VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

APLIKACE MRAVENČÍCH ALGORITMŮ V ROZSÁHLÝCH ÚLOHÁCH TSP

ANT COLONY OPTIMIZATION FOR SOLVING BIG INSTANCES OF TSP

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PATRÍCIA RAMOSOVÁ**
AUTHOR

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

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2019/2020

Zadání bakalářské práce

Studentka: **Ramosová Patrícia**
Program: Informační technologie
Název: **Aplikace mravenčích algoritmů v rozsáhlých úlohách TSP**
Ant Colony Optimization for Solving Big Instances of TSP
Kategorie: Umělá inteligence

Zadání:

1. Nastudujte problematiku mravenčích algoritmů (Ant Colony Optimization - ACO) a jejich použití pro řešení úloh obchodního cestujícího (Traveling Salesman Problem - TSP).
2. Seznamte se s rozšířeními ACO, umožňujícími efektivně řešit rozsáhlé instance TSP.
3. Navrhněte a implementujte systém, využívající vybraná rozšíření ACO, pro optimalizaci tras TSP.
4. Proveďte sadu experimentů pro různé konfigurace systému.
5. Zhodnoťte dosažené výsledky a diskutujte možnosti pokračování projektu.

Literatura:

- M. Dorigo, T. Stützle: Ant Colony optimization. The MIT Press, Cambridge, MA, USA, 2004
- J. Peake, M. Amos, P. Yiapanis, H. Lloyd: Scaling techniques for parallel ant colony optimization on large problem instances. In: *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO '19), ACM, New York, NY, USA, p. 47-54

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 a 2 zadání, demonstrace prototypu aplikace z bodu 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Bidlo Michal, Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 25. října 2019

Abstrakt

V súčasnej dobe je v rade aplikácií kladený dôraz na nájdenie optimálneho riešenia určitého problému. Pre niektoré úlohy je však typické, že sa ich náročnosť stupňuje exponenciálne v závislosti na veľkosti inštancie. Typickým príkladom takéhoto problému je obchodný cestujúci (angl. Traveling Salesman Problem – TSP). Jednou triedou metód, ktoré sa ukázali byť v riešení TSP veľmi nápomocné, sú mravčie algoritmy. Narazili však na svoj limit – vysoký počet miest v inštancii, kedy sa už stali kvôli časovej a pamäťovej náročnosti takmer nepoužiteľné. Cieľom tejto práce je modifikovať mravčí algoritmus a vytvoriť tak systém schopný rýchlo a efektívne riešiť rozsiahle úlohy TSP bez výraznej straty na kvalite nájdeného riešenia. Optimalizácie budú zamerané na redukciu priestorovej zložitosti a celkovému zníženiu výpočtového času.

Abstract

Currently, many applications place emphasis on finding the optimal solution to a particular problem. However, it is typical for some tasks that their complexity increases exponentially depending on the size of the instance. A typical example of such a problem is the Traveling Salesman Problem (TSP). One class of methods that have proven to be very helpful in solving TSPs are ant algorithms. Nonetheless, they reached their limit – a high number of cities in the instance and became almost unusable due to time and memory requirements. This bachelor thesis aims to modify the ant algorithm and create a system capable of quickly and efficiently solve large-scale TSPs without significant loss in the quality of the solution found. Optimization will focus on reducing memory complexity and total execution time.

Kľúčové slová

optimalizácia mravčou kolóniou, problém obchodného cestujúceho, rozsiahle inštancie TSP, MAX-MIN Ant System

Keywords

Ant Colony Optimization, Travelling Salesman Problem, large-scale TSP instances, MAX-MIN Ant System

Citácia

RAMOSOVÁ, Patrícia. *Aplikace mravenčích algoritmů v rozsáhlých úlohách TSP*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Bidlo, Ph.D.

Aplikace mravenčích algoritmů v rozsáhlých úlohách TSP

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracovala samostatne pod vedením pána Ing. Michala Bidla, Ph.D. Uviedla som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpala.

.....
Patrícia Ramosová
11. júna 2020

Poďakovanie

Rada by som poďakovala svojmu vedúcemu, pánovi Ing. Michalovi Bidlovi, Ph.D. za jeho cenné rady a podporu. Táto práca bola podporená Ministerstvom školstva, mládeže a telovýchovy z podpory Veľkých infraštruktúr pre výskum, experimentálny vývoj a inovácie v rámci projektu "IT4Innovations národní superpočítačové centrum - LM2015070".

Obsah

Kapitola 1

Úvod

V súčasnej dobe je v rade aplikácií kladený dôraz na nájdenie optimálneho riešenia určitého problému. Inžinierske postupy poznajú efektívne metódy, ako riešiť rôzne (aj náročné) problémy. Pre niektoré úlohy je však typické, že sa ich náročnosť stupňuje exponenciálne v závislosti na velkosti inštancie. Typickým príkladom takéhoto problému je obchodný cestujúci (angl. Traveling Salesman Problem – TSP). Cielom jeho riešenia je nájsť optimálnu (tj. najkratšiu) cestu obchodného cestujúceho medzi množinou miest, pričom je dovolené navštíviť každé mesto práve jedenkrát. Problémom však je, že s narastajúcim počtom miest sa náročnosť nájdeného riešenia enormne zvyšuje a od istej hranice je nájdenie optimálnej cesty prakticky nemožné. Podobné rysy sú v informatike typické tzv. NP-úplnym problémom, medzi ktoré patrí taktiež TSP. V súčasnej dobe je jedinou možnosťou, ako nájsť rišenie takého problému v rozumnom čase, použitie rôznych heuristických postupov.

Jednou triedou metód, ktoré sa ukázali byť v riešení TSP velmi nápomocné, sú napríklad mravčie algoritmy. Dorigo poprvé predstavil mravčie algoritmy vo svojej dizertačnej práci, v ktorej ukázal možnosť riešenia TSP pomocou pricípov vysledovaných z chovania mravcov v prírode [5]. Ich prvý algoritmus, nazvaný Ant System, bol neskôr rozšírený a doplnený o ďalšie prvky (súhrnne napr. v [8]).

Pre riešenie TSP s veľkým počtom miest je treba zaviesť vhodné optimalizácie, aby sa znížila pamäťová a časová náročnosť výpočtu. Napríklad, pre nájdenie optimálneho riešenia inštancie TSP s 10 000 mestami je potreba približne 380 MB operačnej pamäte, s čím by si väčšina moderného hardvéru vedela poradiť. Avšak, pri inštancii TSP so 100 000 mestami je potrebnej približne až 37 GB operačnej pamäte, čo už možno považovať za menej praktické. Príkladom zníženia pamäťovej náročnosti je redukcia veľkosti miesta potrebného na ukladanie heuristických informácií potrebných k výpočtu.

Hlavným cieľom tejto práce je vytvoriť systém schopný rýchlo a efektívne riešiť rozsiahle úlohy TSP bez výraznej straty na kvalite nájdeného riešenia. V prvej časti budú implementované optimalizácie z článku *Scaling Techniques for Parallel Ant Colony Optimization on Large Problem Instances* [16] pre MAX-MIN Ant System. Ako základ pre implementáciu optimalizácií bude použitá pôvodná implementácia Ant System od pána Doriga a spol.[1], ktorá v základe nie je vhodná pre veľké inštancie obsahujúce niekoľko desiatok až stoviek tisíc miest, najmä pre príliš veľké nároky na pamäťový priestor a dlhý čas jednej iterácie. Optimalizácie budú zamerané na redukciu priestorovej zložitosti metódou takzvanej obmedzenej feromónovej matice. Zrýchlenie výpočtu trasy mravca bude realizované využitím špe-

[1] Dostupné na http://www.aco-metaheuristic.org/aco-code/public-software.html

ciálnych inštrukcií procesora, konkrétne *IMCI* inštrukcií. Celkovému zníženiu výpočtového času pomôže aj paralelizácia, ktorá je už z podstaty algoritmu dobre implementovateľná.

Štatistické dáta o výpočtoch a riešení budú získané pomocou sady experimentov vykonaných na inštanciach o veľkosti niekoľko desiatok tisíc miest. Vyhodnocovaný bude hlavne čas potrebný na jednu iteráciu algoritmu. Ďalej bude skúmaný vplyv optimalizácií na kvalitu nájdeného riešenia. Porovnávané budú pôvodné implementácie a upravená implementácia MAX-MIN Ant System obohatená o optimalizácie.

Kapitola 2

Mravčie algoritmy

V tejto kapitole sa rozoberá podstata mravčích algoritmov, inšpirácia umelých mravcov v ich živých predlohách a experiment ukazujúci schopnosť mravcov nájsť optimálne riešenie zadanej úlohy. Ďalej sú popísané tri základné varianty mravčích algoritmov – Ant System, Ant Colony System, MAX-MIN Ant System a ich ďalšie varianty.

Ant Colony Optimization (ACO) predstavujú triedu algoritmov kolektívnej výpočetnej inteligencie (angl. Computation Swarm Intelligence – CSI), ktoré vychádzajú z pozorovania a študovania chovania prírodných spoločenstiev ako sú vtáčie hejná, včelie roje a mravčie kolónie. V prípade mravčích algoritmov (ďalej „MA“) je sledované práve chovanie mravčích spoločenstiev pri hľadaní potravy (angl. tzv. *foraging* – spásanie), u ktorých je základným faktorom úspechu komunikácia, hlavne cez prostredie pomocou feromónov – chemickej stopy. Postupné vypúšťanie feromónov jednotlivými mravcami pri prehľadávaní priestoru naviguje ostatných mravcov k nájdeniu optimálnej cesty k zdroju potravy, nakoľko preferovaná cesta bude tá s vyššou koncentráciou feromónov. Feromónová stopa je úmerná k dĺžke a náročnosti cesty, no zároveň k prínosnosti konkrétneho zdroja potravy. Rovnako ako živé mravce v prírode, aj umelé mravce v optimalizačných úlohách hľadajú najoptimálnejšie riešenie – najvýhodnejšiu trasu vo váženom grafe [1].

2.1 Experiment dvoch mostov

Ako už bolo spomenuté v úvode kapitoly č. 2, mravce sa riadia nepriamou komunikáciou (tzv. *Stigmergic communication*[1]) pomocou feromónov. Postupným vypúšťaním tejto prchavej chemickej látky vytvoria takzvanú feromónovú stopu. Majú dobre vyvinutý čuch, vďaka ktorému cítia rôznu koncentráciu feromónových stôp na rôznych trasách. Čím vyššia koncentrácia, tým vyššia pravdepodobnosť, že mravec si danú trasu vyberie. Toto chovanie bolo natoľko zaujímavé, že sa viacerí vedci rozhodli podrobiť ho rôznym experimentom.

