VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

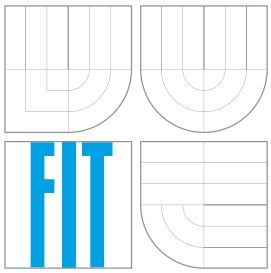
INSTRUKCEMI ŘÍZENÉ CELULÁRNÍ AUTOMATY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE JAROSLAV BENDL
AUTHOR

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

INSTRUKCEMI ŘÍZENÉ CELULÁRNÍ AUTOMATY

INSTRUCTION-CONTROLLED CELLULAR AUTOMATA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — JAROSLAV BENDL
AUTHOR

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

BRNO 2011

Abstrakt

Tato práce se zabývá návrhem nového konceptu řízení celulárního automatu založeného na tzv. instrukcích. Instrukci lze chápat jako určité pravidlo ověřující stavy předem definované skupiny buněk v sousedství vyšetřované buňky, přičemž při splnění stanovené podmínky kladené na danou skupinu je její stav změněn dle daného předpisu. Jelikož je možné v rámci jednoho výpočetního kroku uvažovat sekvenci složenou z více instrukcí, přičemž každá instrukce může změnit stav centrální buňky ihned po své aplikaci, lze jejich posloupnost pokládat za určitou formu krátkého programu. Tento koncept je zároveň možné rozšířit o jednoduché operace aplikované na buněčné okolí a prováděné během interpretace jednotlivých instrukcí – příkladem takové operace může být řádkový nebo sloupcový posun. Výhoda použití instrukcí tkví v redukci vyhledávacího prostoru, neboť oproti obvykle používané tabulkové metodě není nutné prohledávat množinu všech možných konfigurací buněk v okolí, nýbrž pouze několik oblastí vymezených předpisy instrukcí. Zatímco skupiny vyšetřovaných buněk v rámci instrukce jsou navrhovány ručně na základě analýzy řešené úlohy, posloupnost jejich umístění v chromozomu je optimalizována prostřednictvím genetického algoritmu. Úspěšnost navržené metody řízení celulárního automatu je zkoumána na vybraných benchmarkových úlohách – majoritě, synchronizace, samoorganizaci a návrhu kombinačních logických obvodů.

Abstract

The thesis focuses on a new concept of cellular automata control based on instructions. The instruction can be understood as a rule that checks the states of cells in pre-defined areas in the cellular neighbourhood. If a given condition is satisfied, the state of the central cell is changed according to the definition of the instruction. Because it's possible to perform more instructions in one computational step, their sequence can be understood as a form of a short program. This concept can be extended with simple operations applied to the instruction's prescription during interpretation of the instructions – an example of such operation can be row shift or column shift. An advantage of the instruction-based approach lies in the search space reduction. In comparison with the table-based approach, it isn't necessary to search all the possible configurations of the cellular neighbouhood, but only several areas determined by the instructions. While the groups of the inspected cells in the cellular neighbourhood are designed manually on the basis of the analysis of the solved task, their sequence in the chromosome is optimized by genetic algorithm. The capability of the proposed method of cellular automata control is studied on these benchmark tasks - majority, synchronization, self-organization and the design of combinational circuits.

Klíčová slova

Celulární automat, evoluční algoritmus, genetický algoritmus, problém majority, problém synchronizace, problém samoorganizace.

Keywords

Cellular automata, evolutionary algorithm, genetic algorithm, majority task, synchronization task, self-organization task.

Citace

Jaroslav Bendl: Instrukcemi řízené celulární automaty, diplomová práce, Brno, FIT VUT v Brně, 2011

Instrukcemi řízené celulární automaty

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Michala Bidla, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jaroslav Bendl
23. července 2011

Poděkování

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

Obsah

Kapitola 1

Úvod

Tato práce se zabývá návrhem nového konceptu řízení celulárního automatu založeného na tzv. instrukcích. Instrukci lze chápat jako určité pravidlo ověřující stavy předem definované skupiny buněk v sousedství vyšetřované buňky, přičemž při splnění stanovené podmínky kladené na danou skupinu je její stav změněn dle daného předpisu. Jelikož je možné v rámci jednoho výpočetního kroku uvažovat sekvenci složenou z více instrukcí, přičemž každá instrukce může změnit stav centrální buňky ihned po své aplikaci, lze jejich posloupnost pokládat za určitou formu krátkého programu. Tento koncept je zároveň možné rozšířit o jednoduché operace aplikované na buněčné okolí a prováděné během interpretace jednotlivých instrukcí – příkladem takové operace může být řádkový nebo sloupcový posun. Výhoda použití instrukcí tkví v redukci vyhledávacího prostoru, neboť oproti obvykle používané tabulkové metodě není nutné prohledávat množinu všech možných konfigurací buněk v okolí, nýbrž pouze několik oblastí vymezených předpisy instrukcí. Zatímco skupiny vyšetřovaných buněk jsou navrhovány ručně na základě analýzy řešené úlohy, posloupnost jejich umístění v rámci chromozomu je optimalizována prostřednictvím genetického algoritmu.

Text práce je rozdělen následovně: Kapitola 2 se zabývá charakteristikou evolučních výpočetních technik, přičemž detailněji se věnuje konceptu genetického algoritmu. Kapitola 3 vysvětluje principy celulárních automatů a popisuje známé benchmarkové úlohy sloužící k ohodnocení jejich schopností. U těchto úloh je uvedena nejenom formální definice, ale rovněž shrnutí doposud známých metod řešení včetně jejich výsledků. Kapitola 4 je zaměřena na vysvětlení konceptu instrukcemi řízeného celulárního automatu, přičemž výsledky experimentování s tímto modelem jsou popsány v kapitole 5. Kapitola 6 obsahuje závěr a shrnutí celé práce.

Kapitola 2

Evoluční algoritmy

Evoluční algoritmy jsou souhrnným názvem pro třídu netradičních výpočetních postupů hledajících inspiraci v přírodě. Podobně jako u přírodních jevů mají tyto techniky významnou náhodnou složku ovlivňující jejich běh a vlastnosti. Jejich hlavní myšlenka je založena na Darwinově teorii přirozeného výběru implikující přežití nejsilnějších jedinců.

První pokusy o aplikaci metod inspirovaných evolucí se objevily již v první polovině šedesátých let. Tři studenti univerzity v Berlíně zabývající se konstrukcí převodovky, přišli s myšlenkou náhodné kombinace dvojice existujících konstrukcí ve snaze nalézt konstrukci s lepšími užitnými vlastnostmi, což beze sporu připomíná přírodní křížení v rámci téhož biologického druhu [15]. Významnějšího rozvoje se ale evoluční techniky dočkaly až po přelomové publikaci *Adaptation in Natural and Artificial Systems* teoretického biologa Johna Hollanda z roku 1975 [9]. Ve zmíněné práci jsou studovány evoluční principy známé z přírody, kdy jedinci populace navzájem soupeří o zdroje k přežití a usilují o sebe-reprodukci. Délka života a počet potomků přitom odpovídá schopnostem každého individua. Jelikož vlastnosti jedinců jsou součástí jejich genomu předávaného s jistými modifikacemi vlastním potomkům, a protože schopnější jedinci mají zpravidla více potomků, potom lze předpokládat, že průměrná schopnost jedinců populace v dalších generacích se bude zlepšovat. Tento koncept, sledovaný již Darwinem v živé přírodě, Holland převzal a na jeho základě definoval obecnou podobu genetického algoritmu. Podrobnějšímu studiu této techniky z hlediska její aplikace na praktické problémy se později věnoval David Goldberg, který výsledky své práce shrnul v další slavné publikaci *Genetic Algorithms in Search, Optimization and Machine Learning* z roku 1989 [7].

Do třídy evolučních výpočetních technik patří vedle genetického algoritmu také genetické programování, evoluční programování a evoluční strategie. Tyto techniky, z nichž první uvedená bude v rámci této kapitoly podrobně popsána, se vzájemně liší pouze v implementačních detailech a rozdílném způsobu reprezentace kandidátního řešení úlohy, což jim umožňuje specializovat se na odlišné kategorie problémů.

2.1 Základní charakteristiky evolučních algoritmů

Jak již bylo naznačeno v úvodu kapitoly, evoluční výpočetní techniky jsou založeny na Darwinově teorii přirozeného výběru upřednostňující k reprodukci schopnější jedince. Tento princip lze dobře vystihnout následujícím citátem [5]: “V přírodní evoluci je základní úlohou biologického druhu vyhledávání výhodných adaptací vůči složitému a dynamicky se měnícímu prostředí. ‘Znalost’, která charakterizuje každý biologický druh, byla získána vývojem

a je shrnuta v chromozomech každého jedince.” Právě existence jisté ‘znalosti’ přitom odlišuje třídu evolučních algoritmů od slepého stochastického algoritmu, v němž prohledávání stavového prostoru není nijak vázáno ke kvalitě řešení.

Evoluční výpočetní techniky pracují podle schématu uvedeném na obrázku 2.1. Vývoj množiny nezávislých kandidátních řešení, označovaných jako *populace*, probíhá v diskrétních časových krocích zvaných *generace*. Na začátku celého procesu je vytvořena počáteční populace sestávající obvykle z náhodně vygenerovaných jedinců. Těmto jedincům je na základě jejich schopnosti řešit zadanou úlohu přiřazena *fitness hodnota* prostřednictvím *fitness funkce*. S využitím znalosti míry úspěšnosti kandidátních řešení je pak v jednotlivých iteracích evolučního cyklu prováděna *aktualizace* populace. Během aktualizace dochází nejprve k vytvoření nových kandidátních řešení a posléze k jejich následnému sloučení s jedinci původní populace. K vytvoření nových kandidátních řešení se používají *genetické operátory* inspirované rekombinačními procesy známými z přírody. Pro zachování stejného počtu jedinců populace je sloučená množina nových a původních jedinců pomocí vhodné *vývojové strategie* redukována, čímž je napodobováno působení *selekčního tlaku* směřujícího evoluci k zisku kvalitnějších řešení.

Základní typy vývojových strategií jsou (převzato z [15]):

- *generační* (generational evolution), při které je původní populace jedinců zcela nahrazena populací potomků,
- *postupná* (steady state evolution), při které je původní populace jedinců nahrazena populací potomků jen z určité části.

Konkrétní implementace procesu tvorby nových potomků a redukce spojené množiny potomků a rodičů pak závisí na vybrané evoluční technice a zvolené strategii.

```
čas t = 0;
vytvoř počáteční populaci P(t);
vypočti úspěšnost prvků v P(t);
while(nesplněná ukončující podmínka)
{
    t = t + 1;
    P(t) = vytvoř novou populaci z P(t-1);
    vyhodnoť úspěšnost prvků v P(t);
}
```

Obrázek 2.1: Pseudokód evolučního algoritmu. Převzato z [15].

2.2 Genetický algoritmus

Genetický algoritmus je v praxi nejčastěji používanou metodou ze třídy evolučních výpočetních technik. Počátky vývoje této metody sahají až do šedesátých let 20. století a byly podrobně popsány již v úvodu této kapitoly.

2.2.1 Základní charakteristiky genetického algoritmu

Jedinci populace kandidátních řešení daného problému jsou v rámci genetického algoritmu reprezentováni strukturou ve formě řetězců konečné délky, v analogii s obecnou genetikou nazývaných *chromozomy* [15]. Každý chromozom je tvořen sekvencí symbolů, přičemž symboly nacházející se na konkrétních pozicích řetězce se nazývají *geny* a pochází ze stejné abecedy.

Obecný tvar genetického algoritmu, platný pro všechny jeho varianty, je zobrazen na obrázku 2.2. Popis jednotlivých částí tohoto algoritmu bude předmětem následujících dvou sekcí.

```
čas t = 0;
inicializuj chromozomy v P(t) náhodnými alelami;
do {
    vypočti fitness jedinců v P(t);
    vyber jedince do prostoru páření;
    t=t+1;
    do {
        náhodně vyber dva rodiče z prostoru páření;
        pomocí operátoru křížení vytvoř dva potomky;
        aplikuj náhodnou mutaci na potomky;
        utvoř P(t) nahrazením jedinců v P(t-1);
    } until (P(t) je naplněna novými potomky);
} until(splněna ukončující podmínka);
```

Obrázek 2.2: Pseudokód genetického algoritmu. Převzato z [15].

2.2.2 Počáteční populace a operátor selekce

Před vytvořením počáteční populace je nejprve nutné zvolit způsob, jakým budou jednotlivá individua zakódována, tedy jakých hodnot budou moci nabývat jednotlivé geny. Nejčastější je přitom binární kódování, při kterém jsou chromozomy tvořeny dvojkovým řetězcem dané délky. Na základě této znalosti je pak možné začít generovat náhodné chromozomy do počáteční populace, což je typický přístup tvorby jedinců první generace. Ačkoliv většina takto vygenerovaných jedinců nebude v řešení daného problému příliš úspěšná, při volbě dostatečně velké populace je dost pravděpodobné, že se v ní objeví několik alespoň částečně použitelných jedinců. Těmto jedincům je přiřazena větší fitness hodnota než zbytku populace, v důsledku čehož potom mají větší šanci zúčastnit se reprodukčního procesu a šířit svůj genom do dalších generací.

V každém evolučním kroku následuje po etapě tvorby nových potomků proces selekce, při kterém je redukována množina potomků a množina jejich rodičů tak, aby byla zachována původní velikost populace. Přestože má selekce zvýhodňovat úspěšnější jedince před horšími, je vhodné do nové populace zařazovat i určitou část hůře ohodnocených jedinců. Jejich přítomnost dokáže udržovat genovou variaci a zabránit předčasné konvergenci. Nejčastěji se k tomuto účelu využívá *ruletová selekce* nebo *turnajová selekce*.

Ruletová selekce (fitness-proportionate selection) určuje pravděpodobnost výběru jedince na základě jejich fitness hodnoty. V analogii s ruletou má každý jedinec přiřazenou

takovou výseč kola, která odpovídá jeho úspěšnosti. Ačkoliv tedy mají horší jedinci menší výseč, přesto existuje možnost, že budou do nové populace vybráni.

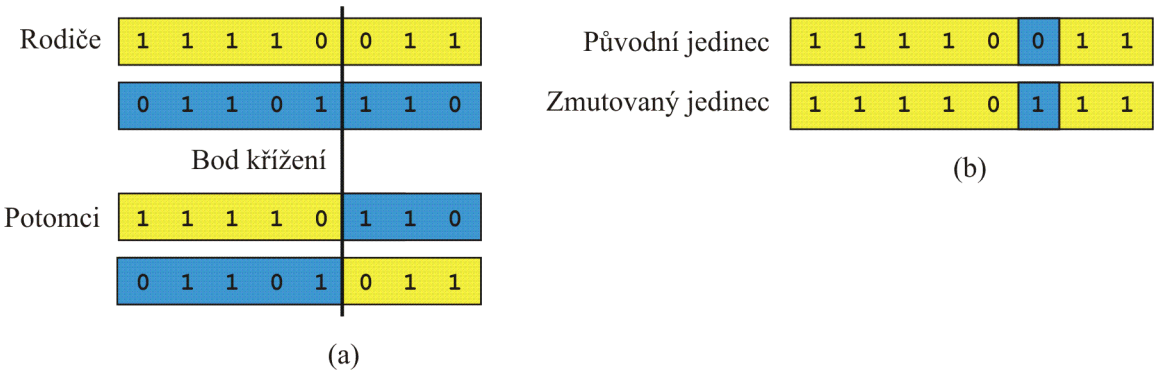
Turnajová selekce (tournament selection) určí náhodně skupinu k ($k \geq 2$) jedinců, z nichž do nové populace vybere jedince s nejvyšší fitness hodnotou. Díky náhodnému výběru jedinců se občas "utkají" dva relativně slabší soupeři, takže i tato metoda připouští do nové populace slabší individua.

2.2.3 Genetické operátory

Nová kandidátní řešení jsou generována prostřednictvím genetických operátorů aplikovaných na původní členy populace. Nejčastěji přitom bývají využity operátory *křížení* a *mutace* (viz obr. 2.3).

Křížení slouží k vytvoření nových kandidátních řešení s genetickou informací pocházející z kombinace chromozomů dvou jedinců původní populace. Obvyklá implementace tohoto operátoru spočívá ve volbě náhodné pozice v chromozomu a následné záměně zbývajících částí chromozomů vybraných rodičů. Dalšími variantami implementace je vícebodové a uniformní křížení. Vícebodové křížení je v podstatě totožné s prezentovaným jednobodovým křížením, s tím rozdílem, že je vybráno více pozic, na kterých dochází k záměnám. Uniformní křížení vytváří potomka náhodnou volbou rodiče pro každý jednotlivý gen potomka. Tento operátor se aplikuje na vybranou dvojici rodičů obvykle jen s určitou pravděpodobností, přičemž tato pravděpodobnost je typicky relativně vysoká (podle [22] se volí nejčastěji $p_c \in < 0.75, 0.95 >$).

Mutace slouží k rozšíření prohledávaného prostoru o řešení, která nelze získat křížením a zabraňuje tak předčasné konvergenci. Obvyklá implementace tohoto operátoru spočívá v náhodné změně hodnoty genu, přičemž této operaci se podrobí každý gen s určitou pravděpodobností, která je typicky velice malá (podle [22] se volí nejčastěji $p_m \in < 0.001, 0.05 >$).



Obrázek 2.3: Genetické operátory: (a) jednobodové křížení, (b) mutace.

Výše popsané metody křížení a mutace představují základní varianty implementace těchto genetických operátorů. Kromě nich však existuje celá řada dalších problémově-specifických variant, které se vzájemně liší paměťovou i časovou náročností. Na základě analýzy řešené úlohy lze mnohdy vybrat ty z nich, které pro daný problém poskytují nejlepší výsledky. Pokud takovou analýzu nelze provést, často se přistupuje k výběru větší množiny genetických operátorů, z nichž se pak v jednotlivých iteracích evolučního cyklu volí konkrétní varianta náhodně. Střídání jednotlivých variant implementace zajišťuje nejednotvárnou obměnu genetického materiálu jedinců a brání tak předčasné konvergenci způsobené nedostatečnou rozmanitostí populace. Tento přístup je použitý rovněž v rámci této

práce, přičemž konkrétní používané varianty genetických operátorů jsou uvedeny v textu níže a znázorněny na obrázcích 2.4 a 2.5.

Operátor křížení s uspořádáním

Operátor křížení s uspořádáním (OX – order crossover) je aplikován na dvojici rodičů a vzniká jeden potomek. Princip lze popsat takto [10]:

1. Náhodně jsou vygenerovány dva body křížení.
2. Geny mezi body křížení prvního rodičovského chromozomu jsou zkopírovány na korespondující pozice v chromozomu potomka.
3. Postupně je od pozice druhého bodu křížení procházen chromozom druhého rodiče a pokud se aktuální gen dosud nenachází v chromozomu potomka, je k němu tento gen zkopírován. V opačném případě se tento gen přeskočí.

Operátor křížení s částečným přiřazením

Operátor křížení s částečným přiřazením (PMX – partial matched crossover) je aplikován na dvojici rodičů a vzniká jeden potomek. Princip lze popsat takto [10]:

1. Náhodně jsou vygenerovány dva body křížení.
2. Geny mezi body křížení prvního rodičovského chromozomu jsou zkopírovány na korespondující pozice v chromozomu potomka.
3. Postupně je procházen úsek prvního rodičovského chromozomu mezi body křížení.
   (a) Aktuální gen je spojen s protilehlým genem druhého rodičovského chromozomu a zároveň se stejnou hodnotou v druhém rodičovském chromozomu.
   (b) Pakliže nemá protilehlý gen stejnou hodnotu jako aktuální, jsou oba geny zkopírovány do pomocného chromozomu potomka a jsou prohozeny jejich pozice.
4. Pomocný chromozom je mimo úsek vymezený body křížení zkopírován do potomka.
5. Zbývající volné pozice v chromozomu potomka jsou obsazeny hodnotami z odpovídajících pozic z druhého rodičovského chromozomu.

