logické funkce jako AND, OR, $NAND$ a další, obsažené v množině funkcí.

Po vytvoření inicializační (i každé následující) populace je chování každého jedince (genotypu, stromu), které spočívá v obarvení prostřední buňky v závislosti na stavu jejího okolí, převedeno na tabulku, v níž je všem kombinacím buněk v okolí přiřazena výsledná barva prostřední buňky. V kódu aplikace se tato tabulka nazývá *pravidlo*. Tento krok pomáhá ušetřit značné množství výpočetního času, protože program dále vyhodnocuje nový stav buňky jen pomocí vyhledání daného okolí v *pravidle* a nevyhodnocuje okolí prostřednictvím dosazení hodnot do uzlů stromu, který může nabývat velkých rozměrů (růst stromů je však omezován v závislosti na parametrech zadaných uživatelem, viz kapitola 4.1 *Nastavení parametrů*).

Každý genotyp, převedený na pravidlo, je následně otestován na 1000 testovacích počátečních konfigurací. Každá počáteční řada je vygenerována náhodně, přičemž každá buňka s 50% pravděpodobností získá černou, nebo bílou barvu. Po vytvoření testovací řady je zjištěno, která barva je převládající. Pravidlo je soustavně aplikováno na poslední řadu až do uživatelem zadaného počtu řad (obvykle 600). V poslední řadě musejí všechny buňky obsahovat tu barvu, která v první řadě převažovala, aby se genotypu mohl připočítat 1 bod k jeho *čisté zdatnosti.* Výsledná čistá zdatnost genotypu se tedy pohybuje v rozmezí $\langle 0;\ 1000\rangle$. Standardizovaná zdatnost se počítá jako

$$s(i,t) = r_{max} - r(i,t)$$

a po dosazení dostáváme $s(i,t) = 1000 - r(i,t)$.

Při tvorbě dalších generací jsou jedinci vybíráni na základě své čisté zdatnosti. K výběru jedinců slouží turnajová selekce pro 3 jedince. Nová generace je tvořena pomocí všech primárních a jedné sekundární genetické operace (decimace) uvedených v kapitolách 2.2.5 a 2.2.6). Podíly jednotlivých primárních operací na tvorbě nové generace určuje uživatel před zahájením běhu genetického algoritmu pomocí parametrů *pravděpodobnost reprodukce*, *křížení* a *mutace*. Decimaci je možné vypnout za účelem zrychlení výpočetního procesu, ale její použití může pomoci k dosažení lepších výsledků. Zahrnuta je také volitelná funkce elitismu, která automaticky kopíruje jedince s nejvyšší zdatností do příští generace.

Proces tvorby nových generací je opakován až do zadaného počtu kroků. Pokud uživatel zadal zobrazování výsledků v průběhu práce algoritmu, bude po každé zadané generaci nejlepší dosažené řešení demonstrováno na náhodně vygenerované řadě. Zdatnost nejlepšího řešení je vždy vypisována na obrazovku. Po ukončení práce algoritmu pak bude optimální řešení znovu ohodnoceno, a to kvůli zvýšení přesnosti výsledku. Upřesnění se provádí pomocí otestování na zvýšeném množství počátečních konfigurací, obvykle na 1 000 000 konfigurací, případně i větším (10 000 000 či 15 000 000 konfigurací [2]). Výsledkem je procentuálně vyjádřená úspěšnost nejlepšího dosaženého řešení a genotyp tohoto řešení, zapsaný v podobě textového řetězce.

4.1 Nastavení parametrů

Aby mohly být výsledky porovnány s výsledky dosaženými v [2], byly před zahájením běhu genetického algoritmu zadány shodné parametry:

počet automatů v řadě	149
počet řad (kroků)	600
velikost okolí	7
počet testovacích konfigurací – běh algoritmu	1 000
počet testovacích konfigurací – určení výsledku	1 000 000

Parametry genetického algoritmu byly při prvním spuštění nastaveny následovně (při dalších pokusech byly obměňovány, na rozdíl od parametrů uvedených výše s výjimkou *počtu kroků*, který byl při závěrečných pokusech snížen, aby došlo k úspoře výpočetního času):

počet generací	50
velikost počáteční populace	200
maximální inicializační hloubka stromu	4
maximální hloubka stromu po křížení	8
pravděpodobnost křížení v uzlu funkce	0,5
pravděpodobnost reprodukce	0,1
pravděpodobnost křížení	0,7
pravděpodobnost mutace	0,2
zachovávání nejzdatnějšího jedince	ano
decimace	ne
inicializační metoda	grow

Parametry *počet automatů v řadě* a *počet řad* udávají rozměry prostoru pro vykreslování celulárních automatů. *Velikost okolí* určuje, kolik buněk je zahrnuto do sousedství každé buňky (včetně buňky prostřední, proto jde vždy o liché číslo) a tím také určuje velikost tabulky reprezentující konkrétní pravidlo. Počty testovacích konfigurací určují, kolik náhodně vygenerovaných počátečních řad bude použito k otestování zdatnosti každého pravidla v populaci. Počet testovacích konfigurací pro určení konečného výsledku je použit pouze jednou, a to po ukončení genetického algoritmu, kdy je s jeho pomocí upřesněna zdatnost nejlepšího dosaženého řešení; i přesto však jde stále o nepříliš přesnou aproximaci, neboť zjištění skutečné zdatnosti by vyžadovalo otestování na všech možných počátečních konfiguracích, kterých je celkem 2^{149}.

Hodnoty parametrů genetického algoritmu byly při prováděných pokusech zadány s ohledem na nastavení, které v mé předchozí, bakalářské práci dosahovalo nejlep-

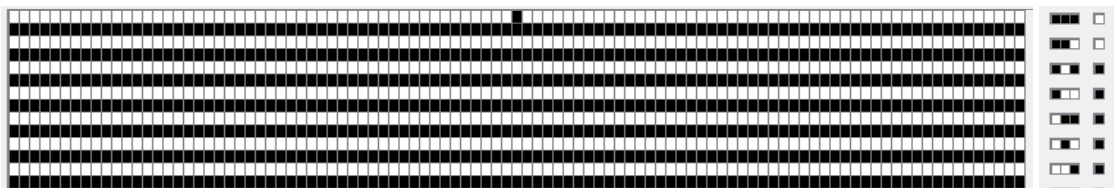
ších výsledků při řešení problému řízení pohybu umělého mravence po prostoru mřížky s umístěnými návnadami.

Bylo zjištěno, že v průběhu genetického algoritmu dochází k růstu počtu uzlů stromu, ale že lze toto chování do značné míry potlačit pomocí parametrů omezujících hloubku stromů.

4.2 Dosažené výsledky

Po spuštění aplikace na počítači s procesorem Intel Core 2 Duo 2,26 GHz, 4 GB RAM, bylo při prvním pokusu dosaženo vytvoření první generace až po 3 hodinách a 22 minutách a druhé generace po dalších 3 hodinách a 28 minutách. Uvedený čas odpovídá skutečnosti, že výpočty celulárních automatů a úlohy genetického programování obecně jsou určeny primárně pro výkonné paralelní počítače. Protože cílem této práce je řešit úlohu klasifikace většiny s využitím dostupných osobních počítačů, bylo před dokončením běhu genetického algoritmu nutné nejdříve tento výkon vylepšit tak, aby mohl být genetický algoritmus dokončen v přijatelném čase.

Ke zlepšení došlo nejprve omezením některých parametrů. Ty byly zvoleny tak, aby nedocházelo ke zkreslení nalezeného řešení. Například parametr *velikost populace* lze snižovat libovolně (avšak se zároveň se snižující pravděpodobností, že bude nalezeno dostatečně dobré řešení), zatímco snížení *počtu konfigurací v průběhu algoritmu* by mohlo vést ke zkreslení – po nastavení například na hodnotu 100 bývá dosahováno vysokých hodnot zdatnosti, a to již v několika prvních generacích (průměrně 60 – 70 %), avšak při následném testu s vysokým počtem testovacích konfigurací tato zdatnost klesá na hodnotu kolem 50 %, což obvykle znamená chování, kdy se celulární automaty po krátkém čase pouze ustálí na jedné a téže barvě a nejsou schopny dosáhnout opačné barvy, a to při žádné z testovaných počátečních konfigurací.



Obrázek 22: střídání černých a bílých řádků, které může být považováno za úspěšné, pokud poslední řádek obsahuje barvu, která převládala v původním řádku.

Přínosným způsobem, jak zlepšit výpočetní výkon, bylo zamezení dalším výpočtům v případě, že již bylo dosaženo jednotné barvy v celé řadě celulárních automatů. V takových případech je vhodné výpočet ukončit předčasně a dále nečekat na vytvo-

ření uživatelem stanoveného počtu řádků. Důležité je však správně určit úspěch či neúspěch tohoto pravidla. Během testů se totiž často objevovala i pravidla, která měla průměrnou úspěšnost a která se po určitém počtu kroků ustálila na opakování černého a bílého řádku (viz obrázek 22).