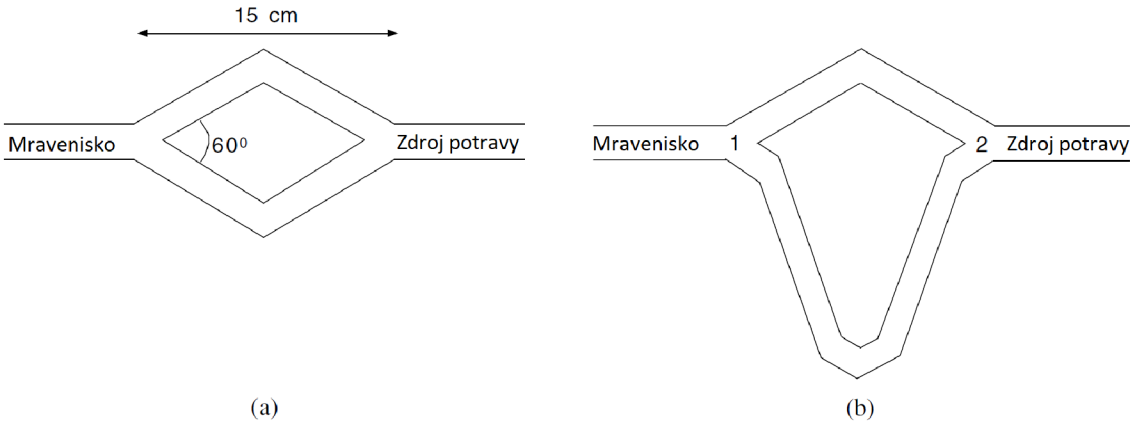
Jeden z najprínosnejších experimentov, ktorý nesie názov Experiment dvoch mostov (angl. Double Bridge Experiment) bol navrhnutý a prevedený pánom Deneubourgom a jeho kolegami [4] v roku 1989. V tomto experimente použili dvojitý most – dve cesty, ktorým preklenuli plochu medzi mraveniskom a zdrojom potravy. Predpokladalo sa, že mravci dokážu nájsť najkratšiu cestu k zdroju potravy postupným prechádzaním ciest a vypúšťaním feromónovej stopy počas pohybu.

Cieľom experimentu bolo vysledovať výsledné chovanie kólonie mravcov. Aby mohli skúmať odlišné chovanie, použili rôzny pomer dĺžky oboch ciest

[1] https://www.sciencedirect.com/science/article/pii/S1389041707000290?via%3Dihubk

$$r = \frac{l_l}{l_s} \tag{2.1}$$

kde l_l je dĺžka dlhšej cesty a l_s je dĺžka kratšej cesty.



Obr. 2.1: Experimentálne zostavenie ciest. (a) Cesty majú rovnakú dĺžku. (b) Cesty majú rozdielnu dĺžku.

V prvom experimente – obrázok 2.1a použili most s rovnako dlhými cestami, teda $r = 1$. Na začiatku sa mravce pohybovali medzi mraveniskom a zdrojom potravy svojvoľne. Vedci skúmali percentuálne rozhodnutia voľby jednotlivých ciest. Pri pozorovaní zistili, že i keď na počiatku sa mravce pohybovali náhodne, časom sa začali mravce prikláňať k jednej ceste a nakoniec sa ňou pohybovali všetky mravce.

Náhodný pohyb na začiatku experimentu bol spôsobený tým, že sa na cestách nevyskytovala feromónová stopa a teda mravce vyberali trasu náhodne – obe cesty s rovnakou pravdepodobnosťou, nakoľko si bez stopy feromónov nemohli vytvárať preferencie. Neskôr sa ale náhodným výberom začalo viac mravcov postupne prikláňať k jednej z dvoch ciest, čo spôsobilo, že feromónová stopa na jednej z nich bola koncentrovanejšia. Tento jav bol spôsobený pravdepodobnostnou fluktuáciou. Vyššia koncentrácia feromónov stimulovala mravcov, ktorý začali preferovať túto konkrétnu trasu. Nakoniec sa celá kolónia priklonila k jednej ceste.

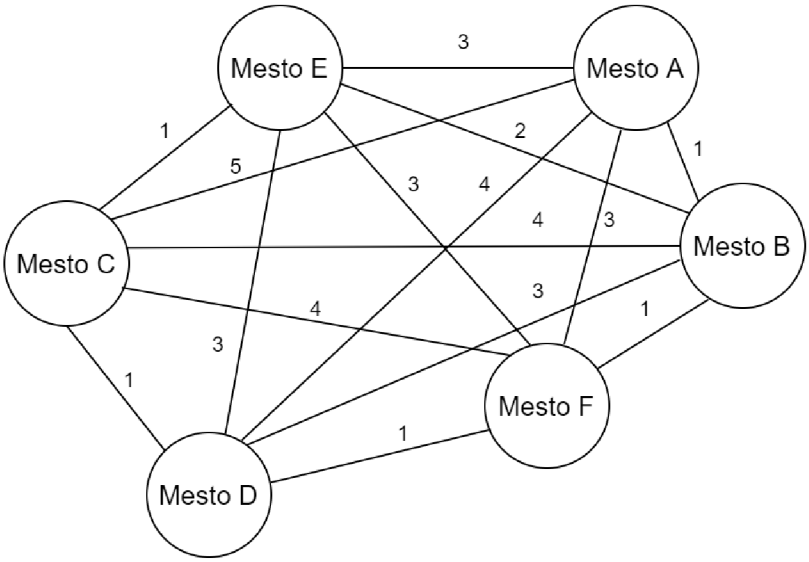
V druhom experimente – obrázok 2.1b vedci použili most s rozdielne dlhými cestami, $r = 2$, teda dlhšia cesta bola dvakrát tak dlhá, ako tá kratšia. Na počiatku sa mravce pohybovali rovnako, ako v prvom experimente – volili obe cesty náhodne s rovnakou pravdepodobnosťou, nakoľko ani jedna neobsahovala feromóny. Zmena nastala v momente, kedy mravce vyprodukovali feromónovú stopu a začali sa riadiť podľa nej. Zistilo sa, že preferovanejšia začala byť kratšia cesta – mravce, ktoré ňou prešli z mraveniska k zdroju potravy tam boli skôr a taktiež sa vrátili skôr do mraveniska. Pri rozhodovaní sa, ktorou cestou sa vrátit boli ovplyvnení vyššou koncentráciou feromónov na tej kratšej. Feromóny sa začali akumulovať a postupne viac mravcov začalo preferovať jednu cestu. Inak povedané, väčší nárast feromónov na kratšej ceste má na svedomí fakt, že za jednotku času dokáže po tejto ceste prejsť viac mravcov, ako po tej dlhej [8, 9].

2.2 Problém obchodného cestujúceho

Problém obchodného cestujúceho (angl. Traveling Salesman Problem – TSP) je NP-úplna kombinačná optimalizačná úloha, ktorá hrá dôležitú rolu vo vývoji mravčích algoritmov. MA boli pôvodne navrhnuté pre a overené práve na úlohe TSP. V roku 1996 bol navrhnutý a implementovaný prvý mravčí algoritmus riešiaci problém obchodného cestujúceho, ktorý bol pomenovaný Ant System (viac v sekcii 2.3). Neskôr bolo TSP použité na overenie a testovanie takmer všetkých navrhnutých MA. Problém obchodného cestujúceho je ľahko porozumiteľný, výborne sa na neho MA aplikujú a v neposlednom rade, ak algoritmus pracuje výkonne a efektívne s úlohou TSP, dokazuje to jeho užitočnosť a prínos [8].

V úlohe TSP sa obchodný cestujúci snaží nájsť optimálnu, v tomto kontexte najkratšiu, cestu medzi množinou miest, pričom navštíviť môže každé mesto práve jedenkrát, kým sa nakoniec vráti do počiatočného mesta, z ktorého začínal. Každé mesto je prepojené so všetkými ostatnými mestami a obchodný cestujúci sa môže pohybovať všetkými smermi.

Na základe tohto opisu môže byť TSP reprezentované ako úplny ohodnotený graf $G = (N, A)$, kde N predstavuje množinu uzlov – miest, a A predstavuje množinu hrán – ciest. Každá hrana $(i, j) \in A$ má príslušnú váhu w_{ij}, ktorá reprezentuje vzdialenosť medzi mestami $i, j \in N$. Ak sa vzdialenosť medzi akýmikoľvek dvomi mestami zmenou smeru nezmení, teda $w_{ij} = d_{ji}$, ide o symetrické TSP. Ak sa však zmenou smeru mení aj vzdialenosť (váha) medzi minimálne jednou dvojicou miest, teda $w_{ij} \neq w_{ji}$, ide o nesymetrické TSP. Táto práca sa zaoberá výhradne symetrickými TSP. Príklad úplneho symetrického grafu TSP je ukázaný na obrázku 2.2.



Obr. 2.2: Príklad úplneho grafu jednoduchej úlohy TSP so šiestimi mestami a cestami medzi nimi, ktoré sú ohodnotené váhou predstavujúcou vzdialenosť.

Cieľom úlohy je nájsť hamiltonovskú kružnicu s najnižším možným ohodnotením, kde hamiltonovská kružnica je uzatvretá prechádzka prechádzajúca každým mestom $n = |N|$ práve raz. Optimálny výsledok je teda permutácia π indexov všetkých miest $\{1, 2, ..., n\}$. Zmyslom hamiltonovskej kružnice je skutočnosť, že obchodný cestujúci sa z posledného

navštíveného mesta vracia do mesta počiatočného [8]. Dĺžka cesty $f(\pi)$ je teda daná ako

$$f(\pi) = \sum_{i=1}^{n-1} w_{\pi(i)\pi(i+1)} + w_{\pi(n)\pi(1)} \tag{2.2}$$

Testovacie inštancie TSP

Pre potreby tejto práce budú symetrické inštancie TSP čerpané zo stránky *TSP Test Data*[2] a zároveň z knižnice *TSPLIB benchmark library*[3]. V týchto dvoch zdrojoch možno nájsť inštancie od pár desiatok miest až po niekoľko stoviek tisíc miest.

Záujímavými a rozsiahlymi inštanciami TSP sú napríklad umelecké inštancie TSP ukázané na obrázku 2.3.

Obr. 2.3: Známe umelecké inštancie TSP, zľava hore `vangogh120k`, `earring200k`, `venus140k`, `mona-lisa100k` [16].

