VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

NOVÉ APLIKACE MRAVENČÍCH ALGORITMŮ

NOVEL APPLICATIONS OF ANT ALGORITHMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. JAKUB KORGO**
AUTHOR

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

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Korgo Jakub, Bc.**
Obor: Bioinformatika a biocomputing
Téma: **Nové aplikace mravenčích algoritmů**
Novel Applications of Ant Algorithms
Kategorie: Umělá inteligence

Pokyny:

1. Nastudujte problematiku mravenčích algoritmů a možnosti jejich využití.
2. Proveďte rešerši vybraných diskrétních problémů, které jsou řešitelné pomocí mravenčích algoritmů.
3. Pro zvolenou doménu (vybrané úlohy) navrhněte vhodnou grafovou reprezentaci tak, aby bylo možno generovat řešení těchto úloh pomocí mravenčích algoritmů.
4. Implementujte systém mravenčích algoritmů pro úlohy z bodu 3 a proveďte sadu experimentů demonstrujících jeho schopnosti.
5. Zhodnoťte dosažené výsledky a diskutujte možnosti pokračování projektu.

Literatura:

- Podle pokynů vedoucího projektu.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání, demonstrace prototypu systému dle bodu 4 zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Bidlo Michal, Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstrakt

Mravenčí algoritmy byly použity na rozličné kombinatorické optimalizační úlohy. Jedna z těchto úloh, která však mravenčími algoritmy řešena nebyla, je návrh přechodových pravidel pro celulární automaty (CA). Což je i úloha, na kterou se zaměřuje tato diplomová práce. Tato práce začíná úvodem do mravenčích algoritmů a přehledem jejich aplikací, po kterém následuje úvod do CA. V další části autor navrhuje způsob, jak zakódovat pravidla CA do grafu, který je použit v mravenčích algoritmech. Poslední část této práce obsahuje aplikaci tohoto kódování pravidel do algoritmů elitist ant system a MAX—MIN ant system. Ta je následována experimentálními výsledky pokusů těchto algoritmů o vytvoření přechodových pravidel pro úlohy CA.

Abstract

Ant algorithms have been used for a variety of combinatorial optimization problems. One of these problems, where ant algorithms haven't been used, is the design of transition rules for cellular automata (CA). Which is a problem that this master's thesis is focused on. This work begins with an introduction into ant algorithms and a overview of its applications, followed by an introduction into CA. In the next part the author proposes a way how to encode rules of CA into a graph which is used in ant algorithms. The last part of this thesis contains an application of encoded graph on elitist ant system and MAX–MIN ant system. This is followed by experimental results of creating transition rules for CA problems by these algorithms.

Klíčová slova

Mravenčí algoritmus, optimalizace mravenčí kolonií, MAX–MIN ant system, celulární automat.

Keywords

Ant algorithm, ant colony optimization, MAX–MIN ant system, cellular automaton.

Citace

KORGO, Jakub. *Nové aplikace mravenčích algoritmů*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Bidlo, Ph.D.

Nové aplikace mravenčích algoritmů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Michala Bidla, Ph.D. a uvedl jsem všechny zdroje, ze kterých jsem čerpal.

.....
Jakub Korgo
21. května 2018

Poděkování

Chtěl bych poděkovat svému vedoucímu Ing. Michalu Bidlovi, Ph.D. za jeho rady a návrhy vedoucí k vyšší kvalitě této práce. Dále bych chtěl poděkovat mému nejbližšímu okolí, bez jejichž podpory a tolerance by tato práce nikdy nebyla úspěšně dokončena. Tato práce byla podpořena Ministerstvem školství, mládeže a tělovýchovy z podpory Velkých infrastruktur pro výzkum, experimentální vývoj a inovace v rámci projektu „IT4Innovations národní superpočítačové centrum - LM2015070“.

Obsah

Kapitola 1

Úvod

Většina z nás určitě byla v určitém období dětství fascinována mravenci. Jak chodili za potravou v jedné cestičce a že jim nedělal žádný problém si vytvořit novou trasu i v případě, kdy jim někdo zahradil cestu botou. Tyto vlastnosti byly zajímavé i pro některé vědce, kteří začali zkoumat chování mravenců a s ním spojený způsob komunikace a tvoření cest. Mezi nejzajímavější výsledky jejich bádání patřilo to, že mravenci nekomunikují přímo mezi sebou. Ke komunikaci využívají feromonovou stopu, kterou označí cestu vedoucí k cíli, a tím dokáží ostatní mravence efektivně navést ke kvalitnímu zdroji potravy. Dále jsou mravenci schopni ovlivnit, kolik feromonu vyloučí. Díky těmto vlastnostem a způsobu, jakým mravenci volí směr cesty, pak dokáží jednotlivci navést celou kolonii na potřebná místa a popřípadě i optimalizovat používanou neefektivní cestu tím, že vyloučí větší množství feromonů na kratší cestu k cíli.

Optimalizační úlohy, a to včetně úlohy hledání nejkratší trasy, jsou často řešeny i v oblasti informatiky. Pro řešení těchto úloh se často nasazují heuristické metody, po kterých se však nepožaduje, aby našly optimální řešení. Díky této vlastnosti heuristiky většinou dokážou nalézt přijatelné řešení za relativně krátkou dobu. Mezi pokusy v oblasti heuristik patřil i výzkum toho, zda neexistuje způsob, jak přenést princip mravenčí komunikace do informatiky. První pokusy proběhly na počátku 90. let, kdy byly uskutečněny demonstrace, jak se princip komunikace mravenců dá převést do počítačové podoby takovým způsobem, aby byla schopna najít uspokojivé řešení. Tento úspěch následně přilákal více výzkumníku a začala tak vznikat celá řada heuristik spadajících pod meta-heuristiku mravenčích algoritmů (v originále *Ant Colony Optimization* – zkráceně ACO).

Implementace počítačových mravenců využívá komunikaci na základě intenzity feromonů – stejně jako mravenci v přírodě. Na rozdíl od živého mravence se počítačový mravenec pohybuje v grafu, kde hledá cestu, která je co nejlépe schopna řešit zadanou úlohu. Průchod jedince grafem je řešen tak, že z vybraného počátečního uzlu se na základě intenzity feromonů, a popřípadě jiných informací, přesouvá do dalších uzlů, dokud se nedostane k cíli. Feromony jsou v mravenčích algoritmech reprezentovány kladnými reálnými čísly. Důležitá přidaná vlastnost počítačového mravence je, že si pamatuje použitou cestu. Ta je využita v dalším kroku, kdy následuje návrat mravence do startovního uzlu po nalezené cestě. V této fázi, na základě svého uvážení, které se může vztahovat třeba na délku cesty, přidává po celé trase zvolené množství feromonů. Těmito feromony následně ovlivňuje rozhodování ostatních mravenců při dalším průchodu grafem. Pro řešení úloh bylo však potřeba počítačové mravence zjednodušit, a proto je prostor mravenčích algoritmů omezen pouze na grafy s konečnou množinou uzlů a hran. Jedná se tedy o algoritmy primárně určené pro kombinatorické optimalizace.

Oproti většině ostatních heuristik a meta-heuristik jsou mravenčí algoritmy unikátní v tom, že řešení je postupně získáváno za pomoci nepřímé spolupráce celé populace mravenců. Heuristiky jako *tabu search* nebo *simulované žíhání* používají prohledávací funkci, která obsahuje jen „jednoho jedince“. Mravenčí algoritmy oproti tomu většinou využívají spolupráci jedinců z celé populace, kteří se navzájem nepřímo ovlivňují v době běhu algoritmu. Nevýhoda mravenčích algoritmů je v tom, že bez větších zásahů do algoritmu a funkčnosti nejsou schopny řešit úlohy, ve kterých probíhá optimalizace na spojité množině.

Pomocí mravenčích algoritmů bylo řešeno mnoho úloh. Nejznámější jsou úlohy zabývající se naplánováním trasy, kde byl úspěšně řešen *problém obchodního cestujícího* [10], nebo *vehicle routing problem* [15]. Mezi další úlohy velmi efektivně řešené mravenčími algoritmy patří skupina přiřazovacích úloh. Šlo zde například o pokus, který se zabývá řešením úlohy *minimálního obarvení grafu*, kdy mravenci dosáhli dobré výkonnosti, ale stále lehce zaostávali za nejlepšími heuristikami [16]. Poslední zde zmíněnou skupinou jsou plánovací úlohy. Pro tuto skupinu lze například uvést upravený mravenčí algoritmus, uvedený v roce 2010, který byl v době vydání článku schopen nejefektivněji řešit úlohu *flexible job shop problem* [22].

Cílem této práce je nastudovat vlastnosti mravenčích algoritmů a následně tyto získané znalosti aplikovat na úlohy, které dosud nebyly řešeny za pomoci mravenčích algoritmů. Pro aplikaci mravenčích algoritmů byl zvolen návrh přechodové funkce pro celulární automaty. Aby autor mohl aplikovat mravenčí algoritmy na tuto úlohu, bylo třeba navrhnout konstrukční graf pro mravenčí algoritmy, ze kterého budou mravenci schopni generovat tyto přechodové funkce. Tento konstrukční graf je následně aplikován v algoritmech *elitist ant system* a $\mathcal{MAX}$–$\mathcal{MIN}$ *ant system*, za účelem ověření funkčnosti a porovnání výkonnosti. Poslední část této práce je srovnání autorem navržené metody s existujícími metodami pro automatické generování přechodových funkcí celulárního automatu.

Kapitola 2

Mravenčí algoritmy

Mravenčí algoritmus, jinak nazýván jako optimalizace mravenčí kolonií, v originále *ant colony optimization* (zkráceně ACO), je meta-heuristika[1], ve které populace počítačových mravenců spolupracuje při hledání co nejlepšího řešení zadané úlohy. Je zde využíváno nepřímé komunikace za pomoci feromonů asociovaných s grafem. To vede k emergentnímu chování celé soustavy mravenců, při kterém je tato populace schopna v průběhu času najít řešení splňující zadané podmínky a popřípadě hledat i efektivnější alternativy k takto nalezeným řešením. Mravenci zde pracují synchronně v diskrétním prostoru grafu. Touto meta-heuristikou lze ve většině implementací řešit pouze úlohy s diskrétním prostorem možných řešení [5].

V mravenčích algoritmech je řešení skládáno za pomoci jedinců, kteří stochasticky procházejí grafem $G_c = (C, E)$, kde C je konečná množina komponent řešení (uzly grafu) a E je množina hran obsahující propojení všech uzlů v grafu. Z této definice lze odvodit, že za pomoci mravenčích algoritmů můžeme řešit libovolnou kombinatorickou úlohu, pokud ji budeme schopni zakódovat do grafu. Další problém při návrhu kódování úlohy je definice omezení $\Omega(t)$, podle které se budou muset mravenci řídit při vytváření řešení dané úlohy. Toto omezení lze následně využít buď na penalizovaní ohodnocení výsledných řešení, jež některé z těchto omezení nesplňují, nebo je můžeme využít pro vytvoření pravidel, která určují způsob pohybu mravenců po grafu.

Všechny komponenty $C_i \in C$ anebo všechny hrany $E_{ij} \in E$ k sobě mají přiřazenu intenzitu feromonu τ (přiřazení feromonů k uzlům značíme τ_i, v případě hran τ_{ij}). Intenzita feromonů je dlouhodobá paměť prostředí, která je měněna za běhu algoritmu a je ovlivněna podle kvality řešení nalezených mravenci. Dále mají tyto komponenty přiřazenou heuristickou informaci η (indexováno podobně jako u feromonů: η_i a η_{ij}). Heuristická informace reprezentuje předem vloženou znalost řešené úlohy, která má pomoci mravencům k rychlejšímu nalezení kvalitních řešení. Hodnota obsažena v heuristické informaci je spočítána na začátku algoritmu a dále se nemění. Tyto dvě hodnoty ovlivňují to, jak konkrétní mravenec vytváří cestu, ze které je následně získáno možné řešení. Je důležité poznamenat, že mezi mravenci neprobíhá žádná přímá komunikace.

Tato meta-heuristika je založena na opakovaném aplikování následujících 3 fází:

- `Sestavení Řešení` – V tomto kroku všichni mravenci z populace projdou grafem G_c a ze sestavené cesty vytvoří řešení. Cesta vznikne tak, že mravenci je určen konkrétní počáteční uzel a z něj je stochasticky přesouván do dalších uzlů. Výběr následujícího uzlu je ovlivněn předem definovanými omezeními Ω, hodnotou feromonů a heuristickou

[1] **Meta-heuristika** je předpis pro vytváření heuristických algoritmů určité třídy.

informací okolních uzlů nebo hran. V momentě kdy je mravencem nalezeno řešení, jeho průchod grafem končí. Tento krok je zakončen ohodnocením kvality sestaveného kandidátního řešení.

- `Aktualizování hodnot feromonů` – V tomto kroku mravenci upravují feromony na cestě, která odpovídá jejich řešení z předchozí fáze. Množství přidaných feromonů se odvíjí podle toho, jak mravenec ohodnotil nalezené řešení. Dále je zde prováděno odpařování feromonů v celém grafu, díky kterému se zmenšuje šance na lokální uváznutí.

- `Hromadné akce` – Zde jsou prováděny akce, k jejichž provedení je třeba znát informace z celé populace. Jedná se například o opětovnou aplikaci feromonů na cestu jedince s nejlepším řešením v algoritmu *elitist ant system*. Popřípadě zde patří získávání statistik a jejich následné využití.

2.1 Problém obchodního cestujícího

Pro zjednodušení vysvětlení konkrétních heuristik, které vznikly za základě mravenčích algoritmů, je zde představen problém obchodního cestujícího (v originále *traveling salesman problem*, zkráceně TSP), na který budeme různé mravenčí algoritmy aplikovat. Tato úloha je vybrána z důvodu, že jde o jednoduše popsatelný $\mathcal{NP}$–těžký optimalizační úkol, který je většinou využíván jako benchmarkový úkol pro mravenčí algoritmy. Navíc tato úloha je již definována grafem, není tedy třeba vytvářet nový graf, ve kterém by mohli mravenci tvořit cesty.

Problém obchodního cestujícího je úloha, ve které je zadána sada měst, mezi nimiž známe vzdálenosti. V této topologii chceme zjistit, jaká je nejkratší trasa potřebná pro průchod všemi městy právě jednou. Do této cesty se počítá i návrat do počátečního města. Tuto úlohu lze reprezentovat jako vážený neorientovaný graf $G_{TSP} = (N, A)$, kde N je konečná množina všech měst (uzly) a A je množinou všech cest mezi městy (vážené hrany).

Omezení Ω zde říká, že mravenec může sestavit jen takové řešení, které obsahuje všechna města právě jednou.