Konzervativní operátor křížení

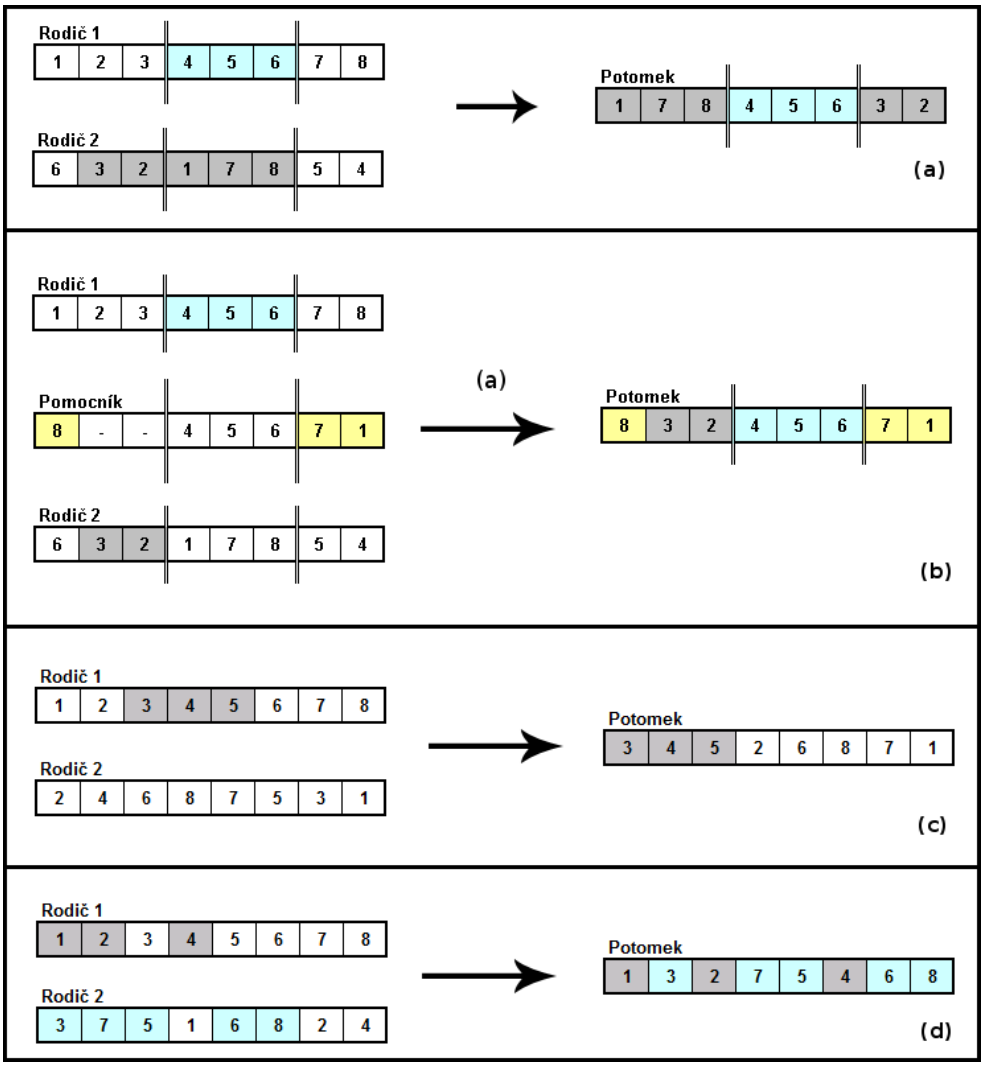
Konzervativní operátor křížení (MPX – maximal preservative crossover) je aplikován na dvojici rodičů a vzniká jeden potomek. Princip lze popsat takto [14]:

1. V chromozomu prvního rodiče je náhodně vybrána sekvence genů.
2. Zvolená sekvence genů je vložena na začátek chromozomu potomka.
3. Zbývající geny v chromozomu potomka jsou nastaveny podle genů druhého rodiče (ten se prochází zleva doprava), přičemž jsou vynechávány ty hodnoty, které již chromozom potomka obsahuje.

Křížení se střídáním pozic

Operátor křížení se střídáním pozic (AP – alternative position crossover) je aplikován na dvojici rodičů a vzniká jeden potomek. Princip lze popsat takto [14]:

1. První gen prvního rodiče se vloží do chromozomu potomka.
2. První gen druhého rodiče (nemá-li stejnou hodnotu jako první gen prvního rodiče) se vloží do chromozomu potomka.
3. Dále se postupuje stejným způsobem jako v předcházejících dvou bodech až do naplnění všech volných pozic v chromozomu, tj. střídají se ve vkládání genů oba rodiče, přičemž je kontrolováno, zda-li se již uvažovaná hodnota v chromozomu potomka nenachází (pokud ano, gen s danou hodnotou není vložen).



Obrázek 2.4: Ilustrace vzniku kandidátního řešení pomocí genetického operátoru křížení typu: (a) OX, (b) PMX, (c) MPX, (d) AP.

Mutace Swap

Operátor mutace Swap [14] spočívá v jednoduché záměně dvou náhodně vybraných genů chromozomu.

Nahrazující mutace

Operátor nahrazující mutace (DM – displacement mutation) [14] spočívá ve výběru náhodné cesty v daném jedinci, vyjmutí této cesty a její umístění na jinou, náhodně zvolenou pozici.

Vkládací mutace

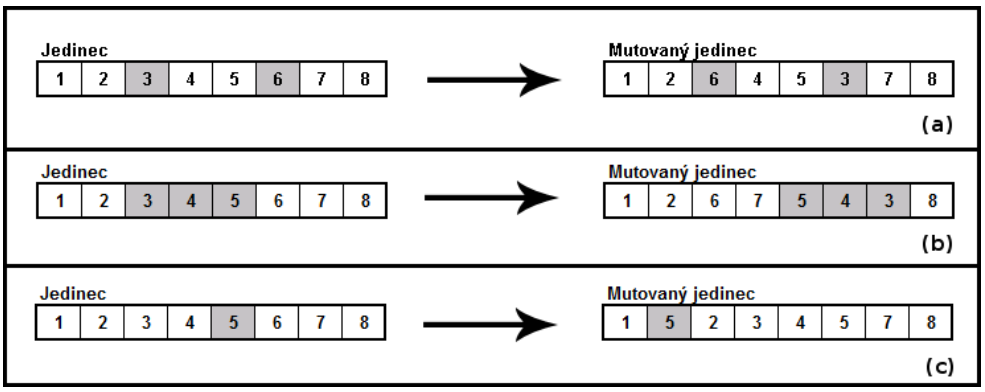
Operátor vkládací mutace (ISM – insertion mutation) [14] spočívá ve výběru náhodného genu, vyjmutí tohoto genu a jeho umístění na jinou, náhodně zvolenou pozici.

Inverzní mutace

Operátor inverzní mutace (IVM – inversion mutation) [14] spočívá ve výběru náhodné cesty v daném jedinci, vyjmutí této cesty, inverzi této cesty a umístěním na jinou, náhodně zvolenou pozici.

Mixující mutace

Operátor mixující mutace (SM – scramble mutation) [14] spočívá ve výběru náhodné cesty v daném jedinci a libovolné záměně hodnot genů této sekvence.



Obrázek 2.5: Ilustrace vzniku kandidátního řešení pomocí genetického operátoru mutace typu: (a) Swap, (b) IVM, (c) DM.

Kapitola 3

Celulární automaty

Celulární automaty jsou souhrnným názvem pro třídu dynamických modelů sloužících k časo-prostorové idealizaci fyzikálních systémů, v nichž stavy sledovaných veličin nabývají pouze diskrétních hodnot. Tento model nachází využití v řadě oborů, zejména však v teorii systémů, matematice a teoretické biologii.

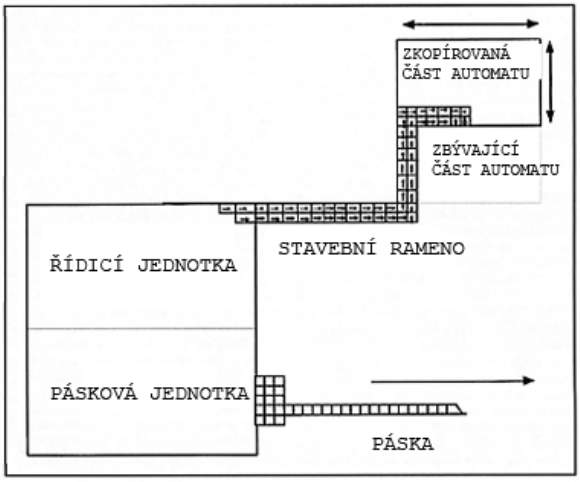
3.1 Historie celulárních automatů

Historie celulárních automatů sahá až do 40. let 20. století. V té době pracoval John von Neumann na konceptu tzv. *kinematického modelu* zabývajícího se srovnáním komplexity počítačů s biologickými systémy (zejména pak mozkem) a směřujícího k vytvoření stroje schopného vlastní replikace. První koncept kinematického modelu vycházel z představy běžného počítače doplněného o několik dalších částí (konkrétně manipulátoru, oddělovače, spojovače, senzoru a nosníků), který je schopen v prostředí náhodně rozložených součástek tyto součástky shromažďovat a montovat, a tímto způsobem vytvářet svoji kopii, kterou na konci tohoto procesu oživí zkopírováním své paměti [15]. Právě představa množícího se robota plujícího v moři součástek spojuje zdánlivě rozcházející se světy nervové tkáně a elektronek prostřednictvím matematické logiky [8]. Kromě samotné sebe-replikace von Neumann připouštěl možnost výskytu drobných chyb (mutací), což by umožňovalo vývoj systému ke komplexnějším strukturám. Tento model byl poprvé představen roku 1948 v přednášce "Obecná teorie automatů" [19] a vycházel z pojmů vypočítatelných čísel a univerzálních počítacích strojů Allana Touringa a dále pak z prací Warrena McCullocha a Waltera Pittse zabývajících se problematikou neuronových sítí. Von Neumann rozšířil Turingovu práci tím, že ukázal, že teoreticky existuje univerzální automat, čili stroj s jistou definovanou mírou komplexity, který bude dělat vše, co může dělat jakýkoli jiný stroj, je-li vybaven správnými instrukcemi [8]. Nad takto definovanou mírou komplexity již pro dosažení složitějšího chování není nutné stroj zesložiťovat, nýbrž mu pouze dát složitější instrukce.

Navržený koncept kinematického stroje sice detailně popisuje funkci jednotlivých součástek, problém však zůstává s jejich vnitřní stavbou (například jak by měl být konkrétně realizován senzor či manipulátor). Řešení tohoto *black-box problému* navrhl von Neumannovi jeho přítel a kolega Stanilaw Ulam [25], který přišel s myšlenkou nahradit "plovoucího robota" "mozaikovým robotem". Koncept "mozaikového robota" nachází inspiraci v růstu krystalu, jenž lze připodobnit ke skládání dílčích bloků do jakési mozaikové struktury. Tento koncept v podstatě představuje celulární automat, který je založen právě na

použití pravidelné prostorové struktury (mozaiky, mřížky, ...), do níž jsou umístěny jednotlivé buňky provádějící v diskrétních časových krocích svoji činnost podle zadaných pravidel. Vývoj automatu je přitom založen na lokálních pravidlech, takže rozhodnutí o další činnosti buňky závisí pouze na malém počtu sousedních buněk, díky čemuž lze jejich stavy počítat souběžně. Na celulární automaty lze tedy pohlížet jako na paradigma paralelních výpočtů, podobně jako je Turingův stroj paradigmatem pro sériové výpočty [8].

Von Neumannův celulární automat byl poprvé představen v roce 1953. Jedná se o dvourozměrný automat složený z těla 80 x 40 buněk, stavebního ramene a dlouhého výrůstku. Dohromady tato struktura obsahuje přibližně 200 000 buněk, které mohou nabývat jednoho z 29 stavů. Zmíněný výrůstek přitom obsahuje zakódované instrukce pro replikaci a je vlastně analogií pásky Turingova stroje. Proces replikace je pak řízen těmito instrukcemi skrze vysunuté stavební rameno (viz obrázek 3.1). Vytvořená kopie samozřejmě funguje zcela samostatně bez jakékoliv vazby na svého rodiče. Ačkoliv je tento model pro simulaci příliš složitý, dostatečně názorně ukazuje, jaký druh komplexity sebe-replikace vyžaduje.

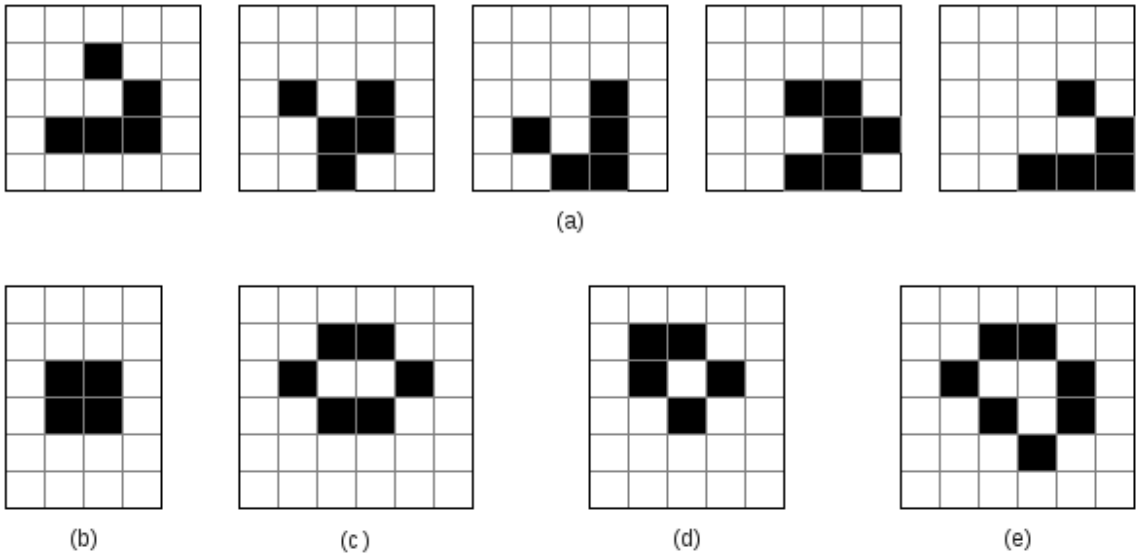


Obrázek 3.1: Ilustrace procesu replikace celulárního automatu podle von Neumanna (převzato z [3]).

Během následujících dvaceti let se v oblasti celulárních automatů mnoho nezměnilo. Tento stav byl zapříčiněn zejména nedostatečnou výpočetní kapacitou tehdejších počítačů a malým zájmem ze strany vědeckých pracovníků. Zlomem byl rok 1970, kdy John Conway sestrojil dvourozměrný binární celulární automat s lokální přechodovou funkcí uvažující všech osm sousedních buněk a tvořenou pravidly umožňující narození, přežití a smrt buňky (za "mrtvou" považoval buňku ve stavu 0, za "živou" pak buňku ve stavu 1). Volba vhodných pravidel je přitom klíčová pro zajištění co nejpestřejší dynamiky vznikajících obrazců a je žádoucí, aby pravidla vylučovala situace, ve kterých obrazce rychle a nekontrolovaně rostou a také situace, ve kterých naopak rychle hynou. Nakonec pravidla pro tento celulární automat nazývaný "Game of Life" definoval takto:

- *narození* - v okolí se vyskytují právě 3 živé buňky,
- *přežití* - v okolí se vyskytují alespoň 2, ale nejvýše 3 živé buňky,
- *smrt* - v okolí se vyskytují alespoň 4 živé buňky.

Obrazce tvořené živými buňkami obvykle po několika generacích buď zcela zanikají, nebo získávají stabilní a neměnnou podobu (viz obrázek 3.2). Další variantou vývinu je cyklicky se opakující obrazec, přičemž nejzajímavější případ nastává tehdy, pokud se obrazec nejen cyklicky opakuje, nýbrž zároveň i posunuje - taková struktura se nazývá *kluzák*. Vznikající obrazce mohou být velmi složité a dokonce se dá ukázat, že tento celulární automat je ekvivalentní Turingovu stroji – jakýkoli výpočet, který se dá provést na jakémkoli myslitelném Turingově počítacím stroji, se dá uskutečnit také pomocí hry Life [8]. Ve snaze o simulaci komplexních jevů byla provedena celá řada dalších experimentů s variantami tohoto automatu, konkrétně například simulace biologického vývoje, šíření požáru, růstu krystalu či vzniku říčních meándrů.



Obrázek 3.2: Ukázka častých struktur hry Life: (a) cyklicky se opakující obrazec s posunem – kluzák, (b) stabilní obrazec – blok, (c) stabilní obrazec – úl, (d) stabilní obrazec – bochník, (e) stabilní obrazec – loď.

Od vzniku hry Life jsou celulární automaty respektovanou oblastí informatiky. Na konci 70. let začala být detailněji zkoumána i teorie celulárních automatů, přičemž o její rozvoj se nejvíce zasloužili zejména Stephen Wolfram a Christopher Langton. Wolfram studoval chování jednorozměrných binárních celulárních automatů a podle složitosti dělil automaty do čtyř tříd jdoucích od stabilního stavu přes periodicitu a komplexní chování až po chaos [26]. Je však nutné zdůraznit, že nejzajímavější třída komplexního chování je v porovnání s velikostí ostatních tříd velmi malá.

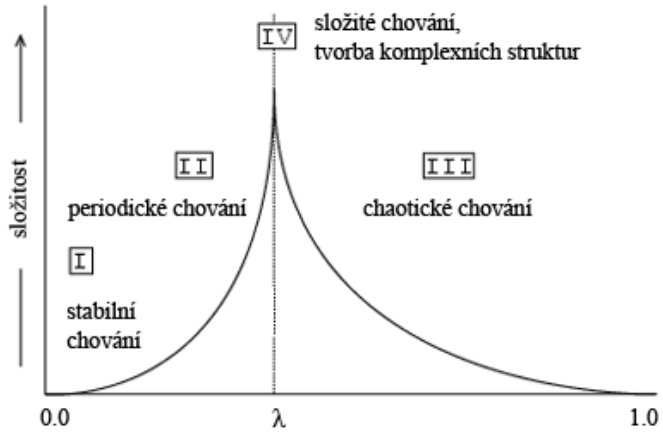
Detailnější kvantifikace se později pokusil dosáhnout Langton tím, že definoval parametr λ (viz obrázek 3.3), pomocí kterého je možné předpovídat chování automatu, přičemž hodnota tohoto parametru se určí následovně [23]:

$$\lambda = \frac{K^N - n}{K^N} \tag{3.1}$$

kde K značí počet možných stavů buňky, N udává počet buněk uvažovaného okolí a n značí počet pravidel lokální přechodové funkce, které vedou k neaktivnímu stavu (za neaktivní stav je označován jeden z možných stavů, nejčastěji stav 0). Je tedy zřejmé, že

λ je určena jako podíl počtu pravidel, jejichž výstupem jsou neklidové stavy k celkovému počtu pravidel [13]. Cílem tohoto dělení je nalézt určitou kritickou mez, při níž je možný přenos informace, ovšem bez ztráty vazby na její původní místo.

Kromě vývoje teorie se rovněž velkého pokroku dosáhlo i v oblasti implementace celulárních automatů, přičemž dnes již dokonce existují i jejich hardwarové realizace. Kupříkladu platforma CAM (cellular automaton machine) umožňuje přibližně tisíckrát rychlejší vývoj automatu, než jakého lze dosáhnout na běžných procesorech osobních počítačů.