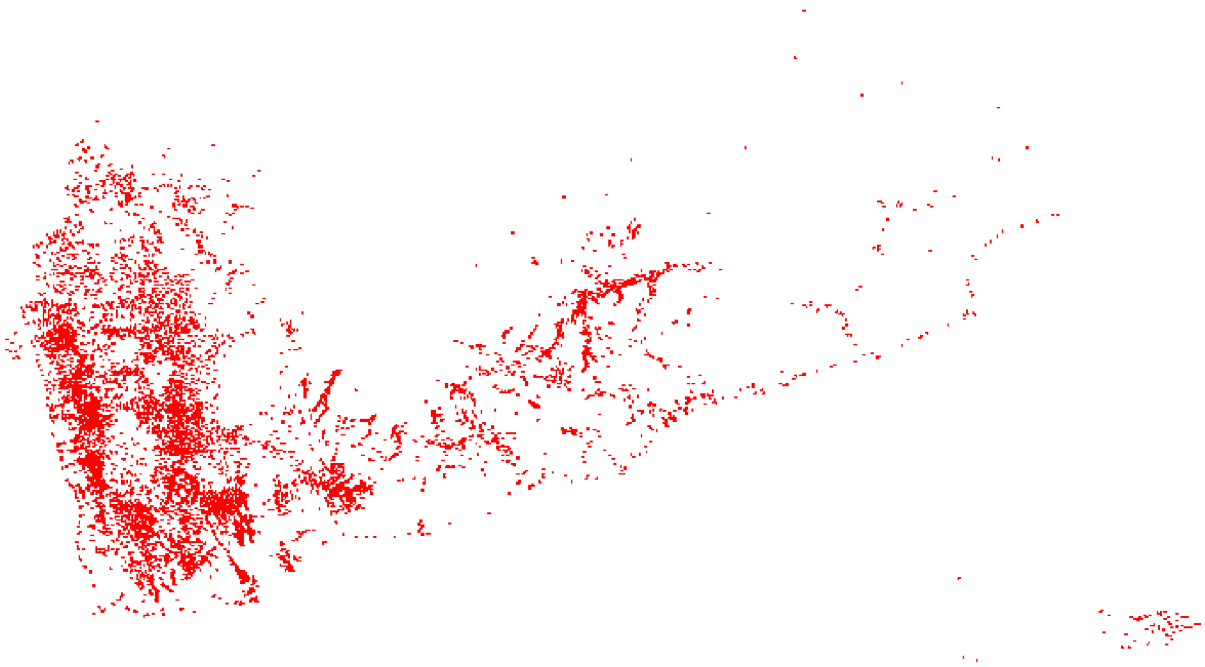
TSP súbor má pevne danú štruktúru a poradie jednotlivých informácií:

- Ako prvé sa uvádza meno inštancie.

[2] http://www.math.uwaterloo.ca/tsp/data/index.html
[3] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/

- Nasledujú voliteľné komentáre, ktoré nijako neovplyvňujú výpočet alebo spracovanie a majú iba informatívny charakter.
- Typ úlohy, v tomto prípade vždy TSP.
- Počet miest v inštancii.
- Ako predposledný sa uvádza spôsob, ktorým sa budú počítať vzdialenosti medzi mestami – napríklad euklidovská vzdialenosť v 2D priestore.
- Posledná časť obsahuje zoznam všetkých miest v inštancii vo formáte: `CISLO_MESTA SURADNICA_X SURADNICA_Y`. Súbor končí slovom EOF. Mestá sú teda zapísané ako čísla od 1 po N a majú svoje súradnice v Euklidovskom priestore.

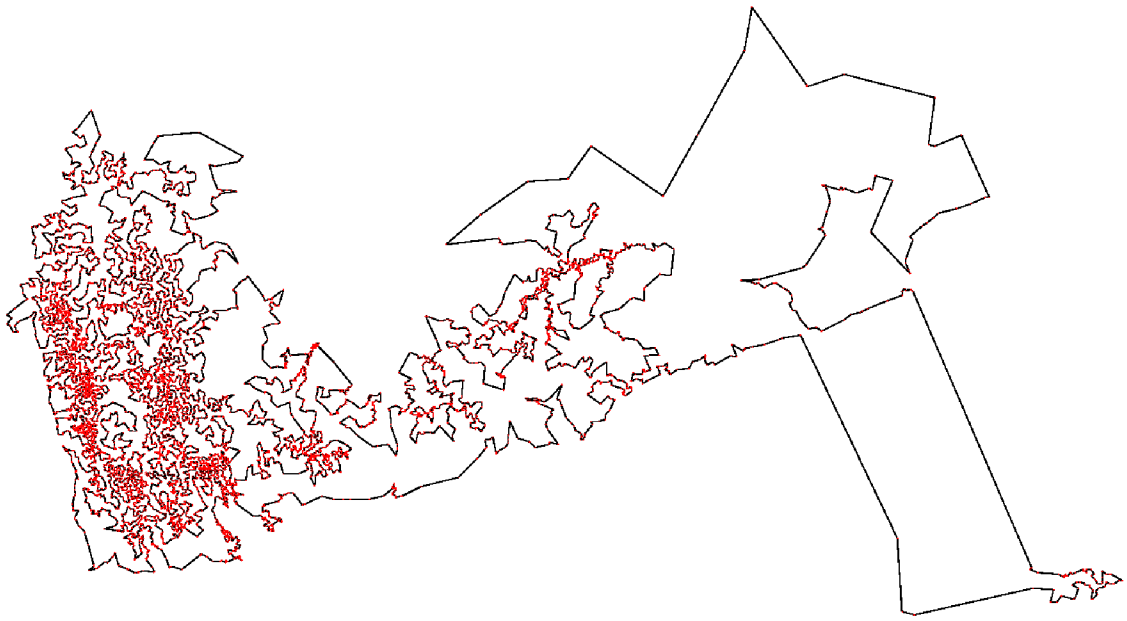
Niektoré inštancie TSP sú doprevádzané *point set* súbormi, ktoré majú formu obrázku. Ako príklad poslúži point set ázijského Jemenu ukázaný na obrázku 2.4.



Obr. 2.4: Point set TSP inštancie Jemenu s 7 663 mestami.[4]

Point set obsahuje pozície jednotlivých miest podla ich súradníc. Do point setu je možné vykresliť cestu, napríklad najlepšie známe riešenie úlohy TSP. Presnosť najlepších znýmych riešení je ovplyvnená zaokrúhľovanim na celé čísla, ale táto nepresnosť je v jednotkách stotín až tisícín percenta. V tejto práci sa však počíta s číslami s pohyblivou desatinnou čiarkou. Príklad optimálnej nájdenej cesty inštancie TSP z obrázku 2.4 možno vidieť na obrázku 2.5.

[4]Zdroj http://www.math.uwaterloo.ca/tsp/world/ympoints.html



Obr. 2.5: Ukážka nájdenej optimálnej cesty TSP inštancie štátu Jemen, ktorej dĺžka je 238 314 kilometrov s nepresnosťou 0,030% oproti exaktnému riešeniu získanému algoritmom používajúcim hodnoty s pohyblivou desatinnou čiarkou.[6]

2.3 Algoritmus Ant System (AS)

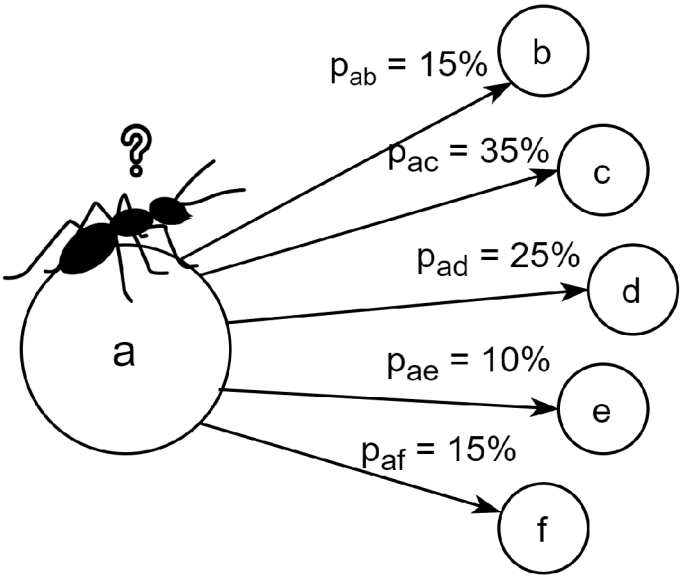
Prvý mravčí algoritmus bol navrhnutý Dorigom v roku 1991 [5] v jeho dizertačnej práci, ktorá bola publikovaná v roku 1992. Ant System (ďalej „AS“) bol pôvodne navrhnutý práve na riešenie problému obchodného cestujúceho. Spočiatku boli navrhnuté tri verzie AS – *ant-density, ant-quantity* a *ant-cycle*. Hlavným rozdielom medzi nimi bol spôsob aktualizovania feromónov. Ant-density a ant-quantity aktualizovali feromónovú stopu pri každej ceste z mesta i do mesta j, zatiaľ čo ant-cycle vykonávalo aktualizáciu až po skoštruovaní všetkých ciest mravcami. Hodnota uložených feromónov vo verzii ant-cycle bola vypočítaná funkciou na základe kvality cesty. Experimentami sa prišlo na to, že prvé dve spomenuté verzie sú výkonovo preukázateľne horšie, preto sa v súčasnosti používa iba ant-cycle.

Mravci sa v AS nazývajú agenti a komunikujú medzi sebou pomocou globálnej komunikácie a feromónov. Na začiatku výpočtu sa jednotliví agenti rozmiestnia do rôznych počiatočných miest. Takto umiestnení agenti začnú iteratívne budovať svoje riešenia prechádzaním jednotlivých miest. Pri výbere ďalšieho mesta si agenti vyberajú vždy z nimi doposiaľ nenavštívených miest. Výber konkrétneho mesta záleží na jeho vzdialenosti a intenzite feromónovej stopy. Proces výberu sa riadi podľa takzvaného *pravidla prechodu* (angl. state transition rule), ktoré sa v terminológii AS nazýva *random-proportional rule*. Udáva pravdepodobnosť s akou sa agent z mesta i presunie do mesta j. Príklad rozhodovania na základe spomenutého pravidla je ukázaný na obrázku 2.6.

Až všetci agenti dokončia svoje cesty, použije sa *globálne aktualizačné pravidlo feromónov* (angl. global pheromone updating rule), na základe ktorého sa na všetkých cestách aktualizuje hodnota feromónovej stopy – určitá časť feromónov sa vyparí v závislosti na nastavenej hodnote koeficientu vyparovania a agenti uložia nové hodnoty feromónových stôp

[6] Zdroj http://www.math.uwaterloo.ca/tsp/world/ymtour.html

na hranách, ktoré patria do ich riešenia v pomere k dĺžke ich trasy (čím kratšia cesta, tým väčší feromónový prírastok na hranách, ktoré ju tvoria). Celý proces sa opakuje, pokým nenastane ukončujúca podmienka [6]. Jednou iteráciou MA sa rozumie postupnosť niekoľkých krokov – náhodné umiestnenie mravcov do počiatočných miest, pohyb mravcov po grafe s cieľom vytvoriť kandidátne riešenie a aktualizovanie feromónových stôp.