Pokud takové pravidlo skončilo řádkem správné barvy, bylo na předložené počáteční konfiguraci považováno za úspěšné; ovšem šlo jen o náhodu a předpokládám, že přibližně v polovině případů bylo naopak považováno za neúspěšné, protože skončilo řádkem opačné barvy. Při nalezení tohoto vzoru chování proto nemá smysl pokračovat v jeho dalších výpočtech. Rovněž je užitečné označit tento vzor jako neúspěšný, aby došlo k potlačení jeho dalšího výskytu v populaci. Výpočet prostoru celulárních automatů lze také předčasně ukončit v případě, že dvě po sobě jdoucí řady mají shodnou barvu. Pak zbývá již jen vyhodnotit, zda barva odpovídá požadovanému výsledku, a tedy zda tento test proběhl úspěšně, a tím ušetřit znatelné množství času potřebného k výpočtu.

Po opětovném spuštění aplikace se ukázalo, že tento princip nenapomáhá snížení potřebného času v prvních generacích, pokud v nich nebyl ve vyšší míře zastoupen program s některým z výše uvedených chování. Avšak v pozdějších fázích běhu genetického algoritmu, kdy se předpokládá vznik programů, které problém řeší pro většinu počátečních konfigurací úspěšně, pomůže tento postup uspořit značné množství času, protože nadále nebudou probíhat výpočty velkého počtu řádků, jejichž další složení bude již dopředu známé.

Po těchto úpravách proběhly 4 experimenty s různě obměňovanými parametry genetického algoritmu (viz tabulka níže, která uvádí i celkové výsledky).

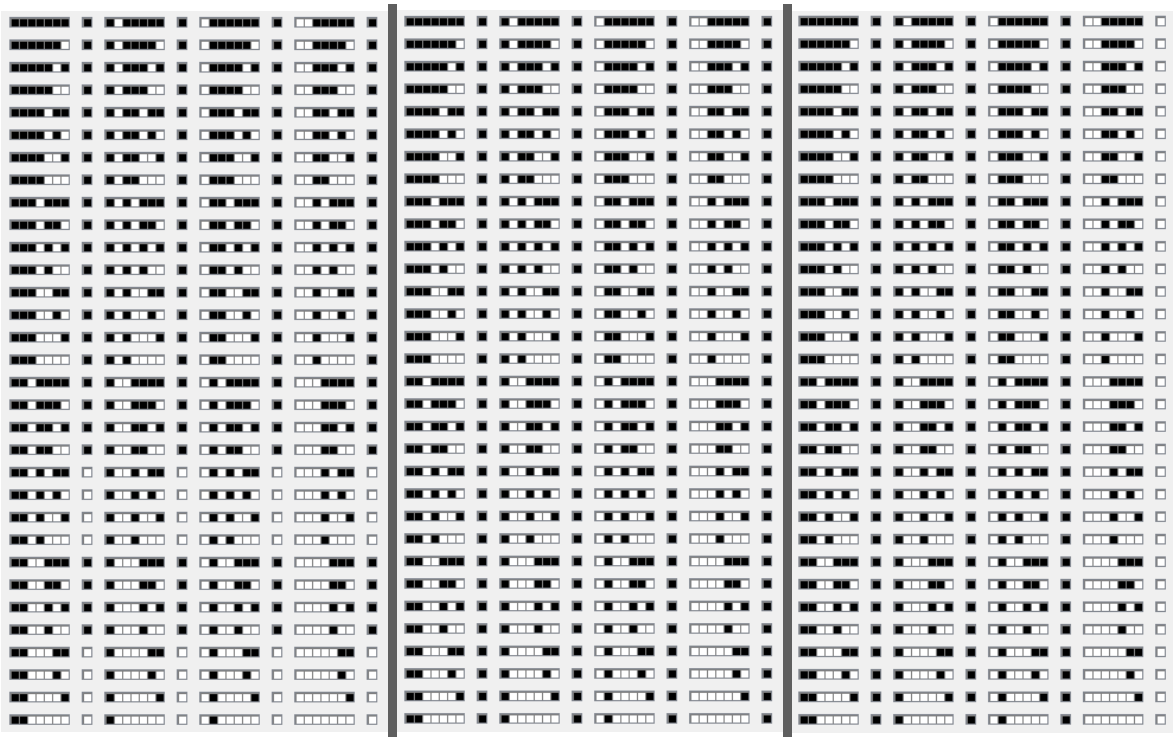
číslo pokusu	1	2	3	4
počet generací	40	50	50	50
velikost počáteční populace	50	60	100	200
max. inicializační hloubka stromu	4	4	4	6
max. hloubka stromu po křížení	8	8	8	14
pravd. křížení v uzlu funkce	0,5	0,5	0,7	0,5
pravd. reprodukce	0,1	0,1	0,1	0,05
pravd. křížení	0,7	0,7	0,7	0,8
pravd. mutace	0,2	0,2	0,2	0,15
elitismus	ano	ano	ano	ano
inicializační metoda	grow	ramped	grow	grow
decimace	ano	ano	ano	ano
výsledná úspěšnost	56,5 %	56,5 %	56,5 %	56,5 %

Výsledná úspěšnost uváděná v posledním řádku výše uvedené tabulky byla zjištěna otestováním každého nejlepšího nalezeného pravidla na 1 000 000 náhodných počátečních konfigurací. Tato úspěšnost (56,5 %) se neměnila, bez ohledu na změny v hodnotách parametrů.

Pravidla, která vznikla jako výsledek těchto 4 pokusů, jsou následující:

pravidlo č. 1	`AND(Z1(),V1())`
pravidlo č. 2	`AND(NOT(Z3()),AND(Z3(),V1()))`
pravidlo č. 3	`AND(OR(Z3(),Z2()),OR(V3(),S()))`
pravidlo č. 4	`AND(XNOR(OR(NOT(AND(V3(),V3())),V3()),Z3()),Z2())`

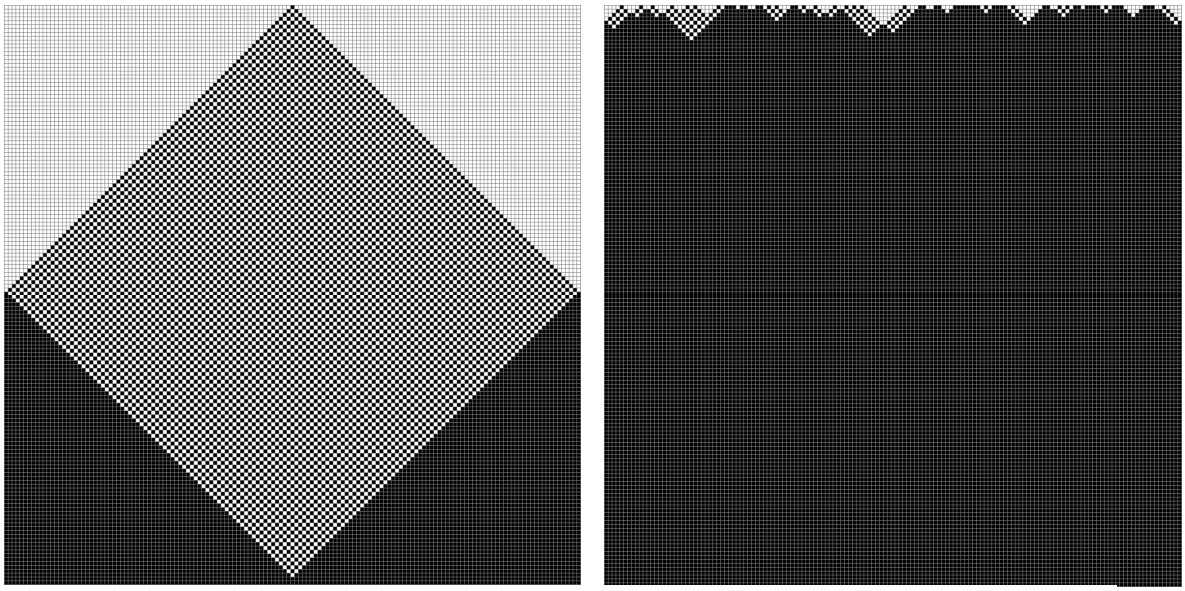
Tato pravidla lze přepsat do tabulek pravidel celulárních automatů. Ukázka převedených pravidel 1, 2 a 4 je na obrázku 23.



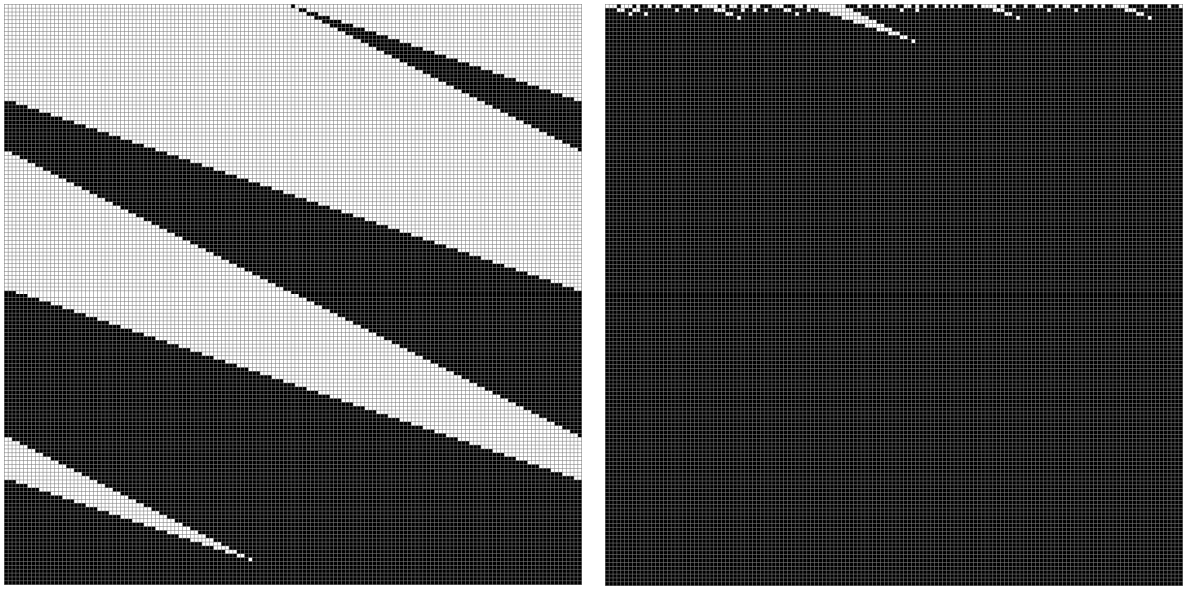
Obrázek 23: tabulky pravidel č. 1, č. 2 a č. 4 (zleva).

