VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

EMERGENTNÍ CHOVÁNÍ CELULÁRNÍCH AUTOMATŮ

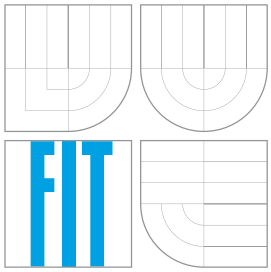
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MICHAL ŘÍHA

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

EMERGENTNÍ CHOVÁNÍ CELULÁRNÍCH AUTOMATŮ

EMERGENT BEHAVIOR OF CELLULAR AUTOMATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MICHAL ŘÍHA
AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAL BIDLO
SUPERVISOR

BRNO 2008

Abstrakt

Práce se zabývá simulací emergentního chování v celulárních automatech, konkrétně problémy majority, synchronizace a šachovnice. Pro řešení je využito evolučních algoritmů.

Klíčová slova

Emergentní chování, celulární automat, evoluční algoritmus, majorita, synchronizace, šachovnice.

Abstract

This work deals with the simulation of an emergent behavior in cellular automata. In particular, density task, synchronization task and chessboard generation problem are investigated. It uses evolutionary algorithm to solve this problem.

Keywords

Emergent behavior, cellular automaton, evolutionary algorithm, density task, synchronization task, chessboard generation.

Citace

Michal Říha: Emergentní chování celulárních automatů, bakalářská práce, Brno, FIT VUT v Brně, 2008

Emergentní chování celulárních automatů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Bidla. Uvedl jsem všechny zdroje ze kterých jsem čerpal.

.....
Michal Říha
11. května 2008

Poděkování

Děkuji vedoucímu mé práce panu Ing. Michalu Bidlovi za jeho připomínky a návrhy, kterými mě podporoval v práci.

© Michal Říha, 2008.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

Kapitola 1

Úvod

S rozvojem informačních technologií se člověk snaží řešit stále složitější problémy. Mezi ně patří také simulace přírody, umělý život a umělá inteligence. Za podstatu veškerého chování v přírodě můžeme považovat emergentní chování, jakožto základ pro společenství jednoduchých živočichů, kterými jsou například brouci a hmyz.

Tato práce se zaobírá simulací emergentního chování v celulárních automatech. K vývoji přechodových pravidel, která povedou ke znakům emergentního chování, je využito evolučních algoritmů, jakožto přístupu také inspirovaného přírodou. Jsou nastíněny 3 problémy emergentního chování, které je možné pomocí celulárních automatů řešit, spolu s dosaženými výsledky pro každý z nich.

V kapitole 2 jsou shrnuty základní poznatky o celulárních automatech a souvisejícím emergentním chování. Kapitola 3 pak popisuje základy evolučních algoritmů. Konkrétněji je nastíněn genetický algoritmus a z něj vycházející celulární programování. Kapitola 4 popisuje konkrétní jevy emergentního chování zkoumané v této práci. Pátá kapitola pak shrnuje dosažené výsledky. Šestá kapitola obsahuje závěr se shrnutím celé práce.

Kapitola 2

Celulární automaty

Celulární automaty (CA) původně zavedli Ulam a von Neumann ve čtyřicátých letech minulého století, aby poskytli formální prostředí pro vyšetřování chování složitých systémů [4]. Jedná se o matematickou idealizaci fyzických systémů, ve kterých je prostor a čas diskrétní a fyzikální veličiny nabírají konečnou množinu diskrétních hodnot [1]. CA sou tvořeny konečnou množinou buněk, z niž každá je právě v jednom stavu z konečné množiny stavů. Nový stav buňky se vyhodnotí na základě stavu, ve kterém jsou okolní buňky, podle lokálního přechodového pravidla.

Buňky jsou uspořádány do mřížky, která může mít libovolný počet rozměrů. Nejvíce používané jsou jedno, dvou nebo tří rozměrné. Čím vyšší je počet rozměrů, tím větší je výpočetní síla automatu. S počtem rozměrů totiž souvisí počet cest, kterými se může informace šířit. Zároveň však roste výpočetní náročnost takového systému.

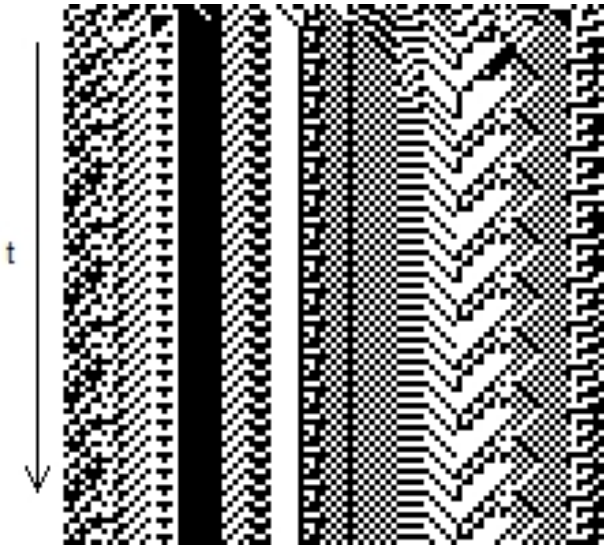
Jak již bylo řečeno dříve, každá buňka přechází do nového stavu na základě lokálního přechodového pravidla. To může být stejné ve všech buňkách celulárního automatu, nebo mohou pravidla být různá. V prvním případě mluvíme o *uniformním* celulárním automatu, v druhém pak o *neuniformním*. Za speciální případ je možné považovat *quasi-uniformní* automat, ve kterém buňky obsahují konečný, nevelký počet různých pravidel. Jako příklad je možné uvést celulární automat obsahující 100 buněk, v niž je obsaženo jen 5 různých přechodových pravidel.

Jestliže mluvíme o konfiguraci celulárního automatu, máme na mysli již konkrétní hodnoty z množiny stavů přiřazené buňkám.

2.1 Jednorozměrné celulární automaty

Jednorozměrné celulární automaty, kterými se tato práce zabývá, jsou základním a nejjednodušším případem celulárních automatů. Buňky v nich jsou uspořádány do jednoho řádku. Při jejich grafickém zobrazení jsou jednotlivé konfigurace poskládány pod sebou tak, jak po sobě v čase postupně následovaly.