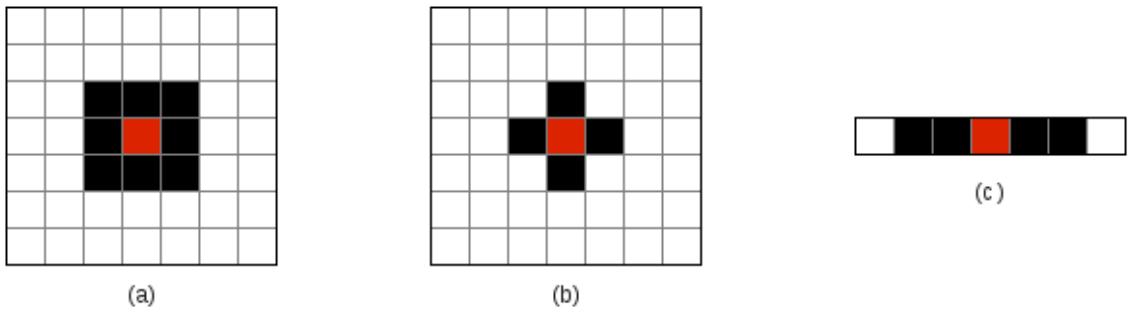
Obrázek 3.3: Schématické znázornění závislosti složitosti chování na parametru λ včetně vyznačení odpovídajících Wolframových tříd. Převzato z [18].

3.2 Základní charakteristiky celulárních automatů

Celulární automaty představují dynamické systémy, diskrétní v hodnotách, prostoru i čase, sestávající z jednotlivých buněk uspořádaných v pravidelné prostorové struktuře. Činnost buněk probíhá paralelně podle lokální přechodové funkce, která určuje stav buňky na základě vlastního stavu a stavů buněk v sousedství. Pravidla tvořící lokální přechodovou funkci mohou být lineární, doplňková nebo neaditivní. Zatímco *lineární pravidla* využívají ke své činnosti pouze operátory součtu modulo 2, *doplňková pravidla* přidávají použití operátorů negace. Automaty řízené výhradně lineárními pravidly jsou označovány jako *lineární celulární automaty*. Pakliže mohou být v automatu obsažena i doplňková pravidla, používá se označení *aditivní celulární automaty*. Konečně pro případ použití pravidel uvažující rovněž jiné logické operátory, kupříkladu logický součin *AND* či exkluzivní disjunkci *XOR*, se používá označení *neaditivní celulární automaty*.

Celulární automaty lze dále rozdělit z hlediska počtu používaných lokálních přechodových funkcí. Zatímco uniformní celulární automat má identická pravidla pro celou mřížku, neuniformní celulární automat dovoluje odlišná pravidla pro každou jednotlivou buňku. Speciálním případem neuniformního celulárního automatu je pak kvazi-uniformní automat, u něhož se v převážné většině buněk vyskytuje pouze několik různých pravidel. Při konstrukci dvourozměrného automatu s výpočetní silou ekvivalentní Turingovu stroji stačí u neuniformních automatů uvažovat pouze von Neumannovo okolí, zatímco u jeho uniformního protějšku je nutné uvažovat alespoň Moorovo okolí [23].

Další dělení celulárních automatů může být provedeno na základě uvažovaného okolí vyšetřované buňky. V případě jednorozměrného celulárního automatu je okolí typicky určeno tzv. poloměrem, který udává počet sledovaných sousedních buněk z obou stran od vyšetřované buňky. V případě dvourozměrného celulárního automatu je okolí obvykle určeno čtyřmi přilehlými buňkami (von Neumannovo okolí), nebo osmi buňkami, kdy se uvažují i buňky sousedící s vyšetřovanou buňkou v rozích (Moorovo okolí). Graficky jsou tato okolí znázorněna na obrázku 3.4.



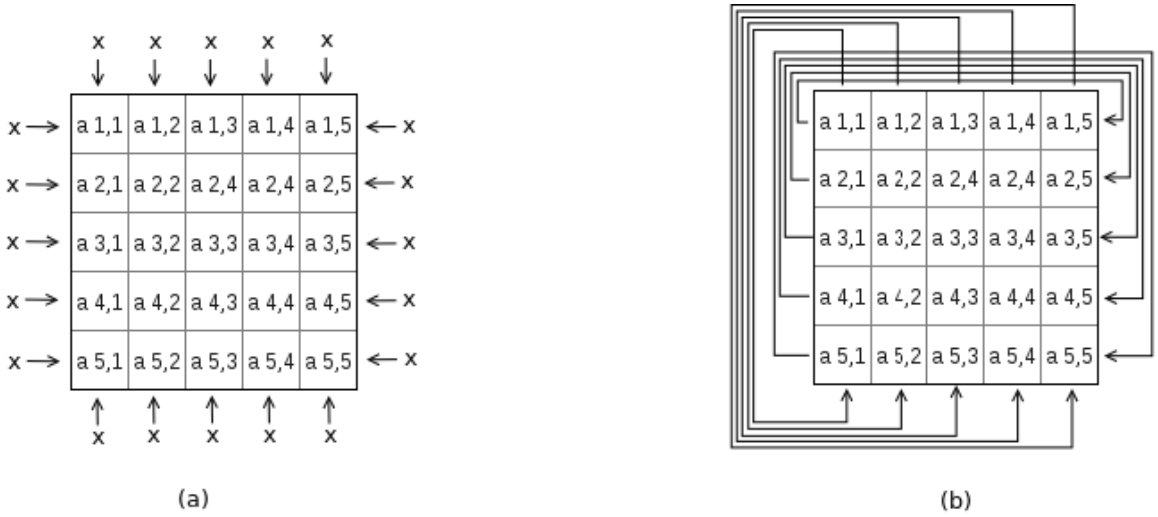
Obrázek 3.4: Nejběžnější typy sousedství: (a) 2D CA – Moorovo sousedství, (b) 2D CA – von Neumannovo sousedství, (c) 1D CA – okolí s poloměrem $r = 2$.

Ačkoliv v teoretické rovině lze uvažovat nekonečný počet buněk, v praktických realizacích je prostor automatu omezen. Buňkám na okraji celulární struktury potom chybí stav jednoho či více sousedů, které by se pomyslně nacházeli právě za hranicí automatu. Podle způsobu řešení této situace rozlišujeme dva typy celulárních automatů (rovněž viz obrázek 3.5):

- *s cyklickými okrajovými podmínkami* (buňky na hranici celulární struktury jsou cyklicky propojeny s buňkami z opačné strany – v případě 1D CA je pak uspořádání podobné kružnici, v případě 2D CA toroidu) a
- *s konstantními okrajovými podmínkami* (buňky na hranici celulární struktury pracují s předem definovanou hodnotu stavu pro neexistující sousední buňky, přičemž typicky se jako výchozí hodnota používá 0).

Důležitými rysy celulárních automatů jsou (převzato z [15]):

- *paralelismus* (výpočet nových hodnot stavů všech prvků probíhá současně – na běžných sériových počítačích se musí tento postup simulovat),
- *lokalita* (nový stav prvku závisí jen na jeho původním stavu a na původních stavech prvků z jeho okolí) a
- *homogenita* (pro všechny prvky platí stejná lokální přechodová funkce – tento rys je však porušen u neuniformních a kvazi-uniformních celulárních automatů, proto je dnes homogenita pokládána pouze za jakousi formální vlastnost vzorového celulárního automatu).



Obrázek 3.5: Obvykle používané typy okrajových podmínek dvourozměrných celulárních automatů: (a) automat s konstantními okrajovými podmínkami, (b) automat s cyklickými okrajovými podmínkami.

Na základě výše uvedeného obecného popisu celulárního automatu lze provést jeho formální definici ve variantě představující platformu pro experimenty realizované v rámci této práce následovně:

Definice 2.1 [20] Dvourozměrný uniformní celulární automat je posloupnost, $(C_t)_{t\in\mathbb{N}}$, definovaná jako pětice (d, Σ, N, f, C_0), kde

1. $d\in\mathbb{N}$ udává velikost mřížky automatu,
2. $\Sigma = \{\sigma_1, ..., \sigma_m\}$ je konečná množina stavů,
3. $N = \{N_1, ..., N_n\}$ udává uvažované okolí, přičemž prvky množiny N jsou tvořené libovolnou dvojicí celých čísel určující relativní polohu vůči vyšetřované buňce (v případě von Neumannova okolí $N = \{(0, -1), (-1, 0), (+1, 0), (0, +1)\}$),
4. $\delta : \Sigma^N \to \Sigma$ je lokální přechodová funkce,
5. $C_0 : \mathbb{Z}^d \to \Sigma$ udává přiřazení počátečních stavů všem buňkám mřížky.

3.3 Celulární programování

Celulární programování, které bylo vynalezeno v 90. letech Moshem Sipperem [23], reprezentuje přístup k evoluci neuniformních celulárních automatů. Stěžejní myšlenka tohoto přístupu je podobná difúznímu modelu paralelního genetického algoritmu, přičemž obecný tvar algoritmu celulárního programování je uveden na obrázku 3.6.

Způsob evoluce celulárního automatu vychází z genetického algoritmu. Na rozdíl od něj však algoritmus nepracuje s celou populací kandidátních řešení – množinou uniformních celulárních automatů, nýbrž mu k činnosti postačuje jediný neuniformní celulární automat. Předmětem evoluce jsou totiž přechodové funkce jednotlivých buněk, které jsou zpočátku

náhodně inicializovány a zakódovány v podobě řetězce obsahujícího výstupní stavy pro všechny možné kombinace stavů buněk z uvažovaného okolí.

Hodnocení úspěšnosti lokální přechodové funkce se provádí pro každou buňku automatu zvlášť. V rámci tohoto procesu je zkoumána schopnost buňky dosáhnout požadovaného cílového stavu z různých počátečních konfigurací stavů buněk celulární struktury po provedení stanoveného počtu kroků automatu. Na základě této schopnosti buňka obdrží fitness hodnotu, která je v každém kroku evolučního cyklu porovnávána s fitness hodnotami sousedních buněk v rámci definovaného okolí. Pokud se v okolí nenachází žádná buňka s vyšší fitness hodnotou, zůstane lokální přechodová funkce nezměněná. Jestliže však existuje právě jedna taková buňka, její lokální přechodová funkce nahradí přechodovou funkci vyšetřované buňky. Existuje-li jich pak více, vyberou se náhodně dvě z nich, aplikuje se na ně operátor křížení a nově vzniklá lokální přechodová funkce je poté uložena na pozici vyšetřované buňky. Před samotným uložením vygenerované lokální přechodové funkce se obvykle ještě provede její mutace.

Takto definované paradigma pomáhá zlepšit kvalitu řešení řady úloh, pro které je obtížné nalézt univerzální lokální přechodovou funkci, která by byla schopná řešit globální problémy pouze na základě lokálních interakcí. Zároveň jsou přitom zachovány charakteristické vlastnosti tradičního modelu celulárního automatu, tedy paralelismus a lokalita.

3.4 Využití celulárních automatů jako modelů vývinu

Umělý development představuje techniku inspirovanou přírodními biologickými jevy. Hlavní myšlenka tohoto přístupu vychází z procesu ontogeneze spočívajícího ve vývinu vyspělého mnohabuněčného organizmu z jediné zárodečné buňky – zygoty. Ontogeneze sestává z několika stádií vývoje, které se typicky alespoň částečně překrývají [11]:

- zárodečné dělení – vytvoření shluku buněk (tzv. blastuly) rapidním dělením zárodečné buňky,
- organizace buněk – prostorové uspořádání buněk v embryu,
- morfogeneze – změna tvaru způsobená přemísťováním buněk za účelem vytvoření základu vnitřností (transformace blastuly na tzv. gastrulu),
- buněčná diferenciace – vznik vlastní struktury a funkce jednotlivých buněk,
- růst – rozmnožování buněk.

Výše popsaný koncept je většinou díky své složitosti aplikován v redukované podobě, kdy jsou zjednodušeně simulovány pouze některé jeho části. Použití masivně paralelních výpočetních systémů, jakými jsou právě celulární automaty, na úlohu umělého vývinu umožňuje alespoň částečně simulovat proces ontogeneze. Jednotlivé pozice v rámci celulární struktury reprezentují buňky vyvíjeného organizmu, přičemž aktuální stav buňky určuje její typ. Počáteční konfigurace buněk mřížky potom představuje zygotu a sekvence přechodových kroků automatu napodobuje proces vývinu.

Umělý development využívá nepřímé mapování genotypu na fenotyp. Na rozdíl od přímého mapování není předmětem evoluce genotyp přímo vázaný na fenotyp, nýbrž samotný postup konstrukce cílového objektu ze zárodečné buňky. Genotypem se označuje genetická informace zakódovaná v chromozomu sestávajícího se z prvků vyhledávacího prostoru. Fenotyp pak představuje řešení úlohy vzniklé zobrazením genotypu do konkrétního jedince.

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

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

Obrázek 3.6: Pseudokód celulárního programování (převzato z [23]).

3.5 Problémy řešené prostřednictvím celulárních automatů

V této kapitole jsou obecně popsány typické benchmarkové úlohy využívané k porovnání různých konceptů celulárních automatů. Ke každé úloze úloze je rovněž uveden souhrn výsledků dosažených jednotlivými koncepty známými z literatury. Tyto výsledky budou později použity k porovnání s hodnotami z vlastních experimentů.

3.5.1 Problém majority

Problém majority (majority task, density task) je rozhodovací úloha určující, zda se v počáteční konfiguraci binárního celulárního automatu vyskytuje více buněk ve stavu 1 než buněk ve stavu 0. Formálně zapsáno [18]: Nechť ρ označuje hustotu výskytu stavu 1 v aktuální konfiguraci. Potom $\rho(t)$ označuje hustotu výskytu stavu 1 v čase t a ρ_c prahovou hodnotu pro klasifikaci (pro problém majority $\rho_c = 0.5$). Požadované chování celulárního automatu je takové, aby po vykonání stanoveného počtu kroků byly všechny buňky jeho konfigurace ve stavu 1, pakliže $\rho(1) > \rho_c$, nebo ve stavu 0, pakliže $\rho(1) < \rho_c$. V případě $\rho(1) = \rho_c$ není požadované chování definované.

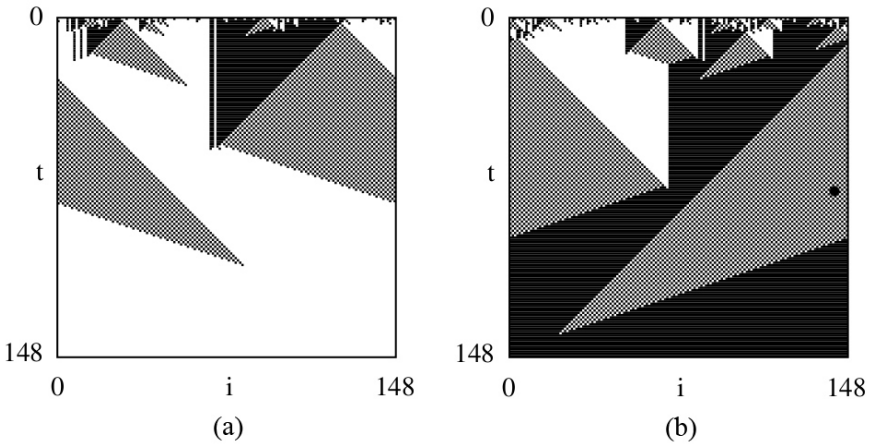
Problém majority je považován za netriviální úlohu z toho důvodu, že celulární automat pracuje podle lokálních pravidel a má proto možnost zjistit stavy buněk pouze u svých sousedů. Hustota výskytu určitého stavu je však globální vlastnost celé konfigurace, tudíž je nutné znát stav ve všech buňkách. Obtížnost řešení pochopitelně narůstá se zvětšujícím se N označujícím velikost mřížky. K získání řešení je nutné uvažovat přenos informace přes všechny buňky mřížky trvající až $2N$ kroků u dvourozměrného automatu uvažujícího Von-Neumannovo okolí, nebo N kroků u dvourozměrného automatu uvažujícího Moorovo okolí. Vyžadované množství paměti (pro uložení počtu buněk v daném stavu) pak rovněž závisí na velikosti mřížky N a odpovídá $O(\log N)$. Jelikož úspěšnost řešení je závislá na schopnosti přenosu informace mřížkou, je problém majority častým a oblíbeným prostředkem k měření schopnosti jednotlivých konceptů celulárních automatů.

V roce 1994 bylo pro jednorozměrný dvoustavový celulární automat s poloměrem r a hustotou výskytu ρ sledovaného stavu dokázáno, že neexistuje žádné takové pravidlo, které by zvládlo problém majority ze všech možných startovacích konfigurací správně vyřešit (pakliže počet buněk mřížky je dostatečně velký – větší než $4r/\rho$) [12]. Toto tvrzení lze zobecnit pro jakýkoliv vícerozměrný uniformní celulární automat. V případě jednorozměrného dvoustavového celulárního automatu je prozatím nejúspěšnějším nalezeným řešením Gacs-Kurdyumov-Levin pravidlo (GKL) definované následovně [24]:

$$s_i(t+1) = \begin{cases} majority[s_i(t), s_{i-1}(t), s_{i-3}(t)] & \text{pro } s_i(t) = 0 \\ majority[s_i(t), s_{i+1}(t), s_{i+3}(t)] & \text{pro } s_i(t) = 1 \end{cases}$$

Na obrázku 3.7 lze sledovat způsob, jakým se šíří informace vzniklá na základě lokálních interakcí předepsaných výše zmíněným pravidlem do svého okolí (každý řádek odpovídá jednomu časovému kroku). To postupně vede k dosažení konfigurace složené výhradně z buněk jednoho stavu. Empiricky zjištěná úspěšnost pravidla GKL pro problém majority na jednorozměrném dvoustavovém celulárním automatu s $N = 149$ buňkami, $M = 149$ časovými kroky a uvažovaným poloměrem $r = 3$, odpovídá fitness ≈ 0.98 [18]. Fitness je přitom vyjádřena jako podíl počtu buněk v očekávaném stavu po stanoveném počtu vývojových kroků automatu vůči počtu všech buněk mřížky. Protože experimenty probíhají typicky na velkém množství počátečních konfigurací, uváděná hodnota fitness je průměrem jednotlivých dílčích výsledků.

Další pokusy o zlepšení úspěšnosti řešení problému majority vedly od ručního návrhu pravidel směrem k automatizovanému návrhu využívající genetický algoritmus. Mitchell [18] provedl řadu experimentů, z kterých vypozoroval, že při aplikaci této metody dochází u většiny problémů k nalezení lepšího než doposud existujícího řešení. Jednou z výjimek je však zde zmiňovaný problém majority, u něhož se nepodařilo nalézt pravidla s úspěšností větší než GKL a úspěšnost nalezeného řešení pro jednorozměrný automat s $N = 149$ buňkami a poloměrem $r = 3$ se pohybovala nejčastěji v rozsahu $0.92 - 0.95$, přičemž hod-



Obrázek 3.7: Problém majority. Graf vývoje dvoustavového jednorozměrného uniformního celulárního automatu s počtem buněk $N = 149$, počtem časových kroků $M = 149$ a uvažovaným poloměrem $r = 3$. Bílá políčka reprezentují buňky ve stavu 0, černá políčka reprezentují buňky ve stavu 1. (a) Vývoj automatu s počáteční konfigurací buněk ve stavu 1 odpovídající $\rho(1) = 0.47$. (b) Vývoj automatu s počáteční konfigurací buněk ve stavu 1 odpovídající $\rho(1) = 0.53$. Převzato z [18].

nota 0.95 nebyla nikdy překročena. Úspěšnost evolučně nalezených pravidel navíc s počtem buněk mřížky N postupně klesá, zatímco s použitím pravidel GKL se takřka nemění. Podobné míry úspěšnosti pohybující se v rozmezí $0.92 - 0.95$ bylo rovněž dosaženo pomocí techniky celulárního programování, která na rozdíl od běžného genetického algoritmu nehledá pravidla univerzálně platná pro celou mřížku, ale snaží se najít optimální pravidla pro každou buňku automatu zvlášť.