Tabulky pravidel č. 1, 2 a 4 na obrázku 23 ukazují, že přestože pravidlo č. 1 vypadá v podobě generované genetickým algoritmem nejjednodušeji, jeho skutečné chování je složitější než u pravidel č. 2 a č. 4, kde pravidlo č. 2 pouze mění buňku v černou a pravidlo č. 4 mění barvu v bílou, pokud jsou buňky *Z*3 a *Z*2 bílé.

Obrázky 24 a 25 zachycují chování pravidel č. 1 a č. 4 na počáteční konfiguraci bílých buněk, v níž jen prostřední je obarvena černě, a také na konfiguraci náhodných buněk.



Obrázek 24: chování pravidla č. 1:
`AND(Z1(),V1())`

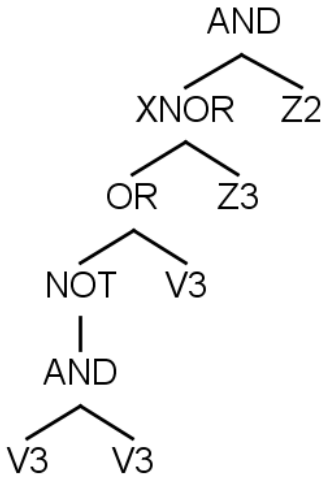


Obrázek 25: chování pravidla č. 4:
`AND(XNOR(OR(NOT(AND(V3(),V3())),V3()),Z3()),Z2())`

Z důvodu dosahované úspěšnosti pouze 56,5 % byl implementovaný genetický algoritmus upraven, a to ve dvou aspektech:

- selekční mechanismus byl změněn z turnajové selekce pro 3 jedince na upravený selekční mechanismus vycházející z výběru podle pořadí,
- elitismus byl modifikován tak, aby kromě přesunutí nejzdatnějšího jedince do další generace vytvořil také náhodný počet (1 až 5) *mutací* tohoto jedince takových, že mutace bude probíhat pouze na terminálech (stromových listech).

V průběhu předchozích 4 pokusů docházelo k nalezení nejzdatnějšího jedince již v časných generacích, ale tento jedinec se již dále neměnil, a to až do konce běhu genetického algoritmu. Přitom například pravidlo č. 4 (obrázek 26 ukazuje strom tohoto pravidla) obsahuje opakovaně terminál *V*3 (v podstromu `AND(V3(),V3())`). Vzhledem ke zdatnosti výsledných pravidel by mutace jejich terminálů mohla v některých případech přinést do populace potřebné zlepšení.



Obrázek 26: strom pravidla č. 4:
`AND(XNOR(OR(NOT(AND(V3(),V3())),V3()),Z3()),Z2())`

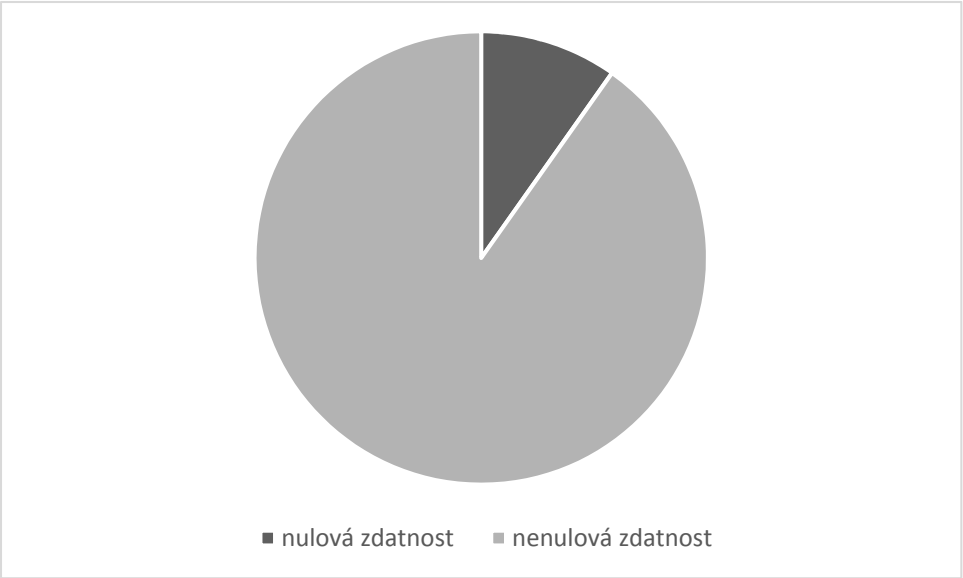
Výběr podle pořadí jsem zvolila z důvodu, že většina relativně zdatných jedinců v populaci má hodnotu zdatnosti kolem 50 % a nejzdatnější jedinci v této populaci měli kvůli turnajové selekci nižší pravděpodobnost výběru, než by byla potřebná vzhledem k jejich zachování a možným genetickým úpravám v dalších generacích. Klasický výběr podle pořadí by ovšem neposkytl požadované zlepšení výběru, a to z důvodu nestandardně rozložených hodnot zdatností v populaci. Bylo zjištěno, že přibližně 70 % populace má zdatnost nulovou a zdatnost zbylých 30 % populace se pohybuje kolem hodnoty 500 (pro 1000 testovacích případů). Při použití klasického výběru podle pořadí by jedinci s nulovou úspěšností celkově získali převahu nad těmi s nenulovou úspěšností. Vliv těchto jedinců bylo nutné potlačit, ale zároveň i nadále umožnit jejich výběr kvůli zachování potřebné různorodosti.

Selekční metodu jsem implementovala jako výběr podle pořadí s následujícími novými úpravami:

- jedinci se zdatností vyšší než 0 budou podle této zdatnosti seřazeni od *n* do 1 (kde *n* je počet jedinců se zdatností vyšší než 0) sestupně podle zdatnosti a budou do populace pro výběr přidáni tolikrát, jaké měli pořadí,

- každý jedinec se zdatností 0 bude do populace pro výběr zahrnut právě k-krát, přičemž $k = \frac{p_p}{p_n} \cdot \frac{1}{10}$ po zaokrouhlení směrem nahoru, kde p_p je počet jedinců se zdatností vyšší než 0 v populaci pro výběr a p_n je počet jedinců s nulovou zdatností.

Tato selekční metoda rozloží pravděpodobnosti výběru jedinců s nulovou a nenulovou zdatností tak, jak je patrné z grafu 3.



Graf 3: Pravděpodobnost výběru jedinců s nulovou a nenulovou zdatností.

Pokud je však prováděna decimace, především je-li prováděna seřazením jedinců sestupně podle zdatnosti a následným zachováním pouze určitého podílu nejzdatnějších z nich, pak po provedení decimace s vysokou pravděpodobností zůstanou v populaci pouze jedinci se zdatností vyšší než 0 a tudíž se v generaci bezprostředně následující po decimaci bude jednat o klasický výběr podle pořadí. V dalších generacích se již opět mohou vyskytnout jedinci s nulovou zdatností a rozložení pravděpodobnosti výběru jedinců s nulovou a nenulovou zdatností bude v takovém případě znovu odpovídat grafu 3.

Po výše uvedené úpravě stávajícího algoritmu byly provedeny poslední dva pokusy. Při nich byl počet řádků snížen, protože z předchozích pokusů vyplynulo, že dochází-li při použití pravidla ke konvergenci řady, pak se tak děje nejčastěji již během prvních 100 – 200 řad. Snížením počtu řad došlo k významné časové úspoře.