Každá hrana $a_{ij} \in A$ má přiřazenou hodnotu d_{ij} značící vzdálenost mezi městy i a j, kde $i, j \in N$. Řešením této úlohy je permutace π obsahující všechna města z množiny N, která je indexována od jedničky. Toto řešení ohodnocujeme funkcí $f(\pi)$ a požadujeme, aby byl výsledek funkce $f(\pi)$ co nejmenší, kde

$$f(\pi) = \sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)} + d_{\pi(n)\pi(1)}. \tag{2.1}$$

Ve všech zde uvedených variantách algoritmu jsou feromony umístěny na hranách grafu. O τ_{ij} můžeme tedy říct, že vyjadřuje jak moc mravenec preferuje město j když se nachází v městě i. Graf je neorientovaný a platí zde $\tau_{ij} = \tau_{ji}$. Jako heuristická funkce η je zde aplikován výběr nejbližšího města, přičemž hodnota této heuristiky η_{ij} mezi dvěma městy i a j je vypočítána následovně:

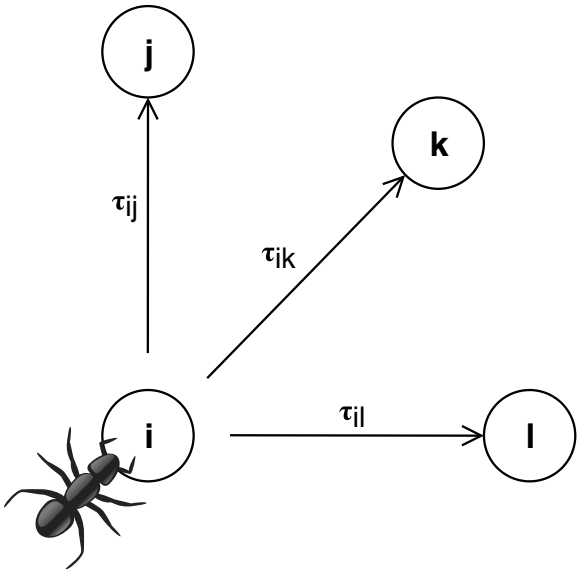
$$\eta_{ij} = 1/d_{ij} \tag{2.2}$$

2.2 Ant System

Ant System (zkráceně AS) [7] je algoritmus, využívající rovnocenné spolupráce všech mravenců v populaci. Před samotným během algoritmu je potřeba provést inicializaci hodnot feromonů v grafu. Inicializace je pro problém obchodního cestujícího řešena tak, že je vytvořena cesta celým grafem z náhodného uzlu za pomoci *nearest neighbour* heuristiky. Následně je celková délka této cesty C^{nn} aplikována jako počáteční hodnota feromonů τ_0, která je shodná pro všechny hrany.

$$\tau_0 = \tau_{ij} = m/C^{nn}, \quad \forall (i,j) \in A, \tag{2.3}$$

kde m je počet mravenců v populaci. Tento postup je zvolen z toho důvodu, že je třeba mít hodnotu τ_0 o něco vyšší, než bude množství feromonů dodaného v prvních iteracích. Díky takto nastavené počáteční hodnotě není algoritmus náchylný k brzkému lokálnímu uváznutí a nedochází ani k situaci, že by mravenci nedokázali dlouhodobě výrazně ovlivnit feromonovou stopu, z důvodu vysoké počáteční hodnoty τ_0. Následuje hlavní smyčka algoritmu, ve které je opakováno *sestavování řešení* a *aktualizace feromonů* do doby, dokud není překročen maximální počet generací nebo není nalezeno dostatečně kvalitní řešení.



Obrázek 2.1: Výběr následujícího uzlu na základě feromonů.

Sestavení řešení

V AS tato fáze začíná náhodným rozložením mravenců do měst. Následuje konstrukce cesty, kdy si každý mravenec k vybírá, do jakého města se přesune při tvorbě jeho cesty. Zde aplikujeme omezující podmínku z Ω, podle které může mravenec navštívit každé město jen jednou. Musíme tedy mít množinu měst $\mathcal{N}_i^k$, které mravenec k ještě nenavštívil, když je ve městě i. Nyní můžeme zjistit pravděpodobnost, s jakou se mravenec k z města i vydá do města j (obrázek 2.1). Tato pravděpodobnost je vyjádřena následujícím vzorcem:

$$p_{ij}^k = \begin{cases} \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, & j \in \mathcal{N}_i^k \\ 0, & j \notin \mathcal{N}_i^k \end{cases} \tag{2.4}$$

,kde parametry α a β určují, jak moc bude mravenec ovlivněn feromony a heuristickou informací. Jako obvyklé hodnoty pro problém obchodního cestujícího bývají používány $\alpha = 1$ a β v intervalu $\langle 2, 5 \rangle$. Takto zvolené parametry způsobí, že bude kladen větší důraz na hodnotu feromonů, než na heuristickou informaci.

Následně si mravenec dle získaných pravděpodobností náhodně vybere město, do kterého se přesune a znovu provede pravděpodobnostní výběr dalšího města. Tento proces se opakuje do té doby, dokud mravenec neprojde všemi městy. Průběžně se během této fáze vytváří uspořádaná množinu T^k, která obsahuje všechny hrany, kterými mravenec k prošel při vytváření svého řešení.

Aktualizování hodnot feromonů

Tato fáze nastává ve chvíli, kdy všichni mravenci z populace mají sestavené řešení. Nejprve proběhne odpaření feromonů v celém grafu. To se děje dle následujícího vzorce:

$$\tau_{ij} \longleftarrow (1 - p)\tau_{ij}, \quad \forall (i, j) \in A, \tag{2.5}$$

kde parametr p určuje rychlost odpařování feromonů. Tato hodnota musí být v intervalu $(0, 1\rangle$. Tento parametr ovlivňuje rychlost, jakou budou „zapomenuty“ nepoužívané cesty.

Dalším krokem je vložení feromonů na cesty, kterými mravenci z populace m prošli. To se dá spočítat pro každou hranu τ_{ij} v grafu následovně:

$$\tau_{ij} \longleftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k, \quad \forall (i, j) \in A \tag{2.6}$$

přičemž množství feromonů $\Delta\tau_{ij}$, které každý mravenec k vyprodukuje vzhledem k množině navštívených hran T^k, je pro úlohu obchodního cestujícího definováno následovně:

$$\Delta\tau_{ij}^k = \begin{cases} 1/C^k, & (i, j) \in T^k \\ 0, & (i, j) \notin T^k \end{cases} \tag{2.7}$$

Kde C^k je délka trasy T^k, kterou mravenec k vytvořil. Zde můžeme vidět, že kratší trasa je ohodnocena více feromony, a tím pádem je větší pravděpodobnost, že v dalším cyklu si libovolný mravenec tuto trasu vybere.

2.3 Elitist Ant System

Elitist Ant System (zkráceně EAS) [7] je jednoduchým vylepšením původního algoritmu AS (podkapitola 2.2). Je zde využito skutečnosti, že si algoritmus pamatuje nejlepší cestu, a ta je v každém cyklu využita ke zvýšení množství feromonů na této trase. Pro zjištění nejlepší cesty EAS využívá fázi hromadných akcí po provedení aktualizace feromonů, ve které je vždy uloženo doposud nejlepší nalezené řešení T^{best}. Algoritmus má shodnou fázi pro vytváření cest s algoritmem AS, která je následována upravenou aktualizací feromonů, ve které je využito hromadné akce.

Aktualizování hodnot feromonů

Stejně jako v předchozím algoritmu je na počátku provedeno odpaření feromonů v celém grafu (rovnice 2.5). Následuje aktualizace feromonů, kdy se při aktualizaci hodnot feromonů navíc využívá trasy T^{best}, která vyjadřuje doposud nejlepší nalezené řešení. Dále je zde nový parametr e, který určuje množství feromonů přidaných z trasy T^{best}. Upravená rovnice vypadá následovně:

$$\tau_{ij} \longleftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} + e\Delta\tau_{ij}^{best}, \quad \forall(i,j) \in A, \tag{2.8}$$

kdy $\Delta\tau_{ij}^{best}$ je definována shodně s $\Delta\tau_{ij}$ v rovnici 2.7.

V případě, že parametr e je vhodně zvolen, elitistická strategie vykazuje podstatně lepší výsledky než původní algoritmus AS [5].

2.4 Rank-based Ant System

Rank-based Ant System (zkráceně AS_{rank}) [3] je dalším vylepšením mravenčích algoritmů, které vychází z EAS (podkapitola 2.3). V této variantě mravenčího algoritmu je množství feromonů uložených jedincem ovlivněno nejen délkou jeho cesty, ale i délkou cest ostatních mravenců v populaci. Další důležitou úpravou je zde skutečnost, že feromony nepřispívají všichni mravenci, ale pouze $w-1$ nejlepších mravenců daného cyklu společně s dosud nejlepším nalezeným řešením. V dalších ohledech je algoritmus shodný s EAS.

Aktualizování hodnot feromonů

Po vytvoření cest a evaporaci feromonů (rovnice 2.5), je pro provedení této fáze třeba celou populaci seřadit dle délky cest. Celá populace je takto vzestupně seřazena od nejkratší cesty k nejdelší. Následně je každému mravenci přiřazena hodnota r odpovídající indexu mravence po seřazení, přičemž první index je roven 1. Následně w-1 nejlepších mravenců z tohoto cyklu a mravenec s doposud nejlepší trasou T^{best} vloží feromony do grafu. Pro každou hranu τ_{ij} z grafu vypadá přiřazení feromonů následovně:

$$\tau_{ij} \longleftarrow \tau_{ij} + \sum_{r=1}^{w-1} (w-r)\Delta\tau_{ij}^{r} + w\Delta\tau_{ij}^{best}, \quad \forall(i,j) \in A \tag{2.9}$$

Tato verze algoritmu vykazuje mírné zlepšení výkonu oproti EAS na úloze TSP [5].

2.5 $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System

$\mathcal{MAX}$–$\mathcal{MIN}$ *Ant System* (zkáraceně $\mathcal{MM}$AS) [20] přináší výraznější implementační rozdíly oproti původní myšlence mravenčích algoritmů. První rozdíl je ten, že feromony do grafu vkládá jen nejlepší řešení t^{bs}. To může být buď nejlepší řešení z dané iterace, nebo doposud nejlepší nalezené řešení. V závislosti na řešené úloze se pak rozhoduje, která z těchto variant je použita. Aby nedošlo k brzkému uváznutí v lokálním maximu, je zavedena další výrazná úprava oproti funkčnosti mravenčích algoritmů. V $\mathcal{MM}$AS existuje rozsah hodnot pro feromony $\langle\tau_{min}, \tau_{max}\rangle$, který nelze překročit. To při správném nastavení zlepšuje prohledávací schopnosti algoritmu. Poslední nová vlastnost je možnost reinicializace feromonové stopy na hodnotu τ_{max}. Ta nastává v momentě, kdy algoritmus stagnuje po dlouhou dobu.

Inicializace feromonů a sestavování cest mravenci probíhá shodně s AS (podkapitola 2.2). Rozdílná je zde fáze aktualizování hodnot feromonů, která může být následovaná reinicializací těchto hodnot. Algoritmus shodně s AS končí po dosažení maximálního počtu iterací, nebo po nalezení dostatečně kvalitního řešení.

Aktualizování hodnot feromonů

Po aplikování evaporace, která je shodná s evaporací v AS (rovnice 2.5), následuje vložení feromonů po trase nejlepšího jedince:

$$\tau_{ij} \longleftarrow \tau_{ij} + \Delta\tau_{ij}^{best}, \quad \forall(i,j) \in A \tag{2.10}$$

$\Delta\tau_{ij}^{best}$ může být vypočítána buď z délky cesty nejlepšího jedince C^{bi} dané iterace, kdy $\Delta\tau_{ij}^{best} = 1/C^{bi}$. Nebo z délky cesty doposud nejlepšího nalezeného řešení C^{bs}. Ve druhém případě je $\Delta\tau_{ij}^{best} = 1/C^{bs}$. Používány jsou obě varianty, přičemž v úloze TSP se ukázalo, že varianta, která používala C^{bi}, byla efektivnější pro menší instance této úlohy. Naopak u velkých instancí začala varianta využívající C^{bs} podávat lepší výsledky [5].

Limity feromonů a reinicializace

$\mathcal{MM}$AS využívá horní (τ_{max}) a dolní (τ_{min}) limit pro možné rozmezí hodnot feromonů v systému. Tyto parametry jsou zde zavedeny z toho důvodu, aby byl podstatně omezen počet situací, při kterých dojde ke stagnaci. Velmi důležité je zajistit správné nastavení parametru τ_{min}, protože tento parametr velkou měrou ovlivňuje četnost stagnací algoritmu a schopnost algoritmu prohledávat prostor.

Další vlastnost, kterou $\mathcal{MM}$AS využívá, je reinicializace algoritmu v momentě, kdy začne stagnovat. Zde je využita hodnota τ_{max}, která určuje, na jakou hodnotu budou všechny cesty v grafu inicializovány. Tento parametr je třeba nastavit dostatečně vysoko, aby nebyla explorační fáze po reinicializaci příliš krátká. Většinou je tato hodnota dynamicky nastavována podle aktuálního nejlepšího řešení.

Díky velmi dobré schopnosti prohledávat prostor byl tento algoritmus využit v mnoha úlohách. To můžeme vidět i v kapitole, která se zabývá aplikacemi mravenčích algoritmů (kapitola 3), kdy se různě upravený algoritmus $\mathcal{MM}$AS objevuje v mnoha řešených úlohách. To se projevilo i v problému obchodního cestujícího, kdy je tento algoritmus při delších bězích schopen najít lepší řešení, než všechny zde zmíněné algoritmy [5].

2.6 Ant Colony System

Ant colony system (zkráceně ACS) [6] je s $\mathcal{MM}$AS dalším algoritmem, který se výrazněji odchyluje od původní myšlenky mravenčích algoritmů implementovaných v AS (podkapitola 2.2). Prvním rozdílem je, že existuje parametr q_0, který umožňuje vynechat pravděpodobnostní výběr a nahradit ho výběrem cesty s nejvyšší pravděpodobností. S narůstající hodnotou q_0 je častěji vybrána přímo tato cesta. Také zde neprobíhá globální evaporace feromonů. Ty zde ubývají v průběhu vytváření cesty, kdy jakmile mravenec projde hranou (i,j), tak je z této hrany ubrána část feromonů. Pro přidání feromonů je používána pouze trasa T^{best}, která shodne s $\mathcal{MM}$AS může být získána jakožto nejlepší cesta mravence z cyklu C^{bi}, nebo může být doposud nejlepší nalezenou cestou C^{bs}.

Sestavení řešení a odpařování feromonů

V této variantě mravenčího algoritmu využíváme pseudonáhodného pravděpodobnostního výběru, kdy se uzel j z uzlu i vybírá následovně:

$$j = \begin{cases} \text{argmax}_{l \in \mathcal{N}_i^k} \{\tau_{il}[\eta_{il}]^\beta\}, & \text{pokud } q \leq q_0 \\ J, & \text{jinak,} \end{cases} \tag{2.11}$$

kde q je náhodná hodnota z rozložení $\langle 0, 1\rangle$ a q_0 (kde $0 \leq q_0 \leq 1$) je parametr ovlivňující četnost vybrání nejlepší dostupné cesty. J značí výběr náhodné cesty dle distribuce získané z rovnice 2.4 pro pravděpodobnostní výběr v ant systému, kde použijeme $\alpha = 1$. Parametrem q_0 je tedy určena míra mezi tím, jak se bude mravenec řídit pouze nejlepší cestou anebo zda využije pravděpodobnostní prohledávání.

Po vybrání hrany i, j, po které se mravenec přesune, následuje odpaření feromonů τ_{ij} na této hraně:

$$\tau_{ij} \longleftarrow (1 - \xi)\tau_{ij} + \xi\tau_0, \tag{2.12}$$

přičemž τ_0 je parametr, který je shodný s inicializační hodnotou feromonů a parametr ξ určuje rychlost, jakou se aktuální hodnota feromonů přibližuje ke zlomku inicializační hodnoty. Experimentálně bylo zjištěno, že pro problém obchodního cestujícího je nejlepší hodnota $\xi = 0,1$. Díky tomuto způsobu odpařování feromonů má tento algoritmus dobré prohledávací schopnosti, protože jedna cesta není stejně lákavá pro všechny mravence. Dále je důležité si u této úpravy algoritmu uvědomit, že zde již záleží na tom, zda jsou mravenci v cyklu spouštěni paralelně, nebo sériově [6].

Aktualizování hodnot feromonů

Jak již bylo zmíněno, koncová úprava feromonů probíhá pouze na cestě T^{best}, což lze vyjádřit následujícím vzorcem:

$$\tau_{ij} \longleftarrow (1 - p)\tau_{ij} + p\Delta\tau_{ij}^{best}, \quad \forall (i, j) \in T^{best}, \tag{2.13}$$

kde $\Delta\tau_{ij}^{best} = 1/C^{best}$. Zajímavá vlastnost tohoto algoritmu je, že není přidávána celá hodnota $\Delta\tau_{ij}^{best}$, ale vzniká zde vážený průměr mezi původní hodnotou feromonu a feromony, které chce daný mravenec přidat. Tato vlastnost vytváří pomyslný feromonový strop, který je další vlastností této varianty mravenčích algoritmů.

Kapitola 3

Známé aplikace mravenčích algoritmů

Po úspěchu mravenčích algoritmů při aplikaci na *problém obchodního cestujícího*, byla tato meta-heuristika aplikována i na jiné $\mathcal{NP}$–těžké optimalizační problémy. V této kapitole se zaměříme na tři větší kategorie optimalizačních problémů. Těmi jsou směrovací, přiřazovací a plánovací problémy. Následně bude zmíněny další úlohy řešené za pomocí mravenčích algoritmů, které nespadají do žádné z předchozích kategorií.