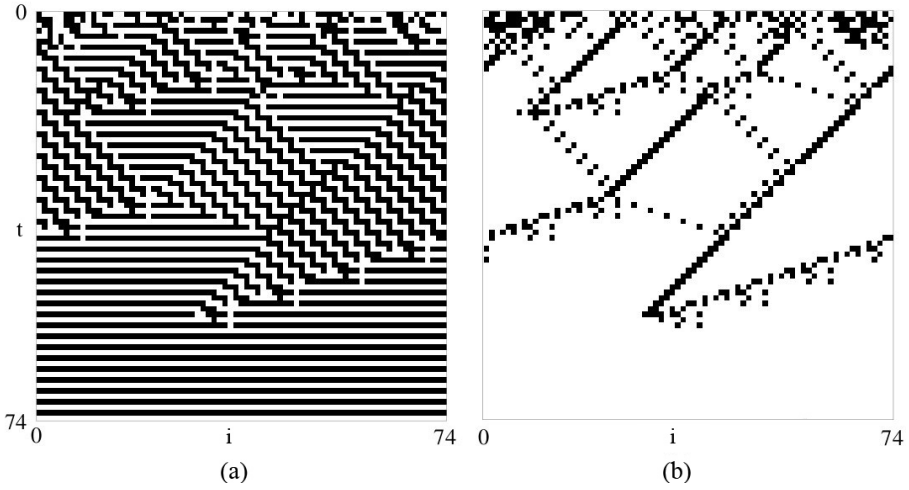
Problém majority je snáze řešitelný při uvažování dvourozměrného celulárního automatu. Každá buňka je totiž namísto pouhých dvou buněk spojena se čtyřmi (von Neumannovo okolí) nebo osmi (Moorovo okolí) sousedními buňkami. Informace se také díky dvourozměrné struktuře mřížky mnohem rychleji šíří k ostatním buňkám automatu, takže počet vývojových kroků nutných k dosažení cílové konfigurace je výrazně nižší. Z těchto důvodů dosáhl Sipper při uvažování dvourozměrného celulárního automatu pracujícího s von Neumannovým okolím za použití techniky celulárního programování průměrné fitness ≈ 0.99 [23].

3.5.2 Problém synchronizace

Problém synchronizace (synchronization task) představuje výpočetní úlohu hledající celulární automat, který je schopen se v konečném počtu výpočetních kroků dostat z jakékoliv počáteční konfigurace do takového stavu, že jeho konfigurace je tvořena výhradně stavy 0 nebo stavy 1, přičemž mezi těmito alternativami neustále osciluje [17]. Formálně zapsáno [23]: Nechť ρ označuje hustotu výskytu stavu 1 v aktuální konfiguraci. Potom $\rho(t)$ označuje hustotu výskytu stavu 1 v čase t a ρ_c prahovou hodnotu (pro problém synchronizace $\rho_c = 0.5$). Požadované chování celulárního automatu je takové, aby po vykonání stanoveného počtu přechodů M měly všechny buňky automatu v následujících 4 výpočetních krocích (tedy $[M + 1] \ldots [M + 4]$) sekvenci stavů $0 \to 1 \to 0 \to 1$, nebo $1 \to 0 \to 1 \to 0$. Úloha je tedy v principu podobná již popsanému problému majority, neboť opět je cílem

dosažení globální vlastnosti pouze na základě lokálních interakcí.

K získání řešení je nutné uvažovat přenos informace zajišťující synchronizaci činnosti lokálních regionů přes celou mřížku, což může trvat až N kroků u jednorozměrného automatu a až $2N$ kroků u dvourozměrného automatu s Von-Neumannovým okolím. Na obrázku 3.8 pak lze sledovat způsob šíření informace. Perfektní řešení je při uvažování jednorozměrného celulárního automatu s $N = 149$ buňkami a poloměrem $r = 3$ nalezeno přibližně v 20% běhů genetického algoritmu [17]. Při stejném zadání dosáhl Sipper s technikou celulárního programování průměrné fitness ≈ 0.84 [23] a pro dvourozměrný celulární automat dokonce ≈ 0.99 [23].



Obrázek 3.8: Problém synchronizace. Graf vývoje jednorozměrného uniformního celulárního automatu s počtem buněk $N = 75$, počtem časových kroků $M = 149$ a uvažovaným poloměrem $r = 3$. Bílá políčka reprezentují buňky ve stavu 0, černá políčka reprezentují buňky ve stavu 1. (a) Vývoj automatu z náhodné počáteční konfigurace. (b) Vývoj automatu ze stejné konfigurace, ovšem s vynecháním pravidelných domén. Převzato z [17].

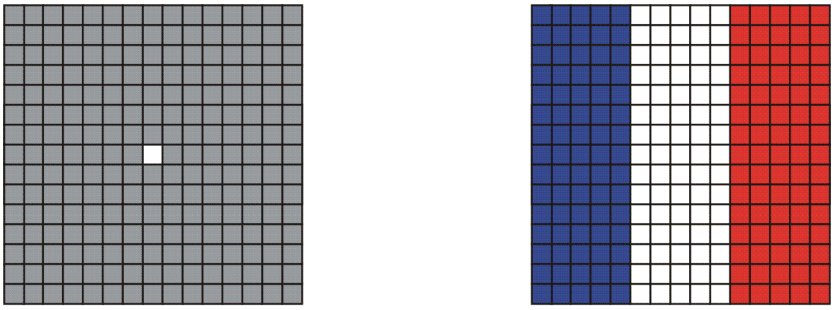
3.5.3 Problém samoorganizace

Problém samoorganizace (self-organization task) bývá často řešen prostřednictvím celulárních automatů jako vývin určitého vzoru z definované počáteční konfigurace – zygoty. Jelikož je vývin určitého uspořádání buněk klíčovým krokem v přirozeném i umělém developmentu, bude řešení využívat jeho principů popsaných v kapitole 3.4.

Úloha “francouzská vlajka”

Úspěšnost jednotlivých metod řešení problému samoorganizace je často testována na úloze generování vzoru francouzské vlajky [6]. Tato úloha, sloužící ke studiu vývinu zygoty do podoby vyspělého vícebuněčného organizmu schopného regulace svého růstu, byla poprvé představena embryologem Lewisem Wolpertem na konci 60. let 20. století [4]. Uvedený problém bývá nejčastěji řešen prostřednictvím morfogenetického či embryogenetického modelu. Ilustrace jedné z možných podob zadání úlohy je znázorněna na obrázku 3.9.

Morfogenetický model (morphogenetic model) je navržený pro multi-celulární systémy, přičemž inspiraci čerpá v biologických procesech genové exprese a buněčné diferenciace.



(a) (b)

Obrázek 3.9: Úloha "francouzská vlajka": (a) počáteční konfigurace – zygota, (b) cílová konfigurace – vyspělý vícebuněčný organizmus.

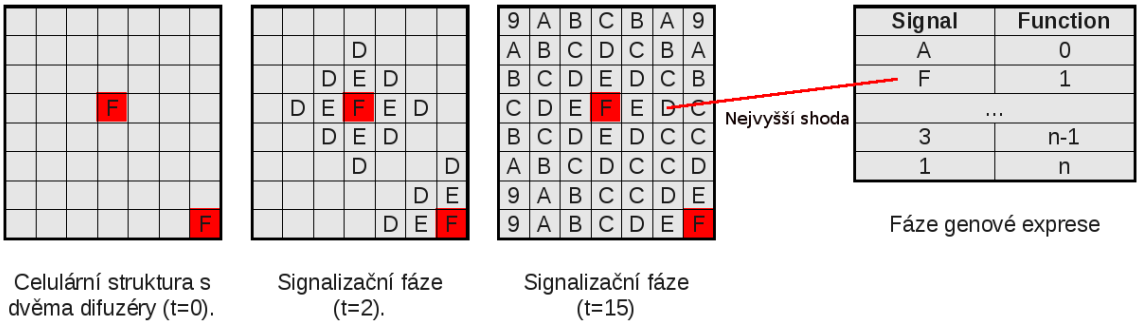
Stěžejní myšlenka spočívá v uvažování koncentrace proteinů na pozici dané buňky pro určení jejího typu [21]. Výpočet obvykle probíhá ve dvou krocích – ve fázi buněčné signalizace (cell signalling phase) a ve fázi genové exprese (gene expression phase). *Signalizační fáze* používá mezibuněčnou komunikaci k výměně signálů mezi sousedními buňkami, čímž je zajištěno šíření informace podobné procesu difuze v dané celulární struktuře. Intenzita signálu je přitom reprezentována numerickou hodnotou vztaženou ke konkrétní buňce automatu. Automat rovněž obsahuje několik speciálních buněk zvaných difuzéry (diffusers), jejichž intenzita signálu je maximální, a na kterých proces difuze začíná. Pravidla šíření signálu pro konkrétní buňku jsou potom odvozena od velikostí intenzity signálu u čtyřech přilehlých sousedů. Obecně platí, že intenzita signálu se snižuje lineárně se vzdáleností od difuzéru. *Fáze genové exprese* stanovuje typ (funkcionalitu) buňky na základě vyhledávání v pomocné tabulce, kde klíčovým vstupem je velikost signálu. Ilustrace tohoto popisu je znázorněna na obrázku 3.10.

Embryogenetický model (embryogeny model) je založený na evoluci regulačního systému buněk, jejich metabolismu a jejich schopnosti duplikace. Ontogeneze je v tomto procesu výsledkem emergentní buněčné interakce a koncentrace chemikálií v okolním prostředí [21]. Jednotlivé buňky automatu jsou charakterizovány interními a externími proměnnými. Interní proměnné značí typ buňky a koncentraci chemikálií. Externí proměnná se pak vztahuje k prostředí a sleduje pravidla procesu difuze. V každém kroku vývoje může jakákoliv žijící buňka uvolnit chemikálie do okolní prostředí, změnit svůj vlastní typ nebo typ metabolismu či se duplikovat do jednoho ze čtyř směrů. Program růstu je přitom řízen dopřednou neuronovou sítí, přičemž jedna z jeho možných podob je zobrazena na obrázku 3.11.

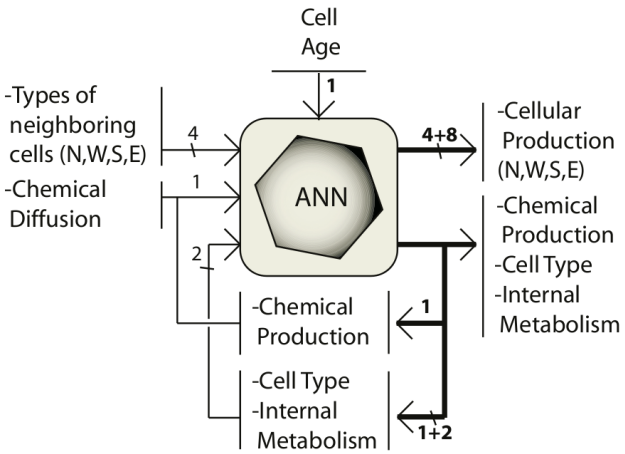
Úspěšnost řešení úloh je u obou popisovaných modelů velmi podobná a prakticky totožné jsou i výsledky testování robustnosti vůči chybám. Nelze proto jednoznačně určit, který z nich je obecně výhodnější použít. Pro oba však platí, že řešení je silně závislé na vhodném nastavení parametrů [21].

3.5.4 Problém návrhu kombinačních logických obvodů

Pro automatizovaný návrh kombinačních logických obvodů již bylo vyvinuto několik odlišných metod. Asi nejznámější a nejúspěšnější z nich je takzvané kartézské genetické programování (CGP) [16], které lze chápat jako speciální variantu genetického programování, v němž je hledané řešení reprezentováno prostřednictvím grafu s pevným počtem uzlů. Za



Obrázek 3.10: Ukázka činnosti morfogenetického modelu. Průběh signalizační fáze je znázorněn na třech mřížkách vlevo, kde červeně podbarvená políčka označují difúzery. Číslo u každé buňky vyjadřuje intenzitu signálu (hexadecimálně). Fáze genové exprese je ukázána napravo – signál D je spojen se vstupem F, neboť z nabízených možností v tabulce je hodnota F nejméně vzdálená. Funkce buňky tedy bude f_1. Převzato z [21].



Obrázek 3.11: Ukázka programu růstu. Každý vstup / výstup je reprezentován desetinným číslem z intervalu $\langle 0, 1\rangle$. Převzato z [21].

jednotlivé uzly jsou přitom považovány hradla plnící předem dané funkce. Konektivita uzlů není libovolná a bývá určena tzv. *l-back* parametrem. Chromozom pak představuje sekvenci celočíselných hodnot vyjadřujících pro každý uzel jeho typ a spojení s ostatními uzly. Během evoluce se pak používá pouze genetický operátor mutace.

Zatímco výše popsaný přístup je zaměřen na evoluci matice přímo reprezentující požadovaný kombinační logický obvod, idea řešení tohoto problému prostřednictvím celulárního automatu je zcela odlišná. V jedné z obvyklých variant implementace je možné podrobit evoluci celulární automat generující požadovaný logický obvod postupně v jednotlivých krocích [1, 2]. Další možností je ovšem zcela se oprostit od idei fyzické implementace pomocí běžných logických hradel a ponechat celulárnímu automatu naprostou volnost ve vytváření přechodových pravidel. V tomto případě slouží první řádek či sloupec automatu k zadání vstupního vektoru, přičemž výstupní vektor je po provedení stanoveného počtu přechodových kroků sejmut z posledního řádku či sloupce (v závislosti na zvolené implementaci). Úspěšné provedení evoluce je podmíněno znalostí kompletní množiny dvojic tvořené vstup-

ními a odpovídajícími výstupními vektory. Ke změnám přechodových funkcí se pak přistupuje na základě schopnosti automatu na dané vstupní vektory odpovídat předepsanými výstupy.

Návrh vhodné metody evoluce automatu není triviální. Jelikož generování výstupního vektoru je založeno na interakci více buněk s různými přechodovými funkcemi, není možné konkrétně určit buňku zodpovědnou za chybné chování. Z toho důvodu se nejčastěji přistupuje k obměně určitého počtu přechodových pravidel náhodně vybraných buněk, přičemž počet takto modifikovaných buněk obvykle závisí na míře chybných odpovědí automatu. Velikost automatu řešícího danou úlohu obvykle odpovídá počtu bitů vstupního vektoru. To však není pravidlem a velikost mřížky bývá, podobně jako počet uvažovaných stavů buňky nebo počet přechodových kroků, předmětem experimentování.

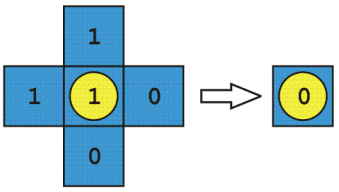
Kapitola 4

Koncept celulárních automatů řízených instrukcemi

Běžné celulární automaty mají lokální přechodovou funkci definovanou ve formě pravdivostní tabulky, kde vstupní proměnné představují stavy buněk v okolí vyšetřované buňky a výstupní proměnná určuje její nový stav po provedení přechodu (viz obrázek 4.1). Evoluce takového celulárního automatu je typicky prováděna prostřednictvím genetického algoritmu či celulárního programování ve snaze nalézt vhodné výstupní stavy pro všechny přípustné konfigurace buněk v sousedství. Je však zřejmé, že počet přípustných konfigurací roste exponenciálně s počtem stavů a z toho důvodu tento přístup vykazuje neuspokojivé výsledky pro vícestavové automaty. Cílem této kapitoly je popis nového konceptu řízení celulárního automatu, který se dokáže s popsaným problémem nárůstu informačního rozsahu lokální přechodové funkce efektivně vypořádat.

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

(a)



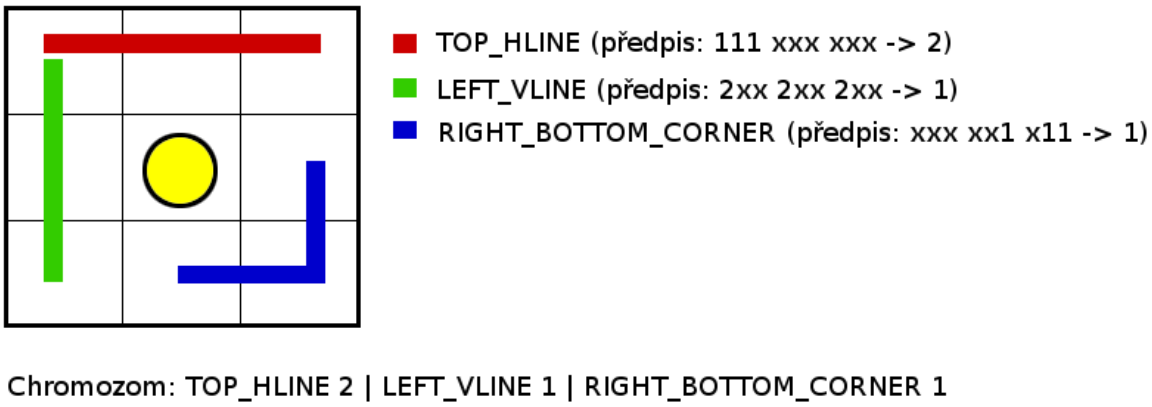
(b)

Obrázek 4.1: (a) Ukázka tabulky přechodových pravidel tvořících lokální přechodovou funkci celulárního automatu uvažujícího von Neumannovo sousedství. Sloupec *Okolí* označuje aktuální stav vyšetřované buňky a sousedních buněk. Sloupec *Výstup* určuje nový stav buňky po provedení přechodu. Pořadí zápisu stavů buněk ve sloupci *Okolí* je následující: aktuální buňka, horní soused, levý soused, dolní soused, pravý soused. (b) Ukázka přechodu jedné buňky automatu podle odpovídajícího pravidla lokální přechodové funkce z tabulky.

4.1 Základní charakteristika konceptu instrukcí

Lokální přechodová funkce instrukcemi řízeného automatu není tvořena pravdivostní tabulkou. Namísto ní se pro výpočet následujícího stavu aplikuje sekvence tzv. instrukcí. Instrukce je označení pro potenciálně zajímavou kombinaci buněk v okolí vyšetřované buňky, nad kterou se provede test stanovené podmínky, při jejímž splnění je stav buňky změněn dle instrukčního předpisu. Vzhledem k relativně malé množině uvažovaných instrukcí v rámci lokální přechodové funkce je zřejmé, že výhoda tohoto přístupu tkví v redukci prohledávacího prostoru, neboť není nutné určovat výstupní stav pro každou kombinaci buněk ze sousedství zvlášť, nýbrž je nový stav vypočten aplikací sekvence instrukcí. Při definici požadované podoby buněčného okolí dané instrukce lze využít pro některé sousedy stav "nedefinováno", který určuje, že na konkrétní hodnotě daného souseda nezáleží.