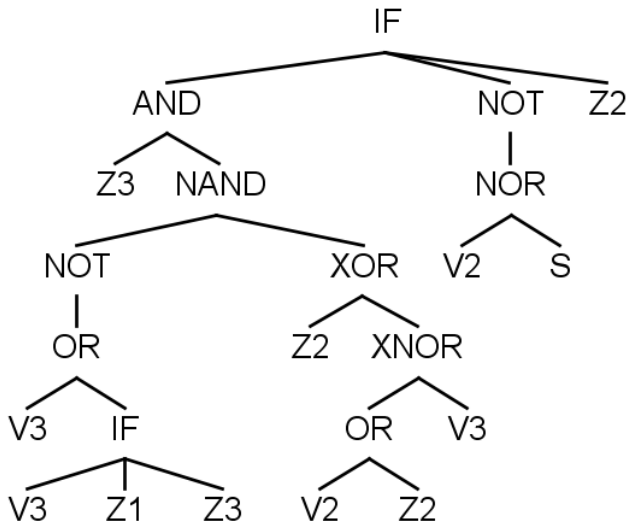
Nastavení těchto dvou pokusů (č. 5, č. 6) uvádí následující tabulka:

číslo pokusu	5	6
počet generací	90	10
velikost počáteční populace	100	100
max. inicializační hloubka stromu	6	6
max. hloubka stromu po křížení	14	14
pravd. křížení v uzlu funkce	0,5	0,5
pravd. reprodukce	0,05	0,05
pravd. křížení	0,8	0,8
pravd. mutace	0,15	0,15
elitismus	ano	ano
inicializační metoda	grow	grow
decimace	ano	ano
počet CA v řadě	149	149
počet řad	300	400

Při 5. pokusu v 5. generaci aplikace nalezla pravidlo s čistou zdatností 657, které až do konce běhu genetického algoritmu (tzn. po dalších 85 generací) zůstalo nejlepším dosaženým řešením. Otestováním na 1 000 000 náhodných počátečních konfigurací jsem získala výslednou úspěšnost 62,6878 %. Zápis tohoto pravidla je:

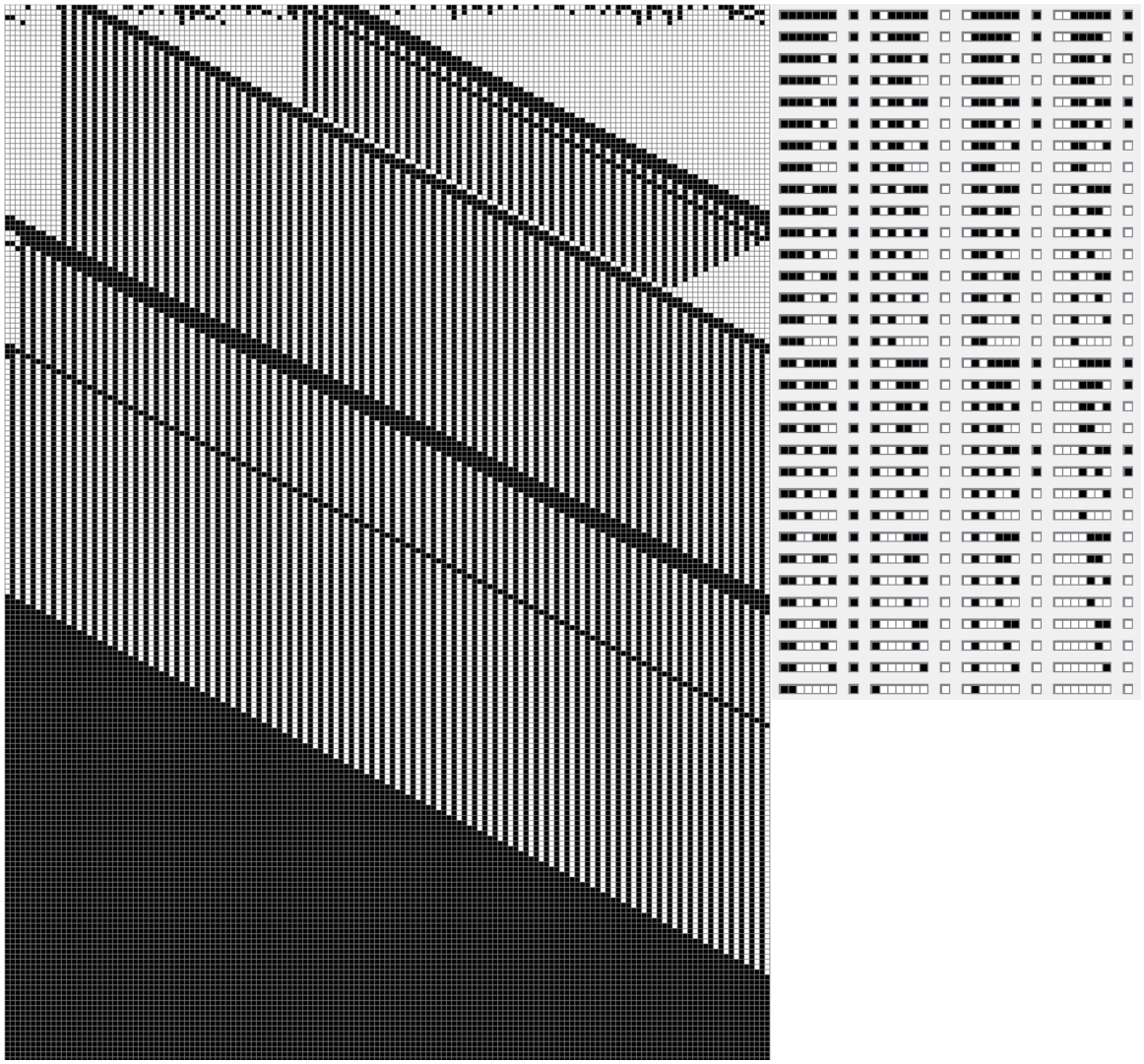
```
IF(AND(Z3(),NAND(NOT(OR(V3(),IF(V3(),Z1(),Z3()))),XOR(Z2(),
XNOR(OR(V2(),Z2()),V3())))),NOT(NOR(V2(),S())),Z2())
```

pro lepší přehlednost lze zápis převést do následující podoby grafu (obrázek 27):



Obrázek 27: strom pravidla č. 5:
`IF(AND(Z3(),NAND(NOT(OR(V3(),IF(V3(),Z1(),Z3()))),XOR(Z2(),XNOR(OR(V2(),Z2()),V3())))),NOT(NOR(V2(),S())),Z2())`

Chování tohoto pravidla na náhodné počáteční konfiguraci ukazuje obrázek 28.

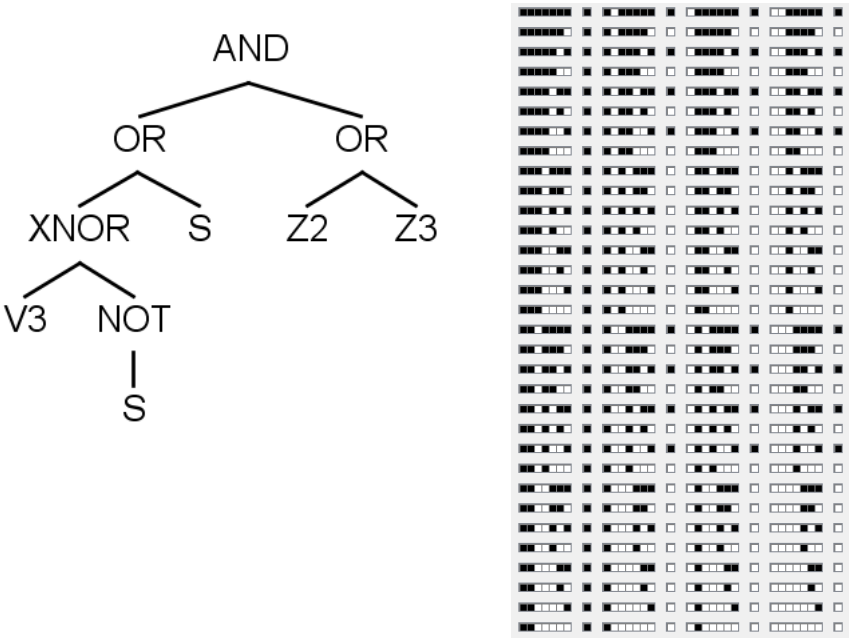


Obrázek 28: Chování nejúspěšnějšího pravidla v pokusu č. 5

Při 6. pokusu bylo již v 7. generaci nalezeno pravidlo, které překonalo pravidla nalezená při předchozích pokusech. Toto pravidlo získalo za běhu čistou zdatnost 661, která měla po otestování na 1 000 000 počátečních konfigurací výslednou úspěšnost 65,4011 %, tedy o 8,9 % vyšší než první čtyři předchozí pravidla. Toto pravidlo lze zapsat jako:

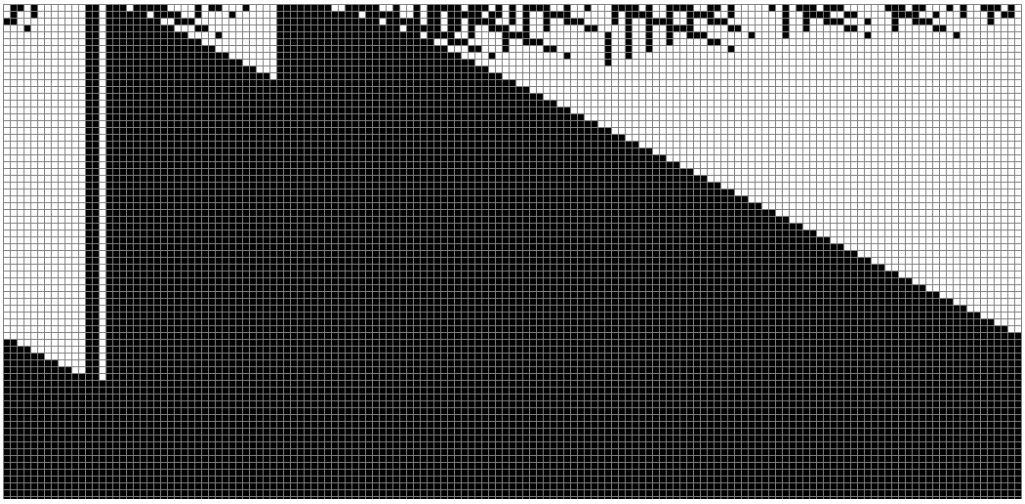
```
AND(OR(XNOR(V3(),NOT(S())),S()),OR(Z2(),Z3()))
```

a přepsat do podoby stromu a tabulky následovně (obrázek 29):



Obrázek 29: strom a tabulka pravidla `AND(OR(XNOR(V3(),NOT(S())),S()),OR(Z2(),Z3()))`

Jak je patrné z obrázku 30, toto pravidlo podobně jako pravidlo z pokusu č. 5 udává komplexnější globální chování, než jaké měla první čtyři získaná pravidla. Strategie pravidel č. 5 a č. 6 se z uvedených obrázků jeví jako velmi podobná, výsledné chování je ale silně ovlivněno konkrétní podobou počáteční konfigurace.