Okolí buňky v jednorozměrném celulárním automatu se skládá z buněk vpravo a vlevo od sledované buňky a z ní samotné. Počet sledovaných buněk z každé strany je označován jako *rádius* r. Počet buněk v okolí lze pak vyjádřit rovnicí $2r + 1$. Z hlediska této definice je třeba rozhodnout, jak se mají chovat buňky na okraji celulární struktury. Existují dvě řešení tohoto problému, a to konstantní okrajové podmínky, kdy je za souseda okrajové buňky stanovena fiktivní buňka ve fixním stavu, nebo cyklické okrajové podmínky. V tomto případě je za levého souseda buňky na levém okraji celulárního automatu považována buňka



Obrázek 2.1: Příklad jednorozměrného celulárního automatu a jeho vývoje v čase

na pravém okraji a obráceně. Řádek si je pak možné představit jako kružnici. V této práci jsou využívány výhradně cyklické okrajové podmínky.

Příklad jednorozměrného celulárního automatu a jeho vývoje v čase je znázorněn na obrázku 2.1.

2.2 Emergentní chování

Emergentní chování je chování systému, které není explicitně popsáno chováním částí systému, a proto je pro pozorovatele nebo návrháře neočekávané [5]. Vlastnosti, které můžeme na systému pozorovat jako na celku, vznikají díky jednoduchým interakcím základních entit. Ty se nechovají podle žádných globálních pravidel. Pro celý systém je typická velká komplexnost. Bez toho by bylo snadné jeho chování předvídat.

Příklady emergentního chování můžeme sledovat v přírodě, a to jak ve fyzikálních, tak živých systémech. Z živých systémů je nejčastějším příkladem kolonie mravenců. V kolonii není žádná vyšší entita, která by udávala rozkazy. Každý mravenec reaguje pouze na chemické stopy ve svém okolí a podle svého genetického naprogramování se chová. I přesto můžeme sledovat komplexní chování celé kolonie, které vzniká jednoduchými interakcemi základních entit, tedy mravenců a jejich okolí.

Přenesení vlastností emergentnícho chování z přírody do výpočetní techniky se nazývá *Emergent computing*. Jsou zachovány nejdůležitější vlastnosti emergentních systémů, tedy jednoduché entity, které tvoří systém vykazující emergentní chování, bez jakékoliv globální kontroly systému. Typickými příklady takovýchto aplikací jsou např. neuronové sítě, agentní systémy nebo celulární automaty [2].

Celulární automaty jsou nástrojem pro demonstraci emergentního chování. Tato vlastnost je dána zejména tím, že jsou tvořeny velmi jednoduchými entitami, buňkami, které samy o sobě nejsou schopny zpracovávat složitější informace. Zároveň dochází k časté změně jejich stavu, čímž je dosaženo značné dynamiky vývoje celého systému.

Kapitola 3

Evoluční algoritmy

Princip evolučních algoritmů vznikl přenesením některých pravidel vývoje organismů v přírodě do informatiky. V přírodě spolu jedinci soupeří o možnost reprodukce na základě své síly a schopnosti přežít. Potomci pak dědí vlastnosti svých rodičů, čímž se zajišťuje vývoj populace k lepším vlastnostem pro dané prostředí.

Evoluční algoritmy využívají tohoto modelu k řešení úloh, které jsou příliš rozsáhlé a náročné pro tradiční postupy. Ve svém průběhu uvažují množinu možných řešení, ze kterých jejich kombinací dostávají lepší. Nevhodná řešení jsou postupně odstraňována.

Základním pojmem těchto algoritmů je populace chromozomů, které jsou obvykle tvořeny řetězcem symbolů [3]. Do nich jsou zakódovány vlastnosti systému. S lepší schopností řešit daný problém souvisí větší ohodnocení chromozomu hodnotou *fitness*. Lépe ohodnocené chromozomy poté mají větší šanci zúčastnit se reprodukčního procesu a tím přenést na novou generaci potomků své vlastnosti. Součástí reprodukce je také proces *mutace*, který náhodně modifikuje malé části chromozomů a dává tak možnost vzniku dalších možných řešení.

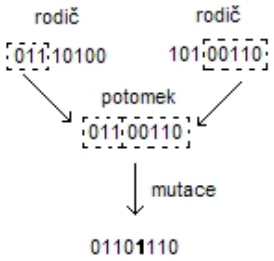
3.1 Genetický algoritmus

Nejčastěji používanou variantou evolučního algoritmu v počítačových modelech je právě genetický algoritmus. Jedná se o iterativní proceduru která zahrnuje konstantě velikou populaci jedinců, každého z nich reprezentovaného konečným řetězcem symbolů, známých jako genom, kódujících možné řešení pro daný prostor problému [4].

Postup algoritmu je následující. Nejprve je vytvořena populace jedinců. Každý z nich je poté ohodnocen fitness podle toho, jak splňuje zadaná kritéria. Následně je vytvořena nová populace. Pro její vznik jsou vybíráni jedinci podle hodnoty své fitness, přičemž platí, že čím je hodnota fitness vyšší, tím větší je pravděpodobnost, že se jedinec bude podílet na tvorbě potomků.

Při vzniku nové populace je využíváno dvou postupů známých s přírody. Jsou jimi *křížení* (crossover) a *mutace* (mutation). Oba tyto procesy probíhají s určitou pravděpodobností. Pojem křížení značí vznik potomka kombinací genomů dvou rodičů. Nejjednodušším případem je použití náhodně zvoleného bodu v genomu. Do tohoto bodu zdědí potomek stejnou část genomu prvního rodiče, od něj pak druhou část genomu druhého rodiče. Mutace pak značí proces náhodné změny malé části genomu. Postup genetického algoritmu je znázorněn na obrázku 3.1.

Pro zakódování přechodových pravidel celulárního automatu je využito reprezentace



Obrázek 3.1: Příklad křížení rodičů. Vzniká nový potomek, který je následně mutován. Náhodný bod křížení je mezi třetím a čtvrtým prvkem genomu.

okolní konfigurace jako binárního čísla. Pokud máme například pravidlo pro přechod $011- > 1$, pak konfiguraci okolí (levou stranu pravidla) můžeme chápat jako číslo 3 zapsané binárně. Toto číslo nám udává, na které bitové pozici je uložena hodnota pravé strany pravidla. Výše uvedený příklad tedy značí, že 3. bit čísla udávajícího obsah genomu je 1. Tímto způsobem můžeme snadno zacházet s pravidly jako s genomem a využít tak evoluční algoritmy. Posloupnost bitů genomu a odpovídajících pravidel je obsažena v tabulce 3.1.