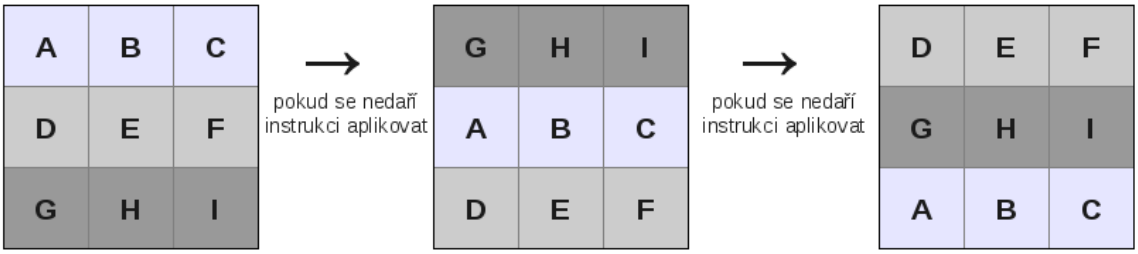
Ukázka popsaného způsobu činnosti automatu je znázorněna na obrázku 4.2. Pro každou vyšetřovanou buňku automatu je známo všech jejich osm sousedů (uvažuje se Moorovo okolí) a rovněž je známo pořadí a podoba instrukcí tvořících lokální přechodovou funkci. Postupně je každá z instrukcí testována, takže například instrukce `TOP_HLINE` ověří stavy v horních třech buňkách a pokud odpovídají stanovené podmínce (v tomto případě musí být jejich stavy nastavené na 1), změní se stav centrální buňky na 2. Bez ohledu na výsledek testu této podmínky se poté pokračuje další instrukcí `LEFT_VLINE`, která otestuje stavy nalevo od centrální buňky a v případě úspěchu nastaví centrální buňku na 1. Analogický postup je pak aplikován i na další instrukce v rámci chromozomu tvořícího lokální přechodovou funkci. Díky sekvenční aplikaci instrukcí, umožňující změnu stavu centrální buňky po jednotlivých dílčích testech, lze považovat interpretaci daného chromozomu za jistou formu programu. Takový program přitom může mít, v porovnání s tabulkovou metodou, stejně komplexní formu chování, avšak k jejímu dosažení postačí mnohem úspornější způsob popisu.



Obrázek 4.2: Ilustrace konceptu instrukcemi řízeného automatu. Chromozom definující lokální přechodovou funkci je tvořen třemi sekvenčně aplikovanými instrukcemi `TOP_HLINE`, `LEFT_VLINE` a `RIGHT_BOTTOM_CORNER`. Testované buňky v rámci jednotlivých instrukcí jsou vyznačené odlišnými barvami v mřížce znázorňující Moorovo okolí vyšetřované buňky (označené žlutým kruhem). Jednotlivé instrukce mají definovanou podobu buněčného okolí určující, jakých stavů musí buňky ze sousedství nabývat pro provedení změny stavu vyšetřované buňky.

V rámci snahy o zvýšení úspěšnosti konceptu instrukcí byla v této práce implementována

modifikace uvedeného konceptu spočívající v řádkové rotaci buněčného okolí. Pokud se při interpretaci chromozomu nedaří na dané pozici v mřížce automatu aplikovat konkrétní instrukci, pak je předpis požadovaného buněčného okolí pozměněn tak, že se jeho první řádek posune na pozici druhého řádku, druhý řádek na pozici třetího řádku a konečně třetí řádek na pozici prvního řádku (viz obrázek 4.3). S takto pozměněným předpisem proběhne další pokus o aplikaci instrukce. Pakliže testovaná konfigurace opět nevyhovuje definovanému předpisu buněčného okolí, je provedena poslední možná rotace řádků a po pokusu o její aplikaci se přechází k další instrukci v pořadí. Tato úprava vedla k rapidnímu zvýšení úspěšnosti konceptu instrukcí na řešených benchmarkových úlohách představených v kapitole 3.5.



Obrázek 4.3: Ilustrace modifikace konceptu instrukcemi řízeného automatu spočívající v řádkové rotaci buněčného okolí. Pokud nelze instrukci aplikovat podle původního předpisu, nepřechází se hned k pokusu o aplikaci další instrukce v pořadí, ale jsou nejprve testovány pozměněné předpisy buněčného okolí původní instrukce.

4.2 Evoluce instrukcemi řízeného celulárního automatu

Instrukce použité pro řízení celulárního automatu jsou navrženy ručně na základě analýzy řešené úlohy. Pro každou řešenou úlohu tedy expert navrhne množinu potenciálně zajímavých instrukcí, z nichž je pak prostřednictvím genetického algoritmu vybrána taková posloupnost, která řeší danou úlohu s co nejvyšší úspěšností. To je důležité z toho důvodu, že každá instrukce může změnit stav centrální buňky, což může ovlivnit výsledek testu další instrukce v pořadí (samozřejmě pouze v případě, že následující instrukce bere v potaz stav centrální buňky). Sekvenční provádění instrukcí je proto svým způsobem podobné interpretaci klasického programu. Účelem této práce je rovněž prozkoumat, jaký vliv má tato vlastnost na celkovou úspěšnost automatu při hledání řešení vybraných problémů.

Konkrétní implementace genetického algoritmu je založena na jeho pseudokódu popsaném v kapitole 2.2.1. Na začátku evoluce je počáteční populace kandidátních řešení utvořena z náhodně vybraných sekvencí instrukcí z předdefinované množiny instrukcí. Takto vzniklí jedinci jsou posléze ohodnoceni fitness hodnotou za pomoci fitness funkce. Evoluce typicky probíhá po stanovený počet generací. Pouze v případě, že je splněna ukončující podmínka, což může být například dosažení perfektního řešení pro všechny možné počáteční konfigurace u problému synchronizace, je provádění evolučního cyklu zastaveno. V každé iteraci evolučního cyklu dochází prostřednictvím genetických operátorů křížení a mutace ke vzniku nových potomků, kteří jsou pak sloučeni s původními členy populace. Pro zachování stejného počtu jedinců populace probíhá nad sloučenou množinou potomků a rodičů proces selekce turnajového typu (viz sekce 2.2.2). Několik jedinců s nejlepším ohodnocením má

však privilegium jisté účasti v další populaci – počet takových jedinců je vyjádřen parametrem *elitismSize*. Kompletní seznam parametrů evoluce je uveden v tabulce 4.1.

Název	**Popis**
Neighbourhood	uvažované okolí buňky (Von Neumannovo / Moorovo)
PopulationSize	velikost populace
GenerationCount	počet generací, po které bude evoluce probíhat
TournamentSize	počet účastníků turnaje při turnajové selekci
AutomatonStep	počet přechodů automatu, po kterých dochází vyhodnocení úspěšnosti řešení úlohy
InstructionCount	počet instrukcí v rámci chromozomu
InstructionSetCount	počet použitých chromozomů (sekvencí instrukcí) v rámci automatu
NopProbability	pravděpodobnost generování prázdné instrukce (NOP = no-operation) do chromozomu
ElitismSize	počet nejúspěšnějších jedinců, kteří jsou beze změn zkopírováni do následující populace
MutationProbability	pravděpodobnost mutace chromozomu, která probíhá při jeho umísťování do následující populace
BoundaryType	typ okrajových podmínek používaný při výpočtu nového stavu u krajních buněk mřížky automatu (konstantní / cyklické)

Tabulka 4.1: Popis používaných parametrů evoluce.

Rekombinační operátory používané v rámci činnosti genetického algoritmu byly obecně popsány v kapitole 2.2.3. Pro výběr množiny instrukcí a nalezení jejich efektivní posloupnosti nachází v rámci této práce uplatnění zejména operátory křížení a mutace. Jelikož každá z konkrétních implementací těchto operátorů má odlišný princip činnosti a generuje tedy nové jedince odlišným způsobem, osvědčilo se vybírat konkrétní implementaci při každém použití daného operátoru zcela náhodně. Při požadavku o provedení operace křížení se tedy vybírá z implementace OX, PMX, MPX a AP. Při požadavku o provedení operace mutace se pak vybírá z implementace SWAP, DM, ISM, IVM a SM. Vyjmenované implementace operátorů křížení a mutace jsou přizpůsobeny pro práci s instrukcí NOP a kupříkladu mutace ISM ji eviduje jako jednu z možností při výběru instrukce vkládané na určenou pozici v chromozomu.

Kapitola 5

Experimenty

V předchozí kapitole byl nejprve připomenut princip tradičního přístupu k řízení činnosti celulárního automatu využívající pravdivostní tabulku. Záhy poté došlo k vysvětlení zcela nového konceptu založeného na instrukcích. Cílem této kapitoly je provést věrohodné testování tohoto nově představeného konceptu a porovnat obdržené výsledky řešení vybraných úloh ze sekce 3.5 s výsledky běžnými pro tradiční přístup.

5.1 Popis metodiky testování a určení hodnot parametrů

Jelikož každá úloha je specifická a klade na evolvované řešení odlišné požadavky, není možné určit globální nastavení evoluce platné pro všechny zkoumané problémy. Pro každou úlohu jsou tedy konkrétní hodnoty parametrů stanoveny individuálně na základě množství provedených experimentů.

Věrohodnost získaných výsledků musí být podložena dostatečným počtem pokusů. Každá úloha je tedy řešena vícekrát, typicky alespoň padesátkrát, a jednotlivá řešení jsou spolu s dosaženou úspěšností ukládána. Po provedení dostatečného počtu experimentů je pak nejlepší dosažené řešení prohlášeno za vítězné. Celý proces testování je zautomatizován pomocí vytvořeného skriptu, který postupně spouští jednotlivé evoluční běhy a zpracovává obdržené výsledky. Vzhledem k výpočetní náročnosti tohoto procesu bylo testování prováděno převážně na serverech `edesign1.fit.vutbr.cz` a `edesign2.fit.vutbr.cz`.

Vyhodnocení úspěšnosti nalezeného řešení pro úlohu majority a synchronizace je komplikováno nutností vyhodnotit vývoj automatu ze všech možných počátečních konfigurací. Jelikož počet takových konfigurací roste exponenciálně s velikostí automatu, je tento požadavek pro větší rozměr mřížky obtížně splnitelný. Z toho důvodu se toto exaktní ohodnocování provádí pouze na mřížkách s počtem buněk $N = 16$ (mřížka 4×4, celkem 2^{16} testovaných konfigurací), $N = 25$ (mřížka 5×5, celkem 2^{25} testovaných konfigurací) a $N = 36$ (mřížka 6×6, celkem 2^{36} testovaných konfigurací). Pro větší rozměr mřížky byla k ohodnocení úspěšnosti navrhnuta statistická metoda.

Princip zmíněné statistické metody je založen na vytvoření několika dostatečně velkých množin náhodně vygenerovaných počátečních konfiguracích. Tyto množiny počátečních konfiguracích jsou postupně využívány během vývoje instrukcí, a to tím způsobem, že po stanoveném počtu generací běhu evolučního cyklu podle obvyklého scénáře se úspěšnost nalezeného řešení vypočítá právě na těchto množinách o relativně velkém počtu konfigurací. Pokud úspěšnost řešení pro všechny zmíněné množiny počátečních konfigurací nepřekročí určitou mez, evoluce opět pokračuje. V opačném případě je evoluce pozastavena a výsledný

odhad úspěšnosti je potom aproximací odhadů pro jednotlivé množiny.

Představená statistická metoda byla testována na mřížce s rozměry 5 x 5 buněk, u které je ještě možné získat relativně rychle přesné ohodnocení vyzkoušením všech možných počátečních konfigurací. Pomocí takto získaného exaktního ohodnocení pak lze snadno spočítat přesnost statisticky získaného výsledku. Na základě deseti provedených experimentů byla vypočítána směrodatná odchylka $\approx 1.291 \cdot 10^{-5}$. Vzhledem ke skutečnosti, že úspěšnost řešení je v rámci prováděných experimentů vyjadřována jako desetinné číslo z intervalu $\langle 0, 1 \rangle$ s přesností na 4 desetinná místa, lze vypočtenou směrodatnou odchylku považovat za nepodstatnou a prohlásit odhady prováděné touto metodou za relevantní.

Aby bylo možné výsledky experimentování s instrukcemi řízeným automatem porovnat s jinými koncepty, byla v rámci této práce implementována i obvykle používaná tabulková metoda. Není tedy nutné se spoléhat na programy vytvořené třetí stranou a dosažené výsledky pak lze pokládat za věrohodné.

5.2 Řešení problému majority

Problém majority byl podrobně popsán v sekci 3.5.1. V rámci této práce je řešení hledáno s hodnotami parametrů uvedených v tabulce 5.1 (význam parametrů byl vysvětlen v sekci 4.2).

Parametr	**Hodnota**
Neighbourhood	Moore
PopulationSize	50
GenerationCount	500
TournamentSize	2
AutomatonStep	2· rozmerMrizky
InstructionCount	8
InstructionSetCount	1 → *uniformní automat*
NopProbability	0.1
ElitismSize	5
MutationProbability	0.2
BoundaryType	cyclic

Tabulka 5.1: Použité hodnoty parametrů evoluce pro hledání řešení problému majority.

5.2.1 Popis experimentů

Testování proběhlo podle schématu popsaného v předcházející sekci. Úspěšnost řešení je vypočítána na základě průměrné fitness nejlepších výsledků v jednotlivých krocích evolučního cyklu. V každé iteraci evolučního cyklu je přitom aktualizována množina konfigurací, na nichž se ohodnocování provádí. Velikost této množiny byla experimentálně stanovena na 100 konfigurací.

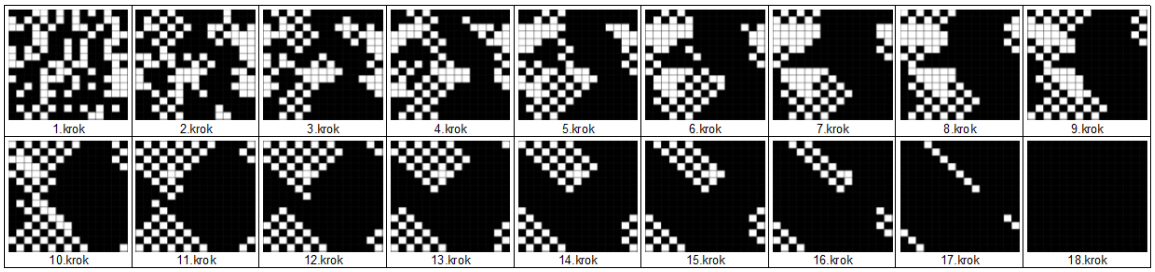
Konečný odhad úspěšnosti je u mřížek do velikosti 6 x 6 buněk prováděn otestováním nejlepšího řešení na množině všech možných počátečních konfigurací. U automatů s velikostí mřížky větším než 6 x 6 je v souladu s metodikou popsanou v sekci 5.1 aplikována statistická metoda, kdy se pro odhad použije celkem 10 množin počátečních konfigurací čítajících po 10^9 konfiguracích.

5.2.2 Výsledky experimentů a diskuze

Experimentálně získané výsledky jsou přehledně shrnuty v tabulce 5.2. Modelová situace vývoje řešení u problému majority je znázorněna na obrázku 5.1. Tento průběh vývoje byl získán s využitím nejlepšího evolučně nalezeného řešení této úlohy při použití přístupu založeném na instrukcích – toto řešení je znázorněno na obrázku 5.2.

Velikost mřížky	**Úspěšnost INST**	**Úspěšnost TM**	**Rozdíl úspěšností**
4 x 4	0.9498	0.8735	0.0763
5 x 5	0.9441	0.8662	0.0779
6 x 6	0.9391	0.8586	0.0805
7 x 7	0.9364	0.8537	0.0827
8 x 8	0.9333	0.8497	0.0836
9 x 9	0.9286	0.8419	0.0867
10 x 10	0.9283	0.8414	0.0869
11 x 11	0.9247	0.8356	0.0891
12 x 12	0.9213	0.8308	0.0904
13 x 13	0.9211	0.8293	0.0918
14 x 14	0.9183	0.8235	0.0948
15 x 15	0.9170	0.8217	0.0953
Průměr	0.9302	0.8438	0.0863

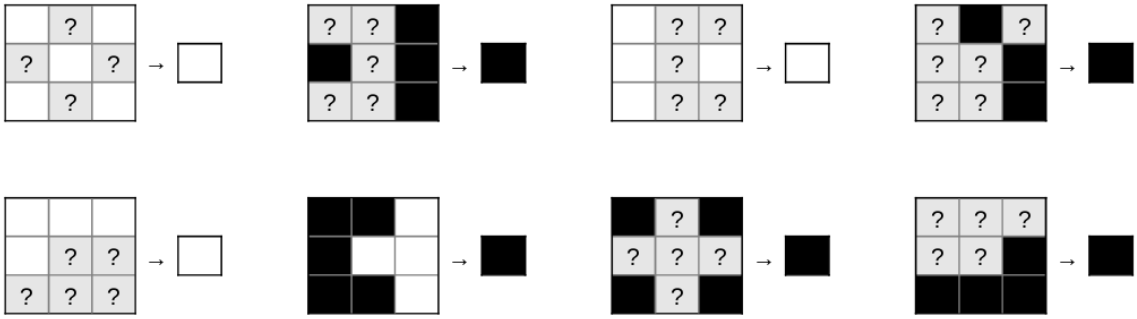
Tabulka 5.2: Tabulka s nejlepšími nalezenými výsledky řešení problému majority pro jednotlivé přístupy. Sloupec *Úspěšnost INST* udává úspěšnost nejlepšího nalezeného řešení s využitím instrukcemi řízeného celulárního automatu. Sloupec *Úspěšnost TM* udává to samé pro tabulkovou metodu. Rozdíl těchto hodnot pro každou velikost mřížky je uveden ve sloupci *Rozdíl úspěšností*.



Obrázek 5.1: Problém majority: Ukázka činnosti dvourozměrného celulárního automatu na náhodné počáteční konfiguraci s počáteční hustotou buněk ve stavu 1 odpovídající $\rho(1) = 0.591$. Velikost mřížky $N = 225$ (15×15).

Na základě provedených experimentů, jejichž výsledky jsou uvedeny v tabulce 5.2, je možné konstatovat, že koncept instrukcemi řízeného automatu vykazuje pro řešení úlohy majority lepší výsledky než tradiční tabulková metoda. Analýzou výsledků bylo rovněž zjištěno, že instrukcemi řízený automat je méně náchylný na problém škálovatelnosti (scalability). Zatímco úspěšnost řešení klesla u tradičního přístupu při změně rozměru mřížky z 4 x 4 na 15 x 15 přibližně o 5.2%, u instrukcemi řízeného automatu tento rozdíl činil přibližně pouze 3.3%. Jelikož v obou případech experimenty proběhly se stejným nastavením

parametrů evolučního algoritmu a trvaly přibližně stejný čas, lze tvrdit, že pro problém majority koncept instrukcemi řízeného automatu jednoznačně vyniká nad tradičním přístupem.



Obrázek 5.2: Problém majority: Grafická podoba instrukcí nejlepšího nalezeného řešení. Řešení je reprezentováno osmi instrukcemi (pořadí jejich aplikace na okolí vyšetřované buňky je zleva doprava a shora dolů). Znak ? vyjadřuje, že na stavu dané buňky pro vyhodnocení podmínky nezáleží.

5.3 Řešení problému synchronizace

Druhou úlohou sloužící pro ověření konceptu instrukcemi řízeného automatu je problém synchronizace, který byl detailně popsán v sekci 3.5.2. V rámci této práce je řešení hledáno s hodnotami parametrů uvedených v tabulce 5.3 (význam parametrů byl vysvětlen v sekci 4.2).

Parametr	**Hodnota**
Neighbourhood	Moore
PopulationSize	50
GenerationCount	200
TournamentSize	2
AutomatonStep	2· rozmerMrizky
InstructionCount	16
InstructionSetCount	1 → *uniformní automat*
NopProbability	0.15
ElitismSize	5
MutationProbability	0.2
BoundaryType	cyclic

Tabulka 5.3: Použité hodnoty parametrů evoluce pro hledání řešení problému synchronizace.