Obrázek 30: Globální chování pravidla `AND(OR(XNOR(V3(),NOT(S())),S()),OR(Z2(),Z3()))`

5 Shrnutí výsledků

Tato kapitola je zaměřena na shrnutí výsledků dosažených v předchozí části a jejich porovnání s úspěšností známých pravidel uvedených v kapitole 3.4 *Problém klasifikace většiny*.

První čtyři pokusy dosahovaly úspěšnosti pouze 56,5 %, což je velmi nízká úspěšnost v porovnání s výsledky známých pravidel (viz kapitola 3.4), které shrnuje tabulka výsledků uvedená níže.

Pravidlo	**Úspěšnost**
Genetické algoritmy (Das, Mitchell a Crutchfield)	76,9 %
GKL (Gacs, Kurdyumov a Levin)	81,6 %
Davisovo pravidlo	81,8 %
Dasovo pravidlo	82,178 %
Genetické programování (Koza)	82,326 %

Z toho důvodu byla před provedením dalšího pokusu stávající implementace upravena, a to modifikací selekčního mechanismu a při použití elitismu také tvorbou mutací terminálových uzlů nejzdatnějšího jedince (podrobnosti jsou uvedeny v předchozí kapitole 4.2 *Dosažené výsledky*).

Následné dva pokusy přinesly výrazné zvýšení dosahované úspěšnosti. V prvním pokusu bylo již v 5. generaci nalezeno řešení, které bylo o 6,2 % úspěšnější než předchozí pravidla; ve druhém bylo v 7. generaci nalezeno řešení lepší o 8,9 %. Tato pravidla jsou z hlediska úspěšnosti srovnatelná s pravidly, která byla vytvořena pomocí genetických algoritmů v [24] a která dosahovala úspěšnosti nejčastěji mezi 65 a 70 %.

Nejlepší nalezené řešení s úspěšností 65,4011 % představovalo pravidlo, které lze zapsat následovně:

```
AND(OR(XNOR(V3(),NOT(S())),S()),OR(Z2(),Z3()))
```

Tento zápis odpovídá stromovému grafu a tabulce uvedených na obrázku 29. Jeho globální chování (viz obrázek 30) se jeví jako výrazně složitější než u prvních čtyř pravidel nalezených v rámci této práce. Po provedení několika aplikací na nezávislých, náhodně vygenerovaných počátečních řadách toto pravidlo svým průběžným globálním chováním vypadá jako značně odlišné od pravidla J. Kozy i pravidla GKL (jejich ukázky viz kapitola 3.4 *Problém klasifikace většiny*). Chování uvedených dvou pravidel vypadá mnohem komplexněji a nabízí mnohem lepší výsledky než pravidla, kterých se podařilo dosáhnout v rámci této práce.

Důvodem nízké úspěšnosti v porovnání s výše uvedenými pravidly je především nízký počet jedinců v populaci, který byl během pokusů nastaven maximálně na 100 (při užití decimace se pro prvních 10 generací zvýšil na 1000 jedinců), zatímco při pokusech prováděných J. Kozou byla tato hodnota nastavena na 51 200 [2]. Protože mé pokusy probíhaly na osobním počítači namísto počítače paralelního, nebylo možné z časových důvodů vytvořit tak velký počet jedinců.

Skutečnost, že vysoký počet jedinců v populaci je důležitější než počet generací, byla potvrzena pokusem s 90 generacemi, během kterých se od 5. generace nejlepší jedinec ani jednou nezměnil. John Koza v [2] uvádí použití pouze 51 generací.

Pozorování, že pokroků je dosahováno pouze během prvních deseti generací před decimací, potvrzuje závěr, že pro nalezení dostatečně úspěšného řešení je vysoký počet jedinců v populaci nejvýznamnější. V případě, že by bylo časově možné podstatně zvýšit velikost populace, by aplikace nalezla mnohem úspěšnější řešení.

6 Závěry a doporučení

Tato diplomová práce si kladla za cíl seznámit čtenáře se základními principy genetického programování a celulárních automatů a získané poznatky aplikovat při řešení problému klasifikace většiny v ukázkové aplikaci. Pravidlo celulárního automatu pro tento problém bylo hledáno právě pomocí genetického programování.

V úvodu práce bylo představeno genetické programování jako jedna z evolučních výpočetních technik, jejíž principy se inspirovaly biologickou genetikou. Následoval popis základních rysů a postupů využití genetického programování včetně činnosti genetického algoritmu, reprezentace jedinců v populaci, hodnocení zdatnosti těchto jedinců a jejich výběr a následná tvorba nových generací. Dále tato práce představila problematiku celulárních automatů a problém klasifikace většiny. Tento problém byl poté řešen pomocí genetického programování implementovaného v aplikaci v programovacím jazyce Java.

Napsání vlastní implementace genetického programování přineslo výhody v podobě rozšiřitelnosti o libovolné vlastní funkce nebo možnosti úprav kterékoli části algoritmu, což umožnilo například následně provést modifikaci elitismu za účelem zlepšení dosahovaných výsledků. Dalším přínosem se stalo získání nových zkušeností s programováním v jazyce Java a prohloubení jeho znalostí.

Pomocí této aplikace se podařilo nalézt dvě pravidla celulárního automatu, která jsou z hlediska úspěšnosti při řešení problému klasifikace většiny srovnatelná s průměrnými pravidly generovanými pomocí genetických algoritmů v [24], která dosahovala nejčastěji úspěšnosti 65 – 70 %. Pravidla nalezená v rámci této práce měla po otestování na 1 000 000 náhodných konfigurací úspěšnost přibližně 65,4 % a 62,2 %. Této úspěšnosti se podařilo dosáhnout po změně selekčního mechanismu a úpravě elitismu; v předchozích pokusech bylo dosaženo úspěšnosti pouze 56,5 %.

Z toho vyplývá závěr, že použitím techniky genetického programování je možné úspěšně nacházet pravidla celulárních automatů s požadovaným globálním chováním, avšak pravidla vytvořená pomocí vlastní aplikace se nedokázala vyrovnat pravidlům nalezeným implementací genetického programování Johna Kozy [2]. Z výsledků vyplývá, že příčinou je s největší pravděpodobností malá velikost populace, která byla dána nutností provádět výpočet na osobním počítači. V menší míře mohly být výsledky negativně ovlivněny také nízkým počtem provedených pokusů, odlišně nastavenými podíly křížení, reprodukce a mutace, nevyužitím automaticky definovaných funkcí nebo také odlišnými dílčími metodami genetického algoritmu, například použitým selekčním mechanismem.

Následující doporučení se týká dalšího možného rozšíření aplikace, která je součástí této diplomové práce.

Genetický algoritmus by bylo možné upravit tak, aby využíval automaticky definované funkce (ADF). Následně by bylo možné porovnávat výsledky dosažené stávající aplikací a aplikací využívající ADF a tím zjistit, nakolik ADF pomáhají při hledání vysoce úspěšných řešení problému klasifikace většiny. Problémem při tomto porovnávání by však mohl být nízký počet provedených experimentů, který by neumožňoval spolehlivě statisticky prokázat, zda použití ADF v případě problému klasifikace většiny vede k nacházení pravidel s vyšší úspěšností než stávající implementace.

I přes výše uvedené možnosti zlepšení lze tímto shrnout, že bylo v rámci této diplomové práce dosaženo stanovených záměrů a cílů.

7 Seznam použité literatury

[1] BANZHAF, Wolfgang, NORDIN, Peter, KELLER, Robert E., FRANCONE, Frank D. *Genetic Programming – An Introduction: On the Automatic Evolution of Computer Programs and Its Applications.* San Francisco: Morgan Kaufmann Publishers, Inc., 1998. 480 s. ISBN 1-55860-510-X.

[2] KOZA, John R., BENNETT III, Forrest H., ANDRE, David, KEANE, Martin A. *Genetic Programming III: Darwinian Invention and Problem Solving.* San Francisco: Morgan Kaufmann Publishers, Inc., 1999. 1154 s. ISBN 1-55860-543-6.

[3] EBERHART, Russell C., SHI, Yuhui. *Computational Intelligence: Concepts to Implementations.* Morgan Kaufmann Publishers, Inc., 2007. 457 s. ISBN 978-1-55860-759-0.