Obr. 2.6: Názorný príklad procesu rozhodovania. Mravec si náhodne vyberá nasledujúce mesto z aktuálneho mesta a. Každý prechod má pridelenú pravdepodobnosť p, ktorej hodnota závisí na váhe prechodu. Týmto sa zabezpečí proporcionalita náhodného výberu.

Konštrukcia riešenia v AS

Uvažujme m agentov, ktorí vytvárajú čiastočné riešenie úlohy súčasne. Ako bolo spomenuté, tí sa náhodne rozmiestnia do počiatočných miest. V každom kroku pri presune z mesta i do mesta j sa agenti rozhodujú podľa pravdepodobnostného pravidla, ktoré je vyjadrené ako

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{c \in C_i^k} [\tau_{ic}]^\alpha [\eta_{ic}]^\beta}, j \in C_i^k \tag{2.3}$$

kde agent k, nachádzajúci sa v meste i, si vyberá mesto j. $\eta_{ij} = 1/w_{ij}$ predstavuje heuristickú informáciu udávajúcu viditeľnosť mesta j z mesta i, w_{ij} vyjadruje vzdialenosť medzi mestami i a j. τ_{ij} udáva intenzitu feromónovej stopy medzi mestami i a j. Konštanty α a β slúžia na určenie miery vplyvu feromónov a vzdialenosti na výber nasledujúceho mesta. C_i^k predstavuje dosiahnuteľné mestá z mesta i pre agenta k, ktoré doposiaľ nenavštívil – pravdepodobnosť navštívenia mesta nenachádzajúceho sa v množine C_i^k je 0. Čím vyššia hodnota feromónovej stopy τ_{ij} a heuristickej informácie η_{ij}, tým vyššia pravdepodobnosť, že agent si vyberie práve mesto j. Ak $\alpha = 0$, agent si s väčšou pravdepodobnosťou vyberie blízke mestá. Ak $\beta = 0$, agent sa riadi výhradne feromónovou stopou. To však vo väčšine prípadov vedie k horším výsledkom a k stagnácii pre hodnoty $\alpha > 1$. *Stagnácia* predstavuje situáciu, kedy všetci agenti nasledujú rovnakú cestu. *Predčasná stagnácia* spravidla nastane v okamihu, kedy agenti preskúmajú málo ciest a príliš rýchlo sa začnú riadiť najvyššími koncetráciami feromónov [9].

Agenti (umelí mravci) majú narozdiel od reálnych mravcov pamäť M^k, v ktorej si po každom kroku ukladajú navštívené mestá v presnom poradí, v akých ich navštívili. To im slúži na zabezpečenie rôznych aplikačno špecifických podmienok – napríklad slúži na vytvorenie množiny C_i^k použitej vo vzorci 2.3, na výpočet dĺžky vytvoreného riešenia a na spätné uloženie feromónov [1, 8, 7].

Aktualizácia feromónových stôp v AS

Po dokončení ciest všetkými agentmi prichádza na rad aktualizácia feromónov na hranách. V prvom kroku sa zníži feromónová stopa na všetkých hranách podľa vzorca

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij}, \forall (i,j) \in A, \tag{2.4}$$

kde $0 < \rho < 1$ značí mieru vyparovania feromónov. Bez evaporácie by mohli byť feromónové stopy nekonečne zvyšované a tým by sa mohol znehodnotiť výsledok. Ak agenti niektoré hrany prestanú využívať, ich feromónová stopa sa s každou iteráciou exponenciálne znižuje, čo dovoľuje algoritmu „zabudnúť“ na predošlý zlý výber hrany. Ďalším krokom aktualizácie je uloženie nových hodnôt feromónov agentami na hrany, ktoré patria do ich riešenia

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k, \forall (i,j) \in A, \tag{2.5}$$

kde m je počet agentov a $\Delta\tau_{ij}^k$ je množstvo feromonóv, ktoré pridal agent k k hrane (i,j). Toto množstvo je dané ako

$$\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{L^k}, & \text{ak hrana } (i,j) \text{ patrí do } T^k; \\ 0, & \text{inak,} \end{cases} \tag{2.6}$$

kde Q značí konštantu a L^k je dĺžka cesty agenta k, vypočítaná ako súčet dĺžok všetkých hrán tvoriacich cestu T^k. Ak je hrana často používaná viacerými agentmi, príbytok feromónov na nej bude väčší a teda v konečnom dôsledku si ju v ďalších iteráciách vyberie viac agentov [14, 8, 9].

2.4 Algoritmus Ant Colony System (ACS)

Mravčí algoritmus Ant System bol veľkým prínosom pre riešenie problému obchodného cestujúceho a ďalší vývoj mravčích algoritmov. Zatiaľ čo menšie inštancie TSP (približne do 30 miest) [6] zvládal bez problémov, rozsiahlejšie úlohy si vyžadovali enormne veľa času a výkon AS bol natoľko zlý, že dosiahnutie optimálneho výsledku v rozumnom čase bolo priam nemožné. Z toho dôvodu bolo veľa času venovaného výzkumu vylepšení, ktoré by zlepšili výkon AS pre väčšie inštancie [8].

Rozšírenie Ant Colony System (ďalej „ACS“) sa od základného AS líši v troch hlavných bodoch:

1. Využíva informácie z ciest zozbierané agentmi viac, ako AS pomocou upraveného pravidla prechodu. To zaručí lepšiu rovnováhu medzi skúmaním nových hrán (exploration) a využívaním nadobudnutých heuristických informácií (exploitation).

2. Aktualizácia feromónov prebieha iba na hranách patriacich do najlepšieho riešenia – *globálne aktualizačné pravidlo.*

3. Pri pohybe jednotlivých agentov z mesta i do mesta j sa aplikuje nové pravidlo – *lokálne aktualizačné pravidlo* [7, 8].

Neformálnym slovným popisom sa dá ACS opísať nasledovne – m agentov je rozmiestnených náhodne do n počiatočných miest. Každý agent vytvorí svoje riešenie preskúmavaním ciest. Pri výbere cesty sa riadi pravidlom prechodu a zároveň upravuje hodnotu feromónov na navštívených cestách pomocou lokálneho aktualizačného pravidla. Po dokončení cesty všetkými agentmi sa feromóny aktualizujú znova, a to aplikovaním globálneho aktualizačného pravidla. Rovnako ako aj v AS, agenti sa pri konštruovaní riešenia riadia dĺžkami hrán (s preferenciou kratších hrán) a feromónovou stopou. Čím silnejšia feromónová stopa je, tým pravdepodobnejšie je, že sa agent danou cestou vyberie. Vďaka vylepšeným aktualizačným pravidlám sa na hrany, ktoré by mali byť agentami navštívené, pridáva väčšie množstvo feromónov [6].

Konštrukcia riešenia v ACS

Pre pochopenie algoritmu ACS treba uviesť dôležité vzťahy. Agent k nachádzajúci sa v meste i si vyberá ďalšie miesto na navštívenie j pomocou pravidla prechodu s názvom *pseudorandom-proportional rule*, ktoré je dané

$$j = \begin{cases} argmax_{l \in C_i^k}\{\tau_{il}[\eta_{il}]^\beta\}, & \text{ak } q \leq q_0; \\ J, & \text{inak,} \end{cases} \tag{2.7}$$

kde q značí nahodnú premennú rovnomerne rozloženú na intervale $[0, 1]$. q_0 v rozmedzí $(0 \leq q_0 \leq 1)$ je parameter a J predstavuje náhodnú premennú danú prabdepodobnostným rozložením podľa vzorca 2.3 s parametrom $\alpha = 1$. C_i^k značí dosiahnuteľné mestá z mesta i pre agenta k, ktoré doposiaľ nenavštívil. [8] V tomto prípade parameter q_0 vyjadruje pravdepodobnosť s akou si agent k vyberie hranu s najvýhodnejšou kombináciou feromónov a heuristickej informácie (exploitation). S pravdepodobnosťou $(1 - q)$ bude agent viac preferovať prieskum svojho okolia (exploration). Vhodným nastavovaním tohto parametru sa dá dosiahnuť rovnováha medzi skúmaním nových hrán a využívaním nadobudnutých heuristických informácií [6].

Globálne aktualizačné pravidlo

Významná zmena v ACS oproti AS spočíva v tom, že iba celkovo najlepší agent (teda ten, ktorý vykonštruoval najkratšiu cestu v danej interácii), môže aktualizovať feromónovú stopu na hranách, ktoré patria do jeho riešenia. Táto zmena spolu s aplikáciou nového pravidla prechodu zaručí, že agenti budú silnejšie prehľadávať okolie dojterajšej najlepšej cesty [6]. Globálna aktualizácia sa aplikuje po ukončení všetkých ciest agentmi, kedy sa hodnota feromónov zmení na základe globálneho aktualizačného pravidla daného vzťahom

$$\Delta\tau(i, j) \leftarrow (1 - \rho) \cdot \tau(i, j) + \rho \cdot \Delta\tau(i, j) \tag{2.8}$$

kde

$$\Delta\tau(i, j) = \begin{cases} (L_{gb})^{-1}, & \text{ak } (i, j) \text{ patrí do globálne najlepšej cesty;} \\ 0, & \text{inak} \end{cases} \tag{2.9}$$

$0 < \rho < 1$ značí mieru vyparovania feromónov a L_{gb} predstavuje dĺžku globálne najlepšej cesty. Parameter ρ neznačí len mieru vyparovania ako v AS, taktiež slúži na výpočet novej feromónovej stopy, ktorá sa vypočíta ako vážený priemer starej feromónovej stopy a hodnoty nových uložených feromónov [8].

Lokálne aktualizačné pravidlo

Zmenou oproti AS je pridanie lokálneho aktualizačného pravidla 2.10, podľa ktorého agenti aktualizujú hodnotu feromónov na hrane (i, j) ihneď po jej prejdení.

$$\tau(i,j) \leftarrow (1-\xi) \cdot \tau(i,j) + \xi \cdot \Delta\tau(i,j) \tag{2.10}$$

$\Delta\tau(i,j)$ sa na začiatku algoritmu nastaví na vhodnú počiatočnú hodnotu r_0. Parameter r_0, ktorý sa udáva v rozmedzí $(0,1)$, značí iniciačnú hodnotu feromónov [6].

2.5 MAX-MIN Ant System (MMAS)

Skúmaním mravčích algoritmov sa prišlo k názoru, že najlepšie výsledky sa dajú dosiahnuť kombináciou efektívneho využívania nájdených najlepších riešení (exploitation) a mechanizmu, ktorý efektívne zabráni stagnácii na počiatku výpočtu [18]. Stützle a Hoos navrhli tzv. *MAX-MIN Ant System (MMAS)*, ktorý bol špecificky vyvinutý aby spĺňal tieto požiadavky. Oproti pôvodnému AS sa líši v štyroch hlavných aspektoch [8]:

1. Aby sa efektívne využívali najlepšie riešenia (exploitation) nájdené v danej iterácii alebo počas behu algoritmu, po každej iterácii môže feromóny aktualizovať iba jeden agent, ktorý našiel najlepšie riešenie buď v danej interácii (*iteration-best ant*), alebo od začiatku výpočtu (*global-best ant*). Táto stratégia však môže viesť k rýchlej stagnácii, pretože privysoký rast feromónovej stopy na vybraných hranách donúti veľa agentov nasledovať jednu a tú istú cestu.