5.3.1 Popis experimentů

Experimentování probíhá stejným způsobem jako u problému majority popsaném v předcházející sekci 5.2. Úspěšnost řešení je opět vypočítána na základě průměrné fitness nejlepších výsledků v jednotlivých krocích evolučního cyklu, přičemž množina konfigurací, na

nichž se ohodnocování provádí, je aktualizována každých 50 generací. Nejúspěšnější řešení u mřížek do rozměru 6 x 6 buněk je testováno na množině všech možných počátečních konfigurací automatu pro dosažení věrohodného výsledku. Automaty s větším rozměrem mřížky pak používají k odhadu úspěšnosti statistickou metodu popsanou v kapitole 5.1.

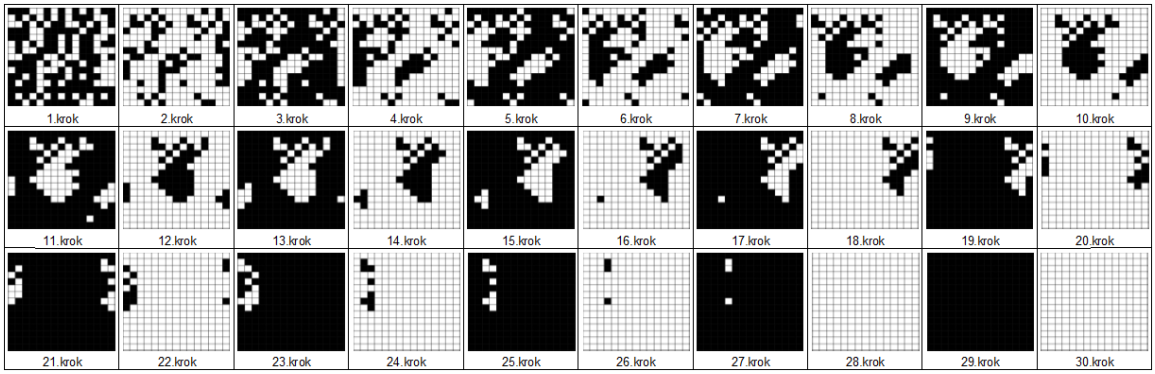
5.3.2 Výsledky experimentů a diskuze

Experimentálně získané výsledky jsou přehledně shrnuty v tabulce 5.4. Modelová situace vývoje řešení u problému synchronizace je znázorněna na obrázku 5.3. Tento průběh vývoje byl získán s využitím nejlepšího evolučně nalezeného řešení této úlohy při použití přístupu založeném na instrukcích – toto řešení je znázorněné na obrázku 5.4.

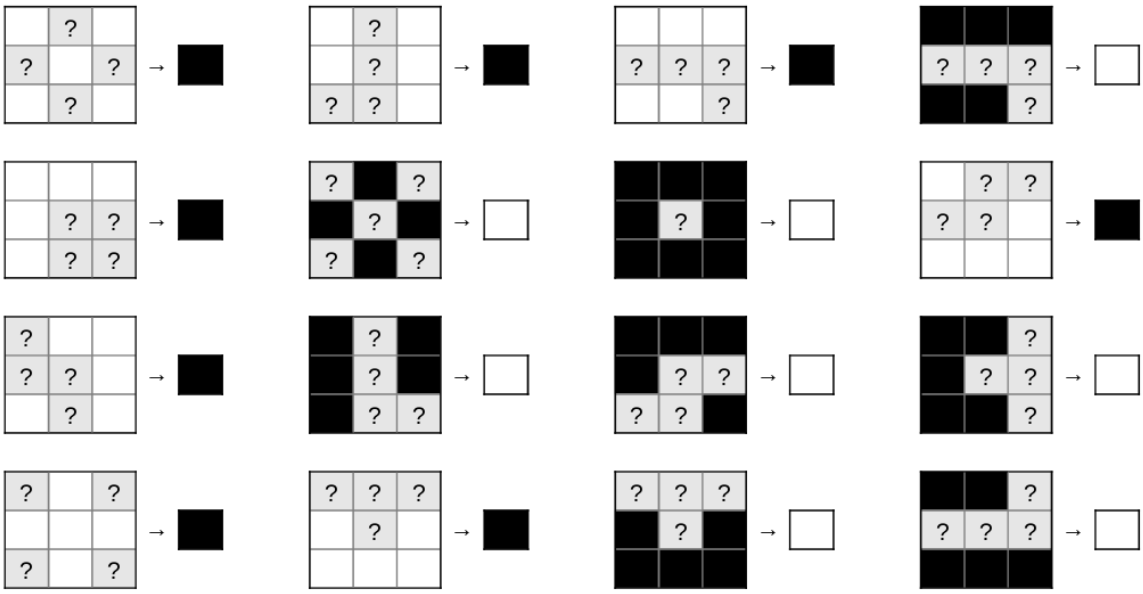
Podobně jako u problému majority je i u problému synchronizace lepších výsledků dosaženo s využitím konceptu instrukcemi řízeného automatu. Z celkových výsledků je rovněž zřejmé, že i v tomto případě je instrukcemi řízený automat méně náchylný na problém škálovatelnosti, neboť zatímco rozdíl mezi nejúspěšnějším řešením pro mřížku 4 x 4 a 15 x 15 je v jeho případě přibližně 3.4%, u tradičního přístupu tato hodnota činí přibližně 4.5%.

Velikost mřížky	**Úspěšnost INST**	**Úspěšnost TM**	**Rozdíl úspěšností**
4 x 4	0.8912	0.8651	0.0261
5 x 4	0.8860	0.8595	0.0265
6 x 4	0.8797	0.8521	0.0276
7 x 4	0.8736	0.8452	0.0284
8 x 4	0.8718	0.8421	0.0297
9 x 4	0.8681	0.8374	0.0307
10 x 4	0.8675	0.8358	0.0317
11 x 4	0.8655	0.8335	0.0320
12 x 4	0.8608	0.8295	0.0313
13 x 4	0.8597	0.8273	0.0324
14 x 4	0.8580	0.8236	0.0344
15 x 4	0.8574	0.8201	0.0373
Průměr	0.8680	0.8369	0.0311

Tabulka 5.4: Shrnutí nejlepších nalezených výsledků řešení problému synchronizace pro jednotlivé přístupy. Sloupec *Úspěšnost INST* udává úspěšnost nejlepšího nalezeného řešení s využitím instrukcemi řízeného celulárního automatu. Sloupec *Úspěšnost TM* udává to samé pro tabulkovou metodu. Rozdíl těchto hodnot pro každou velikost mřížky je uveden ve sloupci *Rozdíl úspěšností*.



Obrázek 5.3: Problém synchronizace: Ukázka činnosti dvourozměrného celulárního automatu na náhodné počáteční konfiguraci s počáteční hustotou buněk ve stavu 1 odpovídající $\rho(1) = 0.591$. Velikost mřížky $N = 225$ (15×15).



Obrázek 5.4: Problém synchronizace: Grafická podoba instrukcí nejlepšího nalezeného řešení. Řešení je reprezentováno 16 instrukcemi (pořadí jejich aplikace na okolí vyšetřované buňky je zleva doprava a shora dolů). Znak ? vyjadřuje, že na stavu dané buňky pro vyhodnocení podmínky nezáleží.

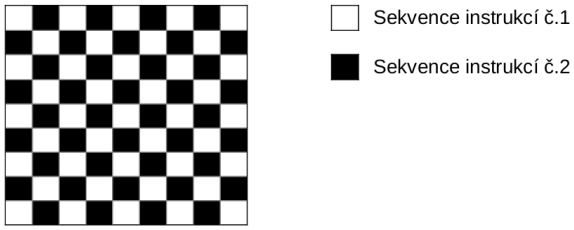
5.4 Řešení problému samoorganizace

Experimenty ověřující schopnost instrukcemi řízeného automatu řešit problém samoorganizace byly provedeny na úloze francouzské vlajky vysvětlené v kapitole 3.5.3. Tato úloha počítá s mřížkou o rozměrech 12 x 12 buněk, přičemž tyto buňky mohou nabývat jednoho ze čtyř stavů. Tři stavy jsou přitom vyhrazeny pro živé buňky (červená, bílá a modrá barva) a čtvrtý stav je vyhrazen pro mrtvou buňku (šedá barva). Na rozdíl od předchozích pokusů nebyl použitý celulární automat uniformní, nýbrž kvazi-uniformní. V rámci vyvinutého programu lze prostřednictvím parametrů konfiguračního souboru zadat maximální počet sekvencí instrukcí, které se mohou pro vývin automatu použít. Kompletní seznam hodnot parametrů evoluce je pak uveden v tabulce 5.5. Daná úprava, tedy zavedení možnosti použití více sekvencí instrukcí pro různé buňky mřížky automatu, výrazně zvyšuje úspěšnost řešení úloh zaměřených na problém samoorganizace. Pro účely této práce se při prováděných experimentech pracovalo se dvěma sekvencemi instrukcí, přičemž určení, jakou sekvencí instrukcí se bude konkrétní buňka řídit, není náhodné. Při přiřazování se pravidelně obě sady sekvencí po každém kroku střídají, čímž je vytvořena pravidelná "mapa pokrytí" (viz obrázek 5.5). Zmíněná mapa pokrytí buněk určenými sekvencemi instrukcí připomíná svou strukturou šachovnici. Toto schéma bylo zvoleno z toho důvodu, že vykazovalo nejlepší výsledky oproti jiným zkoumaným schématům a dokonce předčilo i evolučně vyvíjené schéma, v němž si buňky volily takové sekvence instrukcí, které pro ně byly v daný okamžik nejvýhodnější.

V rámci této úlohy jsou uvedeny výsledky pouze pro instrukcemi řízený celulární automat. Při experimentech využívajících tradiční tabulkovou metodu totiž bylo dosaženo pouze silně neuspokojivých výsledků, což je pravděpodobně způsobeno zejména rapidním nárůstem počtu pravidel lokální přechodové funkce oproti problémům majority a synchronizace. Zatímco ve zmíněných úlohách se počítalo pouze se dvěma stavy na buňku, čemuž odpovídá $2^8 = 256$ pravidel, v případě úlohy francouzské vlajky je se čtyřmi možnými stavy na buňku nutné určit $4^8 = 65535$ pravidel.

Parametr	**Hodnota**
Neighbourhood	Moore
PopulationSize	50
GenerationCount	200
TournamentSize	2
AutomatonStep	20
InstructionCount	12
InstructionSetCount	2 → *neuniformní automat*
NopProbability	0.15
ElitismSize	5
MutationProbability	0.2
BoundaryType	fixed

Tabulka 5.5: Použité hodnoty parametrů evoluce pro hledání řešení problému samoorganizace.



Obrázek 5.5: Problém samoorganizace: Mapa pokrytí buněk mřížky automatu jednotlivými sekvencemi instrukcí použitá při hledání řešení úlohy vývinu vzoru francouzské vlajky.

5.4.1 Popis experimentů

Experimentování proběhlo v souladu s metodikou popsanou v sekci 5.1. Úspěšnost řešení je vypočítána na základě počtu buněk, které dosáhnou cílového stavu po provedení definovaného počtu přechodů. V rámci hodnocení úspěšnosti se rovněž sleduje, zda-li se automat po provedení daného počtu přechodů dále vyvíjí, či zda-li svoji konfiguraci drží v nezměněné podobě.

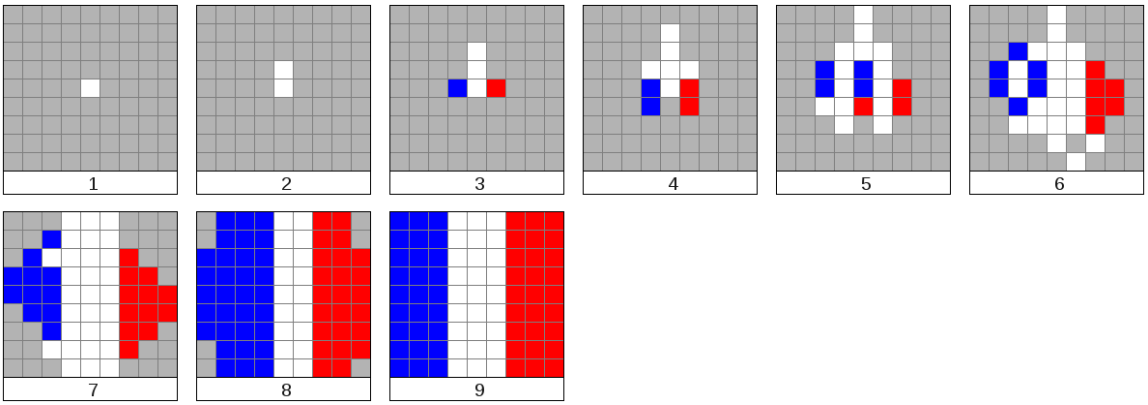
5.4.2 Výsledky experimentů a diskuze

Experimentálně získané výsledky úlohy vývinu vzoru francouzské vlajky jsou pro jednotlivé rozměry mřížky přehledně shrnuty v tabulce 5.6. Z výsledků vyplývá, že pro rozměry mřížky 3 x 3, 6 x 6 a 9 x 9 se podařilo nalézt perfektní řešení. Toto tvrzení již bohužel neplatí pro zkoumané větší velikosti automatu, ovšem i tak lze získané míry úspěšnost považovat za relativně vysoké. Modelová situace perfektního vývoje automatu s rozměrem mřížky 9 x 9 je znázorněna na obrázku 5.6. Na obrázku 5.7 je pak uveden vývoj automatu s rozměrem mřížky 15 x 15 podle nejlepšího evolučně nalezeného řešení uvedeného na obrázku 5.8 s úspěšností přibližně 0.956.

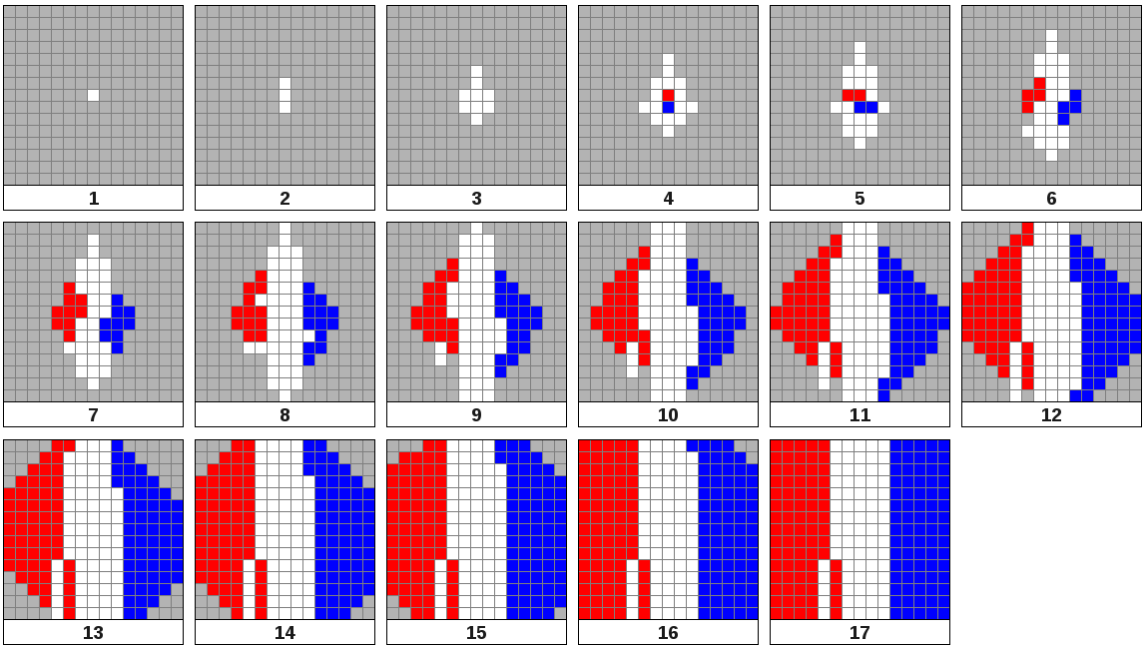
Zatímco v případě předchozích dvou úloh byla úspěšnost řešení vyšší při použití cyklických okrajových podmínek, úloha samoorganizace vykazuje lepší výsledky při konstantních okrajových podmínkách. Příčina tkví pravděpodobně v odlišném charakteru řešených úloh, kdy samoorganizace požaduje rozdílné chování v závislosti na pozici buňky v rámci celulární struktury, s čímž souvisí přiřazování různých instrukčních sad jednotlivým buňkám podle definované “mapy pokrytí”.

Rozměr mřížky	3x3	6x6	9x9	12x12	15x15	18x18	21x21
Úspěšnost	1	1	1	0.965	0.956	0.920	0.896

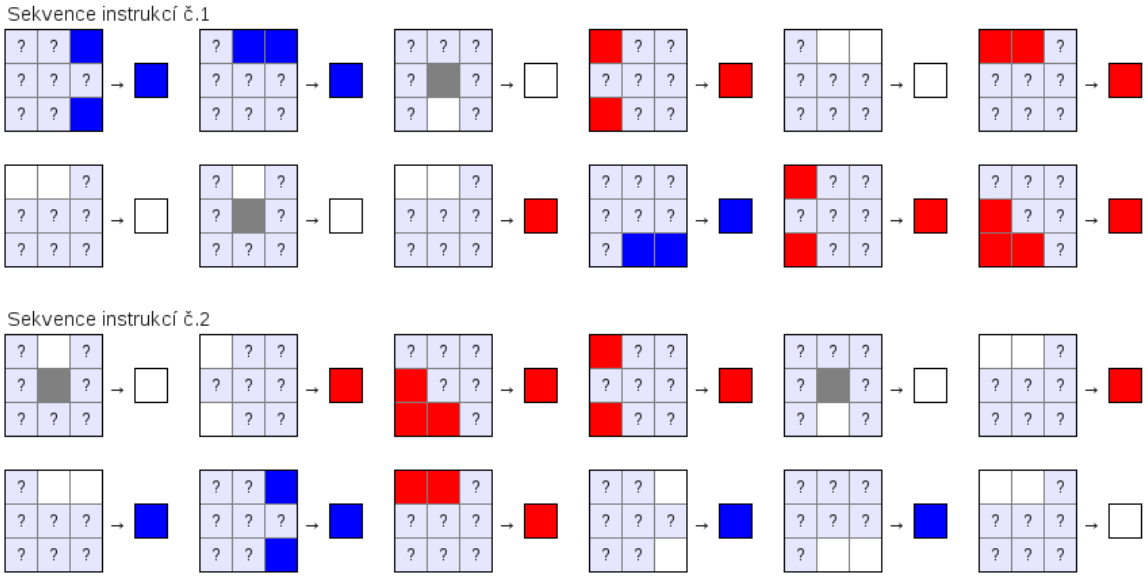
Tabulka 5.6: Shrnutí nejlepších výsledků řešení problému samoorganizace v podobě vývoje vzoru francouzské vlajky pro jednotlivé rozměry mřížky.



Obrázek 5.6: Problém samoorganizace: Ukázka činnosti dvourozměrného celulárního automatu při vývoji vzoru francouzské vlajky. Vývoj probíhá podle nejlepšího evolučně nalezeného řešení pro rozměr mřížky 9 x 9 buněk, které vykazuje úspěšnost 1. Po dosažení 9.kroku vývoje se vzor v mřížce automatu stane stabilní a již se dále nemění.



Obrázek 5.7: Problém samoorganizace: Ukázka činnosti dvourozměrného celulárního automatu při vývoji vzoru francouzské vlajky. Vývoj probíhá podle nejlepšího evolučně nalezeného řešení pro rozměr mřížky 15 x 15 buněk, které vykazuje úspěšnost přibližně 0.956. Po dosažení 17.kroku vývoje se vzor v mřížce automatu stane stabilní a již se dále nemění.