[4] ASHBY, W. Ross. *Principles of the self-organizing dynamic system.* The Journal of general psychology, 1947. s. 125-128.

[5] ENGELBRECHT, Andries P. *Computational Intelligence: An Introduction.* John Wiley & Sons, 2007. 597 s. ISBN 978-0-470-03561-0.

[6] KOZA, John R. *Genetic Programming On the Programming of Computers by Means of Natural Selection*. Cambridge: The MIT Press, 1998. 813 s. ISBN 0-262-11170-5.

[7] KOZA, John R. *Hierarchical Genetic Algorithms Operating on Populations of Computer Programs*. In International Joint Conference on Artificial Intelligence [online]. 1989 [cit 2015-11-2]. s. 768-774. Dostupný z internetu: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.141.1335&rep=rep1&type=pdf>.

[8] SPECTOR, Lee. *Automatic Quantum Computer Programming: A Genetic Programming Approach.* Kluwer Academic Publishers, 2004. 153 s. ISBN 1-4020-7894-3.

[9] POLI, Riccardo, LANGDON, William B., McPHEE, Nicholas F. *A Field Guide to Genetic Programming* [online]. 2008 [cit 2013-2-6]. Dostupný z internetu: <http://www0.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/poli08_fieldguide.pdf>.

[10] MUNAKATA, Toshinori. *Fundamentals of the New Artificial Intelligence: Neural, Evolutionary, Fuzzy and More.* Heidelberg: Springer, 2008. 255 s. ISBN 978-1-84628-838-8.

[11] HYNEK, Josef. *Genetické algoritmy a genetické programování.* Grada Publishing a.s., 2008. 182 s. ISBN 978-80-247-2695-3.

[12] ANGELINE, Peter. J., KINNEAR Kennth E. *Advances in Genetic Programming*, The MIT Press, 1996. 538 s. ISBN 0-262-11188-8.

[13] MITCHELL, Melanie. *Computation in Cellular Automata: A Selected Review.* Non-standard Computation [online]. 1996 [cit 2015-11-6]. s. 95-140. Dostupný z internetu: <http://www.researchgate.net/profile/Melanie_Mitchell2/publication/2700788_Computation_in_Cellular_Automata_A_Selected_Review/links/543543390cf2bf1f1f286267.pdf>.

[14] WOLFRAM, Stephen. *Statistical Mechanics of Cellular Automata*. Reviews of Modern Physics [online]. 1983 [cit 2015-11-3]. Dostupný z internetu: <https://gbic.biol.rug.nl/~rbreitling/dagstuhlpublications/003_Wolfram1983.pdf>.

[15] SCHIFF, Joel L. *Cellular Automata: A Discrete View of the World*. John Wiley & Sons, 2011. 280 s. ISBN 978-0-470-16879-0.

[16] SIEPMANN, Peter. *A Genetic Algorithm Approach to Probing the Evolution of Self-organized Nanostructured Systems.* Nano letters [online]. 2007 [cit 2015-11-3]. Dostupný z internetu: <http://ico2s.org/data/papers/Siepmann2007.pdf>.

[17] WOLFRAM, Stephen. *A New Kind Of Science.* Wolfram Media, 2002. 1197 s. ISBN 1-57955-008-8.

[18] WEISSTEIN, Eric W. *Elementary Cellular Automaton* [online]. 2015 [cit 2015-12-10]. MathWorld - A Wolfram Web Resource. Dostupný z internetu: <http://mathworld.wolfram.com/ElementaryCellularAutomaton.html>.

[19] CAMARA, Daniel. *Non-Uniform Cellular Automata - A Review.* Department of Computer Science, University of Maryland [online]. 1999 [cit 2016-1-28]. <http://www.eurecom.fr/~camara/files/NonUniformCellularAutomataReview.pdf>.

[20] TOMASSINI, M.; SIPPER, M.; PERRENOUD, M. *On the Generation of High-Quality Random Numbers by Two-Dimensional Cellular Automata*. IEEE Transactions on Computers [online]. 2000 [cit 2016-1-27]. Dostupný z internetu: <http://www.cs.bgu.ac.il/~sipper/papabs/twodrand.pdf>.

[21] MITCHELL, Melanie; CRUTCHFIELD, James P.; HRABER, Peter T. *Evolving cellular automata to perform computations: Mechanisms and impediments.* Physica D: Nonlinear Phenomena [online]. 1994 [cit 2016-3-7]. Dostupný z internetu: <http://web.cecs.pdx.edu/~mm/mech-imped.pdf>.

[22] ANDRE, David, BENNET, Forrest H., KOZA, John R. *Discovery by Genetic Programming of a Cellular Automata Rule that is Better than any Known Rule for the Majority Classification Problem.* Proceedings of the First Annual Conference on Genetic Programming [online]. 1996 [cit 2015-3-29]. Dostupný z internetu: <http://www.cs.bham.ac.uk/~wbl/biblio/cache/http___www.genetic-programming.com_jkpdf_gp1996gkl.pdf>.

[23] GACS, Peter. *Reliable computation with cellular automata.* Proceedings of the fifteenth annual ACM symposium on Theory of computing [online]. 1983 [cit 2016-3-7]. Dostupný z internetu: <http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA150656>.

[24] DAS, Rajarshi; MITCHELL, Melanie; CRUTCHFIELD, James P. *A genetic algorithm discovers particle-based computation in cellular automata.* Parallel problem solving from nature [online]. 1994 [cit 2016-3-10]. Dostupný z internetu: <http://csc.ucdavis.edu/~evca/Papers/GA-Particle.part1.pdf>

[25] LAND, Mark; BELEW, Richard K. *No perfect two-state cellular automata for density classification exists*. Physical review letters [online]. 1995 [cit 2016-3-7]. Dostupný z internetu: <http://www.academia.edu/download/31070607/10.1.1.47.8335.pdf>.

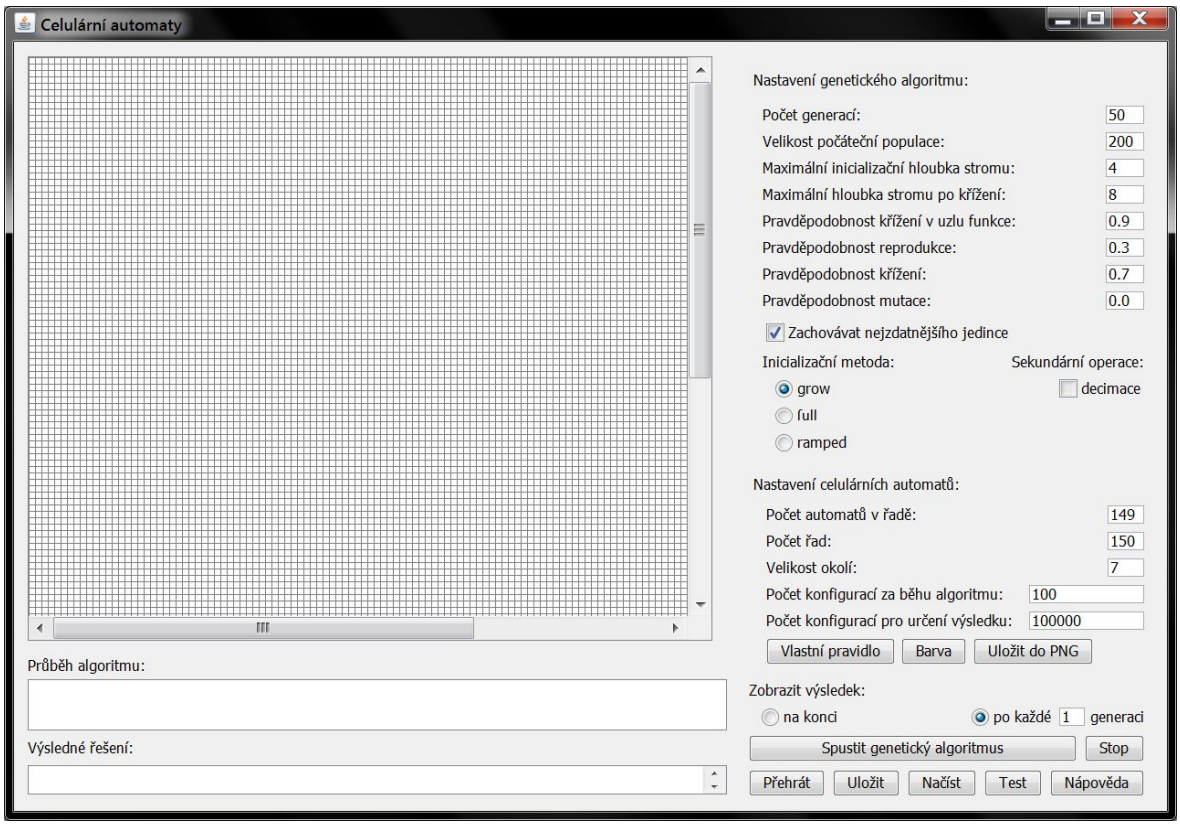
Přílohy

A. Česko-anglický slovník použitých pojmů

čistá zdatnost	*raw fitness*
evoluční výpočetní techniky	*evolutionary computation*
hledání naslepo	*blind search*
konečný automat	*finite state machine*
normalizovaná zdatnost	*normalized fitness*
metoda výstupu na vrchol	*hill climbing method*
ohodnocovací funkce	*fitness function*
optimalizace hejnem částic	*particle swarm optimization*
paprskové prohledávání	*beam search*
prostor hledání	*search space*
přizpůsobená zdatnost	*adjusted fitness*
ruletová selekce	*roulette selection*
samoorganizace	*self-organization*
standardizovaná zdatnost	*standardized fitness*
teorie rojení	*swarming theory*
turnajová selekce	*tournament selection*
výběr podle pořadí	*rank selection*
výběr úměrný zdatnosti	*fitness-proportionate selection*
zdatnost	*fitness*