3.1 Směrovací problémy

V této částí se zaměříme na úlohy, u kterých se budeme snažit zjistit, v jakém pořadí je potřeba projít uzly tak, aby bylo funkční ohodnocení této cesty co nejmenší. Mezi nejznámější úlohy patří *problém obchodního cestujícího*, který byl zmíněn v podkapitole 2.1. Mezi nejúspěšnější varianty řešící tento problém zde patří algoritmus HAS–SOP. Tento upravený mravenčí algoritmus využívá jako základ ACS, který je vylepšen o fázi lokálního prohledávání po vytvoření cesty. Tato modifikace umožňuje prohledání okolí nalezeného řešení algoritmem lokálního prohledávání a což má za následek, že je občas vygenerováno lepší řešení, než bylo původní. V porovnání s nejlepšími heuristikami té doby byl tento algoritmus v době vynalezení schopen poskytnout nejlepší výsledky pro tuto úlohu [10].

Další úlohou z této skupiny je problém okružních jízd s omezenou kapacitou (v originále *capacitated vehicle routing problem*). To je rozšíření problému obchodního cestujícího, kdy máme za úkol z jednoho centrálního uzlu obsloužit n zákazníků, přičemž každý zákazník požaduje určitý počet kusů zboží a kapacita vozidla pro rozvoz je omezena. Kvůli této podmínce je tato úloha náročnější na vyřešení, protože vozidlo se musí vracet do výchozího uzlu pro nové nakládky zboží. Tato úloha je výrazně složitější než problém obchodního cestujícího, protože je zde nutné co nejefektivněji seskupit zákazníky tak, aby při jedné cestě nebyla přeplněna kapacita vozidla, ale zároveň byla konečná ujetá vzdálenost co nejnižší. Tato úloha byla vyřešena za pomocí upraveného algoritmu $\mathrm{AS}_{\mathrm{rank}}$. Tento algoritmus se opět prokázal efektivnějším než do té doby používaná varianta *tabu search*. Tato metoda dokonce dokázala najít do te doby neexistující nejefektivnější řešení [15].

3.2 Přiřazovací problémy

V této kategorii úloh se snažíme n položek (jako jsou například budovy, aktivity, ...) přiřadit k omezenému počtu zdrojů (pro uvedené příklady to mohou být pozemky, agenti, ...), přičemž přiřazení položky ke zdroji má předem dané funkční ohodnocení. Jeden z nejsložitějších úkolů této kategorie je *kvadratický přiřazovací problém*. Zde se snažíme přiřadit množinu zdrojů k množině lokací, přičemž mezi lokacemi známe vzdálenost d a mezi zdroji známe objem provozu f. Cílem je vytvořit takové přiřazení, u kterého bude minimalizována suma $f \cdot d$ mezi všemi zdroji. Zde se mravenčí algoritmy opět projevily jako velmi kvalitní heuristikou, kdy nejlepší výsledky v této úloze podával upravený algoritmus $\mathcal{MM}$AS [20].

Mezi jiné přiřazovací problémy patří sestavování univerzitního rozvrhu (v originále *university course timetabling problem*). Zde algoritmus se snaží vyhovět dvěma třídám podmínek. První třída jsou povinně splnitelné podmínky, mezi které může patřit například, že všechny přednášky musí být vyučovány a žádné dvě přednášky se nesmí překrývat v jedné místnosti. Druhá třída jsou podmínky ovlivňující kvalitu řešení. Zde může třeba patřit například požadavek, aby se co nejmenšímu počtu studentů křížil rozvrh. Splnění podmínek druhé třídy zvyšuje kvalitu návrhu, ale nejprve musí být splněny všechny povinné podmínky. Zde se opět nejlépe prokázal upravený algoritmus $\mathcal{MM}$AS, který exceloval zejména na velikých instancích tohoto problému, kde jiné mravenčí algoritmy nedokázaly najít dostatečně efektivní výsledný rozvrh [19].

Mezi poslední zde zmíněnou úlohu patří *barvení grafu*. Zde hledáme co nejmenší počet barev, kterými lze obarvit graf tak, aby žádné dva sousedící uzly neměly stejnou barvu. Zde mravenčí algoritmy nedosáhly na nejlepší algoritmy, ovšem byly již dostatečně efektivní na to, aby jejich řešení byly kvalitou velmi blízká řešením z nejlepších algoritmů pro tuto úlohu. V nejefektivnější mravenčí variantě se jedna o upravený algoritmus $\mathcal{MM}$AS, který po vytvoření řešení navíc využívá lokální prohledávání [16].

3.3 Plánovací problémy

Všechny úlohy spadající do této kategorie mají společné to, že se musí vytvořit plán, jak rozdělit vykonání úloh na omezený počet zdrojů. Tyto úlohy mohou být složeny z více na sobě závislých operací, ve kterých musí být zachována posloupnost vykonání. U těchto úloh se snažíme dosáhnout vytvoření takového plánu, který bude mít nejkratší celkový čas. Shodné pro všechny úlohy jsou dvě následující vlastnosti:

1. U všech úloh a operací známe dobu vykonání, která je neměnná.
2. Jakmile začne být operace zpracovávána, nesmí dojít k přerušení této operace.

Mravenčí algoritmy byly aplikovány na velmi širokou škálu plánovacích úloh. U některých úloh zaznamenaly velké úspěchy, mezi ty patří například úloha *single-machine total weighted tardiness problem* [4] nebo *flexible job shop problem* [22]. Dále byly prováděny výzkumy nad *open shop schedule problémy* a *job shop schedule problémy*, kdy pro první zmíněný problém algoritmus dokázal nacházet ideální řešení, ovšem v druhém případě byla tato implementace algoritmu ACO poražena heuristikou *Tabu Search* [2]. Další problém, kde jsou schopny mravenčí algoritmy nalézt kvalitní řešení, ale nedokáží se vyrovnat nejlepší známe heuristice, je *Permutation Flow Shop Problem*, kde byl použit jako základní algoritmus $\mathcal{MM}$AS [14].

3.4 Další úlohy

Jedna z dalších úloh, řešena za pomocí mravenčích algoritmů, je *hledání nejdelší kliky v grafu*. Tedy hledáme takovou největší podmnožinu uzlů v grafu, pro které platí, že jsou všechny uzly navzájem propojeny hranou. Tato úloha byla řešena v článku S. Feneta [9] za pomoci $\mathcal{MM}$AS. Princip algoritmu spočíval v tom, že na začátku byl vybrán náhodný uzel a vytvářela se cesta stejně jako v problému obchodního cestujícího. S jediným rozdílem, že množina povolených uzlů byla omezena tím, že následující uzel musí být propojen hranou se všemi předcházejícími. Tento přístup byl schopen řešit tuto úlohu relativně dobře, ovšem nepřekonal dosud nejlepší algoritmus pro tuto úlohu *Reactive Local Search*.

Další aplikace mravenčích algoritmů se týká úlohy *Bin Packing Problem*, kde se snažíme zabalit předem daný počet objektů o různé váze do košů s předem danou nosností. V této úloze se snažíme minimalizovat počet košů, které bude potřeba použít na zabalení všech objektů. Zde byl opět použit jako základ algoritmus $\mathcal{MM}$AS s využitím lokálního prohledávání. V době vydání článku měl tento algoritmus srovnatelné a v některých případech i lepší výsledky, než do té doby nejlepší používané algoritmy [12].

Velmi zajímavá implementace mravenčích algoritmů se týká *detekce hran v obraze*. Konkrétně si zde zmíníme aplikaci na hledání hran v obrázcích listů stromů za účelem vyhledání jejich žilnatiny. Zde je využit základní algoritmus AS, který byl ovšem použit neobvyklým způsobem. Řešení zde totiž není výsledkem průchodu grafem některého jedince, ale výsledek je samotná feromonová stopa v grafu, která vznikne po určitém počtu kroků. Graf je zde samotný vstupní obrázek. Algoritmus je založen na náhodném počátečním rozložení mravenců po grafu, ve kterém se následně pohybují dle definovaných přechodových pravděpodobnost. Tito mravenci ovšem vždy udělají jen jeden krok, po kterém nastává vložení feromonů na vkročenou buňku a evaporace feromonů v celém obraze. Výsledkem je obraz, kde pro každý pixel určuje hodnota feromonů jaká je pravděpodobnost, že v daném místě je hrana [13].

Kapitola 4

Celulární automaty

Celulární automaty (zkráceně CA) jsou dynamické systémy, ve kterých je simulační čas a obor hodnot diskrétní. CA je pole buněk, které mohou nabývat libovolné hodnoty z konečné množiny oboru hodnot. Aktualizace buněk probíhá synchronně na základě předem daných přechodových pravidel. Ta jsou shodná pro všechny buňky v CA. Nový stav buňky se odvíjí od aktuálního stavu konkrétní buňky a stavu nejbližších sousedů této buňky [18].

Celulární automaty mají tři důležité parametry. První z těchto parametrů je *dimenze*, která většinou nepřesahuje hodnotu 3. Tato vlastnost velmi ovlivňuje vlastnosti chování celulárního automatu, protože se zvyšující se dimenzí automatu má každá buňka větší počet sousedních buněk, se kterými interaguje a podle kterých tato buňka určuje nový stav.

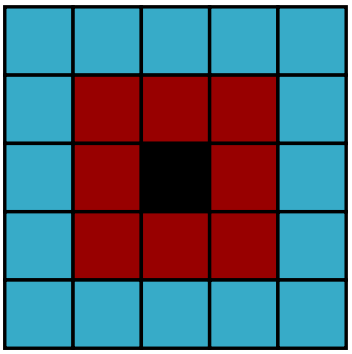
CA obvykle uvažují teoreticky nekonečná pole buněk. V těchto případech se pracuje s automaty, které se dynamicky zvětšují při výpočtu a lze je v tomto smyslu považovat za nekonečné. Při některých úlohách je však žádoucí pracovat s konečnými poli, kde však musíme řešit co dělat v momentě, kdy vyhodnocujeme nový stav pro okrajovou buňku. Část buněk v sousedském okolí totiž nebude definováno. Tyto okrajové podmínky jsou většinou řešeny jednou z následujících variant:

- **Periodicita** – Tato varianta využívá toho, že nejlevější a nejpravější buňka v 1D automatu jsou prohlášeny za sousedy. Vzniká nám tedy pole ve tvaru kruhu. Je možné tuto metodu použit i na vícerozměrné prostory, kdy se tato metoda analogicky používá i pro ostatní směry.
- **Konstantní hodnota** – Při vyhodnocování pravidel daného pole CA, jsou všechny buňky za hranicemi tohoto pole nastaveny na konstantní hodnotu a nemění se po celou dobu simulace.
- **Reflexe** – Zde se pro vyhodnocení pravidel se buňky za hranicemi CA nastavují na hodnotu, která odpovídá hodnotě v nejbližší definované buňce. Pro 1D automat tedy nalevo od výpočetní hranice automatu bude kopie nejlevější hodnoty tohoto automatu a stejný princip se aplikuje i pro pravou stranu.

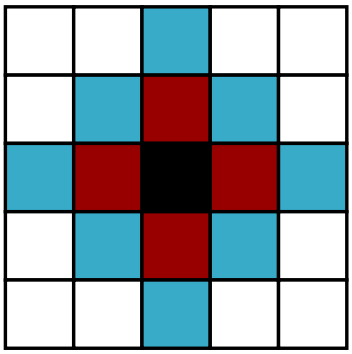
Další vlastností CA je *počet stavů* k, který určuje, kolik hodnot může každá z buněk nabývat. Vždy platí, že $k \geq 2$. V této množině stavů má jeden stav speciální vlastnost, která se nazývá „klidový stav“ nebo „nulový stav“. Tento stav má většinou hodnotu 0 a odlišuje nám neaktivní buňky od buněk aktivních.

Poslední důležitý parametr, podle kterého se bude odvíjet následující stav, je *velikost okolí* buňky r v celulárním automatu. U automatů s dimenzí 1 je okolí většinou definováno

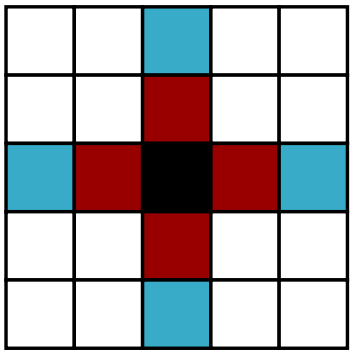
(a) Okolí v 1D CA



(b) Mooreovo okolí v 2D CA



(c) Von Neumannovo okolí v 2D CA



(d) Křížové okolí v 2D CA

Obrázek 4.1: Příklady různých okolí r pro černou centrální buňku. Červené okolí znázorňuje $r = 1$, modré okolí znázorňuje $r = 2$.

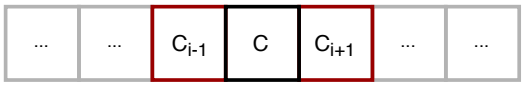
vzdáleností od výchozí buňky, viz obrázek 4.1a. Složitější je situace u automatů s dimenzí 2 a vyšší, kdy lze zadefinovat výpočet vzdálenosti pro okolí různými způsoby. To může být definováno třeba Mooreovým okolím, které odpovídá vzdálenosti buněk, když povolíme pohyb po úhlopříčce (obrázek 4.1b). Další definice může být Von Neumannovo okolí, u kterého je vzdálenost definována Manhattanskou metrikou [17] (obrázek 4.1c). Speciální případ okolí může být křížové okolí, kdy povolíme rozšiřování jen ve směru os od centrální buňky (obrázek 4.1d).

4.1 Přechodová funkce CA

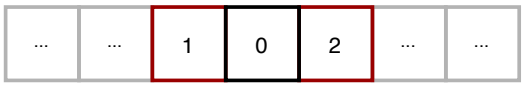
Pro názorné vysvětlení přechodových funkcí zde zvolíme 1D celulární automat. Úlohy, které budou zkoumány v této diplomové práci, jsou založeny právě na 1D automatech. Sousedství buňky c definujeme okolím r, kdy toto sousedství obsahuje buňku c a všechny buňky, které jsou do vzdálenosti r od této buňky vlevo i vpravo. Pro zjištění počtu pravidel je dále třeba znát počet stavů k, kterých může buňka nabývat. Obecně pak lze vypočítat, že sousedství každé buňky je tvořeno $2r+1$ buňkami, počet možných kombinací stavů buněk v sousedství je k^{2r+1} [17].

To vede zejména k tomu, že v závislosti na počtu buněk v sousedství každé buňky a na počtu stavů roste počet možných přechodových funkcí CA exponenciálně, což zásadně snižuje efektivitu návrhu komplexních CA. Celkem existuje $k^{k^{2r+1}}$ rozdílných přechodových funkcí. Vzhledem k tomu, že při takovém množství přechodových funkcí je ruční návrh velmi složitý, tak se přechází k automatizovaným metodám, pod které spadají různé metaheuristiky. To je i jeden z důvodu, proč se tato práce zaměřuje na návrh těchto funkcí pro CA za pomoci mravenčích algoritmů.

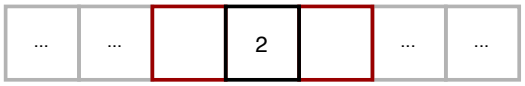
Je mnoho způsobů, jak zadefinovat přechodovou funkci CA. Zde si uvedeme dva nejčastější způsoby definice pravidel, přičemž budeme využívat notaci $c_i(t)$, která určuje hodnotu stavu buňky na pozici i v čase t.



(a) Indexování okolních buněk pro buňku c_i



(b) Ukázkový vstup CA



(c) Výstup CA 4.2b na základě tabulky 4.2d

$\mathbf{c_{i-1}}$	$\mathbf{c_i}$	$\mathbf{c_{i+1}}$	**výstup**
...			...
1	0	0	0
1	0	1	0
1	**0**	**2**	**2**
1	1	0	0
1	1	1	1
...			...

(d) Tabulka přechodových pravidel

Obrázek 4.2: Způsob aplikování tabulkové přechodové funkce na buňku v CA

První varianta, jak určit stav buňky v čase $t+1$, je využití tabulky přechodových pravidel φ, kdy zadefinujeme výstupy pro všechny možné kombinace. Pokud zvolíme $k = 2$ a $r = 1$, potřebujeme zadefinovat 8 pravidel, které určují nový stav buňky c_i pro danou kombinaci stavů v jejím okolí. Nový stav $c_i(t + 1)$ je získán následovně:

$$c_i(t + 1) = \varphi[c_{i-1}(t), c_i(t), c_{i+1}(t)] \tag{4.1}$$

Aplikaci tabulkových pravidel lze vidět v obrázku 4.2, kde vidíme aplikaci tabulky přechodových pravidel 4.2d na ukázkový vstup 4.2b.