Obrázek 5.8: Problém samoorganizace: Grafická podoba instrukcí nejlepšího nalezeného řešení pro vývin vzoru francouzské vlajky na mřížce 15 x 15 buněk. Řešení je reprezentováno dvěma sekvencemi, z nichž každá obsahuje 12 instrukcí. Přiřazení sekvencí instrukcí jednotlivým buňkám je provedeno podle "mapy pokrytí" v podobě znázorněné na obrázku 5.5. Znak ? vyjadřuje, že na stavu dané buňky pro vyhodnocení podmínky nezáleží.

5.5 Řešení problému návrhu kombinačních logických obvodů

Poslední úloha řešená v rámci této práce je zaměřená na automatizovaný návrh kombinačních logických obvodů (teorie vztažená k této problematice se nachází v sekci 3.5.4). Schopnost instrukcemi řízeného automatu provádět takový návrh je zkoumána na úloze konstrukce úplné jednobitové sčítačky a dvoubitové sčítačky. V rámci této práce je řešení hledáno s hodnotami parametrů uvedených v tabulce 5.9 (význam parametrů byl vysvětlen v sekci 4.2).

Úplná jednobitová sčítačka pracuje tak, že obdrží na vstupu trojici (A, B, C_{IN}), kde A a B jsou jednobitová čísla a C_{IN} je přenos z nižšího řádu. Výstupem je dvojice (S, C_{OUT}), kde S je jednobitový součet a C_{OUT} je přenos do vyššího řádu. Fyzická realizace sčítačky prostřednictvím hradel AND, XOR a OR je znázorněna na obrázku 5.9a. Úplné sčítačky lze zřetězit vedle sebe propojením výstupu C_{OUT} s vstupem C_{IN} sčítačky vyššího řádu (viz obrázek 5.9b). Booleovský zápis jednobitové sčítačky je následující:

$$\begin{aligned} S &= A \oplus B \oplus C_{IN} \\ C_{OUT} &= (A \cdot B) + (C_{IN} \cdot (A \oplus B)) = (A \cdot B) + (B \cdot C_{IN}) + (C_{IN} \cdot A) \end{aligned}$$

Definice pravdivostních tabulek obou řešených kombinačních logických obvodů, tedy úplné jednobitové sčítačky a dvoubitové sčítačky, jsou uvedeny v tabulce 5.7, respektive 5.8.

Vstup			**Výstup**	
A	B	C_{IN}	S	C_{OUT}
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Tabulka 5.7: Pravdivostní tabulka úplné jednobitové sčítačky.

Vstup				**Výstup**		
A_1	A_0	B_1	B_0	C_{OUT}	S_1	S_0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

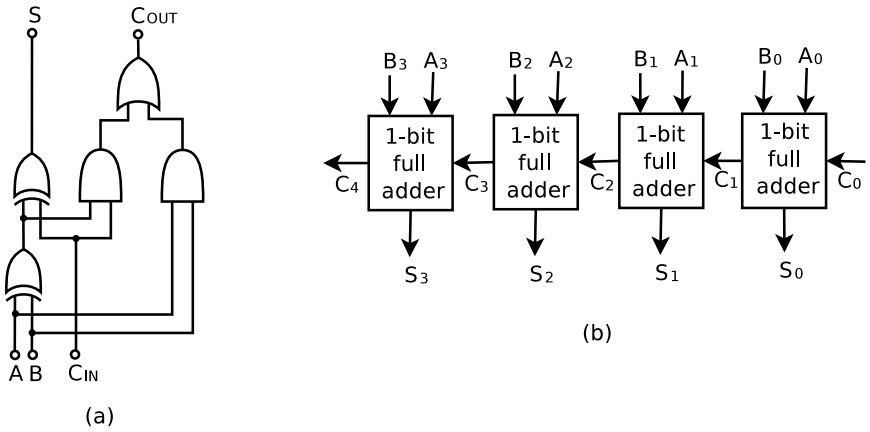
Tabulka 5.8: Pravdivostní tabulka dvoubitové sčítačky.

5.5.1 Popis experimentů

Experimentování probíhá podle obvyklého schématu použitého u dříve představených úloh. Při testování zkoumaného řešení se na první řádek či sloupec automatu zadávají vstupní vektory, přičemž výstupní vektory jsou po provedení stanoveného počtu přechodových kroků sejmuty z posledního řádku či sloupce (v závislosti na zvolené implementaci). Porovnáním takto získaných výstupů s předepsanými výstupy pro všechny kombinace vstupních proměnných z pravdivostní tabulky je vypočítána úspěšnost vyšetřovaného řešení. Úloha se pokládá za vyřešenou pouze v případě, že automat správně odpovídá na každý přiložený vstup.

Parametr	Hodnota
Neighbourhood	Moore
PopulationSize	50
GenerationCount	500
TournamentSize	2
AutomatonStep	6
InstructionCount	10
InstructionSetCount	1 → *uniformní automat*
NopProbability	0.15
ElitismSize	5
MutationProbability	0.2
BoundaryType	fixed

Tabulka 5.9: Použité hodnoty parametrů evoluce pro hledání řešení problému návrhu kombinačních logických obvodů.



Obrázek 5.9: Úplná sčítačka: (a) kombinační obvod úplné jednobitové sčítačky složený z hradel AND, OR a XOR, (b) 4-bitová sčítačka s postupným přenosem složená ze 4 jednobitových sčítaček.

5.5.2 Výsledky experimentů a diskuze

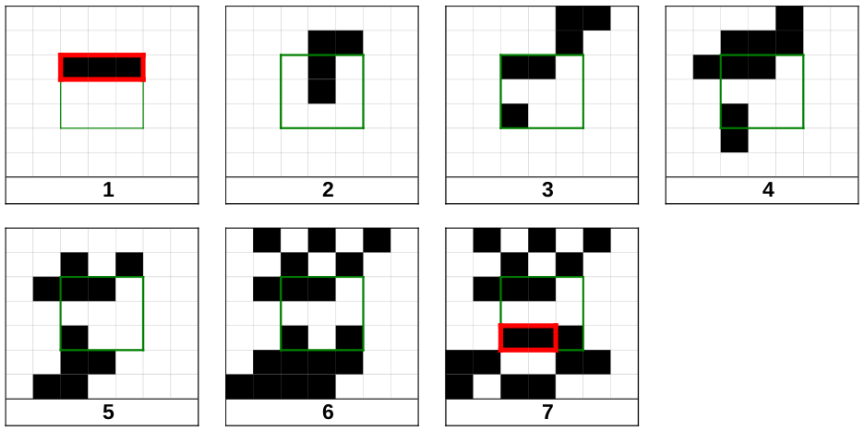
V průběhu experimentování se podařilo najít perfektní řešení problému návrhu úplné jednobitové sčítačky. Modelová situace práce automatu pro vstup $(1, 1, 1)$ je znázorněna na obrázku 5.10, na němž automat po 6 krocích vygeneruje požadovaný výstup $(1, 1)$. Grafická podoba instrukcí výsledného řešení je uvedena na obrázku 5.11.

Bezchybné řešení se podařilo nalézt rovněž pro úlohu dvoubitové sčítačky. Modelová situace práce automatu pro vstup $(1, 1, 0, 0)$ je znázorněna na obrázku 5.12, na němž automat po 6 krocích vygeneruje požadovaný výstup $(0, 1, 1)$. Grafická podoba instrukcí výsledného řešení je uvedena na obrázku 5.13.

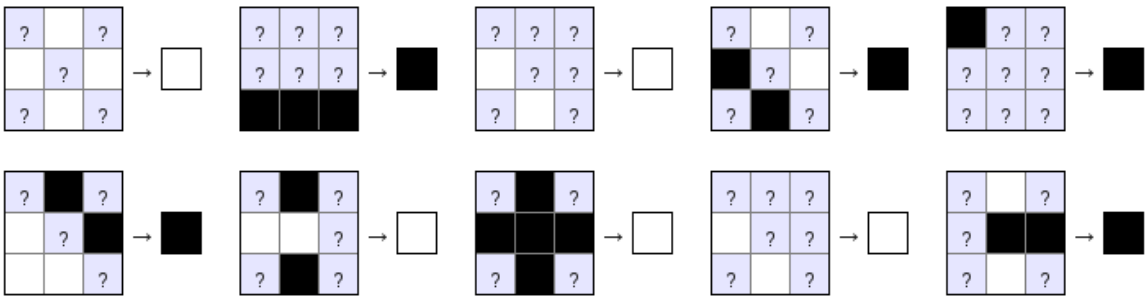
Experimenty hledající řešení pro větší než dvoubitovou sčítačku bohužel již nedokázaly nalézt perfektní řešení. Tento neúspěch by možná mohl být vyřešen navržením jiné sady instrukcí, z kterých se sestavuje výsledná sekvence. Rovněž by mohlo být zajímavé vyzkoušet

využít k řešení neuniformní celulární automat.

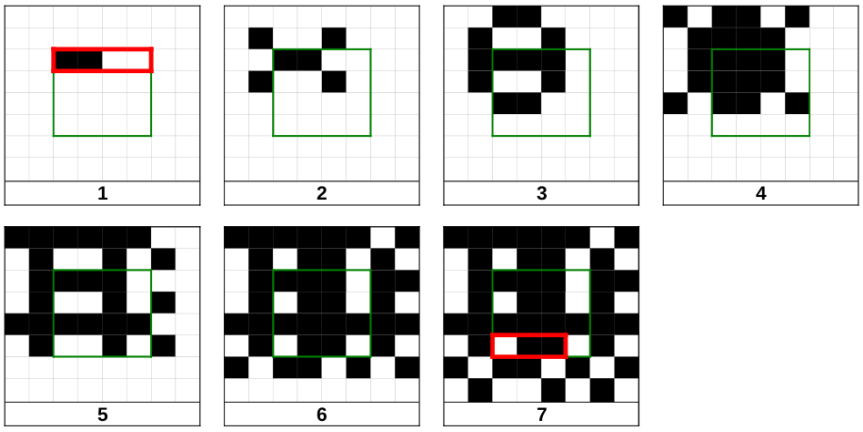
Jelikož se nepodařilo nalézt literaturu, jež by se dostatečně detailně věnovala návrhu kombinačních logických obvodů v podobě úplné jednobitové sčítačky či dvoubitové sčítačky, a která by k tomuto účelu využívala celulární automat řízený tabulkovou metodou, chybí v tomto testu srovnání obou přístupů.



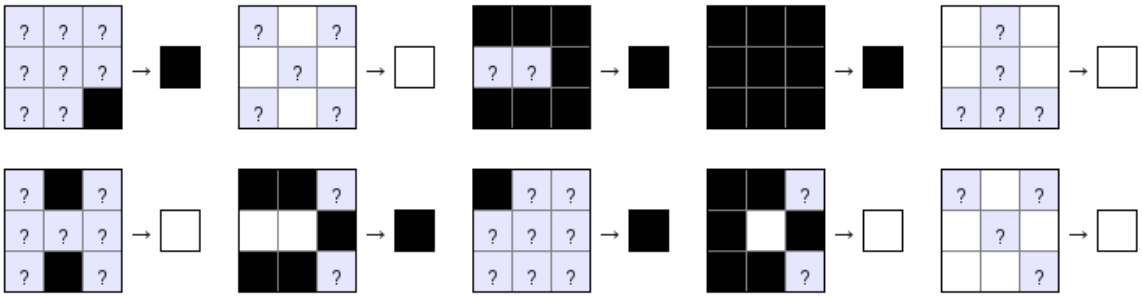
Obrázek 5.10: Problém návrhu kombinačních logických obvodů: Ukázka činnosti dvourozměrného celulárního automatu simulujícího práci úplné jednobitové sčítačky pro vstup $(1, 1, 1)$. Zelené ohraničení značí oblast s rozměry 3×3 buňky, která by teoreticky měla být sama o sobě postačující k provedení výpočtu – vstup je přikládán na první řádek této oblasti, výstup je čten z prvních dvou buněk třetího řádku této oblasti (černá buňka reprezentuje logickou jedničku, bílá buňka reprezentuje logickou nulu). Ostatní buňky se řídí stejnou sekvencí instrukcí a mají simulovat okolí, do kterého je automat zasazen.



Obrázek 5.11: Problém návrhu kombinačních logických obvodů: Grafická podoba instrukcí nejlepšího nalezeného řešení pro simulaci práce úplné jednobitové sčítačky. Řešení je reprezentováno 10 instrukcemi (pořadí jejich aplikace na okolí vyšetřované buňky je zleva doprava a shora dolů). Znak `?` vyjadřuje, že na stavu dané buňky pro vyhodnocení podmínky nezáleží.



Obrázek 5.12: Problém návrhu kombinačních logických obvodů: Ukázka činnosti dvourozměrného celulárního automatu simulujícího práci dvoubitové sčítačky pro vstup $(1, 1, 0, 0)$. Zelené ohraničení značí oblast s rozměry 4×4 buňky, která by teoreticky měla být sama o sobě postačující k provedení výpočtu – vstup je přikládán na první řádek této oblasti, výstup je čten z prvních třech buněk čtvrtého řádku této oblasti (černá buňka reprezentuje logickou jedničku, bílá buňka reprezentuje logickou nulu). Ostatní buňky se řídí stejnou sekvencí instrukcí a mají simulovat okolí, do kterého je automat zasazen.



Obrázek 5.13: Problém návrhu kombinačních logických obvodů: Grafická podoba instrukcí nejlepšího nalezeného řešení pro simulaci práce dvoubitové sčítačky. Řešení je reprezentováno 10 instrukcemi (pořadí jejich aplikace na okolí vyšetřované buňky je zleva doprava a shora dolů). Znak `?` vyjadřuje, že na stavu dané buňky pro vyhodnocení podmínky nezáleží.

5.6 Další výsledky experimentů

Mimo prezentované výsledky byla nalezena další řešení uvedených problémů. Jejich kvalita je z pohledu celkové fitness srovnatelná či horší než u výsledků demonstrovaných v této kapitole – cílem bylo do textu práce uvádět řešení dosahující nejlepších výsledků. Kompletní souhrn je možné nalézt na přiloženém CD v adresáři `results`. Jednotlivá řešení pak lze zobrazit a analyzovat ve vyvinutém programu (návod k ovládání je umístěn v příloze B).

Kapitola 6

Závěr

V rámci této práce byl navržen nový koncept řízení celulárního automatu založený na instrukcích. Schopnost tohoto konceptu byla zkoumána na několika klasických benchmarkových úlohách – majoritě, synchronizaci, samoorganizaci a návrhu kombinačních logických obvodů.

S ohledem na výsledky experimentování na uvedených problémech lze konstatovat, že koncept instrukcemi řízeného celulárního automatu vykazuje vyšší úspěšnost při hledání řešení než obvykle používaná tabulková metoda. Při srovnání výsledků řešení úloh majority a synchronizace na mřížkách s rozměry od 4 x 4 do 15 x 15 buněk totiž vykazoval přístup založený na instrukcích v průměru přibližně o 8.6%, respektive o 3.1% vyšší úspěšnost. Zároveň bylo na těchto úlohách zjištěno, že instrukcemi řízený celulární automat není tak náchylný na problém škálovatelnosti, neboť u něj není pozorovatelný tak výrazný rozdíl mezi nejvyšší úspěšností dosaženou na největší a nejmenší testované mřížce – zatímco pro instrukcemi řízený celulární automat tento rozdíl činil 3.3%, respektive 3.4%, u celulárního automatu řízeného pravdivostní tabulkou činil 5.2%, respektive 4.5%.

Výsledky experimentů zaměřených na problém samoorganizace ukázaly, že navržený přístup je schopen se efektivně vypořádat i s úlohami vyžadujícími větší počet stavů na buňku. Tento typ úloh je přitom problematický pro celulární automaty řízené pravdivostní tabulkou, neboť jejich evoluce je typicky prováděna prostřednictvím genetického algoritmu, který se snaží najít vhodné výstupní stavy pro všechny přípustné konfigurace buněk v sousedství. Počet takových kombinací však roste exponenciálně s počtem možných stavů. Instrukcemi řízený celulární automat dokázal při řešení této úlohy bezchybně vyvinout vzor francouzské vlajky do rozměru mřížky 9 x 9 buněk a při rozměrech 21 x 21 buněk vykazovalo nejlepší nalezené řešení stále přijatelnou úspěšnost 0.896.

Řešený problém poskytuje dostatek prostoru pro další výzkum. Kupříkladu by bylo možné začlenit do sady instrukcí rovněž další operace prováděné na buňkách v sousedství. Kromě již implementované modifikace spočívající v řádkovém posunu předpisu buněčného okolí by mohla být instrukční sada obohacena o množinu dalších instrukcí podobného typu, například: `seřaď daný řádek dle hodnot stavů` nebo `rotuj buňky v sousedství`. Jiné pojetí instrukcí by mohlo využívat aritmetických operací, například: `změň stav centrální buňky na X pokud je součet stavů v horní trojici větší než Y`. Další možnosti výzkumu nabízí změna použité evoluční výpočetní techniky, kdy by mohlo být zajímavé sledovat, zda-li není možné nalézt efektivní množinu instrukcí například pomocí genetického programování konstruujícího instrukce ve formě stromových struktur.

Literatura

[1] Bidlo, M.; Vašíček, Z.: Gate-Level Evolutionary Development Using Cellular Automata. In *2008 NASA/ESA Conference on Adaptive Hardware and Systems*, IEEE Computer Society Press, 2008, s. 11–18.

[2] Bidlo, M.; Vašíček, Z.: Investigating Gate-Level Evolutionary Development of Combinational Multipliers Using Enhanced Cellular Automata-Based Model. In *Proc. of 2009 IEEE Congress on Evolutionary Computation*, IEEE Computational Intelligence Society, 2009, s. 2241–2248.

[3] Burks, A. W.: *Essays on Cellular Automata.* Urbana, Illinois: University of Illinois Press, 1970.

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

[5] Davis, L.: *Genetic Algorithms and Simulated Annealing.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1987.

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

[7] Goldberg, D. E.: *Genetic Algorithms in Search, Optimization and Machine Learning.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[8] Highfield, R.; Coveney, P.: *Frontiers of Complexity: The Search for Order in a Chaotic World.* Faber and Faber, 1995.

[9] Holland, J. H.: *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[10] Hynek, J.: *Genetické algoritmy a genetické programování.* Grada Publishing, a.s., 2008.

[11] Kumar, S.; Bentley, P. J.: On Growth, Form and Computers. Elsevier Academic Press, 2003.

[12] Land, M.; Belew, R. K.: No perfect two-state cellular automata for density classification exists. *Physical Review Letters*, ročník 74, 1995: s. 5148–5150.

[13] Langton, C.: Studying artificial life with cellular automata. *Physica D*, ročník 2, č. 1-3, 1986: s. 120–149.

[14] Larranaga, P.; Kuijpers, C. M. H.; Murga, R.; aj.: Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators. *Artificial Intelligence Review*, ročník 13, 1999: s. 129–170.

[15] Mařík, V.; Štěpánková, O.; Lažanský, J.: *Umělá Inteligence (3)*. Academica, 2001.

[16] Miller, J. F.; Thomson, P.: Cartesian Genetic Programming. In *Proceedings of the 3rd European Conference on Genetic Programming*, ročník 1802, Springer Verlag, 2000, s. 121–132.

[17] Mitchell, M.; Das, R.; Crutchfield, J. P.; aj.: Evolving Globally Synchronized Cellular Automata. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, Morgan Kaufmann, 1995, s. 336–343.

[18] Mitchell, M.; Hraber, P. T.; Crutchfield, J. P.: Revisiting the Edge of Chaos: Evolving Cellular Automata to Perform Computations. *Complex Systems*, ročník 7, 1993: s. 89–130.