Tabulka 3.1: Závislost konfigurace přechodového pravidla na bitu v genomu

bit	0	1	2	3	4	5	6	7
konfigurace	000	001	010	011	100	101	111	111

V dalším textu jsou genomy reprezentovány jako hexadecimální číslo. Pořadí jednotlivých přechodových pravidel celulárního automatu je určeno jejich binární hodnotou. Názorný příklad převodu je zachycen v tabulce 3.2, kde je ukázka přechodového pravidla 0x6D.

Tabulka 3.2: Příklad převodu přechodových pravidel na hexadecimální číslo, v tomto případě 0x6D

konfigurace	000	001	010	011	100	101	110	111
přechod	1	0	1	1	0	1	1	0

3.2 Celulární programování

V této práci je použit postup, který využívá modifikací výše popsaného genetického algoritmu. Tento přístup je nazýván *celulární programování* (cellular programing) [4].

Oproti klasickému genetickému algoritmu není brána v potaz populace všech pravidel, ale pouze pravidla v okolí každé buňky. Rodiče tak mohou být vybráni pouze z $2r + 1$ buněk. Pokud se v tomto okolí vyskytují pravidla s nízkým ohodnocením fitness v porovnání s více vzdálenými pravidly, budou i přesto vybrána.

Pro každou buňku probíhá evoluce zvlášť. Jestliže má buňka nejvyšší ohodnocení ze svého okolí, zůstanou její přechodová pravidla nezměněna. Pokud v okolí existuje pouze jedna buňka s vyšší hodnotou fitness, převezme původní buňka její přechodová pravidla. V případě výskytu více buněk v okolí s lepším ohodnocením fitness budou nová přechodová

pravidla v uvažované buňce vytvořena křížením dvou náhodně vybraných lepších přechodových pravidel.

Důležitým aspektem je také to, že nová přechodová pravidla jsou do buňky zapsána ihned po evoluci. Následující evoluce okolních buněk tak již pracuje s těmito novými pravidly. Můžeme tak dosáhnout rychlého rozšíření pravidel a zkrácení doby potřebné pro to, aby měla příležitost projít celou populací.

Po každé proběhlé evoluci je spuštěn proces mutace, který s pravděpodobností 0,01% invertuje hodnotu bitu. Následně je vynulována fitness všech buněk.

Pseudokód algoritmu celulárního programování, inspirovaného [4], můžeme vidět na obrázku 3.2.

```
počet_projítých_buněk = 0
while(počet_projítých_buněk <= počet_všech_buněk){
  inicializuj přechodová pravidla buňky
  fitness = 0
  počet_projítých_buněk += 1
}
počet_nových_konfigurací = 1
while(počet_nových_konfigurací <= maximální_počet_nových_konfigurací){
    vytvoř novou konfiguraci
    proveď běh pro danou konfiguraci
    vyhodnoť fitness
    if(počet_nových_konfigurací % evoluce_po == 0){
     počet_projítých_buněk = 0
        while(počet_projítých_buněk <= počet_všech_buněk){
            n = počet sousedů s vyšší fitness
            if(n == 0)
             pokračuj
            else if(n == 1)
             nahraď genom v aktuální buňce genomem z buňky s lepší fitness
            else if(n == 2)
             nahraď genom v aktuální buňce křížením genomů buněk s lepší fitness
            else if(n > 2)
             náhodně vyber dvě buňky s lepší fitness a nahraď genom v aktuální
             buňce jejich křížením

         počet_projítých_buněk += 1
        }
    proveď mutaci pro všechny buňky
    vynuluj fitness všech buněk
    }
}
```

Obrázek 3.2: Pseudokód modifikovaného algoritmu celulárního programování tak, jak je v této práci využit. Původní návrh vychází z [4].

Kapitola 4

Projevy emergentního chování v celulárních automatech

Tato kapitola se zabývá některými typickými projevy emergentního chování v celulárních automatech, jakými jsou například problém majority nebo synchronizace. Dalším problémem je pak generování šachovnice. Tento problém není tak význačný jako předchozí dva, jeho specifikace je navržená pro tuto práci.

4.1 Majorita

Majorita je základním netriviálním problémem řešeným pomocí celulárních automatů. Její podstatou je dostat automat do stavu, kdy obsahuje pouze samé nuly, pokud byl v počáteční konfiguraci počet nul větší než jedniček, případně do stavu, kdy obsahuje pouze jedničky, pokud byl v počáteční konfiguraci počet nul menší než počet jedniček.

Pro formální definici je třeba si zavést konstantu λ. Ta značí počet jedniček v počáteční konfiguraci, vyjádřeno jako desetinné číslo značící procenta. Problém majority pak lze popsat následujícím způsobem [1]:

> Najdi přechodová pravidla, která pro danou počáteční konfiguraci CA s lichým počtem buněk a v daném počtu iterací přejdou do stavu 'samé nuly', pokud je $\lambda < 0,5$, a do stavu 'samé jedničky', pokud je $\lambda > 0,5$. Stav 'samé nuly' je stav, kdy je každá buňka celulárního automatu ve stavu nula, stav 'samé jedničky' je stav kdy je každá buňka ve stavu jedna.

Při výpočtu majority je třeba si uvědomit důležitost počáteční konfigurace. Při uniformním rozložení pravděpodobnosti je vysoký počet konfigurací s λ blízkou 0,5, které jsou pro automat těžko zařaditelné [1]. Pro řešení majority jsou tyto konfigurace nevhodné, pro celulární automat jsou nejednoznačné.

Pro účely této práce si tak zavedeme omezení počátečních konfigurací, se kterými budeme pracovat. Konfigurace je považována za vhodnou v následujícím případě (hodnota byla získána experimentálně):

$$\lambda \leq \frac{(N/2)-(N/8)}{100} \vee \lambda \geq \frac{(N/2)+(N/8)}{100}$$

N značí celkový počet buněk celulárního automatu.