Další možnou variantou je výpočet za pomoci předem definované matematické funkce, která splňuje, že výsledek je pro každou vstupní kombinaci v oboru hodnot celulárního automatu. Pro výše zvolené parametry lze například použít součet všech buněk v sousedství, na který je aplikována operace modulo:

$$c_i(t + 1) = (c_{i-1}(t) + c_i(t) + c_{i+1}(t)) \bmod 2, \tag{4.2}$$

přičemž můžeme z této rovnice velmi jednoduše vytvořit tabulku a aplikovat stejný postup jako pro rovnici 4.1. Záleží pouze na implementační a výpočetní náročnosti, která z těchto variant je vhodnější pro využití v konkrétní aplikaci.

4.2 Úlohy řešené CA

CA se potýkají většinou se dvěma typy úloh. První jsou modelovací úlohy, kdy si vytvoříme přechodovou funkci charakterizující daný model a zjišťujeme, jaký výstup se vytvoří ze zadaného vstupu. Vstupem se rozumí počáteční nastavení modelu v CA a výstup zde může být buď samotný průběh, kdy sledujeme chování CA, nebo jako výstup můžeme brát odpověď na otázku, jestli je se schopný CA se zadaným vstupem dostat do stabilního stavu. V této oblasti je například zajímavý model predikce toho, jak se bude šířit požár v lese. Pro tento model je využit 2D CA, do kterého je na počátku uložena různorodost lesu, povětrnostní podmínky a topologie půdy. Výstupem je dynamika šíření požáru v lese [8].

Další úloha je získání modelu na základě předem daného vstupu a výstupu. Takto získanou přechodovou funkci lze následně aplikovat na predikci vstupů u kterých neznáme výstup. Zde si představíme pár úloh, které se využívají jako benchmarky při návrhu přechodové funkce pro CA.

Problém majority

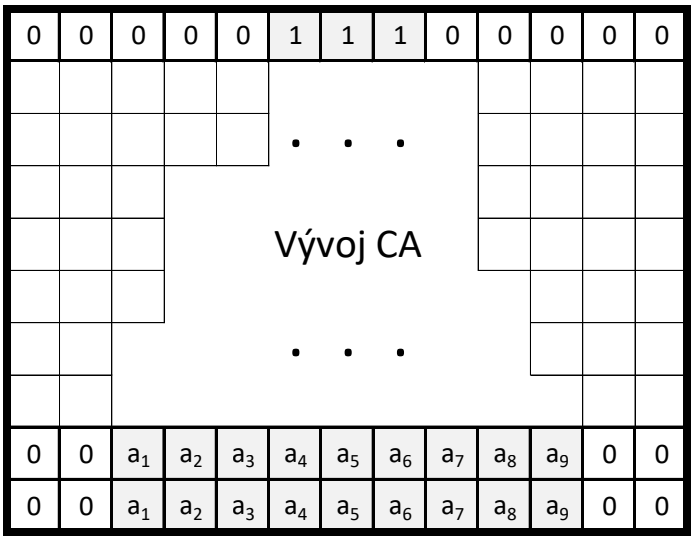
Jednou z nejčastěji používaných úloh, pro zjištění efektivity automaticky navrhovaných přechodových funkcí CA je problém majority. Zde se snažíme v jednorozměrném dvoustavovém automatu navrhnout takovou přechodovou funkci, která je schopna CA nastavit do hodnoty samých 0, nebo 1 na základě toho, která z těchto hodnot je majoritní na počátku simulace. V této úloze se snažíme navrhnout takovou přechodovou funkci, která řeší tento problém pro co nejvíce případů. Je totiž dokázáno, že tuto úlohu nelze vyřešit v CA dokonale [11].

Problém synchronizace

U těchto úloh se snažíme dosáhnout v CA takového chování, která po určitém počtu vykazuje synchronní změny všech stavů v automatu. Tyto úlohy jsou buď definovány pro konkrétní počáteční stav, nebo jako úlohy, které mají řešit synchronizaci libovolného vstupu. Mezi nejznámější úlohy patří *firing squad synchronization problem*. V této úloze máme na počátku jednu aktivní buňku a požadujeme, aby po co nejkratším počtu kroků se všechny buňky v automatu nastavily do stejného nenulového stavu. Hodnota tohoto výsledného stavu musí byt unikátní pro celý automat a nepoužita nikde jinde krom výsledného „výstřelu“ [17].

Výpočet druhé mocniny v CA

V této úloze se snažíme navrhnout přechodovou funkci takovou, aby nám ze vstupu délky x vytvořila stabilní výstup o délce x^2. Je mnoho způsobu jak vstup do CA zakódovat, ať už jako posloupnost jedniček o délce x (viz obrázek 4.3), nebo lze tuto posloupnost buněk rozšířit o jednu buňku jiné hodnoty [21]. To samé platí pro výsledek, kdy můžeme striktně vyžadovat posloupnost délky x^2 stejných výsledných hodnot, nebo to může byt posloupnost délky x^2, kde stačí aby obsahovala jen nenulové hodnoty. Dále je žádáno, aby tento výsledný stav byl stabilní.



Obrázek 4.3: Ukázka vstupu a výstupu při výpočtu druhé mocniny pro číslo 3 za pomoci CA. Platí zde, že každé $a_x \neq 0$.

Cílem této práce bude návrh přechodové funkce pro řešení úlohy výpočtu druhé mocniny v CA. Tato úloha byla zvolena z důvodu škálovatelné náročnosti podle požadavků na řešení, protože zde můžeme buď řešit výpočet druhé mocniny konkrétního čísla, nebo se můžeme pokusit o hledání takového řešení, které je schopno vypočítat druhou mocninu libovolného čísla. Toho je využito hlavně při počátečních implementacích, kdy na jednoduchých úlohách můžeme ověřit funkčnost autorem navrženého algoritmu. Navíc tato úloha nemá dlouhé simulační výpočty, na rozdíl od problému majority, takže je zde jednodušší spustit více generací za stejnou dobu, a tím efektivněji testovat navržený algoritmus. Přesné definice experimentálních úloh jsou uvedeny v kapitole 7.

Kapitola 5

Návrh celulárních automatů za pomoci mravenčích algoritmů

V této kapitole se zaměříme na to, jak lze využít mravenčí algoritmy k získání pravidel pro námi zadanou úlohu v celulárním automatu. Mezi nejdůležitější části patří zjištění možností, jak lze pro řešenou úlohu vytvořit konstrukční graf, definování všech omezujících podmínek Ω a určení významu feromonové stopy ve vztahu k řešení. Po vyřešení těchto počátečních problémů je dále vhodné zjistit, jaké heuristické ohodnocení této úlohy lze vytvořit a jaký mravenčí algoritmus zvolit pro efektivní řešení dané úlohy. Po implementaci algoritmu dle návrhu je nutné zjistit parametry pro jeho optimální funkci.

5.1 Konstrukční graf celulárního automatu

První je třeba definovat, jak budou reprezentovány přechodové funkce CA v konstrukčním grafu a jak následně bude tento graf dekódován po vytvoření cesty mravencem. Zde autor zvolil kódování, které obsahuje všechna možná pravidla pro daný CA. Díky tomu lze průchodem grafem získat libovolnou přechodovou funkci obsahující tato pravidla. K takto navrženému grafu je však potřeba vytvořit omezující podmínky Ω. $\Omega(1)$ říká, že pro každou vstupní kombinaci hodnot musí být definováno pravidlo a $\Omega(2)$ zakazuje vznik dvou různých pravidel se shodnou kombinací vstupních hodnot.

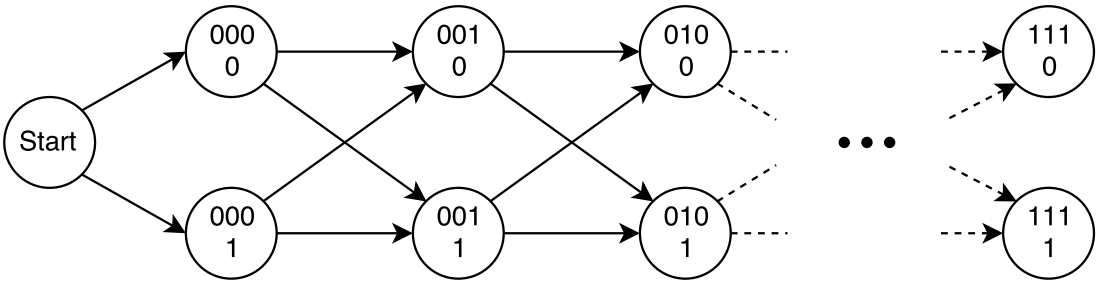
Když známe omezující podmínky Ω a způsob vytváření přechodových pravidel, tak je potřeba navrhnout samotný konstrukční graf, ve kterém se mravenci budou pohybovat. Pro tento návrh autor vycházel z vlastnosti, že nelze o žádných dvou pravidlech s jistotou říct, že mají mezi sebou nějaký vztah, který bude vždy platit nezávisle na zbytku vybraných pravidel. Třeba u úlohy obchodního cestujícího velmi často platí, že výsledná trasa bude obsahovat v posloupnosti takové cesty, které jsou globálně nejkratší. Zde je vzdálenost mezi 2 městy předem danou neměnnou vlastností, která se při zvolení jiné trasy také nemění. Ovšem toto nelze říct o vztahu pravidel v CA, kdy užitečnost dvou pravidel se zadanými výstupy může být velmi rozdílná v kontextu ostatních vybraných pravidel. Jedno řešení, kde se objevují tato dvě pravidla, nám mohou dát maximální fitness[1] a mohou být řešení, kdy s těmito pravidly bude fitness nulová.

[1] **Fitness** je kvantitativní míra ohodnocení kvality řešení.

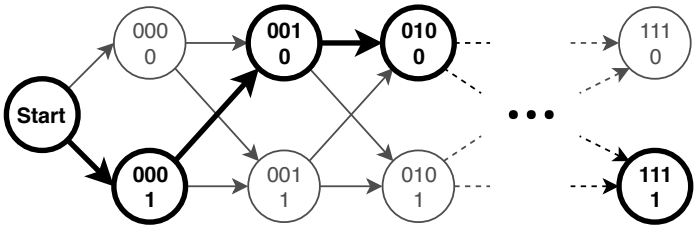
5.1.1 Aplikovaný konstrukční graf

Z předchozích poznatků autor navrhl konstrukční graf, kde mravenec průchodem grafu vytváří úplnou tabulkovou přechodovou funkci. Množina uzlů C je v tomto grafu definována jako množina všech existujících pravidel v daném CA. Navíc do této množiny C je přidán ještě startovací uzel, který je zde využit jako počáteční uzel pro všechny mravence. Tento graf dále využívá množinu orientovaných hran E, které dělí tento graf na segmenty. Pro CA s počtem stavů k a okolím r bude počet segmentů k^{r+1}, přičemž v segmentu se vybírá z k pravidel. To odpovídá tomu, že v každém segmentu jsou pravidla se stejnou vstupní kombinaci a rozdíly mají jen ve výstupu.

Pro znázornění je zde uveden příklad konstrukčního grafu pro 2D CA s počtem stavů ($k = 2$) s okolím $r = 1$ (obrázek 5.1). Zde lze vidět segmentové oddělení pravidel, kdy každý sloupec obsahuje vždy jen pravidla se stejnou vstupní kombinací. Průchodem mravence od startovního uzlu přes uzly definující pravidla získá mravenec úplnou tabulkovou přechodovou funkci. V obrázku 5.2a máme ukázkový průchod mravence grafem a k němu vytvořenou tabulku přechodových pravidel (tabulka 5.2b).



Obrázek 5.1: Konstrukční graf pro CA s počtem stavů $k = 2$ a velikostí okolí $r = 1$. V uzlech jsou popsány pravidla, kdy první řádek reprezentuje vstupní kombinaci hodnot a druhý řádek popisuje výstup daného pravidla.



(a) Ukázkový průchod mravence grafem 5.1.

c_{i-1}	c_i	c_{i+1}	výstup
0	0	0	1
0	0	1	0
0	1	0	0
...			...
1	1	1	1

(b) Výsledná tabulka přechodových pravidel.

Obrázek 5.2: Ukázka vygenerované tabulky přechodových pravidel na základě průchodu mravence grafem.

Důležité rozhodnutí v návrhu je volba uložení feromonů do uzlů místo hran. To bylo primárně zvoleno z důvodu, aby hodnota feromonů udávala preferovanost daného výstupu pro danou vstupní kombinaci hodnot, bez libovolné závislosti na výběru předchozího pravidla. Kdyby feromony byly uloženy na hranách, tak výběr dalšího pravidla by byl závislý na předchozím vybraném pravidlu.

Heuristická funkce pro návrh přechodových funkcí CA není používána. U CA není totiž známa nějaká apriori neměnná vlastnost, která by byla využitelná od počátku běhu algoritmu v celém grafu. Jediná možná heuristická funkce by mohla být definice pravidel, která se musí v přechodové funkci objevovat, dle typu řešené úlohy pro CA. Například je mnohdy žádoucí, aby pravidlo obsahující jen nulové vstupní stavy mělo nulový výstup. Avšak tato heuristika ovlivňuje jen konkrétní uzly a bylo by ji potřeba nastavit pro každou řešenou úlohu v CA zvlášť.

Tento konstrukční graf byl použit z důvodu výrazného urychlení výpočtů pravděpodobnostního výběru a celkového zrychlení běhu, pokud nevyžadujeme ukládání vztahů mezi všemi uzly, ve srovnání s konstrukčním grafem uvedeným v podkapitole 5.1.2. Takto navržený graf ma výhodu, že automaticky splňuje podmínky Ω, protože při celém průchodu grafem získáme kompletní tabulku přechodů, která je definována pro všechna pravidla s unikátními vstupy.

Další užitečná vlastnost tohoto grafu je, že složitost na jeho průchod a vyhodnocení narůstá lineárně s počtem možných vstupních kombinací, a jeho velikost se lineárně odvíjí od počtu pravidel. Kdyby byl použit graf s plným propojením, tak bychom se zde museli potýkat s vyšší výpočetní náročností, z důvodu většího počtu uzlů v grafu, které by byly uváženy pro výběr následujícího uzlu, což je podrobněji popsáno v následující podkapitole 5.1.2.

5.1.2 Alternativní návrh konstrukčního grafu

Další autorem navrženou variantou byl plně propojený graf, kde jsou uzly definovány stejně jako v návrhu grafu 5.1.1. V případě uložení feromonů na hrany by se dal tento model využít pro hledání skrytých závislostí mezi konkrétními uzly obsahujícími pravidlo. Ovšem paměťová a výpočetní náročnost takového modelu by byla velmi vysoká. Například pro 1D CA, s velikost okolí $r = 1$ a počtem pravidel $s = 8$ by byl počet uzlů $N = s^{2+2r} = 4096$ a počet hran v tomto úplném grafu by byl $E_{full} = 8386560$. Pokud bychom použili v tomto modelu uložení na uzlech, tak bychom snížili paměťovou náročnost tohoto algoritmu. Využitá paměť v tomto případě by byla shodná s náročností na paměť, kterou má konstrukční graf na obrázku 5.1. Ovšem počet uzlů, ze kterých se vybírá následující krok mravence, by byl podstatně větší. Navíc by se muselo ošetřovat, aby mravenec nevybíral uzly s pravidly, které by měly shodnou vstupní hodnoty jako již vybrané pravidla, takže generování cesty mravence by trvalo delší dobu.

5.2 Použité mravenčí algoritmy

Pro řešení návrhu přechodových funkcí pro CA byly zvoleny dva algoritmy, přičemž první experimenty byly provedeny s algoritmem EAS, který je jednoduchý na implementaci a ladění. Po zjištění limitů tohoto algoritmu byly dále prováděny pokusy s $\mathcal{MM}$AS, který byl schopen dosáhnout podstatně lepších výsledků.