[19] von Neumann, J.: The general and logical theory of automata. In *Collected Works*, editace A. H. Taub, Elmsford, New York: Pergamon Press, 1963, s. 288–328.

[20] Ormerod, C.: Cellular Automata as Automata. *Paradox*, 2009.

[21] Roggen, D.; Federici, D.: Multi-Cellular Development: Is There Scalability and Robustness to Gain? In *Proceedings of Parallel Problem Solving from Nature 8, Parallel Problem Solving from Nature (PPSN) 2004*, Springer Verlag, 2004, s. 391–400.

[22] Schaffer, J. D.; Caruana, R.; Eshelman, L. J.; aj.: A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization. In *ICGA*, 1989, s. 51–60.

[23] Sipper, M.: *Evolution of Parallel Cellular Machines: The Cellular Programming Approach*. Springer-Verlag, 1997.

[24] de Sá, P. G.; Maes, C.: The Gacs-Kurdyumov-Levin automaton revisited. *Journal of Statistical Physics*, ročník 67, 1992: s. 507–522.

[25] Ulam, S.: Random processes and transformations. In *Proceedings of the International Congress of Mathematicians*, ročník 2, Providence, Rhode Island: American Mathematical, 1952, s. 264–275.

[26] Wolfram, S.: *A New Kind of Science*. Wolfram Media, 2002.

Příloha A

Obsah CD

Přiložené CD má následující strukturu:

`/doc/`	programová dokumentace
`/program/install.sh`	instalační skript
`/program/readme.txt`	popis instalace programu
`/program/src.zip`	zdrojové soubory programu
`/results`	kompletní souhrn řešení jednotlivých úloh
`/text/dp.pdf`	text diplomové práce
`/text/src/`	zdrojové soubory textu diplomové práce (LaTeX)

Příloha B

Návod k použití

Program je rozdělen do následujících čtyř částí:

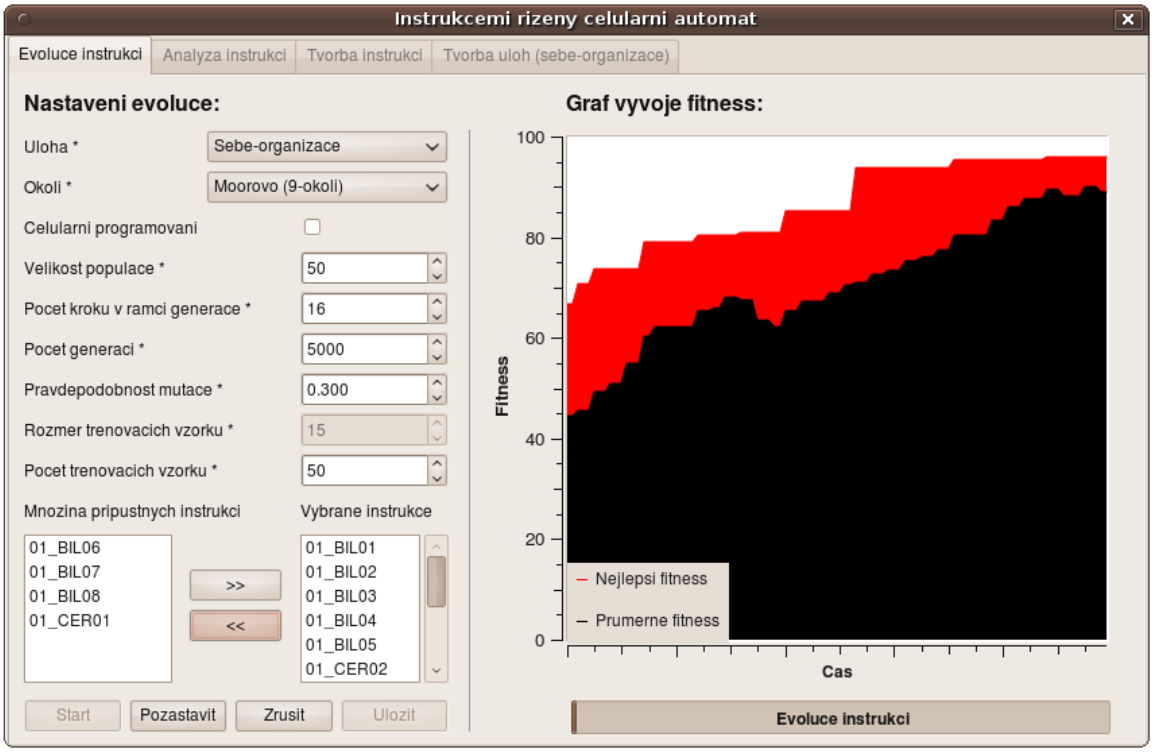
- **Evoluce instrukcí** – slouží k nalezení efektivní posloupnosti instrukcí pro řešení zvolené úlohy.
- **Analýza instrukcí** – slouží k analýze a testování funkčnosti vyvinutých řešení – posloupností instrukcí.
- **Tvorba úloh** – slouží k vytvoření nové úlohy pro problém sebe-organizace (dvojice počáteční konfigurace – cílová konfigurace).
- **Tvorba instrukcí** – slouží k návrhu instrukcí pro jednotlivé úlohy.

B.1 Evoluce instrukcí

Záložka "Evoluce instrukcí" (viz obrázek B.1) slouží k nalezení co nejlepšího řešení zvolené úlohy. Takové řešení je reprezentováno posloupností instrukcí, přičemž k jejich výběru a optimalizaci jejich pořadí se používá genetický algoritmus. Kvalita řešení závisí jednak na navržené množině instrukcí a dále také na konkrétních parametrech evoluce, přičemž metodika výpočtu hodnot parametrů pro jednotlivé úlohy je popsána v textu práce u jednotlivých experimentů uvedených v kapitole 5. Nastavení hodnot pro všechny parametry úlohy (dohromady jich je 37) lze provést manuálně v souboru `configuration.txt`. Několik nejdůležitějších však lze měnit rovněž v okně aplikace, kde je umístěn také graf znázorňující aktuální průběh evoluce. Zobrazené ovládací prvky mají následující význam:

- **Úloha** – výběr typu úlohy (majorita / synchronizace / sebe-organizace / návrh kombinačních logických obvodů).
- **Okolí** – výběr uvažovaného okolí: von Neumannovo / Moorovo.
- **Celulární programování** – zapnutí rozšíření dovolující ukládat pro každou buňku vlastní posloupnost instrukcí.
- **Velikost populace** – počet současně vyvíjených kandidátních řešení.
- **Počet kroků v rámci generace** – počet kroků automatu řízeného posloupností instrukcí daného kandidátního řešení, po kterých dojde k vyhodnocení jeho úspěšnosti.

- **Počet generací** – celkový počet iterací evolučního cyklu.
- **Pravděpodobnost mutace** – pravděpodobnost aplikace operátoru mutace na každého jedince začleňovaného do nové populace.
- **Rozměr trénovacích vzorků** – velikost mřížky automatu, na které je vyvinuté řešení testováno (má význam pouze pro úlohu majority a synchronizace).
- **Počet trénovacích vzorků** – počet náhodně vytvořených počátečních konfigurací automatu, na kterých je ověřována úspěšnost nalezeného řešení (má význam pouze pro úlohu majority a synchronizace).
- **Množina přípustných instrukcí** – kompletní množina instrukcí navržených pro řešení zvolené úlohy.
- **Vybrané instrukce** – množina vybraných instrukcí, které budou uvažovány evolucí při sestavování efektivní posloupnosti instrukcí tvořící řešení.



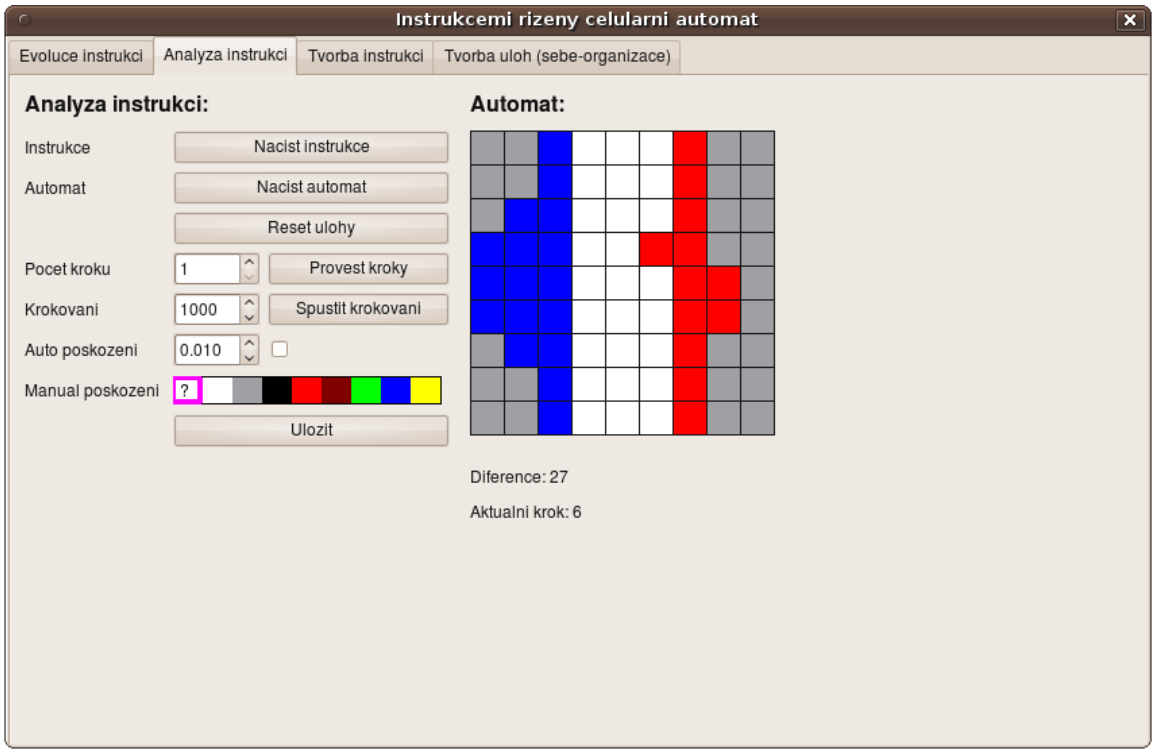
Obrázek B.1: Ukázka aplikace – záložka "Evoluce instrukcí"

B.2 Analýza instrukcí

V záložce "Analýza instrukcí" (viz obrázek B.2) lze ověřit kvalitu vyvinutého řešení sledováním chování automatu v jednotlivých krocích vývoje. Toto sledování lze provádět manuálním krokováním, nebo nastavením automatického krokování po zvoleném časovém

intervalu. V případě úlohy sebe-organizace je možné rovněž vyhodnocovat odolnost vyvinutých instrukcí vůči vlivům externího prostředí – v jednotlivých krocích vývoje lze pomocí štětce a palety barev měnit stav zvolených buněk, případně přímo nastavit procentuální část mřížky, která bude automaticky v každém kroku vývoje poškozována. Ovládací prvky této části aplikace mají následující význam:

- **Načíst instrukce** – výběr souboru s řešením (posloupností instrukcí), které bude analyzováno.
- **Načíst automat** – výběr souboru s konfigurací automatu, ve které vývoj začne (alternativně je možné tuto konfiguraci vytvořit pomocí štětce a palety barev).
- **Reset úlohy** – nastavení konfigurace automatu do výchozí podoby.
- **Počet kroků** – počet kroků vývoje automatu, který bude proveden po stisku tlačítka "provést kroky".
- **Krokování** – počet milisekund, po kterých bude automaticky proveden další krok vývinu při zapnutém automatickém krokování.
- **Automatické poškození** – velikost části mřížky, jejíž buňky v každém kroku náhodně změní svůj stav.

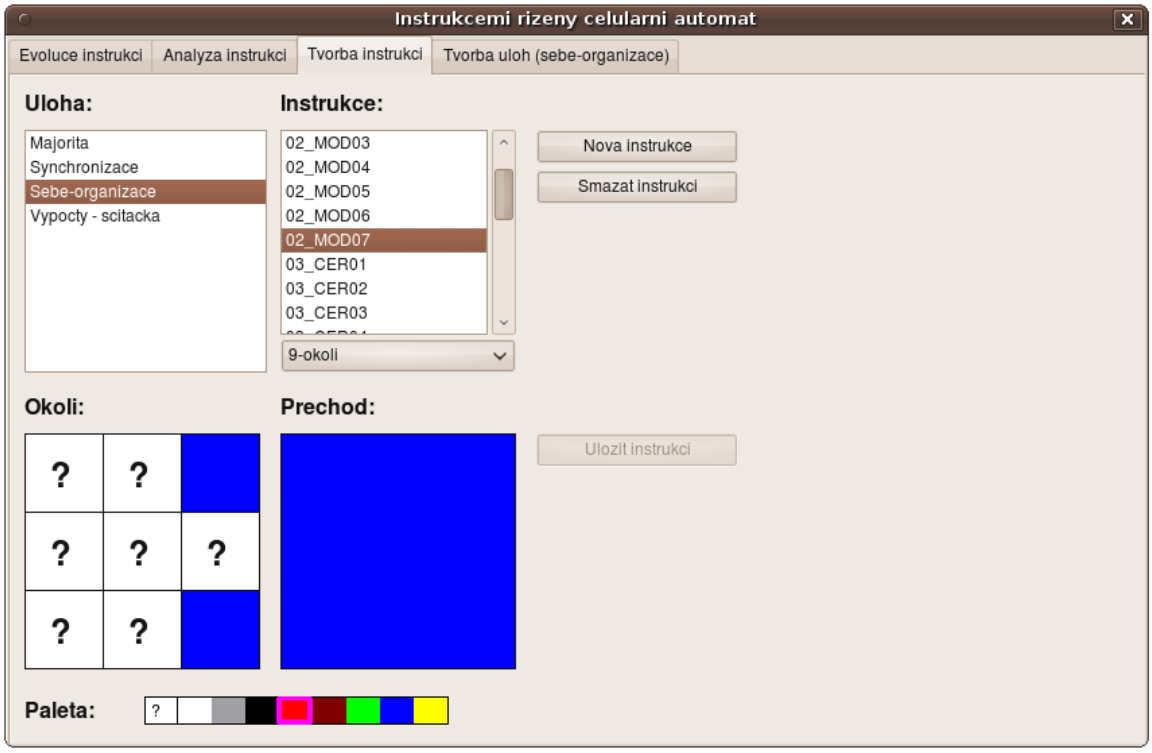


Obrázek B.2: Ukázka aplikace – záložka "Analýza instrukcí"

B.3 Tvorba instrukcí

V záložce "Tvorba instrukcí" (viz obrázek B.3) lze vytvářet instrukce použitelné při hledání řešení vybrané úlohy. Každá úloha disponuje vlastní množinou instrukcí, která se navíc liší pro Von-Neumannovo a Moorovo okolí. Instrukce se vytváří do mřížky znázorňující okolí centrální buňky pomocí štětce a palety barev. Barvy reprezentují stavy, které musí buňky v okolí mít, aby došlo k aplikaci instrukce. V paletě lze však vybrat položku se znakem ? vyjadřující, že na stavu dané buňky při rozhodování o aplikaci instrukce nezáleží. Ovládací prvky v této části mají následující význam:

- **Úloha** – výběr úlohy, pro kterou chceme manipulovat s množinou jejich instrukcí.
- **Instrukce** – výběr instrukce z množiny instrukcí zvolené úlohy (po výběru se instrukce zobrazí v spodních ovládacích prvcích "Okolí" a "Přechod").
- **Okolí** – nastavení instrukce (výběr požadovaných stavů pro jednotlivé buňky okolí).
- **Přechod** – nastavení výstupního stavu aplikovaného na centrální buňku v případě splnění předpisu instrukce.



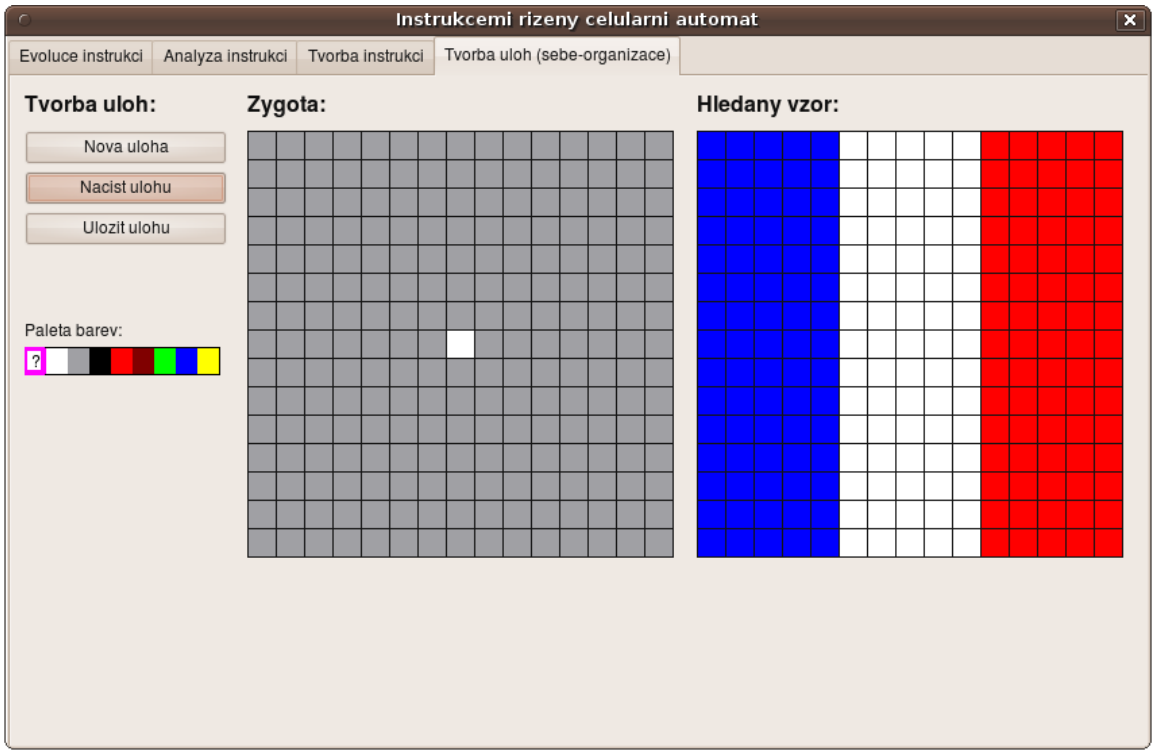
Obrázek B.3: Ukázka aplikace – záložka "Tvorba instrukcí"

B.4 Tvorba úloh

V záložce "Tvorba úloh" (viz obrázek B.4) lze vytvářet či editovat zadání úloh – dvojic počáteční konfigurace, cílová konfigurace pro problém sebe-organizace. Jejich konkrétní

podoba se vytváří pomocí štětce a palety barev. Ovládací prvky v této části mají následující význam:

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



Obrázek B.4: Ukázka aplikace – záložka "Tvorba úloh"

B.5 Instalace a spuštění programu

Instalace programu včetně systémových požadavků je detailně popsána v instalačním manuálu, který se nachází na přiloženém CD v umístění `/program/readme.txt`.

Program lze spustit s parametry příkazové řádky mající následující význam:

Synopsis: `./automaton [-u usage]`

`-u` Určuje využití programu (1 = evoluce instrukcí, 2 = analýza instrukcí, 3 = tvorba instrukcí, 4 = tvorba úloh).

Výchozí nastavení parametrů evoluce a analýzy instrukcí je určeno konfiguračním souborem `configuration.txt` nacházejícím se v kořenové složce programu. Při analýze lze využít připravených souborů s automaty a s řešeními ve složkách `example_tasks` a `example_solutions`.