2. Aby sa predišlo stagnácii, hodnota feromónovej stopy na každej hrane je obmedzená spodnou a hornou hranicou $[\tau_{min}, \tau_{max}]$.

3. Aby sa podporilo prehľadávanie ciest (exploration) na začiatku výpočtu, feromónové stopy sú inicializované na najvyššiu možnú hodnotu τ_{max}.

4. Ďalším mechanizmom na zabránenie stagnácie je reinicializácia feromonóvých stôp zakaždým, keď sa systém blíži k stagnácii alebo keď sa po určitom počte po sebe nasledujúcich iterácií nenájde lepšie riešenie.

MMAS je jedným z najviac študovaných ACO algoritmov a existuje mnoho jeho rozšírení. Vzhľadom na preukázaný dobrý výkon aj z hľadiska paralelizácie bude použitý pre potreby tejto práce.

Konštrukcia riešenia v MMAS

Na začiatku výpočtu sú agenti náhodne rozmiestnení do počiatočných miest. Pri výbere ďalšieho mesta sa riadia podľa pravdepodobnostného pravidla 2.3, teda rovnakého, ako v AS.

Aktualizácia feromónových stôp v MMAS

Po dokončení ciest všetkými agentmi prichádza na rad aktualizácia feromónov. Tá začína procesom vyparovania podľa vzorca 2.4, a následne aktualizáciou feromónovej stopy podľa nového vzťahu

$$\tau_{i,j} = \tau_{i,j} + \Delta\tau_{i,j}^{best}, \tag{2.11}$$

kde $\Delta\tau_{i,j}^{best}$ je množstvo feromónov pridané najlepších agentom x na hranu (i, j) dané vzťahom

$$\Delta\tau_{i,j}^{best} = \begin{cases} \frac{1}{L^{gb}}, & \text{ak } x \text{ je global-best ant;} \\ \frac{1}{L^{ib}}, & \text{ak } x \text{ je iteration-best ant;} \\ 0, & \text{inak,} \end{cases} \tag{2.12}$$

kde L^{ib} je dĺžka najlepšej cesty nájdenej v danej iterácii a L^{gb} je dĺžka najlepšej cesty nájdenej od začiatku výpočtu [8].

Rozsah feromóvej stopy v MMAS

Nezávisle na výbere agenta, ktorý aktualizoval feromónovú stopu sa môže riešenie uberať k stagnácii. Ak má nejaká hrana vyššiu feromónovú stopu, agent si podľa pravdepodobnostného pravidla prechodu 2.3 vyberie práve ju. Tým ju ešte viac „posilní“ a následne bude preferovaná viacerými agentmi – čo vedie k vytváraniu toho istého riešenia znova a znova bez posunu k lepšiemu výsledku. Jedným z riešení tohoto problému je ovplyvniť pravdepodobnosť výberu ďalšieho mesta, ktorá je závislá na heuristickej informácii a feromónovej stope. Nakoľko heuristická informácia sa počas výpočtu nemení, mechanizmus sa zameriava na limitovanie dopadu feromónov na výber mesta tým, že predchádza príliš veľkým rozdielom v koncentrácii feromónov na hranách [18].

Hranice τ_{min} a τ_{max} ohraničujú hodnotu feromóvej stopy τ_{ij} vo vzťahu $\tau_{min} \leq \tau_{ij} \leq \tau_{max}$. Tieto hranice upravujú pravdepodobnosť p_{ij} výberu mesta j, keď agent je v meste i vo vzťahu $0 < p_{min} \leq p_{ij} \leq p_{max} \leq 1$. Iba v prípade, že agent má na výber iba jedno mesto, $p_{min} = p_{max} = 1$. Ak hodnota feromóvej stopy je vyššia ako horná hranica, nastaví sa na τ_{max}. Analogicky, ak je nižšia, ako dolná hranica, nastaví sa na τ_{min}. Experimentálne výsledky [19] naznačujú, že na predchádzanie stagnácie je dôležitejšia dolná hranica. Tá horná hrá významnú rolu pri reinicializácii feromónových stôp [8].

Inicializácia a reinicializácia feromónových stôp v MMAS

Na začiatku algoritmu sa feromónové stopy inicializujú na odhad hornej hranice τ_{max}. Vďaka tomuto kroku v kombinácii s nízkou mierou vyparovania feromónov sa hodnoty feromónových stôp pomaly zvyšujú, čo zaručí, že v počiatočných iteráciách sa bude preskúmavať veľa riešení. Experimentálne výsledky potvrdujú, že intenzívnejšie skúmanie priestoru zlepšuje výkon MMAS [18].

Hrany s nízkou feromónovou stopou majú malú šancu, že ich agenti zahrnú do svojho riešenia. Aby sa pravdepodobnosť ich výberu zvýšila, a tým boli preskúmané ďalšie riešenia, feromónové stopy sa občas reinicializujú. Reinicializácia nastáva zvyčajne v okamihu, kedy sa systém blíži k stagnácii, alebo sa po určitom počte po sebe nasledujúcich iterácií nenašlo lepšie riešenie [8].

2.6 Ďalšie varianty mravčích algoritmov

Existuje mnoho variant mravčích algoritmov líšiacich sa vo viacerých aspektoch – najčastejšie v konštrukcii cesty agentmi, výpočtu hodnôt feromónových stôp, prípadne ďalšími pridanými metódami a technikami. Avšak všetky varianty majú rovnaký základ, všeobecný postup nazývaný *ant foraging*, zhrnutý v algoritme 1. Nižšie sú uvedené známe a úspešné varianty, ktoré do veľkej miery prispeli k riešeniu mnohých úloh a k ďalšiemu vývoju MA.

Algoritmus 1: Všeobecný algoritmus ACO [1]

```
Initialize pheromone trails to positive value;
repeat
    for each ant in turn do
        construct a solution (using optional heuristic);
        local pheromone update (optional step);
        measure quality of solution;
    end
    Update pheromone trails via pheromone deposit and evaporation;
until terminating condition;
```

Fast Ant System (FANT) – bol navrhnutý špecificky na riešenie tzv. *Quadratic Assignment Problem – QAP.* Hlavným rozdielom medzi FANT a ostatnými mravčími algoritmami je využívanie iba jedného mravca a aplikovanie rozdielneho feromónového aktualizačného pravidla, v ktorom sa nepoužíva vyparovanie feromónov [9].

Elitist Ant System (EAS) – bol prvým vylepšením pôvodného AS. Zaviedol tzv. *elitist strategy* – stratégiu, ktorá zintenzívňuje feromónovú stopu na hranách patriacich do doposiaľ najlepšieho riešenia nájdeného od začiatku algoritmu [8].

Ant-Q – varianta ACS, v ktorej je lokálne aktualizačné pravidlo inšpirované algoritmom *Q-learning*[7]. Využíva *AQ-hodnoty*, ktoré upredňostňujú skúmanie ďalších najkratších ciest agentmi namiesto konvergovania k jednej spoločnej [10].

Continuous Ant Colony System (CACS) – rozšírenie ACO metaheuristiky pre riešenie spojitých problémov. Metóda je založená čisto na feromónových stopách bez využitia heuristických informácií. Taktiež obsahuje menej riadiacich parametrov, vďaka čomu je počiatočné nastavovanie parametrov oveľa jednoduchšie, ako v ostatných metódach [17].

[7] http://incompleteideas.net/sutton/book/ebook/node65.html

Kapitola 3

Modifikácie mravčích algoritmov pre rozsiahle úlohy TSP

Táto kapitola vychádza z článku *Scaling Techniques for Parallel Ant Colony Optimization on Large Problem Instances* [16], ktorý predstavuje hlavný zdroj optimalizácií implementovaných v praktickej časti tejto práce. V tejto kapitole sa rozoberajú modifikácie a rozšírenia mravčích algoritmov pre riešenie rozsiahlych inštancií problému obchodného cestujúceho s cieľom ušetriť výpočetný čas a operačnú pamäť. Spomenuté sú modifikácie zaoberajúce sa výberom nasledujúceho mesta – Independent Roulette, vRoulette-1, a bližšie vysvetlená je ich vylepšená varianta Vectorized Candidate Set Selection. Taktiež je popísaná situácia, kedy zoznam susedných miest na výber nasledujúceho mesta je prázdny. Ďalej sú spomenuté modifikácie, ktoré sa zaoberajú znižovaním pamäťovej náročnosti – P-ACO, PartialACO a bližšie vysvetlená metóda Restricted Pheromone Matrix.

Ako už bolo spomenuté, mravčie algoritmy prispeli významným spôsobom k riešeniu úloh TSP. Narazili však na svoj limit – vysoký počet miest v inštancii, kedy sa už stali kvôli časovej a pamäťovej náročnosti takmer nepoužiteľné. Napríklad, pre nájdenie optimálneho riešenia inštancie TSP s 10 000 mestami je potreba približne 380 MB operačnej pamäte, s čím by si väčšina moderného hardvéru vedela poradiť. Avšak, pri inštancii TSP so 100 000 mestami je potrebnej približne až 37 GB operačnej pamäte, čo už možno považovať za nepraktické [16]. Postupným vylepšovaním hardvéru sa síce limity veľkosti inštancie TSP podarilo mierne posunúť, no aj napriek tomu je treba navrhovať a testovať nové optimalizácie MA, ktoré zrýchlia a zefektívnia riešenie rozsiahlych inštancií TSP bez výraznej straty na kvalite nájdeného riešenia.

Hlavnými oblasťami, ktorými sa optimalizácie v tejto kapitole zaoberajú sú:

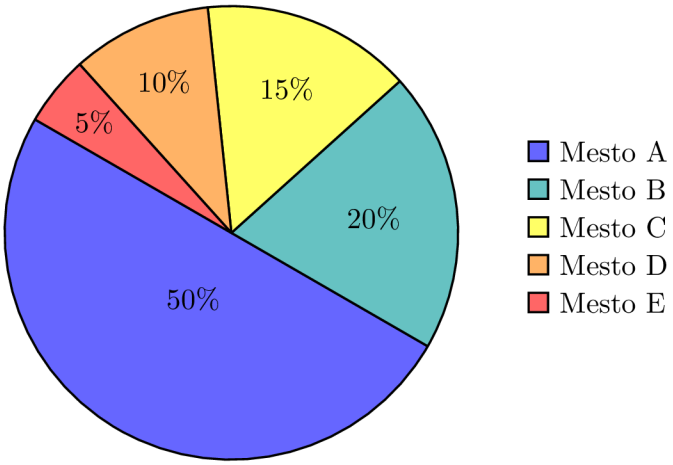
- Modifikácie výberu nasledujúceho mesta, ktoré minimalizujú čas potrebný na skonštruovanie kandidátneho riešenia.
- Metódy výberu nasledujúceho mesta v prípade, že všetky susedné mestá už boli navštívené.
- Paralelizácia konštruovania ciest mravcami, nakoľko výber nasledujúceho mesta je výpočtovo najnáročnejšia časť algoritmu.
- Modifikácie dátových štruktúr ukladajúcich informácie o inštancii, napríklad úprava feromónovej matice s cieľom zníženia pamäťovej náročnosti.