Dalším problémem je běh celulárního automatu mezi dvěma po sobě jdoucími evolucemi. Může nastat situace, kdy v průběhu hodnocení fitness dochází k příliš častému generování konfigurací, které obsahují majoritu jedniček, popřípadě nul. Pravidla jsou pak hodnocena bez ohledu na svou schopnost řešit majoritu a nastává situace, kdy získáváme fitness, která neodpovídá našemu zadání.

Při běhu proto budeme hlídat rovnováhu mezi konfiguracemi s majoritou jedniček a nul. Maximální rozdíl mezi nimi je 10%. Tato hodnota je navržena pro dosažení velmi úzkých odchylek a ověřena experimentálně.

Úspěšné řešení problému majority je zachyceno na obrázku 4.1.

Obrázek 4.1: Úspěšné řešení problému majority, $r = 1$, $\lambda = 0,61$

4.2 Synchronizace

Synchronizace je druhým ze základních netriviálních problémů řešeních pomocí celulárních automatů. Jejím cílem je v průběhu automatu dosáhnout toho, aby se v jednotlivých krocích střídal stav, kdy automat obsahuje samé jedničky, a stav, kdy obsahuje samé nuly.

Pro formálnější definici je opět nutné použít koeficient λ tak, jak byl zaveden v případě majority. Zároveň si problém pro potřeby této práce upravíme, aby bylo možno navázat na problém majority. Definice pro modifikovaný problém synchronizace je pak následující:

> Najdi taková přechodová pravidla, která pro danou počáteční konfiguraci CA s lichým počtem buněk v každé z nich budou procházet stavy $1- > 0- > 1- > 0$, pokud je $\lambda > 0,5$, nebo stavy $0- > 1- > 0- > 1$ při $\lambda < 0,5$.

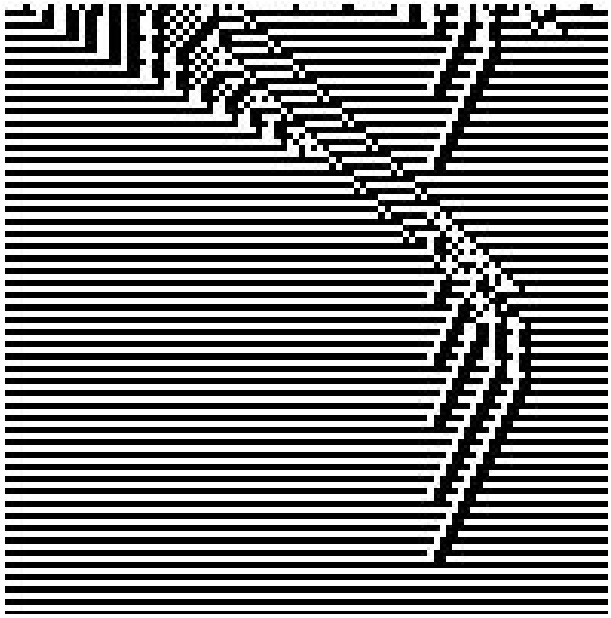
Oproti běžnému popisu synchronizace obsahuje tato definice i podmínku týkající se toho, jaké bude pořadí střídání buněk v závislosti na počáteční konfiguraci. Takto specifikován je tento problém o něco složitější.

Pro počáteční konfigurace využijeme pravidla popsaná v kapitole o majoritě. Budou platit stejná omezení λ. Také bude platit maximální rozdíl mezi počtem konfigurací obsahujících více jedniček a konfiguraci s více nulami, který je taktéž 10%.

Zásadní rozdíl však bude ve výpočtu fitness. Z definice vyplývá, že pokud je v počáteční konfiguraci $\lambda > 0,5$, měly by se střídat stavy $1- > 0- > 1- > 0$, pro $\lambda < 0,5$ pak stavy $0- > 1- > 0- > 1$. Jestliže však hodnotíme fitness pouze v konečném stavu běhu automatu se zadanou konfigurací, ohodnocování buněk se může tvářit stejně jako při majoritě. Vezmeme-li v úvahu automat, jehož fitness se vyhodnocuje po 100 krocích, bude pro $\lambda > 0,5$ ohodnocena každá buňka ve stavu 1 zvýšením fitness, při $\lambda < 0,5$ pak každá buňka ve stavu 0. Jak je vidět, toto přímo odpovídá problému majority. Nic nenutí celulární automat aby 'blikal' mezi stavy 0 a 1.

Pro řešení tohoto problému je využijeme proměnnou délku běhu pro každou novou konfiguraci, přičemž rozhoduje to, zda je délka běhu sudá či lichá. Střídáním lichých a sudých délek běhu pro konfiguraci s větším počtem jedniček určujeme, že požadujeme střídání stavů. To samé samozřejmě platí i pro konfiguraci s větším počtem nul.

Příklad úspěšného vyřešení problému synchronizace celulárním automatem můžeme vidět na obrázku 4.2.



Obrázek 4.2: Úspěšné řešení problému synchronizace, $r = 2$, $\lambda = 0,37$

4.3 Šachovnice

Problém šachovnice částečně navazuje na synchronizaci. Úkolem je dosáhnout toho, že každá buňka bude mít hodnotu opačnou než její sousedé. Ti musí mít hodnotu naopak stejnou. Zjednodušeně se dá říct, že řada buněk jednorozměrného celulárního automatu obsahuje konfiguraci 01010101, popřípadě 10101010. Problém si pro tuto práci nadefinujeme takto:

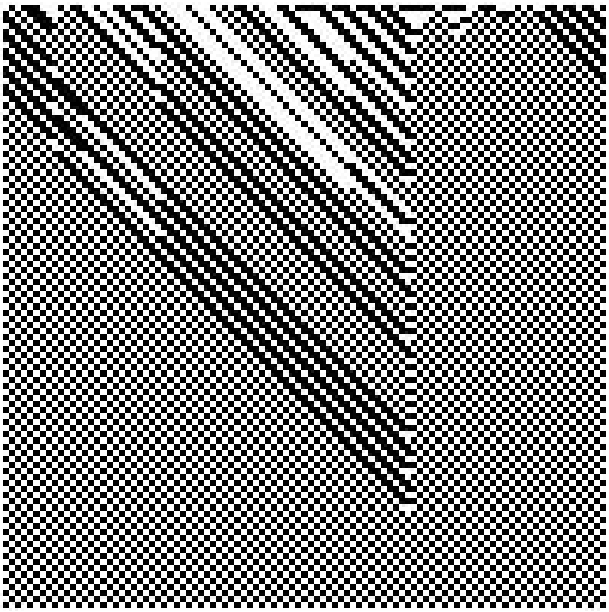
Najdi taková pravidla celulárního automatu, která pro každou počáteční konfiguraci celulárního automatu se sudým počtem buněk vyústí ve stav, kdy každá

buňka bude obsahovat hodnotu opačnou než bezprostředně sousední buňky vlevo a vpravo. Stavy sousedních buněk bude stejný. Hodnota buňky se přitom bude střídat v každém kroku. Pokud je $\lambda > 0,5$, bude posloupnost hodnot v první buňce $1- > 0- > 1- > 0$, pří hodnotě $\lambda < 0,5$ bude posloupnost $0- > 1- > 0- > 1$.