B. Uživatelská dokumentace

Po spuštění aplikace Celulární automaty se zobrazí hlavní okno, rozdělené na tři části. Mřížka vlevo je prostorem pro vykreslování jednorozměrných celulárních automatů v řadách pod sebou. Aktuální nastavení buněk lze ovlivnit buď klikáním přímo do mřížky, které při každém kliknutí přepne barvu vybrané buňky z bílé na černou, anebo pomocí tlačítka *Barva* (v pravé části uprostřed), které změní barvu všech buněk v mřížce na černou nebo bílou (se dvěma mezikroky: v prvním jsou přebarveny všechny buňky a buňka v první řadě uprostřed dostane barvu opačnou – tato konfigurace slouží k testování chování pravidla při co nejjednodušší počáteční konfiguraci; ve druhém je vygenerována náhodná řada). V případě, že je konfigurace zadávána ručně myší, bude při následujících aplikacích pravidla využita pouze první řada a ostatní budou překresleny.

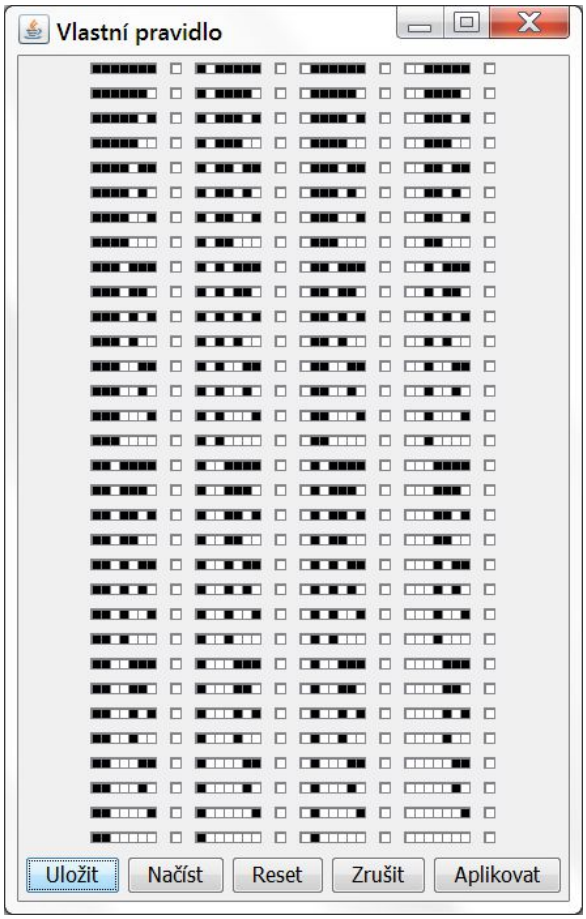


Pod mřížkou se nacházejí dvě textová pole *Průběh algoritmu* a *Výsledné řešení*. *Průběh algoritmu* poskytuje informace o běžícím genetickém algoritmu, jako jsou například průběžné výpisy zdatnosti nejzdatnějšího dosud nalezeného pravidla. Pole *Výsledné řešení* obsahuje nejzdatnější dosud nalezené pravidlo ve formátu stromu přepsaného do textového řetězce (syntaxe tohoto řetězce je obdobná syntaxi programovacího jazyka LISP).

Panel vpravo umožňuje úpravu některých parametrů, a to zvlášť pro genetický algoritmus a zvlášť pro celulární automaty. V pravém dolním rohu se nacházejí tlačítka pro další práci s aplikací, z nichž nejdůležitější je *Spustit genetický algoritmus*, které zahájí hledání nejlepšího řešení problému klasifikace většiny. Výstupy budou viditelné v levé části okna. Uživatel může pomocí přepínače *Zobrazit výsledek* ovlivnit, zda bude nalezené řešení demonstrováno v mřížce vlevo až po skončení běhu genetického algoritmu, nebo zda bude mřížka zobrazovat chování nejlepšího dosavadního řešení po každé zadané generaci. Tlačítko *Stop* umožňuje předčasně zastavit běh genetického algoritmu a vypsat výsledek.

Zcela dole se nacházejí tlačítka *Nápověda* (pro zobrazení podrobných informací o parametrech a funkcích aplikace) a trojice tlačítek *Přehrát, Uložit* a *Načíst*, která pracují s řetězcem uvedeným v textovém poli *Výsledné řešení*. Je možné jej uložit do textového souboru, opětovně nahrát a znovu přehrát její chování, a to na libovolné počáteční konfiguraci, kterou uživatel zadá pomocí myši a tlačítka *Barva*.

Tlačítko *Vlastní pravidlo* zobrazí nové okno, které umožňuje ručně zvolit pravidlo a následně jej aplikovat na uživatelem nastavenou počáteční konfiguraci. V okně jsou



zobrazeny všechny kombinace buněk v okolí; jejich počet tudíž závisí na velikosti okolí (která je jedním z nastavitelných parametrů, ve výchozím stavu je nastavena na hodnotu 7). Po pravé straně každé kombinace buněk v okolí se nachází jedna bílá buňka, kterou lze kliknutím myší změnit na černou a naopak. Tato buňka představuje prostřední buňku v okolí po aplikaci pravidla. Po nastavení pravidla jej můžeme *Uložit* pro pozdější použití (bude uloženo do textového souboru v podobě sekvence hodnot 0 a 1) a poté znovu *Načíst*. Stisknutím *Reset* dojde k přebarvení všech nastavitelných buněk zpět na bílou barvu. *Zrušit* uzavře okno bez uložení změn a *Aplikovat* aktuální pravidlo použije na uživatelem zadanou počáteční konfiguraci v mřížce.

Poslední tlačítko v hlavním okně, *Uložit do PNG*, ukládá jako obrázek ve formátu PNG aktuální obsah mřížky vlevo, a to jak po aplikování vlastního pravidla, tak i kdykoli za běhu genetického algoritmu.

V hlavním okně jsou parametry rozděleny do dvou sekcí. V první sekci, *Nastavení genetického algoritmu*, jsou umístěny parametry:

- *počet generací* – maximální počet generací vytvořených za běhu genetického algoritmu, pokud nebude optimálního řešení dosaženo dříve (vzhledem k povaze problému klasifikace většiny bude vždy vytvořen uvedený počet generací, nezastaví-li uživatel běh algoritmu předčasně),
- *velikost počáteční populace* – počet jedinců v jedné generaci. Pokud je zaškrtnuto pole *decimace*, bude počet uvedený zde vynásoben deseti a po vytvoření desáté generace bude počet jedinců snížen na uvedený počet,
- *maximální inicializační hloubka stromu* – v inicializační populaci budou stromy vytvořeny tak, aby jejich hloubka nepřekročila uvedenou hodnotu,
- *maximální hloubka stromu po křížení* – udává hranici, které mohou dosáhnout stromy vznikající po křížení; uzly na rodičovských stromech jsou vybírány tak, aby po zkřížení nedošlo ke vzniku stromů s hloubkou překračující tuto hodnotu,
- *pravděpodobnost křížení v uzlu funkce* – hodnota v procentech určující, nakolik bude při křížení upřednostňován výběr uzlu funkce oproti uzlu terminálu,
- *pravděpodobnost reprodukce* – podíl genetického operátoru reprodukce na tvorbě nové generace (v součtu s pravděpodobností křížení a mutace musí vycházet 1),
- *pravděpodobnost křížení* – podíl genetického operátoru křížení na tvorbě každé nové generace, měl by být vyšší než reprodukce a mutace,
- *pravděpodobnost mutace* – podíl mutace na tvorbě nové generace.

Zaškrtávací políčko *Zachovávat nejzdatnějšího jedince* zapíná funkci *elitismu*, která zajišťuje, že nejlepší dosud nalezené řešení bude beze změny automaticky zkopírováno do příští generace. Navíc pro toto řešení vytváří náhodný počet (1 – 5) mutací na terminálových uzlech a přidává je do následující generace.

Inicializační metoda nabízí tři možnosti – *grow*, *full* a *ramped*. Jedná se o způsob, jakým jsou tvořeny stromy v první generaci. Metoda *grow* vytváří stromy tak, že prvek z množiny terminálů může být přidán se stejnou pravděpodobností jako prvek z množiny funkcí, nejpozději však v hloubce stanovené parametrem *maximální inicializační hloubka stromu.* Naproti tomu stromy vytvořené metodou *full* mají větve utvo-

řené až do maximální povolené hloubky, protože v předchozích vrstvách hloubky tato funkce přidává pouze prvky z množiny funkcí a terminály jsou připojeny až v největší povolené hloubce. Metoda *ramped* vytvoří polovinu jedinců v první generaci pomocí metody *grow* a druhou polovinu pomocí metody *full*.