3.1 Modifikácie výberu nasledujúceho mesta

Efektivitu MA významne ovplyvňuje spôsob výberu nasledujúceho mesta. Význam zvolenej metódy rastie úmerne s veľkosťou inštancie, pretože možností na výber ďalšieho mesta rapídne pribúda. Pre tieto situácie sa osvedčilo zúžiť počet ďalších možných miest, ktoré sú na výber pomocou tzv. *kandidátnych zoznamov*. Myšlienkou kandidátnych zoznamov je, že dobré riešenie TSP sa snaží predchádzať dlhým „skokom“ medzi mestami a volí skôr lokálnejšie prechody medzi jednotlivými mestami. Avšak kandidátne zoznamy sa ukázali ako problémové pri návrhu výberových metód, ktoré sa snažia využívať inštrukcie procesora pracujúce s viacerými dátami súčasne *(SIMD – Single Instruction, Multiple Data)*. *Vektorizácia* je proces úpravy algoritmu tak, aby využíval pri svojom výpočte inštrukcie procesora, ktoré pracujú s vektormi. Vektor je jednorozmerné pole obsahujúce až 16 hodnôt s pohyblivou desatinnou čiarkou. Upravená reprezentácia zoznamu najbližších miest a modifikovaná metóda výberu následujúceho mesta vedie k výraznému zrýchleniu práve efektívnym využitím vektorizácie.

3.1.1 Roulette Wheel a Independent Roulette

Algoritmus *Roulette Wheel*, ktorý je využívaný v implementáciách ACO pre výber ďalšieho mesta, berie pri náhodnom výbere následúceho mesta do úvahy váhu hrany medzi mestami. Čím má prechod väčšiu váhu, tým je pravdepodobnejšie, že bude náhodne vybraný. Príklad výberu nasledujúceho mesta pomocou techniky Roulette Wheel je znázornený grafom 3.1. Avšak, tento sekvenčný algoritmus je len ťažko paralelizovateľný.



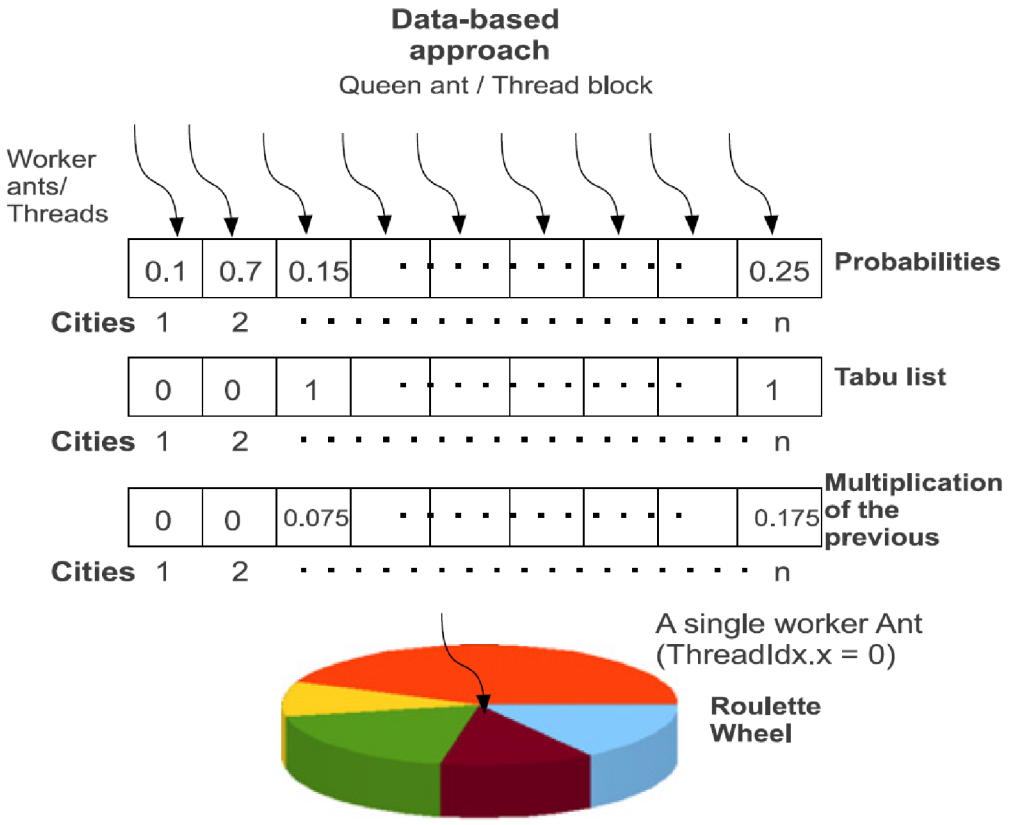
Obr. 3.1: Graf zobrazujúci príklad výberu nasledujúceho mesta pomocou techniky Roulette Wheel Selection. Každému mestu pripadá proporcionálna oblasť, ktorá vyjadruje pravdepodobnosť výberu na základe váhy prechodu do daného mesta.

Vzrastajúca dostupnosť a možnosti vysoko výkonných výpočtových platforiem, ako sú napríklad grafické karty (angl. Graphical Processing Units – GPU), viedli k záujmu o ich využitie pre paralelizované mravčie algoritmy. Plné využitie ponúkaného potenciálu vyžaduje vysoký stupeň paralelizmu, čo viedlo k vývoju modifikácie *I-Roulette (Independent Roulette)*. Pre jednoduché využitie výpočtových prostriedkov GPU bol využitý framework *CUDA* (Compute Unified Device Architecture).

Obe hlavné časti mravčieho algoritmu – konštrukcia riešenia aj ukladanie feromónovej stopy sú výzvou pre implementáciu v paralelnom prostredí. Existuje veľmi jednoduchý spôsob ako paralelizovať fázu vytvárania kandidátneho riešenia – priradením jedného mravca do práve jedného vlákna, tzv. *úlohový paralelizmus* (angl. task-paralelism), avšak takéto paralelné riešenie nie je dostatočne granulárne aby efektívne využilo ponúkaný masívne paralelný hardvér.

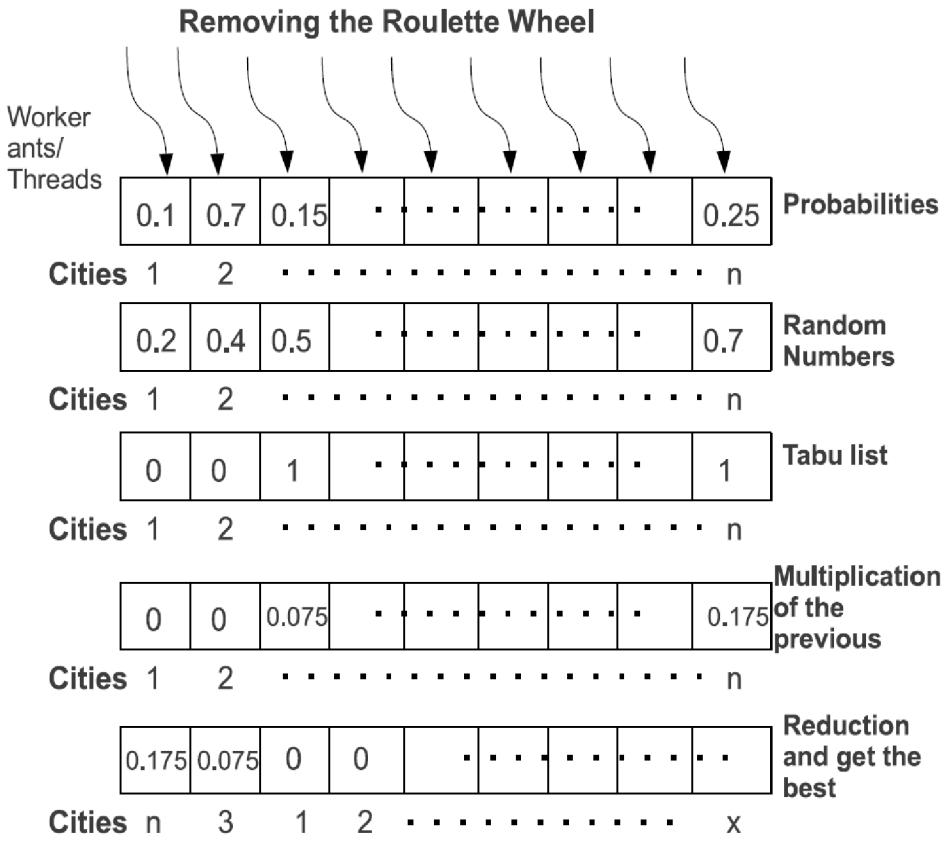
Ako riešenie bola navrhnutá implementácia ACO, ktorá používa tzv. *dátový paralelizmus* (angl. data-paralelism). Tento prístup rozdeľuje vlákna do blokov, ktoré sú súčasťou CUDA frameworku. Každý blok vlákien reprezentuje jednotlivé mravce, ktoré hľadajú kandidátne riešenie. Tieto bloky sú nazývané *„mravčie kráľovné“* (angl. Queen ants) a využívajú *„mravčích robotníkov“* (angl. Worker ants), ktorí potom spolupracujú so svojím nadriadeným mravcom (vláknom) na hľadaní riešenia. S každým robotníckym vláknom je spojené jedno mesto (alebo aj viacej miest), ktoré môže mravec navštíviť [2].

Ako je ukázané na obrázku 3.2, všetky robotnícke vlákna spočítajú váhy všetkých prechodov medzi aktuálnym a dostupnými mestami. Váha prechodu je vyjadrená ako násobenie heuristickej informácie (Probabilities) s informáciou zo zoznamu navštívených miest mravca (Tabu list). Ak mesto nebolo navštívené, je táto informácia vyjadrená číslom 1, inak je to 0. Takto sa predíde podmienkam pri vytváraní poľa pre simulovanú ruletu. Nasleduje výber nasledujúceho mesta pomocou simulácie ruletového kolesa. Do takto vybraného mesta sa mravčia kráľovná presunie a proces sa opakuje.