Na rozdíl od obou předchozích popsaných problému musí celulární automat v tomto případě obsahovat sudý počet buněk. To je zapříčiněno cyklickými okrajovými podmínkami. Při řešení dále vycházíme z poznatků z výše popsaných problému majority a synchronizace. Použijeme tedy omezení λ popsané v kapitole 4.1 o majoritě a proměnnou délku běh tak, jak je popsáno v kapitole 4.2 o synchronizaci.

Zásadní rozdíl oproti předchozím problémům je v hodnocení fitness. Zatímco u majority a synchronizace se vyhodnocovala fitness na konci běhu každé nové počáteční konfigurace, zde je tento způsob nedostatečný. Důvodem je nejednoznačnost takovéhoto hodnocení vzhledem k zadanému problému. Takto získaná pravidla mají tendenci řešit problém synchronizace více než problém šachovnice. Pro dosažení lepšího chování je nutné zavést dvojí hodnocení fitness pro každý běh, a to v předposledním a posledním kroku běhu. S touto přidanou informací pak snáze dosáhneme řešení šachovnice, což bylo ověřeno i v provedených experimentech.

Ukázku úspěšného vyřešení problému generování šachovnice lze vidět na obrázku 4.3.



Obrázek 4.3: Úspěšné řešení šachovnice, $r = 1$

Kapitola 5

Dosažené výsledky

Celulární automat byl nastaven tak, že obsahoval 99 nebo 100 buněk, v závislosti na řešeném problému. Běh pro jednu konfiguraci trval 100 kroků. Evoluce probíhala po každých 300 nově vygenerovaných počátečních konfiguracích. Celkově automat provedl generaci 60000 počátečních konfigurací. Při tomto nastavení postupně proběhne 200 evolucí pravidel, což se ukázalo jako dostatečné pro získání výsledků. Veškeré počáteční konfigurace v průběhu automatu byly generovány s omezením λ.

V závislosti na řešeném problému se vyskytlo také množství řešení, která nesplňovala požadavky zadání. To je způsobeno tím, že buňky daného celulárního automatu mohou nabývat pouze stavů 0 a 1. Z hlediska pravděpodobnosti tedy není nutné, aby špatné pravidlo skončilo ve špatném stavu, stejně tak jako dobré pravidlo ve správném stavu. Tyto vlastnosti je možné částečně omezit velkým počtem kroků při simulaci. Pokud je však pravidlo, byť špatné, ze začátku ohodnoceno velmi vysokou fitness, rozšíří se do všech buněk automatu. Ten se pak z tohoto stavu ke správnému řešení již nedostane.

Testování jednotlivých řešení probíhalo na 10000 náhodných konfiguracích. Opakování jednotlivých konfigurací nebylo nijak ošetřeno, bylo tedy možné. Vyhodnocování správnosti řešení pak probíhalo kontrolou posledních dvou kroků každého běhu, přičemž se sčítal počet buněk ve stavu, který nebyl správný. Výsledky jsou pak zapsány jako počty konfigurací, které skončily tak, že žádná buňka nebyla ve špatném stavu, nebo bylo maximálně 5%, 10% a 20% buněk bylo ve špatném stavu. Stav, který byl zahrnut v do lepšího hodnocení, se k horšímu již nepřičítal. Příklad výpočtu: V automatu majícím 100 buněk bylo v předposledním kroku špatně 5 buněk, v posledním pak 7. Celkem bylo špatně 12 stavů buněk z 200 možných, odchylka tedy spadá do kategorie 10%.

Pro testování výsledků byly použity dva přístupy. První z nich využívá pro generaci testovacích konfigurací omezení λ tak, jak byla využita při evoluci. Druhý pak generuje počáteční konfigurace bez omezení.

5.1 Majorita

Při řešení problému majority bylo dosaženo několika zajímavých výsledků. Dosažená řešení ukazují obecné principy chování celulárních automatů, které jsou základem pro úspěšné zvládnutí dalších úloh. Pro majoritu bylo spuštěno 50 evolučních experimentů.

Základní úvahou pro přechodová pravidla jednorozměrného celulárního automatu s $r = 1$ při řešení problému majority je přejít do stavu 1, pokud je většina buněk v okolí ve stavu 1, jinak přejít do stavu nula. S využitím tohoto pravidla vzniká automat, jehož buňky

obsahují pouze jedno, stejné pravidlo, a to 0xE8. Z neuniformního automatu se nám evolucí pak stává automat pseudo-uniformní. Tento výsledek evoluce byl velmi častým i přesto, že jeho úspěšnost je minimální.

Při bližším prozkoumání pravidla zjistíme jeho nedostatek, a to nemožnost šíření informace. Pravidlo zredukuje počáteční konfiguraci na větší shluky jedniček a nul, avšak mezi nimi již nedokáže dále rozšiřovat informaci o majoritě. Typické chování tohoto pravidla je znázorněno na obrázku 5.1.

Lepšího chování, vykazujícího některé důležité rysy pro vyřešení problému majority, bylo dosaženo v případě, kdy automat obsahoval 5 různých pravidel. Můžeme tedy říct že se jednalo o automat quasi-uniformní. Pravidla v něm obsažená jsou zachycena v tabulce 5.1.