Sekundární operaci lze zahrnout do genetického algoritmu zaškrtnutím příslušného políčka. *Decimace* desetkrát zvýší počet jedinců v inicializační populaci a před tvorbou jedenácté generace ponechá v populaci pouze desetinu jedinců s nejvyšší zdatností.

Sekce *Nastavení celulárních automatů* obsahuje pět následujících volitelných parametrů:

- *počet automatů v řadě* – udává počet buněk v mřížce v každém řádku,
- *počet řad* – počet buněk v mřížce v každém sloupci, jinými slovy kolikrát bude vypočítán další krok,
- *velikost okolí* – též velikost *sousedství*; udává počet sousedících buněk, které ovlivňují výslednou barvu prostřední buňky v příštím kroku,
- *počet konfigurací za běhu algoritmu* – v průběhu genetického algoritmu bude každé nalezené řešení ohodnoceno otestováním na zadaném počtu náhodných konfigurací; vyšší číslo sníží rychlost výpočtu, ale poskytne přesnější výsledky,
- *počet konfigurací pro určení výsledku* – po ukončení genetického algoritmu je nejlepší nalezené řešení (výsledné řešení) znovu otestováno, a to na zadaném počtu náhodných konfigurací. Výsledkem je jeho procentuálně vyjádřená úspěšnost.

C. Dokumentace k programu

Aplikace *Celulární automaty* je vytvořena v programovacím jazyce Java. Její kód je pro lepší přehlednost rozmístěn do čtyř balíčků – *aplikace, genetika, model* a *rozhrani*. Balíček *aplikace* obsahuje pouze třídu pro spuštění celé aplikace. Balíček *rozhrani* je souborem tříd odpovědných za zobrazení a obsluhu jednotlivých oken (*OknoAplikace* obsluhuje hlavní okno aplikace, *OknoPravidlo* zobrazuje okno pro práci s uživatelskými pravidly). Také se zde nachází soubor *Mrizka.java*, který vykresluje prostor celulárních automatů.

Balíček *model* obsahuje především třídy související s prací s celulárními automaty. Pomocná třída *Bunka* reprezentuje jednotlivou buňku (automat) a ukládá infor-

mace o její pozici a barvě. Třída *Konfigurace* uchovává posloupnost celulárních automatů v jednom kroku, tzn. právě jeden z řádků v mřížce. Tato posloupnost je ukládána jako pole hodnot *boolean*. Kromě metod pro nastavení hodnot jednotlivých buněk a jejich čtení obsahuje také metodu pro vygenerování nové, náhodné posloupnosti, která současnou posloupnost nahradí posloupností *boolean* hodnot, kde prvek 0 i 1 je pro každou pozici vybírán s 50% pravděpodobností. Třída *Pravidlo* představuje konkrétní pravidlo a ukládá se do ní velikost okolí, tabulka okolí se všemi kombinacemi polí v okolí vzhledem k jeho velikosti (jako *java.util.List* polí hodnot typu *boolean*; tyto kombinace dokáže třída *Pravidlo* také sama vygenerovat) a tabulka výsledků (jako *java.util.List* hodnot typu *boolean*). Poslední důležitou třídou je *PocitacCA*, který slouží ke zpracování předložené konfigurace (řady) CA. Jeho metody umožňují zjistit barvu řady, počet bílých polí v řadě nebo zjistit převládající barvu; nejdůležitější metodou však je výpočet další řady na základě obdrženého pravidla a konfigurace.

Genetika je balíčkem, který shrnuje všechny třídy týkající se genetického programování. Základní třídou pro reprezentaci stromových uzlů je *Gen*, obsahující údaje jako arita, hloubka (vrstva, v níž se tento uzel nachází ve stromovém grafu), textový příkaz a další. Gen má rekurzivní metodu pro výpis celého podstromu genů počínajícího aktuálním genem nebo například metodu pro převod podstromu na objekt třídy *Pravidlo* (tato metoda se používá u kořene stromu). Z této třídy dědí *Terminal* a *Funkce*. Zatímco *Terminal* pouze upřesňuje údaje třídy Gen (nastavuje nulovou aritu apod.), *Funkce* obsahuje konstruktor vedoucí k rekurzivní tvorbě náhodného stromu pomocí metod genetického programování zvaných *grow*, *full* a *ramped*. Další třída, *Genotyp*, slouží pro uchovávání konkrétního jedince v populaci. Obsahuje odkaz na kořen stromu genů tvořících jedince a číselný údaj o zdatnosti tohoto jedince. Nejdůležitější třídou v balíčku je pak *GenetickyAlgoritmus*, která shrnuje celý průběh genetického algoritmu - vytvoření inicializační populace, provedení decimace, ohodnocení jedinců, modifikovaný výběr podle pořadí pro selekci jedinců, provedení reprodukce, křížení a mutace a vytvoření dalších generací.

Podklad pro zadání DIPLOMOVÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Bc. Dibitanzlová Iva	Na Nivách 642, Trhové Sviny	I14282

TÉMA ČESKY:

Genetické programování a celulární automaty

TÉMA ANGLICKY:

Genetic Programming and Cellular Automata

VEDOUCÍ PRÁCE:

prof. RNDr. Josef Hynek, Ph.D., MBA - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl práce:
Cílem práce je shrnutí principů genetického programování, popsání celulárních automatů a následné využití získaných poznatků k řešení problému klasifikace většiny celulárních automatů za použití genetického programování. Za tímto účelem bude v programovacím jazyce Java implementován genetický algoritmus pracující s celulárními automaty. Úspěšnost výsledných pravidel bude porovnána s úspěšností pravidel nalezených v předchozích pracích.

Osnova:
1) Úvod
2) Genetické programování
3) Celulární automaty
4) Problém klasifikace většiny
5) Implementace
6) Shrnutí výsledků
7) Závěry a doporučení

SEZNAM DOPORUČENÉ LITERATURY:

ALESHUNAS, John. Building Block Emergence in Genetic Programming. 2008.

ANDRE, David, BENNET, Forrest H., KOZA, John R. Discovery by Genetic Programming of a Cellular Automata Rule that is Better than any Known Rule for the Majority Classification Problem. In Proceedings of the First Annual Conference on Genetic Programming. MIT Press, 1996.

ANDRE, David; BENNETT III, Forrest H.; KOZA, John R. Evolution of Intricate Long-Distance Communication Signals in Cellular Automata Using Genetic Programming. In: Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems. MIT Press, 1996.

ANGELINE, Peter J.; KINNEAR, Kenneth E. (ed.). Advances in Genetic Programming. Cambridge: MIT Press, 1996.

DELORME, M., MAZOYER., J., OLLINGER, N., THEYSSIER, G. Classifications of Cellular Automata. Université de Savoie, 2010.

DENNUNZIO, Alberto, FORMENTI, Enrico, PROVILLARD, Julien. Non-Uniform Cellular Automata: Classes, Dynamics, And Decidability. Université Nice-Sophia Antipolis, 2000.

EBERHART, Russell C.; SHI, Yuhui. Computational Intelligence: Concepts to Implementations. Elsevier, 2011.

EKÁRT, Anikó. Emergence in Genetic Programming. Genetic Programming and Evolvable Machines, 2014.

ENGELBRECHT, Andries P. Computational Intelligence: An Introduction. John Wiley & Sons, 2007.

FERREIRA, Candida. Gene Expression Programming: A New Adaptive Algorithm for Solving Problems. I Complex Systems, 2001.

HYNEK, Josef. Genetické algoritmy a genetické programování. Grada Publishing a.s., 2008.

KOZA, John R., BENNETT III, Forrest H., ANDRE, David, KEANE, Martin A.. Genetic Programming III : Darwinian Invention and Problem Solving. San Francisco: Morgan Kaufmann Publishers, Inc., 1999.

LUKE, Sean; SPECTOR, Lee. Evolving Teamwork and Coordination with Genetic Programming. In: Proceedings of the 1st Annual Conference on Genetic Programming. MIT Press, 1996.

MITCHEL, Melanie, CRUTCHFIELD, James P., DAS, Rajarshi. Evolving Cellular Automata with Genetic Algorithms: A Review of Recent Work. In Proceedings of the First International Conference on Evolutionary Computation adn Its Applications, 1996.

MITCHEL, Melanie. Computation in Cellular Automata: A Selected Review. In Nonstandard Computation, 1998.

MITCHELL, Melanie; CRUTCHFIELD, James P.; HRABER, Peter T. Evolving Cellular Automata to Perform Computations: Mechanisms and Impediments. Physica D: Nonlinear Phenomena, 1994.

MITCHELL, Melanie; HRABER, Peter; CRUTCHFIELD, James P. Revisiting the edge of chaos: Evolving Cellular Automata to Perform Computations. In Complex Sytems, 1993.

MUNAKATA, Toshinori. Fundamentals of the New Artificial Intelligence. Heidelberg: Springer, 1998.

O'NEILL, Michael. Open Issues in Genetic Programming. Genetic Programming and Evolvable Machines, 2010.

SCHIFF, Joel L. Cellular Automata: A Discrete View of the Word. John Wiley & Sons, 2011.

SPECTOR, Lee. Automatic Quantum Computer Programming: A Genetic Programming Approach. Springer Science & Business Media, 2004.

WOLFRAM, Stephen. A New Kind of Science. Wolfram Media, 2002.

Podpis studenta: ..... **Datum:** .....

Podpis vedoucího práce: ..... **Datum:** .....