5.2.1 Návrh CA pomoci EAS

První pokusy byly provedeny na algoritmu vycházející z EAS, který je popsán v podkapitole 2.3. Tento algoritmus byl vybrán kvůli nízké implementační náročnosti a dobré schopnosti využít znalosti získané z nejlepšího řešení. CA mají totiž velmi mnoho přechodových funkcí, pro které bude výsledná fitness hodnota velmi blízká nebo rovna nule. Proto je důležité,

aby algoritmus začal co nejdříve efektivně využívat nalezené kvalitní řešení a prozkoumávat jeho „okolí“. Kdyby byl použit AS, mohlo by se stát, že toto nalezené řešení by bylo ztraceno v záplavě ostatních nekvalitních řešení.

Další důvod pro zvolení tohoto algoritmu byl relativně nízký počet parametrů, které je potřeba ladit oproti pokročilejším alternativám. To znamená, že bylo jednodušší zjistit správné nastavení parametrů a následně ověřit, jestli lze vůbec navrhovat CA za pomoci mravenčích algoritmů. Tento přístup se ukázal jako správný, kdy algoritmus pro návrh pravidel CA využívající EAS byl schopen vyřešit úlohu, kdy je potřeba navrhnout přechodovou funkci pro výpočet druhé mocniny ze zadaného čísla.

V této úloze se ovšem i objevila limitace tohoto algoritmu, kdy algoritmus byl velice náchylný k uváznutí a velmi často končil v lokálních extrémech. To bylo způsobeno vysokou hodnotou feromonů na některých uzlech grafu po delším běhu algoritmu, což omezilo prohledávací schopnosti v pozdějších fázích běhu. Například tento algoritmus měl velké problémy s návrhem přechodové funkce pro výpočet druhé mocniny, kdy už předpokládáme funkčnost v předem zadaném rozsahu čísel. Další cesta tedy byla použít algoritmus, který obsahuje limitaci minimálních a maximálních hodnot feromonů, které by měly uváznutím zabránit.

5.2.2 Návrh CA pomocí $\mathcal{MM}$AS

Další pokusy byly prováděny na algoritmu $\mathcal{MM}$AS (vysvětlen v podkapitole 2.5) z důvodu lepší schopnosti se vyhnout uváznutí. Algoritmus $\mathcal{MM}$AS lze často vidět jako nejefektivnější pro řešení různých úloh za pomoci mravenčích algoritmů, což lze potvrzeno i v kapitole 3. Díky vlastnostem tohoto algoritmu a pár přidaným úpravám byl tento algoritmus schopen řešit návrh pravidel pro výpočet druhé mocniny při zadání rozsahu čísel na rozdíl od implementace EAS zmíněné v podkapitole 5.2.1.

Tento algoritmus byl rozšířen o dvě vlastnosti, které jsou inspirovány článkem *MAX–MIN Ant System* [20]. První rozšíření bylo upravení původního formátu reinicializace, který nastaví hodnotu všech feromonů na τ_{max}. Ta byla nahrazena parametrizovanou reinicializací, kdy dokážeme ovlivnit množství feromonů, které bude navráceno do hodnoty τ_{max} za pomoci parametru s. Výpočet parametrizované reinicializace je následovný:

$$\tau_{ij} \longleftarrow \tau_{ij} + (\tau_{max} - \tau_{ij}) \cdot s \tag{5.1}$$

Druhé rozšíření se týká výběru jedince bude přidávat feromony. Jestli jedinec s nejlepším globálním řešením C^{bs}, nebo nejlepší jedinec z dané iterace C^{bi}. To je zde řešeno náhodným výběrem, kdy jsme parametrem schopni ovlivnit šanci výběru jedince C^{bs}.

5.3 Výpočet druhé mocniny v CA mravenčími algoritmy

V této diplomové práci budeme řešit úlohu návrhu přechodové funkce CA pro výpočet druhé mocniny pro zadaný vstupní rozsah čísel, zmíněnou v podkapitole 4.2. Tato varianta byla zvolena kvůli možnosti si variabilně vybírat náročnost zadané úlohy. Pokud chceme vytvořit jednodušší zadání, tak se budeme snažit navrhnout přechodovou funkci pro výpočet druhé mocniny z konkrétní hodnoty v CA. Těžší varianta zahrnuje částečné zobecnění této úlohy, kdy požadujeme aby nám algoritmus vytvořil přechodovou funkci, která je schopna řešit výpočet druhé mocniny v CA pro zadaný rozsah čísel. Těžší varianta této úlohy byla řešena i v článku *Evolution of Generic Square Calculations in Cellular Automata* [1], kde

touto metodou byly nalezeny přechodové funkce umožňující vypočítat druhou mocninu z libovolného čísla.

Pro simulaci využíváme konečný CA, který má okrajovou podmínku nastavenou na konstantní hodnotu rovnu nule. Šířka automatu w a maximální počet kroků s v simulaci se vypočítá podle aktuálně řešené hodnoty x následovně:

$$w = x^2 \cdot 3 \tag{5.2a}$$

$$s = x^2 \cdot 4 \tag{5.2b}$$

,kde hodnoty 3 a 4 jsou parametry, přičemž tyto hodnoty byly zvoleny experimentálně podle testovacích běhu. Konstanta x^2 byla autorem zvolena z důvodu, že výpočetní náročnost se odvíjí od počtu buněk, které popisují výslednou hodnotu.

Důvod zavedení dynamického výpočtu hodnot w a s bylo zamezení plýtvání výpočetního času pro výpočty s velkým rozsahem vstupních čísel. Při statickém nastavení těchto hodnot by bylo potřeba automat nastavit dle největší hledané hodnoty. To by při simulaci úloh, kde je zadán velký rozsah čísel, znamenalo vysokou výpočetní náročnost. Například při výpočtu hodnot 3–10 hodnotu 3 zde simulujeme na CA o šířce $w = 27$. V CA o statické velikosti by bylo $w = 300$. Vzhledem k tomu, že úspora je u všech hodnot kromě maximální, tak ušetření času na simulacích je při této metodě markantní.

Dále bylo potřeba navrhnou výpočet fitness funkce. Ta hodnotí kvalitu nalezené přechodové funkce pro výpočet druhé mocniny z čísla x. Hodnotu fitness získáme tak, že spustíme simulaci CA pro zadané x a čekáme, která z následujících tří ukončovacích podmínek nastane:

1. Změna okrajové buňky automatu.
2. Automat překročil maximální počet kroků s.
3. Dva po sobě následující kroky jsou stejné.

První dvě varianty ukončení nastaví hodnotu $fitness_{pow\_x} = 0$. V prvním případě přechodová funkce nesplňuje podmínku obecnosti, kdy simulace je závislá na přesné šířce automatu. Tato podmínka je zde zavedena díky informacím získaných z prvních pokusů s návrhy CA za pomocí EAS. Autor při nich zjistil, že část řešení má tendenci využívat okrajové podmínky s permanentní nulou, což není žádoucí pro tuto úlohu, kdy se snažíme dosáhnout řešení nezávislého na šířce automatu. V druhém případě zde nevzniká stabilní výstup a jako takový nesplňuje podmínky výpočtu druhé mocniny v CA.

Při splnění podmínky 3., tedy že CA se dostal do stabilního stavu, nastává fáze výpočtu hodnoty fitness dle hodnot buněk CA v posledním kroku. Výpočet probíhá tak, že pro vstupní číslo x je vytvořeno okno o velikosti x^2 buněk. Toto okno posouváme přes všechny možné pozice v automatu a zjišťujeme počet buněk značící výsledek v okně, označených h, a počet buněk značící výsledek mimo okno, označených z. Zde hledáme takovou pozici okna p, kde hodnota $h_p - z_p$ je maximální. Tato funkce byla navržena s ohledem na to, že šířka automatu se může v průběhu ladění měnit a takto definovaná fitness funkce dá vždy stejný výsledek nezávisle na šířce automatu[2].

V moment kdy známe pozici okna pm, kde hodnota $h_{pm} - z_{pm}$ je pro daný výsledek simulace CA maximální, následuje výpočet normalizované hodnoty fitness. Ta je navíc následně odmocněna, což se projevilo jako velmi výhodné hlavně při aplikaci algoritmu

[2]Může nastat situace, kdy se změní fitness hodnota, protože automat nebude dost široký pro účely simulace podle daných pravidel.

EAS, kdy nelineární křivka fitness funkce zrychlila konvergenci algoritmu. Funkci pro získání normalizované fitness hodnoty lze zapsat následovně:

$$fitness_{pow\_x} = \begin{cases} \sqrt{\frac{h_{pm} - z_{pm}}{x^2}}, & \text{pokud } (h_{pm} - z_{pm}) \geq 0 \\ 0, & \text{pokud } (h_{pm} - z_{pm}) < 0 \end{cases} \tag{5.3}$$

V případě, že hledáme druhou mocninu konkrétního čísla, tak hodnota $fitness_{pow\_x}$ je použita jako kvalitativní ohodnocení řešení mravence, které se aplikuje ve fázi přidávání feromonů.

Pokud hledáme přechodovou funkci pro výpočet druhé mocniny ze zadaného rozmezí čísel, tak využíváme vážený průměr přes všechna získaná $fitness_{pow\_x}$ pro každé x. Vážený průměr byl zde použit z důvodu, že algoritmy měly tendenci vyřešit úlohu pro malé vstupní hodnoty z rozsahu. Po vyřešení těchto nižších hodnot mnohdy nastalo uváznutí řešení a algoritmus nedokázal vytvořit přechodovou funkci, která by tuto úlohu řešila obecněji v celém rozsahu zadaných hodnot. Pro rozmezí $x_{min} - x_{max}$ vypadá vzorec pro výpočet fitness mravence následovně:

$$fitness_{ant} = \frac{\sum_{x=x_{min}}^{x_{max}} fitness_{pow\_x} \cdot x}{\sum_{x=x_{min}}^{x_{max}} x} \tag{5.4}$$

,kde hodnota $fitness_{ant}$ je v tomto případě využita jako kvalitativní ohodnocení řešení daného mravence.

Kapitola 6

Implementace

V této kapitole se zaměříme na to jak byly algoritmy EAS a $\mathcal{MM}$AS implementovány dle předchozích poznatků na úlohu výpočtu druhé mocniny v CA. Obě implementace byly vytvářeny s ohledem na to, že testy budou spouštěny paralelně, takže lze je zkompilovat jak pro standardní tak i pro paralelní spouštění. Programy do uživatelem zadané složky tisknou informace o nastavených parametrech běhu a podrobné informace o každém spuštěném běhu. Dále ještě program po dokončení na výstup vytiskne souhrnné informace o běhu.

6.1 Elitist ant system

Heuristika EAS nebyla nijak upravena pro tuto úlohu a je procesně shodná s popisem v podkapitole 2.3. Jako řešení, které je používáno jako elitistcké, je zde použito globální nejlepší řešení, z důvodu rychlejší konvergence k požadovanému výsledku. Konstrukční graf a výpočet fitness ze simulací CA je zde implementováno dle kapitoly 5.

Vlastnosti algoritmu lze upravit dle následujících parametru v hlavičkovém souboru, přičemž některé hodnoty mohou byt nastaveny i parametrem při spuštění programu, kdy přepínač parametru je popsán v závorce:

- `PRINT_INTENSITY` – Nastavení četnosti výpisu podrobných informaci o běhu. Hodnota udává který násobek generace se bude tisknout.
- `ANT_COUNT` (`-a`) – Počet mravenců v populaci.
- `INIT_PHEROMONE` – Počáteční hodnota feromonů.
- `EVAPORATION` – Míra evaporace feromonů v grafu. Hodnota musí být v rozmezí $\langle 0; 1)$.
- `ELITIST_MULT` – Multiplikátor přidaných feromonů pro elitistické řešení.
- `MAX_STEPS_ANTS` (`-s`) – Maximální počet generací EAS.
- `RULE_VARIABLES` – Počet stavů, který může buňka nabývat. Tato hodnota musí být větší než 1. Množina stavů je určena následovně: $S = \{0, 1, \ \ldots\ , \texttt{RULE\_VARIABLES} - 1\}$.
- `CA_WIDTH_MULT` – Multiplikátor šířky simulovaného CA.
- `CA_STEPS_MULT` – Multiplikátor maximálního počtu kroků simulovaného CA.

  Výpočet šířky a počtu kroků CA vychází z vyhodnocovaného čísla x, kdy jako základ je použita hodnota x^2 a ta je násobena multiplikačním parametrem.
- `START_RULE_VARIABLE` – Počáteční vstupní hodnota pro řešení úlohy.

- `VARIABLES_TESTED` (`-x`) – Počet testovaných proměnných pro řešení úlohy.

  Testovány jsou tedy hodnoty patřící do následovného celočíselného rozsahu: ⟨`START_RULE_VARIABLE` ; `START_RULE_VARIABLE` + `VARIABLES_TESTED` - 1⟩.
- `OUT_FOLDER` (`-o`) – Výstupní složka, do které se generují výsledky. Tato složka musí být vytvořena před spuštěním programu.
- `JOB_ARRAY` (`-r`) – Zadání pořadového čísla běhu. Pří paralelním spouštění využito jako označení pořadí této skupiny běhů.

6.2 $\mathcal{MAX}$–$\mathcal{MIN}$ ant system

Implementace zde vychází z popisu algoritmu v podkapitole 2.5 s využitím vylepšení zmíněných v podkapitole 5.2.2.

Dále bylo potřeba vyřešit nastavení hodnot τ_{min} a τ_{max}. Pro počátek byla hodnota τ_{max} nastavena jako fitness, kterou bychom získali při vypočítání fitness vstupu. τ_{min} je následně odvozena z hodnoty τ_{max}, v závislosti na velikosti parametru `PHEROMONE_RANGE`. Výpočet hodnoty τ_{min} je následovný:

$$\tau_{min} = \tau_{max}/\texttt{PHEROMONE\_RANGE} \tag{6.1}$$

τ_{max} je vždy aktualizován na novou hodnotu maximální fitness, pokud tato hodnota přesahuje aktuální τ_{max}. Zároveň je případně aktualizována i τ_{min} opět dle vzorce 6.1.

Reinicializace feromonů zde probíhá na základě průměrné fitness populace. Experimenty prokázaly, že tato vlastnost dokáže velmi odhalit stagnaci algoritmu na této úloze. Reinicializace feromonů používá dva parametry. `PTS_CHECK`, který říká jak často si algoritmus bude zjišťovat průměrnou fitness populace a `RESET_LIMIT`, který určuje, kolikrát musí byt stagnace detekována, před provedením reinicializace. Primární roli zde hraje uložená průměrná fitness. Ta je inicializována na 0 a novou hodnotu získá pokud je při kontrole stagnace detekována jedna z následujících situací:

1. Zvýšila se globální maximální fitness hodnota.
2. Aktuální průměrná fitness je vyšší než uložená průměrná fitness.

V momentě kdy se aktualizuje uložená průměrná fitness, tak se resetuje čítač stagnace t na hodnotu 0. Jinak nastává potvrzení stagnace inkrementací hodnoty t. V momentě kdy t == `RESET_LIMIT` nastává reinicializace feromonů v grafu. Reinicializace je provedena podle vzorce 5.1, kdy za hodnotu s je dosazena hodnota z parametru `TRAIL_RESET_RATIO`. Konstrukční graf a výpočet fitness byl použit podle kapitoly 5.

- `PRINT_INTENSITY` – Nastavení četnosti výpisu podrobných informaci o běhu. Hodnota udává který násobek generace se bude tisknout.
- `ANT_COUNT` (`-a`) – Počet mravenců v populaci.
- `MAX_STEPS_ANTS` (`-m`) – Maximální počet generací MMAS.
- `EVAPORATION` (`-e`) – Míra evaporace feromonů v grafu. Hodnota musí být v rozmezí $\langle 0; 1)$.
- `PHEROMONE_RANGE` (`-f`) – Určení rozsahu mezi τ_{min} a τ_{max}.
- `RULE_VARIABLES` (`-v`) – Počet stavů, který může buňka nabývat. Tato hodnota musí být větší než 1. Množina stavů $S = \{0, 1, ..., \texttt{RULE\_VARIABLES} - 1\}$.

- `CA_WIDTH_MULT` (`-w`) – Multiplikátor šířky simulovaného CA.
- `CA_STEPS_MULT` (`-c`) – Multiplikátor maximálního počtu kroků simulovaného CA.