Tabulka 5.1: Pravidla celulárního automatu pro řešení majority

buňky	0-23,89-98	24-47	48-54	55-79	80-88
pravidla	0xF0	0xAA	0xEA	0xE2	0xE8

Chování tohoto automatu je znázorněno na obrázku 5.2. Je vidět že řešení pro tuto počáteční konfiguraci není optimální, neboť buňky na středu příliš propagují jedničku, a zamezují případnému průchodu 0 do levé části. Můžeme si také povšimnout dvou oblastí buněk, u kterých se v počáteční konfiguraci nevyskytovala jasná majorita jedniček nebo nul, a proto zůstávají v neutrálním stavu. Ten změní až po propagaci jedničky nebo nuly ze strany.

Z výsledků testů pro vygenerované počáteční konfigurace, které byly omezeny podle λ stejně jako při evoluci, vidíme, že i přes to je úspěšnost velmi dobrá. Výsledky testů jsou zachyceny v tabulce 5.2.

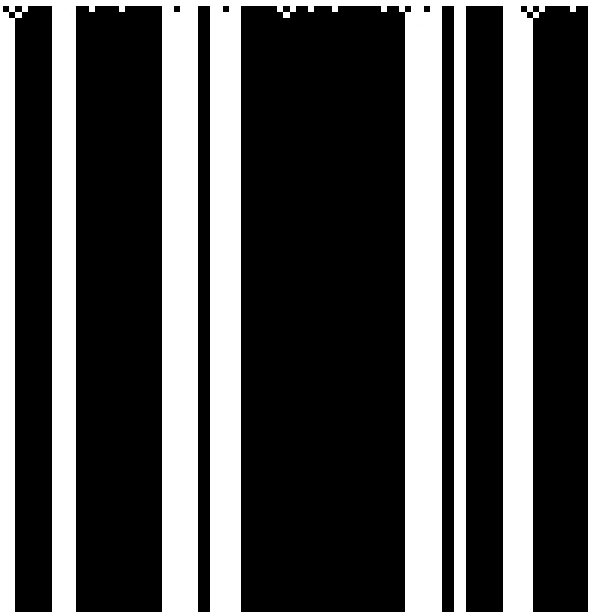
Tabulka 5.2: Výsledky testů pro pravidlo z tabulky 5.1, konfigurace omezené podle λ. Maximální chyba je měřena na posledních dvou konfiguracích každého běhu. V případě 0% byly všechny buňky v obou konfiguracích ve správném stavu. Ostatní případy značí, kolik se vyskytovalo špatných stavů (vyjádřeno v procentech). Bližší popis viz. úvod do kapitoly. Procentuální vyjádření pak ukazuje v procentech, kolik stavů skončilo s danou chybou.

Maximální chyba	Počet konfigurací	Procentuální vyjádření
0%	2166	21,66%
5%	1355	13,55%
10%	415	4,15%
20%	1243	12,43%
více	4821	48,21%

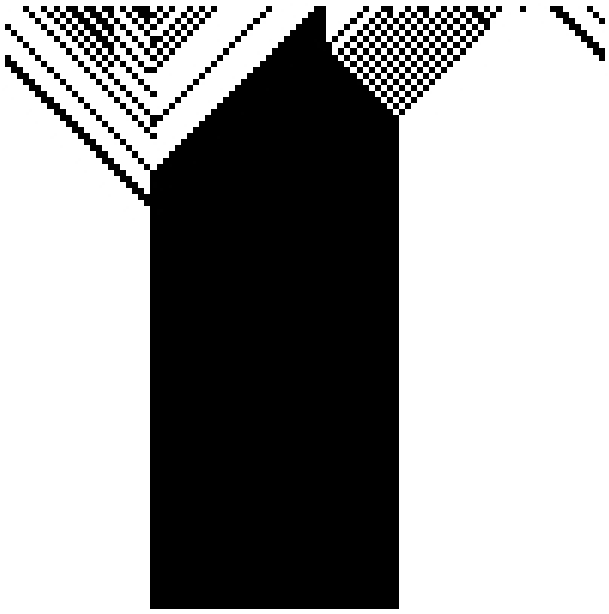
Výsledky testů pro počáteční konfigurace bez omezení jsou v tabulce 5.3. Je vidět že výsledky jsou horší než v prvním případě. Jelikož pravidla nebyla pro takovéto konfigurace evoluována, neměla možnost se správně naučit jak je vyřešit.

Tabulka 5.3: Výslekdy testů pravidla z tabulky 5.1, bez omezení počátečních konfigurací.

Maximální chyba	Počet konfigurací	Procentuální vyjádření
0%	923	9,23%
5%	771	7,71%
10%	49	0,49%
20%	1058	10,58%
více	7199	71,99%



Obrázek 5.1: Příklad chování celulárního automatu, jehož veškeré buňky obsahují pravidlo 0xE8.



Obrázek 5.2: Chování automatu obsahujícího pravidla zobrazená v tabulce 5.1 v náhodně vybrané konfiguraci, $\lambda = 0,37$.

5.2 Synchronizace

V případě řešení problému synchronizace byly dosažené výsledky více úspěšné než v případě majority. U většiny řešení se vyskytovaly shluky buněk, které se chovaly dle požadovaného schématu. Byly však odděleny buňkami, které nebyly schopny se chování okolí přizpůsobit. Podobné chování lze vidět na obrázku 5.3. Celkem bylo pro synchronizaci spuštěno 20 testů.

Prvním podrobněji zkoumaným výsledkem je automat obsahující pravidla zapsaná v tabulce 5.4. Příklad jeho chování je na obrázku 5.4. Nejčastějším pravidlem je zde 0x1B, které je obsažené v téměř polovině buněk.

Tabulka 5.4: Pravidla celulárního automatu pro řešení synchronizace

buňky	0-2,94-98	3-8	9-18,21-22	19,25-26	20,27-72	23-24	73-75,82-93	76-81
pravidla	0x13	0x17	0x1F	0x3F	0x1B	0x2F	0x53	0x43

Celkové chování tohoto automatu v testech je možné vidět v tabulkách 5.5 a 5.6. Tabulka 5.5 obsahuje výsledky testů, v nichž byly počáteční konfigurace omezeny podle λ, stejně jako při vývoji pravidel. Méně než polovina všech počátečních konfigurací skončila tak, že bylo více jak 20% buněk ve špatném stavu. Nejméně konfigurací skočilo ve skupině s maximální chybou 20%. Toto nízké číslo značí, že buňky se buď byly schopny většinově 'dohodnout' na synchronizaci, nebo se nebyly schopny zesynchronizovat vůbec.