Obr. 3.2: Príklad výberu nasledujúceho mesta pomocou techniky I-Roulette so simuláciou ruletového kolesa, ukazujúci dátový paralelizmus [2].

Napriek dosiahnutému stupňu paralelizmu je simulácia ruletového kolesa náročná na implementáciu a výkonovo neefektívna. Preto bol navrhnutý spôsob ako ju úplne nahradiť. Tento postup je ukázaný na obrázku 3.3. Každé robotnícke vlákno vypočíta hodnotu prechodu medzi aktuálnym a dostupnými mestami. Táto hodnota je vypočítaná ako vynásobením troch čísel – heuristickej informácie (Probabilities), náhodného čísla (Random Numbers) a informácie vyjadrujúcej, či mesto bolo alebo nebolo navštívené (rovnaké hodnoty ako v prechádzajúcej verzii). Výsledok sa uloží do poľa spoločného pre všetky robotnícke mravce. Po vyhodnotení všetkých prechodov algoritmus pokračuje na výber najvyššej hodnoty zo spoločného poľa. Simulácia ruletového kolesa bola nahradená pridaním náhodného čísla do výpočtu hodnoty prechodu. Do mesta s najvyššou hodnotou sa mravčia kráľovná presunie a proces sa opakuje [13].



Obr. 3.3: Príklad výberu nasledujúceho mesta pomocou techniky I-Roulette bez simulácie ruletového kolesa, ktoré je nahradené pridaním náhodného čísla do výpočtu [2].

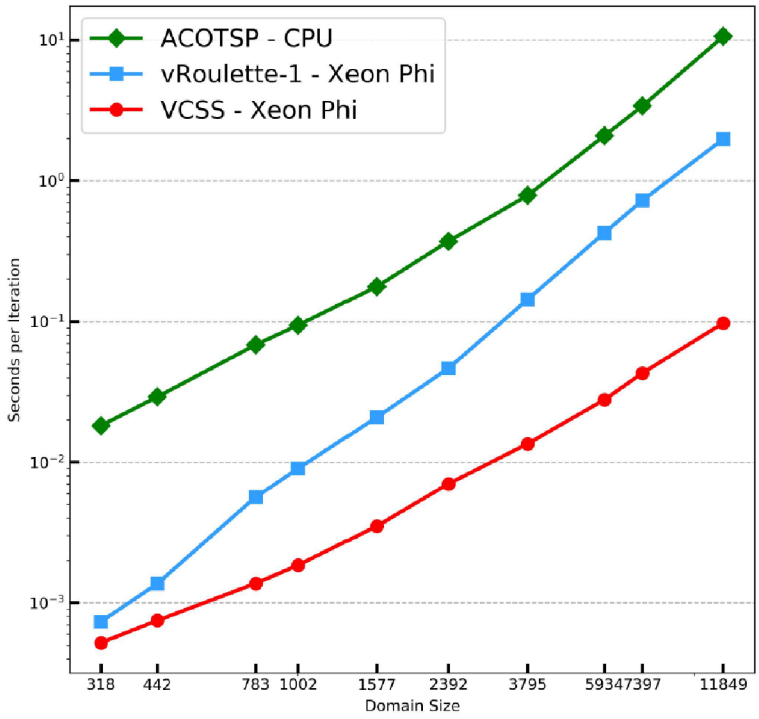
3.1.2 vRoulette-1

Technika Independent Roulette bola neskôr vektorizovaná, aby mohla využiť vektorizačný potenciál nových procesorov – napríklad viacjadrový procesor od Intelu Xeon Phi, pomocou jeho jednotky na spracovávanie vektorov (Vector Processing Unit – VPU) a *IMCI inštrukcií* (IMCI – Initial Many Core Instructions). Nová, vektorizovaná verzia sa nazýva *vRoulette-1* a vďaka nej je možné použiť techniku I-Roulette na viacjadrových architektúrach s využi-

tím inštrukcií SIMD, pracujúcich s viacerými dátami súčasne (SIMD – Single Instruction, Multiple Data) [12].

vRoulette-1 si zachovalo základy I-Roulette. Váhy všetkých hrán a náhodné čísla sú načítané do vektorov a násobia sa súčasne. Počas celého procesu je udržovaný vektor obsahujúci najvyššie dosiahnuté výsledky násobenia. Po dokončení výpočtu je vektor redukovaný s cieľom nájsť najvyššiu hodnotu. Mesto, ktorému táto hodnota pripadá sa stane nasledujúcim mestom, ktoré mravčia kráľovná navštívi. vRoulette-1 nevyberá hrany s pravdepodobnosťou výberu priamoúmernej ich váhe, avšak hrany s väčšou váhou budú vybrané častejšie, ako tie s nižšou [12, 16].

Technika vRoulette-1 používa *zoznam najbližších miest* (angl. Nearest Neighbour List) na dosiahnutie vyššej kvality riešenia. Zoznam je vytvorený pri inicializácii algoritmu pre každé mesto, ako jednorozmerné pole indexov ostatných miest, zoradené podľa ich vzdialenosti od mesta, pre ktoré je zoznam vytvorený. Pre zaradenie týchto zoznamov jednotlivých miest do výpočtu je následne vytvorené dvojrozmerné pole hodnôt s pohyblivou desatinnou čiarkou. Jednotlivé prvky tohto dvojrozmerného poľa svojím indexom (i, j) reprezentujú prechod medzi mestom i a j. Hodnota prvku (i, j) je 1 ak je mesto j v zozname najbližších miest mesta i, inak 0.



Obr. 3.4: Výpočetný čas potrebný na jednu iteráciu výpočtu riešenia TSP pri použití pôvodnej implementácie MMAS na procesore, vRoulette-1 na Xeon Phi a VCSS (sekcia 3.1.3) na Xeon Phi. Použité inštancie sú v rozsahu 318-11849 miest. Výsledky pochádzajú z experimentov uvedených v článku [15].

3.1.3 Vectorized Candidate Set Selection (VCSS)

Doterajšie snahy o vylepšienia výkonnosti ACO algoritmov smerujúce k zníženiu času potrebného na vytvorenie kandidátneho riešenia boli z časti úspešné, no ani jedna modifikácia

nebola dostatočne ekektívna aby si poradila aj s veľmi rozsiahlymi inštanciami (100 000 miest a viac) v rozumnom čase. Tzv. vektorizácia výberu miest pri zostavovaní ciest mravcami (angl. Vectorized Candidate Set Selection – VCSS) vychádza z postupného vývoja modifikácií, začínajúc prvou paralelnou implementáciou výberu nasledujúceho mesta, I-Roulette (sekcia 3.1.1), nasledovanou vektorizovanou verziou tohto algoritmu nazvanou VRoulette-1 (sekcia 3.1.2). VRoulette-1 bol značne rýchlejší ako jeho predchodca, avšak nedokázal naplno efektívne využiť potenciál, ktorý ponúkajú procesory v oblasti vektorizácie výpočtu. Tento rozdiel je viditeľný aj na obrázku 3.4, kde je porovnávané pôvodné MMAS bez úprav, MMAS s úpravou vRoulette1 a modifikáciou VCSS. Nedostatkom bolo použitie zoznamu najbližších miest iba na zvýšenie kvality riešenia a nevyužitie tohto zoznamu na zrýchlenie výpočtu. Tieto nedostatky rieši VCSS upraveným algoritmom výpočtu následujúceho mesta a definovaním novej dátovej štruktúry, ktorá slúži ako zoznam najbližších miest [15]. Nakoľko VCSS je doposiaľ najefektívnejšia modifikácia výberu nasledujúceho mesta, je implementovaná a testovaná v tejto práci.

Vylepšený zoznam najbližších susedných miest

Kľúčovým prínosom VCSS je vektorizovaný algoritmus spolu s príslušnou štruktúrou údajov, ktoré urýchlujú výber mesta použitím vylepšeného zoznamu najbližších miest, tzv. *kandidátnych setov* (angl. candidate sets). Oproti pôvodným verziám je výber nasledujúceho mesta upravený tak, že z aktuálnej pozície sa berú do úvahy iba uzly v zozname najbližších miest. Iba v prípade, že na výber nie je žiadne mesto (všetky blízke už boli raz navštívené), bude výber nasledujúceho uzlu umožnený zo zoznamu všetkých dostupných miest (doposiaľ nenavštívených). Zvyčajne sa zoznam najbližších susedných miest obmedzuje na približne 20 miest, čo pre rozsiahle inštancie môže znamenať urýchlenie výberového procesu mnohonásobne. Príklad, ako môže vyzerať najbližšie okolie aktuálneho mesta je zobrazený na obrázku 3.5 [15].

Počas inicializácie algoritmu sú mestá zoradené podľa vzájomnej vzdialenosti tak, aby boli jednotlivé bitové masky čo najviac využité. Týmto spracovaním sa docieli kratších zoznamov objektov najbližších susedov a urýchli sa výpočet algoritmu.

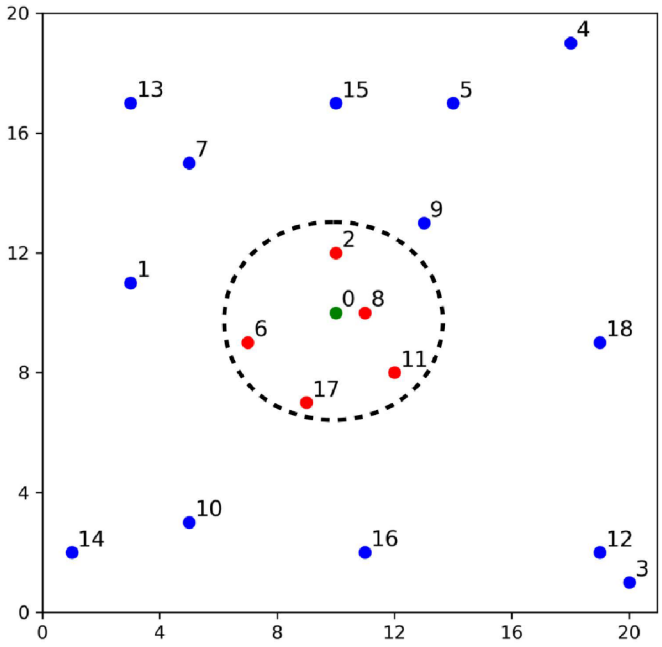
Ďalej je vytvorená $n \times n$ matica (kde n je počet miest v inštancii), ktorá obsahuje vzdialenosti jednotlivých miest. Informácie o vzdialenostiach z tejto matice sa použijú na vytvorenie zoznamu najbližších miest pre každé jedno mesto v inštancii. Ako prvé je vypočítaný potrebný počet *objektov najbližších susedov*, do ktorých sa uložia jednotlivé zoznamy najbližších miest. Toto je jednoducho vypočítané podľa vzorca

$$N_p = \lceil \frac{N_{nn}}{p} \rceil, \tag{3.1}$$

kde N_p je potrebný počet SIMD vektorov na uloženie zoznamu, N_{nn} je dĺžka zoznamu najbližších miest a p je šírka SIMD vektoru na danej architektúre. Zoznamy sú v týchto objektoch uložné ako bitové masky, kde podľa indexu mesta je nastavený príslušný bit v maske. Ak je mesto j nabližším susedom mesta i, tak je v bitovej maske určenej vzťahom 3.2 nastavený bit na pozícii j mod p na hodnotu 1.

$$ivec = \frac{J}{p} \tag{3.2}$$

Záznam pre každé mesto je naplnený prvými N_{nn} najbližšími mestami podľa vzdialeností z matice. Pre každé mesto je spočítaná hodnota *ivec* – ak táto hodnota už v zázname



Obr. 3.5: Príklad grafu TSP inštancie, kde aktuálne mesto (označené ako 0) je v strede a jeho 5 najbližších susedných miest je vyznačených červene v prerušovanom kruhu [15].