  Výpočet šířky a počtu kroků CA vychází z vyhodnocovaného čísla x, kdy jako základ je použita hodnota x^2 a ta je násobena multiplikačním parametrem.
- `PTS_CHECK` (`-p`) – Nastavuje jak často bude prováděna kontrola na stagnaci.
- `RESET_LIMIT` (`-l`) – Nastavuje kolikrát musí být detekována stagnace před reinicializací algoritmu.
- `TRAIL_RESET_RATIO` (`-t`) – Nastaví poměr feromonů, které se vrátí do hodnoty τ_{max} při reinicializaci. Hodnota musí být v rozmezí $\langle 0; 1)$.
- `GLOBAL_SOLUTION_PREFERENCE` (`-g`) – Nastavuje pravděpodobnost výběru nejlepšího globálního řešení. Hodnota musí být v rozmezí $\langle 0; 1)$.
- `START_RULE_VARIABLE` (`-s`) – Počáteční vstupní hodnota pro řešení úlohy.
- `VARIABLES_TESTED` (`-x`) – Počet testovaných proměnných pro řešení úlohy.

  Testovány jsou tedy hodnoty patřící do následovného celočíselného rozsahu: $\langle$`START_RULE_VARIABLE` ; `START_RULE_VARIABLE` + `VARIABLES_TESTED` - 1$\rangle$.
- `OUT_FOLDER` (`-o`) – Výstupní složka, do které se generují výsledky. Tato složka musí být vytvořena před spuštěním programu.
- `JOB_ARRAY` (`-r`) – Zadání pořadového čísla běhu. Pří paralelním spouštění využito jako označení pořadí této skupiny běhů.

6.3 Spouštění

Obě dvě implementace lze přeložit za pomocí programu *make*, kdy pokud chceme získat verzi programu pro paralelní spouštění, tak je potřeba zavolat `make` s parametrem `MPI`. Pro ukázkové spuštění jsou ve složce `scripts` nachystány skripty `run.sh` a `runMPI.sh`. Oba skripty vytvoří výstupní složku dle parametru v hlavičce skriptu, v této složce vytvoří soubor pro souhrnné výsledky a program přeloží. Pokud byl spuštěn `run.sh`, tak je přeložena a spuštěna standardní verze algoritmu. V případě `runMPI.sh` je přeložena verze pro paralelní spouštění a následně je spuštěna paralelní verze algoritmu. Pro toto přeložení a spuštění paralelní verze je potřeba mít nainstalovaný balíčky `mpicc` a `mpiexec`.

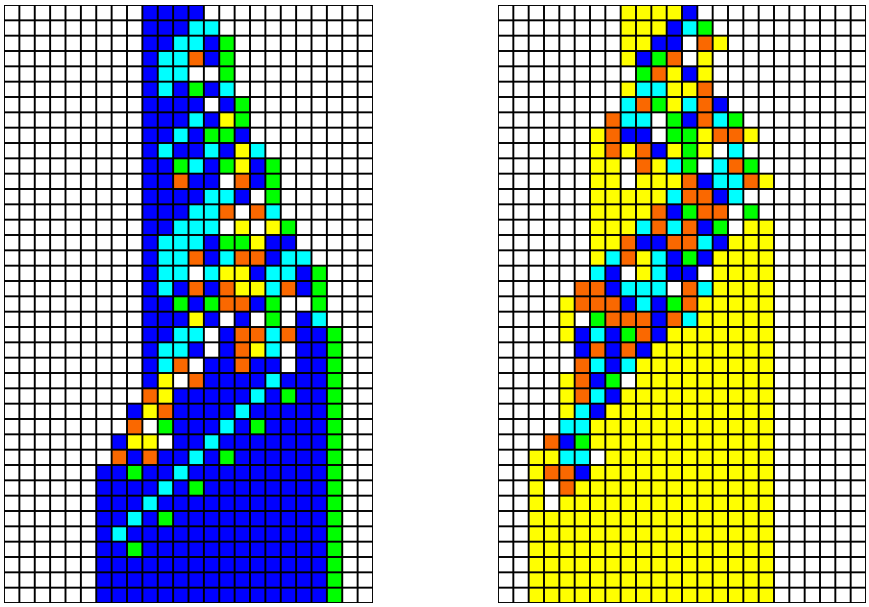
Kapitola 7

Experimentální výsledky

Vyhodnocení navrženého řešení probíhalo na úloze výpočtu druhé mocniny (viz podkapitoly 4.2 a 5.3). Pro tuto úlohu byly zvolena dvě různá zadání, které mají rozdílné kódování vstupní hodnoty x a výsledné hodnoty.

1. Vstupní počítaná hodnota x je uložena do CA automatu jako posloupnost jedniček. Ve výstupu hledáme nenulovou posloupnost čísel o délce x^2, přičemž všechny ostatní buňky musí být nulové (obrázek 7.1a).

2. Vstupní počítaná hodnota x je uložena do CA automatu jako posloupnost dvojek, která je následována jedničkou. Ve výstupu hledáme posloupnost čísel o délce x^2 obsahující čísla $i > 1$ a pro všechny buňky mimo tuto posloupnost musí platit, že jejich hodnota $i \leq 1$. Tato úloha vychází z knihy *A new kind of science* [21], kde tato konkrétní úloha byla řešena (obrázek 7.1b).



(a) Zadání 1 (b) Zadání 2

Obrázek 7.1: Ukázkové řešení obou variant zadání v CA, pro $x = 4$.

Tyto úlohy byly testovány na výpočetním clusteru Anselm, který patří pod instituci IT4Innovations národní superpočítačové centrum. Tyto výpočetní uzly jsou osazeny procesory Intel Sandy Bridge E5-2665, 2.4 GHz, na kterých probíhaly vyhodnocovací běhy.

Počty generací pro algoritmus EAS a $\mathcal{MM}$AS byly vybrány tak, aby oba měly stejnou výpočetní náročnost v ohledu na počet vytváření cest v grafu. Toho bylo dosaženo tak, že se požadovalo aby součin velikosti populace a počtu generací musí být pro oba algoritmy stejný. Ostatní parametry byly nastaveny podle zkušeností z počátečních experimentů.

Parametr	**Hodnota**
Počet mravenců v populaci (`Ant_count`)	60000
Maximální počet generací	250
Množství počátečních feromonů	`Ant_count`/100
Míra evaporace	0,3
Elitistický multiplikátor	`Ant_count`/200
Šířka simulovaného CA	$x^2 \cdot 4$
Maximální počet kroků CA	$x^2 \cdot 5$

Tabulka 7.1: Nastavení parametrů EAS pro experimenty při vstupní hodnotě x.

Parametr	**Hodnota**
Počet mravenců v populaci	750
Maximální počet generací	20000
Míra evaporace	0,005
Rozsah mezi τ_{min} a τ_{max}	10000
Šířka simulovaného CA	$x^2 \cdot 4$
Maximální počet kroků CA	$x^2 \cdot 5$
Velikost kroku pro kontrolu na reinicializaci	20
Počet detekovaných stagnaci pro reinicializaci	6
Množství navrácených feromonů do τ_{max}	0,025
Preference globálního nejlepšího řešení	0,85

Tabulka 7.2: Nastavení parametrů $\mathcal{MM}$AS pro experimenty při vstupní hodnotě x.

7.1 Srovnání EAS a $\mathcal{MM}$AS

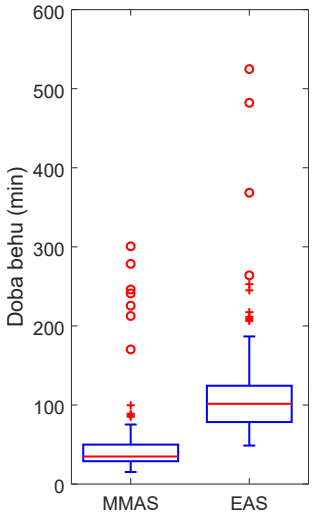
K srovnání výkonnosti těchto algoritmů bylo využito 1. varianty zadání úlohy výpočtu druhé mocniny v CA, pro které bylo hledáno řešení druhé mocniny čísla 15. Další vyhodnocení bylo provedeno opět na 1. variantě úlohy, kde vstup pro tuto úlohu bylo rozmezí hodnot 4 – 9. Tyto dvě úlohy byly zvoleny kvůli jejich rozdílné náročnosti na vytvoření přechodové funkce. Nastavení EAS a $\mathcal{MM}$AS odpovídalo tabulkám 7.1 a 7.2. Pro každý experiment bylo provedeno 96 běhů programu.

Hledání druhé mocniny čísla 15

Tuto úlohu byly oba algoritmy schopny vyřešit bez problému za daných podmínek. Všechny běhy zde našly přechodovou funkci řešící tento výpočet. Kde se ovšem tyto algoritmy výrazně lišily, tak byla doba potřebná k nalezení řešení. Zde algoritmus $\mathcal{MM}$AS znatelně

převyšoval schopnosti algoritmu EAS, kdy potřeboval kratší výpočetní čas pro získání vyhovujícího řešení. Když porovnáme mediány počtu generací $\mathcal{MM}$AS a EAS z tabulky 7.2b, tak pro získání v EAS bylo potřeba vygenerovat $34 \cdot 60000 = 2,04$ miliónů cest v grafu. Oproti tomu $\mathcal{MM}$AS stačilo pro nalezení výsledného řešení jenom $963 \cdot 750 \approx 722$ tisíc cest v grafu, což je zhruba 3x menší počet výpočtů. To odpovídá i časové náročnosti, kdy EAS mělo výpočetní náročnost zhruba 3x vyšší, což lze vidět v grafu 7.2a.

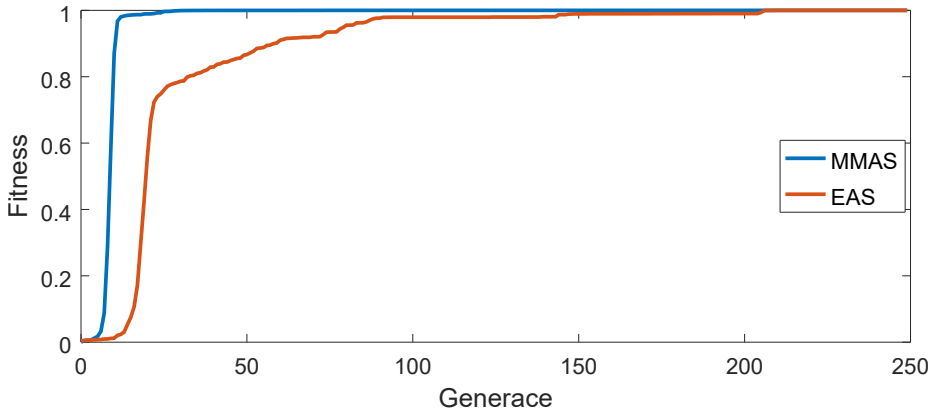
Zajímavý je i graf průměrného vývoje fitness 7.2c, kde lze vidět výrazný rozdíl v konvergenci obou algoritmů. Jeden z hlavních rozdílu je vlastnost algoritmu EAS, který potřebuje více času než se začnou objevovat první řešení s vyšší fitness a následné tendence k uváznutí v lokálních maximech. Popisná osa zde byla zvolena podle počtu generací EAS, přičemž 1 generace EAS odpovídá 80 generacím v $\mathcal{MM}$AS.



(a) Porovnání délky běhů

Algoritmus	$\mathcal{MM}$AS	EAS
Počet úspěšných běhů	100 %	100 %
Počet generací – medián	963	34
Doba výpočtu – medián (min)	34,8	101,5

(b) Porovnání výsledků běhů algoritmů



(c) Průměrný vývoj fitness, generace $\mathcal{MM}$AS jsou škálovány vůči EAS

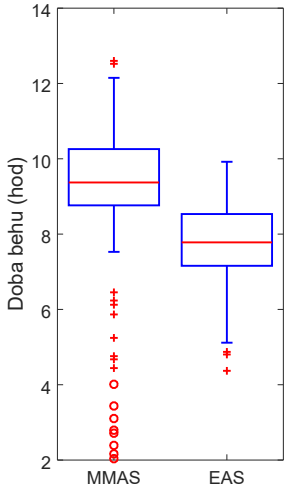
Obrázek 7.2: Porovnání algoritmů $\mathcal{MM}$AS a EAS pro zadanou hodnotu 15

Hledání druhé mocniny pro čísla 4 – 9

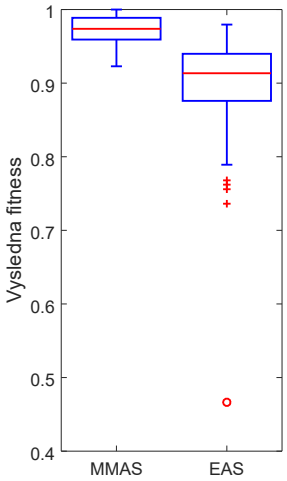
Na této úloze se projevila slabina algoritmu EAS. Algoritmus zde postrádal schopnost dostat se z lokálních extrému při řešení zadaného rozsahu. To lze pozorovat na celkové nižší fitness

v grafu 7.3b, kde pro algoritmus EAS se objevují extrémy s fitness hodnotou nižší než 0,8. To je dále potvrzeno v grafu průměrného vývoje fitness hodnoty 7.3d, ve kterém lze vidět podstatně pomalejší růst této hodnoty než v algoritmu $\mathcal{MM}$AS.

Jediné místo, kde by se mohlo zdát, že algoritmus EAS je lepší než $\mathcal{MM}$AS, je při porovnání délky běhů algoritmů (graf 7.3a). Tento jev však není důsledkem toho, že by algoritmus EAS byl schopen provádět prohledávání rychleji. S nižší průměrnou hodnotou fitness jsou častěji generována řešení, která mohou být při simulaci CA předčasně ukončena, viz podkapitola 5.3. Simulace CA je v návrhu přechodových pravidel pro CA ta nejnáročnější výpočetní část a tak množství předčasně ukončených simulací dokáže výrazně ovlivnit rychlost běhu, zde v prospěch EAS. Avšak i u $\mathcal{MM}$AS v grafu 7.3a můžeme vidět mnoho krátkých běhů, které jsou v grafu ukázány jako minimální extrémy. Ty jsou zde většinou způsobeny tím, že algoritmus byl ukončen po nalezení řešení.



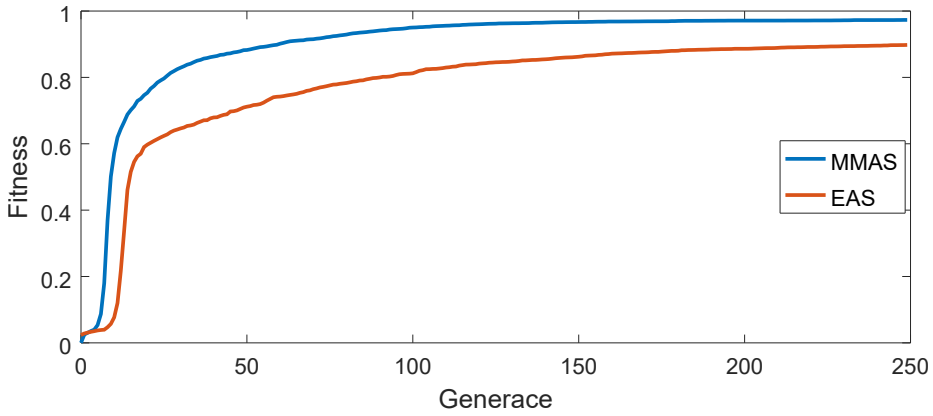
(a) Porovnání délky běhů



(b) Porovnání výsledných fitness hodnot

Algoritmus	$\mathcal{MM}$AS	EAS
Počet úspěšných běhů	19 %	0 %
Výsledná fitness – medián	0,974	0,913
Doba výpočtu – medián (hod)	9,37	7,78
Chybné výsledné buňky – medián	3,5	11

(c) Porovnání výsledků běhů algoritmů



(d) Průměrný vývoj fitness, generace $\mathcal{MM}$AS jsou škálovány vůči EAS

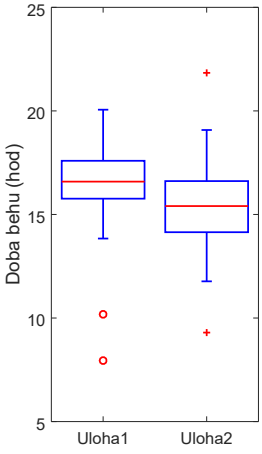
Obrázek 7.3: Porovnání algoritmů $\mathcal{MM}$AS a EAS pro zadaný rozsah hodnot 4 – 9

$\mathcal{MM}$AS se ovšem i přes poměrně nízký počet úspěšných řešení velmi blížil s nevyhovujícími výsledky k požadovanému chování. Medián počtu buněk o který neodpovídalo řešení vůči hledanému výsledku byl 3,5. To znamená, že průměrný automat navrhl výpočet pro

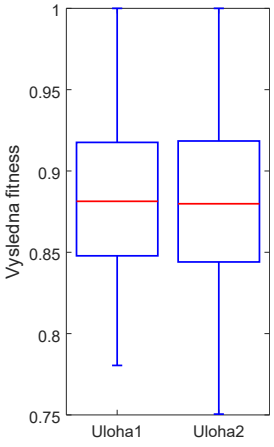
2 až 3 hodnoty v rozsahu správně a pro zbylé vstupní hodnoty se výsledek odchyloval jen o 1 buňku. V porovnání EAS mělo průměrnou odchylku zhruba 2 buňky pro každou vstupní hodnotu.