Tabulka 5.5: Výsledky testů pro pravidla z tabulky 5.4, počáteční konfigurace omezeny podle λ.

Maximální chyba	Počet konfigurací	Procentuální vyjádření
0%	1081	10,81%
5%	1805	18,05%
10%	2046	20,46%
20%	762	7,62%
více	4306	43,06%

Tabulka 5.6 obsahuje výsledky testů pro konfigurace, které byly generovány bez jakéhokoliv omezení. Je vidět že úspěšnost řešení je daleko menší než v předchozím případě. Téměř 70% všech konfigurací skončilo s odchylkou větší než 20%.

Tabulka 5.6: Výsledky testů pro pravidla z tabulky 5.4, bez omezení počátečních konfigurací.

Maximální chyba	Počet konfigurací	Procentuální vyjádření
0%	57	0,57%
5%	551	5,51%
10%	1360	13,60%
20%	1116	11,16%
více	6916	69,16%

Pravidla druhého zajímavého řešení synchronizace jsou popsána v tabulce 5.7. Nejčastějším pravidlem je zde 0x15. V obou řešení se vyskytují stejná pravidla, kterými jsou 0x13

a 0x17. Ostatní pravidla jsou rozdílná.

Tabulka 5.7: Pravidla celulárního automatu pro řešení synchronizace

buňky	0-7,69-98	8-13,23-30	14-22	31-32	33-46	47-58	59-68
pravidla	0x15	0x75	0x55	0x13	0x03	0x07	0x17

V tabulce 5.8 je vidět úspěšnost pravidla pro testy s omezením možných konfigurací podle λ. Výsledky jsou v porovnání s předchozím automatem velmi podobné. Počet konfigurací, které skončily bez jediné chyby, je téměř stejný. Zároveň trend malého počtu výsledných konfigurací s maximální procentuální chybou 20% zůstal zachován. Masivní změnu je možné pozorovat u konfigurací, které skončily s maximální chybou 5%, kam patří téměř polovina všech konfigurací.

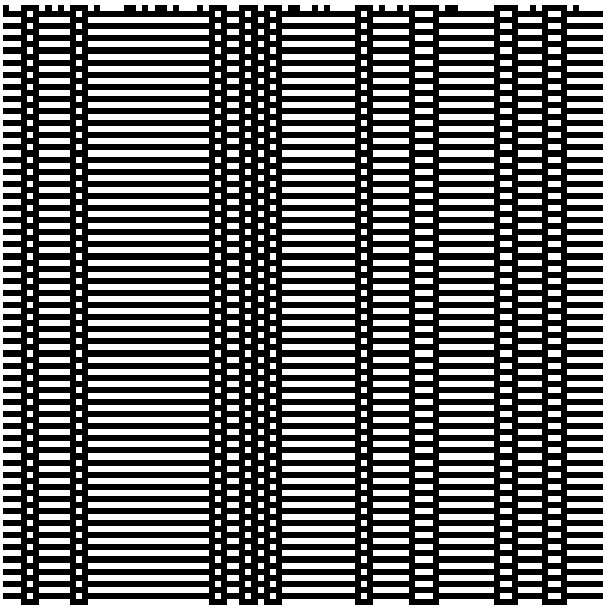
Tabulka 5.8: Výsledky testů pro pravidla z tabulky 5.7, počáteční konfigurace omezeny podle λ.

Maximální chyba	Počet konfigurací	Procentuální vyjádření
0%	1012	10,12%
5%	4813	48,13%
10%	506	5,06%
20%	0	0%
více	3669	36,69%

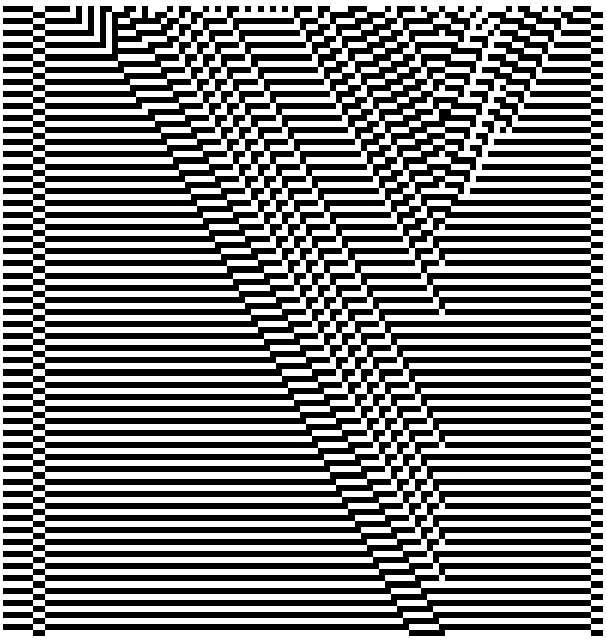
Tabulka 5.9 zachycuje testy, ve kterých počáteční konfigurace nebyly nijak omezeny. Oproti předchozímu řešení je vidět celkový nárůst úspěšnosti.

Tabulka 5.9: Výsledky testů pro pravidla z tabulky 5.7, bez omezení počátečních konfigurací.

Maximální chyba	Počet konfigurací	Procentuální vyjádření
0%	643	6,43%
5%	2226	22,26%
10%	1560	15,60%
20%	10	0,1%
více	5561	55,61%



Obrázek 5.3: Částečně úspěšné řešení synchronizace, větší shluky stejně zesynchronizovaných buněk jsou odděleny buňkami, které zamezují průchodu informace. Zároveň je vidět, že chování automatu je definitivně určeno již druhým krokem, což je také nežádoucí vlastnost. Nedochází tak k rozšiřování informací mezi buňkami. Jedná se o automat obsahující ve všech buňkách pravidlo 0x7F.



Obrázek 5.4: Chování automatu obsahujícího pravidla z tabulky 5.4. V průběhu je vidět postupná korekce středu automatu, kde vzniklo několik skupin buněk s rozdílnou synchronizací. Ve výsledku je 19 buněk ve špatném stavu.

5.3 Šachovnice

Řešení dosažená pro tento problém byla převážně nedostatečně vyhovující. Často se objevovala pouze částečná funkčnost, kdy požadavky chování plnila méně než třetina buněk automatu. Pro šachovnici bylo celkem spuštěno 20 evolučních experimentů.