7.2 Aplikace $\mathcal{MM}$AS na větší rozmezí hodnot

V tomto experimentu jsme aplikovali $\mathcal{MM}$AS na 1. a 2. variantu úlohy výpočtu druhé mocniny v CA, kde pro obě varianty byl zadán rozsah hodnot 4 – 12. Běhy byly zde zkráceny na 15000 generací, z důvodu kompenzace výpočetní náročnosti, zbylé nastavení parametrů odpovídá tabulce 7.2. Pro obě zadání byl $\mathcal{MM}$AS schopen najít alespoň jedno vyhovující řešení z 96 testovacích běhů. Co lze zde pozorovat, tak $\mathcal{MM}$AS není ovlivněn způsobem, jakým je tato úloha zadána. Na obrázku 7.4 lze vidět, že tento algoritmus má pro obě úlohy velmi podobné výsledky. Jediné místo kde lze vidět rozdíl, je u doby výpočtu, ale to opět by mohlo být způsobeno o něco pomalejším růstem fitness pro úlohu 2, což můžeme pozorovat v grafu 7.4d.



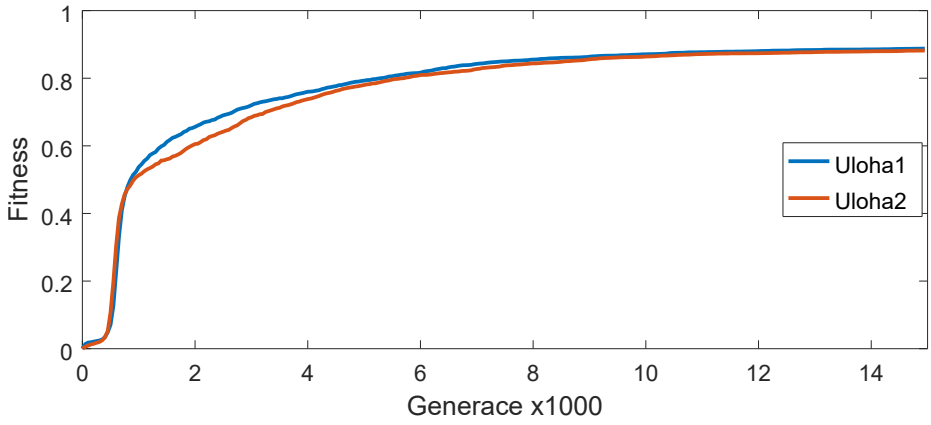
(a) Porovnání délky běhů



(b) Porovnání výsledných fitness hodnot

Varianta úlohy	**1.**	**2.**
Počet úspěšných běhů	2 %	1 %
Výsledná fitness – medián	0,881	0,880
Doba výpočtu – medián (hod)	16,6	15,4
Chybné výsledné buňky – medián	38	39

(c) Porovnání výsledků běhů algoritmu $\mathcal{MM}$AS

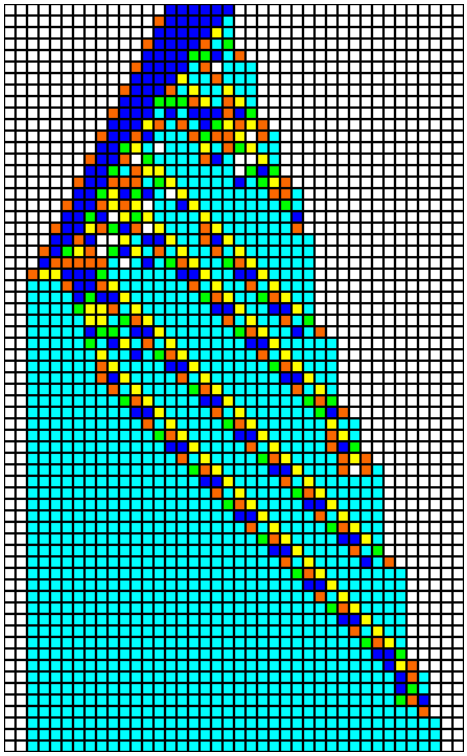


(d) Průměrný vývoj fitness algoritmu $\mathcal{MM}$AS

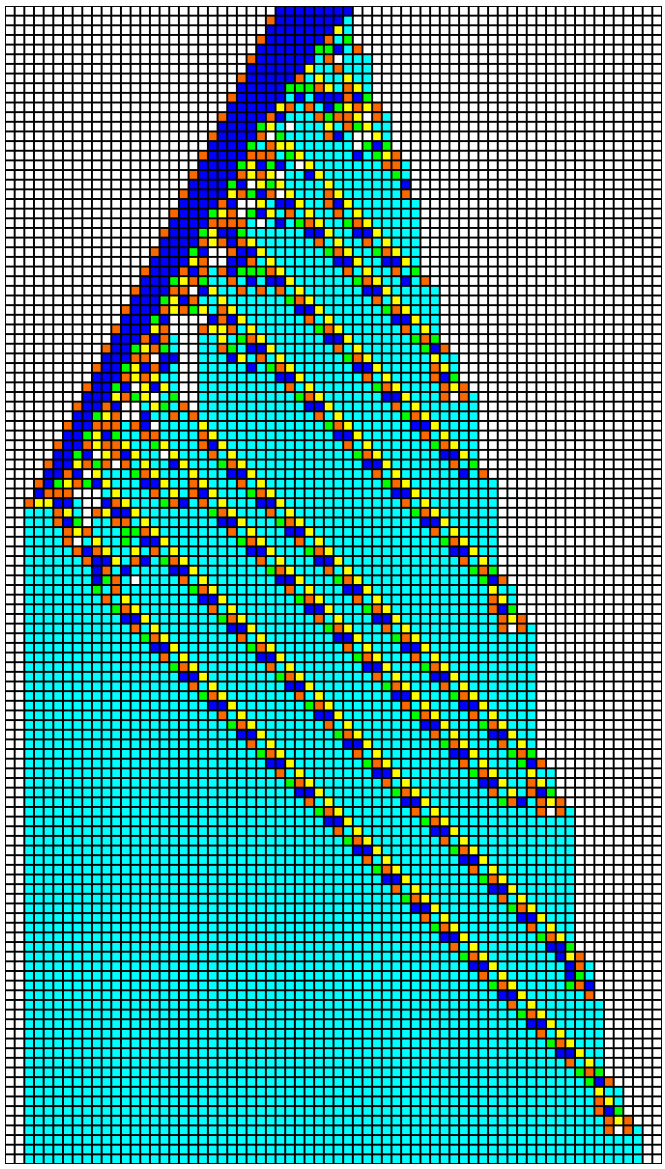
Obrázek 7.4: Porovnání algoritmů $\mathcal{MM}$AS a EAS na výpočtu druhé mocniny čísla 4 – 12

V obrázcích 7.5 a 7.6 můžeme sledovat chování přechodových funkcí na vybraných hodnotách x z rozsahu 4– 12. U obrázku 7.5 řešící první variantu úlohy by se mohlo zdát, že tento automat má pravidelné chování, které se postupně rozšiřuje podle velikosti vstupní hodnoty. Ovšem při bližším pozorování vzniku šikmých „čar“ lze vidět, že nevznikají pravidelně a některé se navzájem ovlivňují. To také způsobuje, že tato přechodová funkce není schopna řešit hodnoty mimo původní hledaný rozsah.

Chaotičnost je podstatně viditelnější v přechodové funkci řešící 2. variantu úlohy na obrázku 7.6b, kdy v druhé polovině simulace vidíme chaotický šum, který se objevuje ještě výrazněji ve vyšších hodnotách x. Přičemž zde opět platí, že tato přechodová funkce nedokáže řešit hodnoty, mimo vstupní rozsah $4 - 12$.

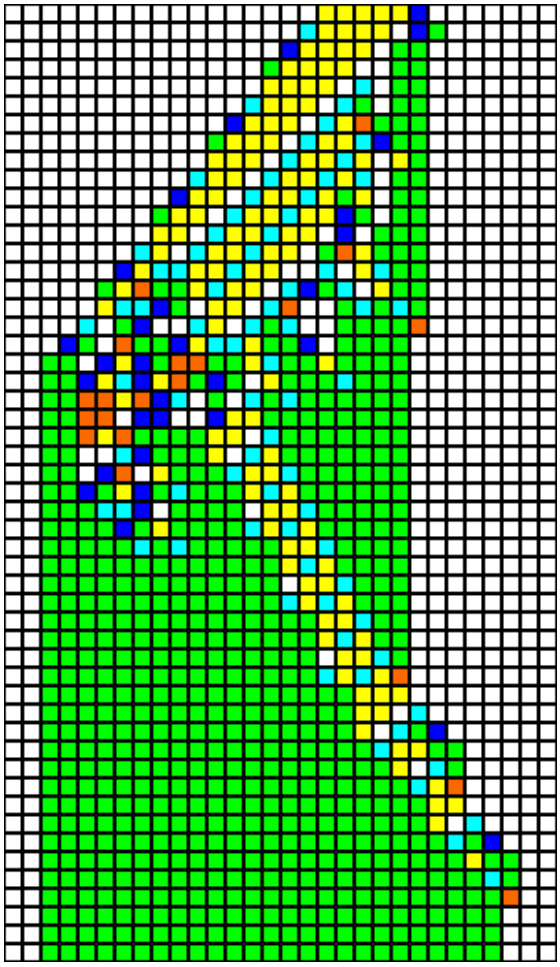


(a) Simulace pro $x = 6$

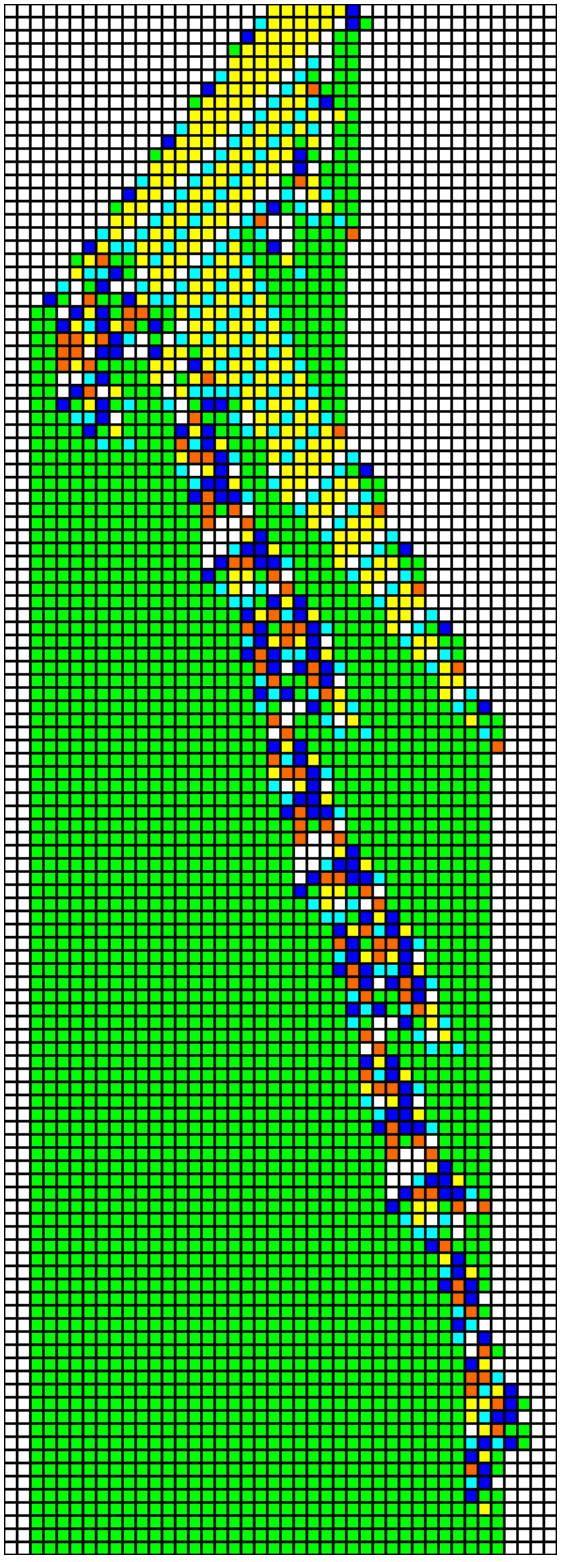


(b) Simulace pro $x = 8$

Obrázek 7.5: Ukázka vývoje přechodové funkce CA pro 1. variantu úlohy. Pravidla přechodové funkce jsou umístěny v příloze.



(a) Simulace pro $x = 5$



(b) Simulace CA $x = 6$

Obrázek 7.6: Ukázka vývoje přechodové funkce CA pro 2. variantu úlohy. Pravidla přechodové funkce jsou umístěny v příloze.

7.3 Srovnání $\mathcal{MM}$AS s výsledky návrhu za pomocí evolučních algoritmů

Jako poslední experiment bylo zvoleno srovnání zde navrženého algoritmu $\mathcal{MM}$AS s již funkčním řešením, který využívá k návrhu přechodové funkce evoluční algoritmy. Významný rozdíl o proti algoritmu $\mathcal{MM}$AS je, že vygenerované přechodové funkce jsou zde vytvářeny ve formátu CMR, který je blíže popsán v článku *Evolution of Generic Square Calculations in Cellular Automata* [1]. Tento článek byl zaměřen na návrh přechodové funkce, která je schopna řešit výpočet druhé mocniny v CA pro libovolné zadané číslo, přičemž učení probíhá na vstupních hodnotách 2 – 6.

Ve výše zmíněném článku pro hledání přechodové funkce za pomocí evolučního algoritmu byla použita populace o velikosti 10 a bylo generováno 2 milióny generací. To počtem simulací CA zhruba odpovídá výpočetní náročnosti $\mathcal{MM}$AS, při použití velikosti populace 750 a počtu generací 20 tisíc. Dále bylo potřeba upravit parametry určující šířku simulovaného CA a počtu kroků které tento automat provede. Pro výpočet druhé mocniny z čísla 2 zde používané automatické škálování nefunguje dobře pro nízké hodnoty těchto parametrů. Proto obě hodnoty byly zvýšeny na 6. Zbytek nastavení parametrů algoritmu $\mathcal{MM}$AS je shodný s nastavením v tabulce 7.2.

Zde se ukázalo, že algoritmus $\mathcal{MM}$AS není schopen vytvořit obecné řešení na základě úlohy, kde je pevně zadaný rozsah hodnot. Ani v jenom běhu nebyla nalezena přechodová funkce, která by byla schopna vypočítat druhou mocninu pro libovolnou zadanou hodnotu. To může být způsobeno tím, že algoritmus $\mathcal{MM}$AS používá úplnou tabulkovou funkci pro definici přechodové funkce. Další možností může být nevhodně zvolená fitness funkce s ohledem na získání pravidel pro výpočet druhé mocniny pro libovolné číslo.

Kde však $\mathcal{MM}$AS exceloval, tak byl počet úspěšných řešení, které ve srovnání s EA byl schopen nalézt pro zadanou v rozsahu 2 – 6, při všech variantách počtů stavů (tabulka 7.3). V porovnání délky výpočtu potřeboval $\mathcal{MM}$AS menší počet simulací v úlohách, s menším počtem stavů, při instancích s větším počtem stavů byl však evoluční algoritmus rychlejší, viz tabulka 7.4.