Příklad dosaženého řešení je v tabulce 5.10. Výsledná pravidla se vyskytují převážně pouze v menších skupinkách buněk. Vyjímkou je pouze pravidlo 0x70, které je v automatu nejčastější. Zajímavostí je pak pravidlo 0xF2, které se vyskytuje ve čtyřech menších seskupeních buněk v automatu.

Tabulka 5.10: Pravidla celulárního automatu pro řešení šachovnice

buňky	0-7,9-11, 87-89,91-99	8	12	13-25,28-66	26-27,90	67-73	74	75-78	79-86
pravidla	0xF2	0xB0	0xF0	0x70	0x72	0x32	0x22	0x23	0xB3

Tabulka 5.11 ukazuje výsledky pro testy s omezenými počátečními konfiguracemi. Bez chyby skončila pouze jediná konfigurace z celkového počtu 10000, což je číslo velmi malé, stejně tak jako počet konfigurací s maximálně 5% odchylkou. Na druhou stranu vidíme poměrně velké číslo u 20% odchylky, což je změna oproti synchronizaci, kde tato hodnota nabývala malých hodnot.

Tabulka 5.11: Výsledky testů pro pravidla z tabulky 5.10, konfigurace omezené podle λ.

Maximální chyba	Počet konfigurací	Procentuální vyjádření
0%	1	0,01%
5%	201	2,01%
10%	1349	13,49%
20%	2488	24,88%
více	5961	59,61%

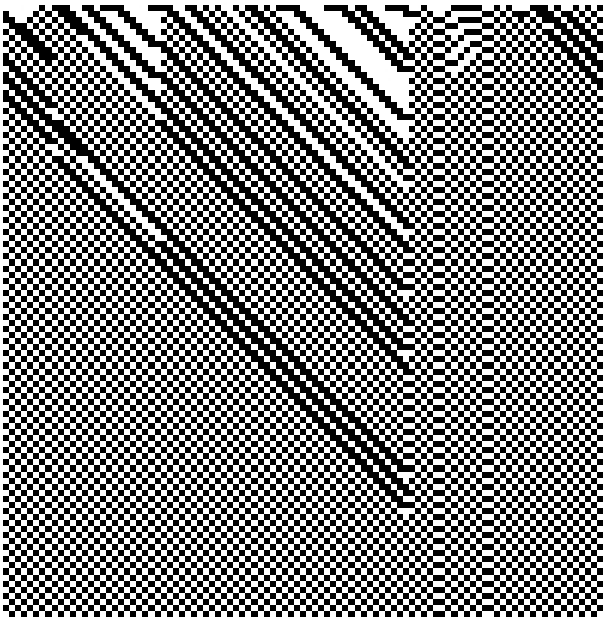
V tabulce 5.12 jsou pak shrnuty výsledky pro testy bez počátečních omezení. Oproti předchozímu testu se značně zvýšil počet konfigurací, které skončily v naprosto správném stavu, stejně tak jako ve stavu s maximální odchylkou 5%. Počet stavů které skočili s 20% odchylkou se pak snížil na úkor předchozích. Vidíme tak zajímavý jev, kdy se automat chová lépe při počátečních konfiguracích, pro které nebyl přímo evoluován.

Tabulka 5.12: Výsledky testů pro pravidla z tabulky 5.10, bez omezení počátečních konfigurací

Maximální chyba	Počet konfigurací	Procentuální vyjádření
0%	223	2,23%
5%	1066	10,66%
10%	1775	17,75%
20%	1829	18,29%
více	5107	51,07%

Celkově jsou však výsledky pro šachovnici horší než tomu bylo u synchronizace. Tento jev je důsledkem složitějších předpokladů pro správnost stavu buňky. Zatímco u synchronizace se toto rozhoduje podle kroku, ve kterém se automat nachází, a který je pro všechny buňky stejný, u šachovnice vstupuje v potaz i pořadí buňky v automatu.

Zajímavým jevem u tohoto problému je šíření informace v řadě buněk výlučně zleva doprava, které bylo typické pro většinu řešení. Toto chování je možné vidět např. na obrázcích 4.3 a 5.5.



Obrázek 5.5: Příklad chování automatu obsahujícího pravidla zaznamenaná v tabulce 5.10. Obsahuje dva sloupce s opačným chováním než zbytek, přičemž první z nich způsobuje 'desinformaci' pro další buňky, zatímco druhý ji opravuje.

Kapitola 6

Závěr

Práce shrnuje některé poznatky o jevu emergentního chování v celulárních automatech na konkrétních příkladech, kterými jsou majorita, synchronizace a šachovnice. Všechny tyto problémy byly odsimulovány a dosažené výsledky shrnuty. Žádné řešení není úplně úspěšné, což se však ani nedalo očekávat.

Při práci bylo největším problémem připravit vhodné podmínky v automatu pro to, aby bylo snadno dosaženo úspěchu při simulaci. S tím souvisí omezení λ, proměnná délka běhu a další problémy popsané v této práci. Bez těchto opatření by sice stále bylo možné nalézt řešení, ovšem počet nutných simulací by značně narostl. Velké množství času bylo také věnováno na zkoumání získaných výsledků, hledání typických vlastností kterých řešení dosahují pro daný problém a výběr ukázkových řešení, které jsou shrnuty v kapitole 5.

Řešený problém lze dále rozšiřovat. Je možné navrhnout další případy emergentního chování a pokusit se je řešit. Druhou možností je pak zvyšování rozměrů automatu, případně hledání optimálního počtu dimenzí pro řešení některých problému.

Literatura

[1] R. Breukelaar and Th. Bäck. Using a genetic algorithm to evolve behavior in multi dimensional cellular automata: emergence of behavior. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 107–114, New York, 2005. ACM.

[2] K. A. Brunner. What's emergent in emergent computing? In *Proc. 16th European Meeting on Cybernetics and Systems Research*, pages 189–192, Vienna, AT, 2002. Vienna University of Technology.

[3] V. Kvasnička, J. Pospíchal, and P. Tiňo. *Evolučné algoritmy*. STU Bratislava, 2000.

[4] M. Sipper. *Evolution of Parallel Cellular Machines The Cellular Programming Approach*. Springer, 1997.

[5] WWW stránky. Emergent behavior. http://c2.com/cgi/wiki?EmergentBehavior, Říjen 2005.