Počet stavů	**4**	**6**	**8**	**10**
EA (CMR) [1]	3 %	30 %	45 %	35 %
MMAS	69 %	97 %	86 %	51 %

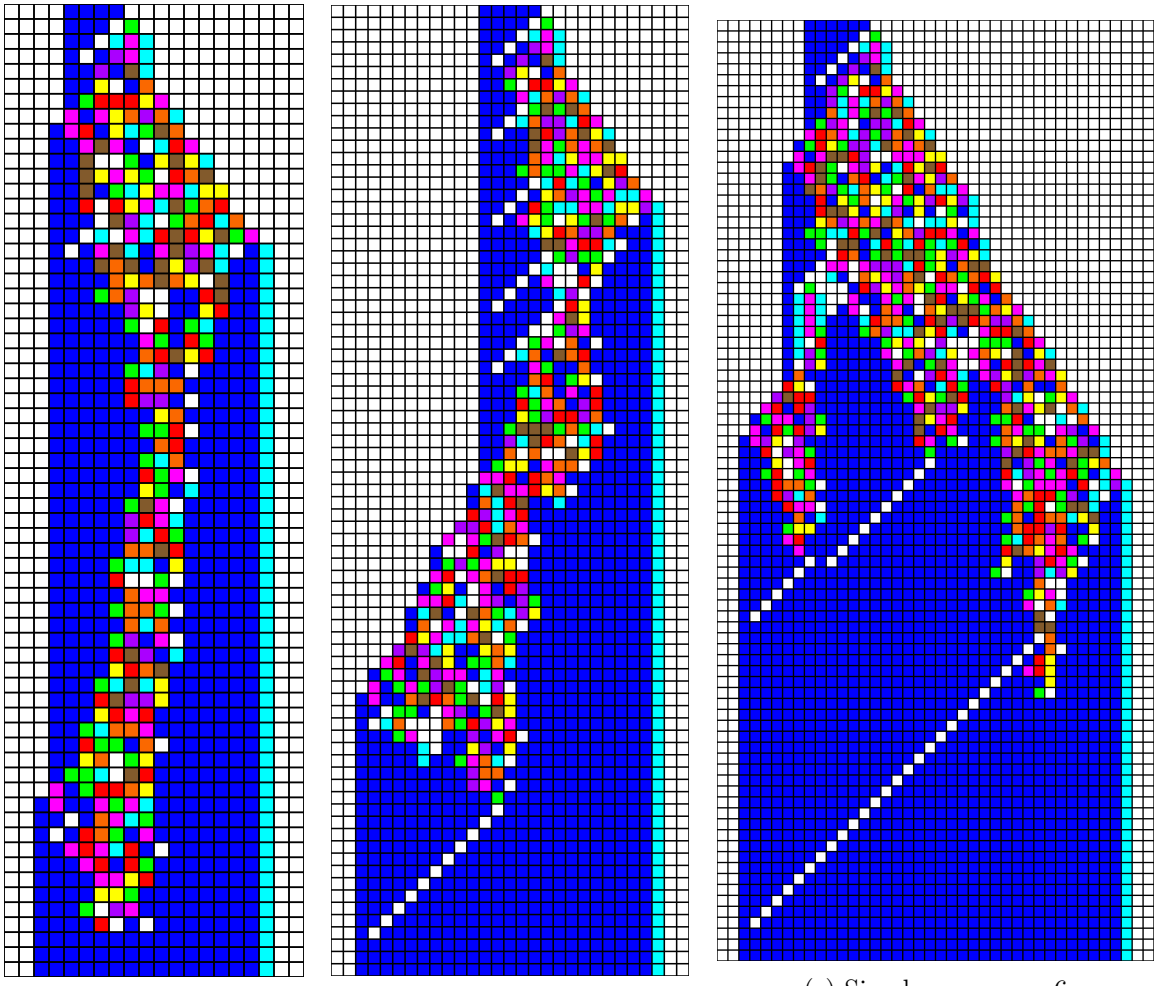
Tabulka 7.3: Porovnání počtu úspěšných běhů pro zadané počty stavů

Počet stavů	**4**	**6**	**8**	**10**
EA (CMR) [1]	844364	769440	570939	328210
MMAS	8402	5913	9643	16375
Poměr simulací EA/MMAS	1,34	1,74	0,79	0,27

Tabulka 7.4: Porovnání průměrného počtu generací pro zadané počty stavů. Velikost populace $\mathcal{MM}$AS je 750 a u EA je 10.

V obrázku 7.7 můžeme vidět jeden důvodů, proč je zde navržený algoritmus $\mathcal{MM}$AS neúspěšný v hledání obecných řešení. Lze zde pozorovat vysokou chaotičnost způsobu, jakým se navržená přechodová funkce dostává k řešení pro výpočet druhé mocniny ze zadaného čísla. Ta je v při zadání počtu stavů $s = 10$ umocněna širokým prohledávacím prostorem.

To je potvrzeno i na úlohách, kde byla hledána přechodová funkce pro větší rozsah čísel v CA s menším počtem stavů (obrázek 7.6). Tato chaotičnost byla pozorována ve většině nalezených řešení během experimentů.



(a) Simulace pro $x = 4$ (b) Simulace pro $x = 5$ (c) Simulace pro $x = 6$

Obrázek 7.7: Ukázka vývoje přechodové funkce CA, kde počet stavů $s = 10$.

Kapitola 8

Závěr

Autor v této diplomové práci aplikoval mravenčí algoritmy na úlohu návrhu přechodové funkce pro celulární automaty (CA), která doposud mravenčími algoritmy řešena nebyla. Aby bylo možné tuto úlohu řešit, bylo potřeba navrhnout konstrukční graf a následně tento graf implementovat do vybraných mravenčích algoritmů.

Autorem navržený graf, který byl schopen generovat přechodové funkce pro zadané úlohy, byl vytvořen s ohledem na generování úplné přechodové tabulky. Množina uzlů v grafu obsahuje všechna pravidla, která lze dle zadaných parametrů CA vytvořit. Množina hran je zde navržena tak, aby při každém kroku bylo vybíráno jen z takových uzlů, které obsahují stejné vstupní hodnoty pro jejich pravidla. Průchodem tímto grafem se postupně vytváří úplná tabulková přechodová funkce CA. Feromony jsou v tomto návrhu uloženy na uzlech, aby výběr výstupu dalšího pravidla nebyl ovlivněn výstupem předchozího. Tento graf byl následně implementován do algoritmů *elitist ant system* (EAS) a $\mathcal{MAX}$–$\mathcal{MIN}$ *ant system* ($\mathcal{MM}$AS), se kterými byly prováděny experimenty.

Tyto experimenty probíhaly na úloze výpočtu druhé mocniny v CA. Na jednoduchých úlohách, kde se navrhovala přechodová funkce pro výpočet mocniny z konkrétního čísla, byly schopny oba algoritmy navrhnout funkční přechodovou funkci. Avšak EAS potřeboval na vytvoření těchto pravidel v průměru 3x více času než $\mathcal{MM}$AS.

U složitějších úloh, kde navrhujeme přechodovou funkci pro výpočet druhé mocniny z čísel v předem daném rozmezí, byly rozdíly mezi těmito algoritmy značné. Pro rozmezí 4–9 EAS nebyl schopen najít žádné řešení, které by správně provedlo výpočet pro všechna zadaná čísla. To bylo způsobeno lokálním uváznutím v důsledku příliš velké hodnoty feromonů na některých uzlech. Oproti tomu $\mathcal{MM}$AS byl schopen toto řešení nalézt v 19% případů.

Dále se ukázalo, že pro tuhle řešenou úlohu je algoritmus $\mathcal{MM}$AS schopen se vypořádat s různými typy zakódování vstupu, kde tento algoritmus měl stejnou úspěšnost i při odlišných stylech kódování vstupu. Při dvou rozdílných testovaných vstupech pro úlohu výpočtu druhé mocniny byl algoritmus schopen navrhnout přechodovou funkci, která řeší výpočet mocniny pro rozmezí čísel 4–12. V obou variantách měl algoritmus $\mathcal{MM}$AS podobnou výkonnost i úspěšnost.

Další zajímavé poznatky vyšly při srovnání s článkem [1]. V tomto článku byla nalezena přechodová funkce řešící výpočet druhé mocniny pro libovolné číslo. Při využití stejného zadání, jaké je zmíněno ve výše uvedeném článku, se zde navrženému algoritmu $\mathcal{MM}$AS nepodařilo takové zobecněné řešení najít. Avšak ve srovnání s výsledky nacházející se v článku [1] na úlohách, které zde byly řešeny, byl algoritmus $\mathcal{MM}$AS podstatně úspěšnější v četnosti nalezení řešení pro zadanou úlohu a to při využití méně generací.

Další práce by mohla být zaměřena na zjištění, proč není zde navržený algoritmus schopen generovat obecná řešení pro výpočet druhé mocniny, jestli je to například způsobeno návrhem konstrukčního grafu nebo způsobem výpočtu kvality řešení získaného z cesty mravence. Další možnost rozvoje této diplomové práce by mohla být implementace zde navrženého algoritmu na jiné úlohy, zabývající se návrhem přechodové funkce pro CA, jako je třeba generování obrázků v 2D CA.

Literatura

[1] Bidlo, M.: Evolution of Generic Square Calculations in Cellular Automata. In *Proceedings of the 8th International Joint Conference on Computational Intelligence - Volume 3: ECTA*, SciTePress - Science and Technology Publications, 2016, ISBN 978-989-758-201-1, s. 94–102.

[2] Blum, C.; Sampels, M.: An Ant Colony Optimization Algorithm for Shop Scheduling Problems. *Journal of Mathematical Modelling and Algorithms*, ročník 3, č. 3, Sep 2004: s. 285–308.

[3] Bullnheimer, B.; Hartl, R. F.; Strauss, C.: A new rank based version of the Ant System. A computational study. *Central European Journal for Operations Research and Economics*, ročník 7, 1997: s. 25–38.

[4] Den Besten, M.; Stützle, T.; Dorigo, M.: Ant colony optimization for the total weighted tardiness problem. In *Parallel Problem Solving from Nature PPSN VI*, Springer, 2000, s. 611–620.

[5] Dorigo, M.: *Ant colony optimization*. Cambridge, Mass: MIT Press, 2004, ISBN 0-262-04219-3.

[6] Dorigo, M.; Gambardella, L. M.: Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, ročník 1, č. 1, 1997: s. 53–66.

[7] Dorigo, M.; Maniezzo, V.; Colorni, A.: Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, ročník 26, č. 1, 1996: s. 29–41.

[8] Encinas, A. H.; Encinas, L. H.; White, S. H.; aj.: Simulation of forest fire fronts using cellular automata. *Advances in Engineering Software*, ročník 38, č. 6, 2007: s. 372–378.

[9] Fenet, S.; Solnon, C.: Searching for maximum cliques with ant colony optimization. In *Workshops on Applications of Evolutionary Computation*, Springer, 2003, s. 236–245.

[10] Gambardella, L. M.; Dorigo, M.: An ant colony system hybridized with a new local search for the sequential ordering problem. *INFORMS Journal on Computing*, ročník 12, č. 3, 2000: s. 237–255.

[11] Land, M.; Belew, R. K.: No perfect two-state cellular automata for density classification exists. *Physical review letters*, ročník 74, č. 25, 1995: str. 5148.

[12] Levine, J.; Ducatelle, F.: Ant colony optimization and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society*, ročník 55, č. 7, 2004: s. 705–716.

[13] Mullen, R. J.; Monekosso, D.; Barman, S.; aj.: Artificial ants to extract leaf outlines and primary venation patterns. In *International Conference on Ant Colony Optimization and Swarm Intelligence*, Springer, 2008, s. 251–258.

[14] Rajendran, C.; Ziegler, H.: Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research*, ročník 155, č. 2, 2004: s. 426–438.

[15] Reimann, M.; Doerner, K.; Hartl, R. F.: D-ants: Savings based ants divide and conquer the vehicle routing problem. *Computers & Operations Research*, ročník 31, č. 4, 2004: s. 563–591.

[16] Salari, E.; Eshghi, K.: An ACO algorithm for the graph coloring problem. *Int. J. Contemp. Math. Sciences*, ročník 3, č. 6, 2008: s. 293–304.

[17] Schiff, J.: *Cellular Automata: A Discrete View of the World*. Hoboken, N.J: Wiley-Interscience, 2008, ISBN 978-0-470-16879-0.

[18] Sipper, M.: *Evolution of parallel cellular machines : the cellular programming approach*. Berlin New York: Springer, 1997, ISBN 9783540683407.

[19] Socha, K.; Sampels, M.; Manfrin, M.: Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. *Applications of evolutionary computing*, 2003: s. 334–345.

[20] Stützle, T.; Hoos, H. H.: MAX–MIN ant system. *Future generation computer systems*, ročník 16, č. 8, 2000: s. 889–914.

[21] Wolfram, S.: *A new kind of science*. Wolfram media Champaign, 2002, ISBN 1579550088.

[22] Xing, L.-N.; Chen, Y.-W.; Wang, P.; aj.: A knowledge-based ant colony optimization for flexible job shop scheduling problems. *Applied Soft Computing*, ročník 10, č. 3, 2010: s. 888–896.

Příloha A

Přechodové funkce použitých ukázek

Přechodová funkce je zde zaznamenána jako pořadí výstupních stavu pro jednotlivý pravidla. Pro automat o počtu stavů s je pořadí vstupních kombinací hodnot následovný:

000 001 ... 00s 010 011 ... 0ss 100 101 ... 111.

Vstup	000	001	010	011	100	101	110	111
Výstup	0	0	1	1	1	0	1	1

Tabulka A.1: Ukázková tabulka přechodových pravidel

Pro tabulku A.1 by byly výstupní stavy zaznamenány jako „0 0 1 1 1 0 1 1“.

Přechodová funkce obrázku 7.5

Počet stavů $s = 6$, velikost okolí $r = 1$.
Výstupní stavy:

0 4 5 0 3 3 5 4 0 2 0 0 3 2 5 2 0 5 4 3 1 1 2 5 4 1 1 5 1 2 2 1 1 3 0 4 0 1 1 5 0 5 1 1 1 1
3 1 0 4 2 5 0 5 0 1 2 1 2 3 1 1 5 1 5 3 1 1 3 5 0 1 4 4 2 1 3 5 5 4 3 3 3 1 3 1 2 2 5 5 4 0 2 5
4 5 1 5 1 5 1 1 2 5 3 4 5 2 1 5 0 0 1 2 2 1 1 2 1 0 1 3 4 0 5 1 5 4 5 0 3 3 2 4 3 0 2 0 3 2 2 2
3 1 1 1 3 2 5 0 1 4 2 0 2 4 4 2 1 2 2 1 1 5 5 0 5 4 0 4 5 0 4 0 0 0 4 0 2 2 1 4 1 3 5 2 1 1 1 5
3 3 2 0 0 2 4 1 3 1 5 5 5 1 5 2 3 2 4 4 4 5 1 0 2 1

Přechodová funkce obrázku 7.6

Počet stavů $s = 6$, velikost okolí $r = 1$.
Výstupní stavy:

0 3 0 2 3 1 3 5 4 3 1 4 2 5 2 0 4 0 2 1 2 5 5 5 5 0 1 0 1 2 3 2 5 2 4 3 0 4 1 5 4 5 3 5 3 1
0 2 0 0 1 0 5 2 3 1 1 0 3 1 5 5 4 0 1 1 2 4 5 1 5 5 0 4 2 2 2 1 3 0 2 0 2 2 2 2 2 0 2 2 5 4 3 3
4 0 2 0 2 5 3 2 2 3 2 4 0 4 2 1 1 5 0 0 2 0 3 2 1 4 1 4 1 1 0 5 2 1 3 5 4 5 4 0 2 2 5 5 1 0 2 3
4 0 5 4 4 1 5 3 5 5 1 0 3 5 4 4 5 3 0 2 1 2 5 3 0 4 1 1 5 1 4 1 5 2 4 5 1 3 2 0 0 1 3 4 5 4 4 3
2 1 2 3 4 2 2 0 4 5 5 0 1 5 4 2 1 1 4 5 5 1 0 1 3 3