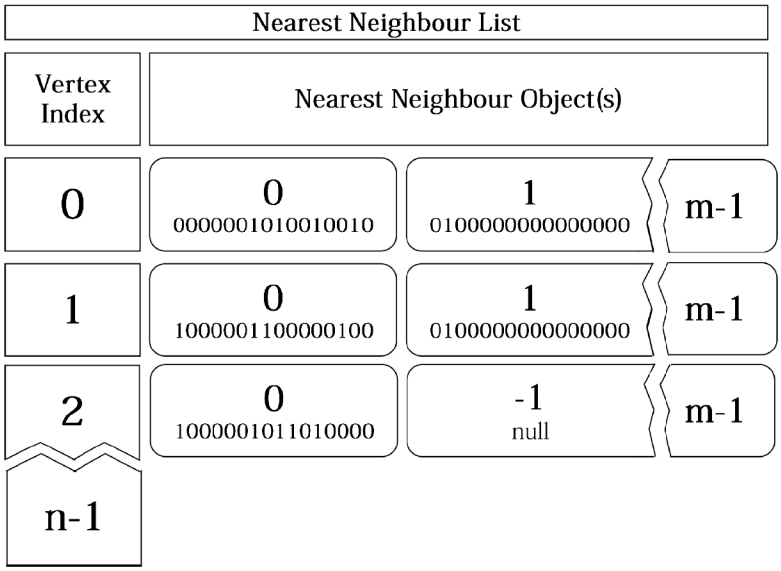
existuje, tak sa nastaví príslušný bit; ak ešte neexistuje, je pridaná do záznamu a následne je nastavený príslušný bit. Príklad ako môže vyzerať takáto dátová štruktúra je na obrázku 3.6 pre vektory, ktoré majú šírku 16 hodnôt s pohyblivou desatinnou čiarkou [15].

Konštrukcia riešenia s použitím VCSS

Metóda VCSS sa vyvinula upravením metódy I-Roulette (sekcia 3.1.1) pre použitie na procesoroch (I-Roulette bola implementovaná pre GPU). Taktiež sa vytvorili nové dátové štruktúry, aby sa zlepšila vektorizácia kódu. Konštrukcia riešenia je paralelizovaná pomocou knižnice *OpenMP*, ktorá ponúka jednoduché ale efektívne viacvláknové programovanie. Každému mravcovi je pridelené vlastné vlákno v rámci ktorého realizuje výber nasledujúceho mesta paralelne s ostatnými mravcami. Nakoľko mravce v mravčom algoritme MMAS implementovanom v tejto práci používajú len vlastnú lokálnu pamäť a teda nezapisujú do zdielanej pamäte, nie je treba žiadna synchronizácia počas konštrukcie riešenia. Na začiatku sa mravce rozmiestnia náhodne do počiatočných miest a následne sa opakovane volá funkcia výberu nasledujúceho mesta. Každý mravec si udržuje vlastný zoznam navštívených miest. Vstupné parametre algoritmu VCSS sú:

- zoznam doposiaľ navštívených miest (tabu list),
- pole váh prechodov do nasledujúcich miest,
- zoznam najbližších susedných miest pre mesto i.

Výpočet algoritmu potom prebieha nasledovne – ako prvé sa inicializujú dva vektory, ktoré reprezentujú doposiaľ nájdené maximum váhy prechodov a indexy miest, do ktorých tieto prechody vedú. v ďalšom kroku algoritmus iteruje cez všetky vektory tvoriace zoznam najbližších miest. V každom kroku načíta váhy prechodov ako jeden vektor o dĺžke p. Po načítaní

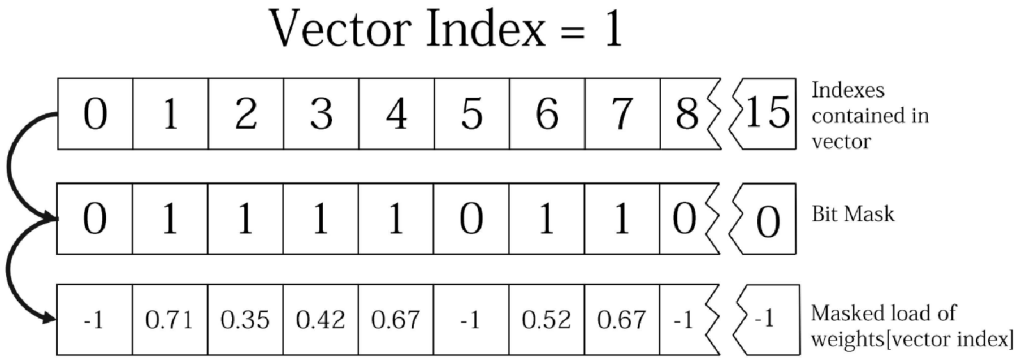


Obr. 3.6: Štruktúra zoznamu najbližších susedných miest. Každý uzol (mesto) má priradené pole objektov predstavujúcich najbližšie mestá. Objekty obsahujú vektorový index *ivec* a bitovú masku. Hodnota $ivec = -1$ je príznakom konca zoznamu pre dané mesto. n značí počet uzlov (miest), a n_{16} znaží najvyšší počet záznamov pre dané mesto (pre vektory o šírke 16 hodnôt s pohyblivou desatinnou čiarkou) [15].

sa použije bitová maska uložená v zozname najbližších miest na odfiltrovanie váh prechodov do miest, ktoré nie sú susedné aktuálnemu mestu. Príklad načítania a následného filtrovania je možné vidieť na obrázku 3.7. Ďalej sa vytvorí vektor obsahujúci po sebe idúce celé čísla, ktoré reprezentujú indexy miest, ktorých váhy sú práve načítané vo vektore váh. Volaním funkcie *Random()* sa vytvorí vektor obsahujúci náhodné čísla v intervale $(0, 1)$. Ten je potom vynásobený s vektorom načitaných váh a výsledok je uložený znovu ako vektor. Výsledok násobenia je porovnaný s doposiaľ nájdenými maximálnymi hodnotami prechodov, a to prvok po prvku pomocou jednej inštrukcie. Výsledkom porovnania je modifikovaný vektor maximálnych hodnôt a bitová maska, ktorá vyjadruje, ktoré prvky boli nahradené za nové väčšie a ktoré zostali nezmenené. Bitová maska je následne použitá na aktualizovanie vektoru indexov prislúchajúcim najvyšším hodnotám doposiaľ nájdeným. Po spracovaní všetkých susedných miest je nájdené maximum medzi hodnotami uloženými vo vektore navyšších hodnôt. Toto hľadanie je vykonané v $log_2 p$ krokoch pomocou takzvaných *premiešavacích a porovanávacích vektorových inštrukcií* (angl. *bit-swizzling and compare instructions*). Ak nebol vybraný žiadny prechod, je použitá niektorá z metód výberu mesta pri prázdnom zozname najbližších susedných miest popísaných v nasledujúcej časti [15].

Výber mesta pri prázdnom zozname najbližších susedných miest

V štandardných algoritmoch ACO sa väčšinou pri prázdnom zozname najbližších susedných miest vyberie prechod s najvyšším ohodnotením spomedzi všetkých nenavštívených miest. Avšak pri použití modifikácie *Restricted Pheromone Matrix* (popísaná v sekcii 3.2.2) nie sú váhy prechodov (feromóny) miest nenachádzajúcich sa v zozname k dispozícii. Spôsob riešenia tohto problému spočíva v technike s názvom *Fallback* (tzv. záložný spôsob výberu



Obr. 3.7: Načítavanie váh prechodov s použitím masky najbližších miest. Pod jednotlivými indexmi sú uložené váhy prechodov. Po aplikovaní masky sa vytvorí vektor váh prechodov do susedných miest, kde -1 indikuje, že mesto nie je susedné a nebude sa s ním ďalej pracovať; kladné číslo indikuje platného suseda, ktorý bude zahrnutý do konštrukcie riešenia [15].

nasledujúceho mesta pri prázdnom zozname nasledujúcich miest). V tejto práci sú implementované dve rôzne varianty:

- *Heuristic Fallback* – spočíva v nájdení najbližšieho doposiaľ nenavštíveného mesta tak, že vzdialenosti medzi aktuálnym a ostatnými nenavštívenými mestami je vypočítaná priamo zo súradníc miest. Aby sa vyhlo počítaniu odmocniny, počíta sa s najnižšou štvorcovou euklidovskou vzdialenosťou [16].
- *Pheromone Map Fallback* – čo najvernejšie napodobňuje chovanie algoritmu MMAS tým, že sa snaží o ohodnotenie všetkých hrán feromónovou stopou. Tieto hodnoty sú ukladané do C++ objektu *map* predstavujúci asociatívne pole, v ktorom sú hodnoty ukladané v pároch kľúč-hodnota. Ukladané sú hrany, ktoré tvoria alebo tvorili najlepšie riešenie. Každá hrana má svoj unikátny kľúč, ktorý ju identifikuje medzi ostatnými hranami grafu. Hodnota kľúča sa dá spočítať jednoduchým matematickým vzorcom $(A \times N) + B$, kde A a B sú indexy miest tvoriace počítanú hranu a N je počet miest. Jedinou podmienkou je, aby A bolo väčšie ako B. Pri použití tejto metódy sa predpokladá, že každá hrana, ktorá je uložená v *map* má priradenú feromónovú hodnotu. Ak sa feromónová hodnota pre danú hranu nenájde, použije sa hodnota τ_{min}. Pomocou cyklu sú spočítané hodnoty kľúča pre každé jedno mesto v inštancii. Feromonová hodnota asociovaná s touto hranou (kľúčom) je vyhľadaná v mape a následne je spočítaná s euklidovskou vzdialenosťou daných miest. Index spojený s najvyššou hodnotou zo všetkých je použitý ako následujúce mesto [16].

3.2 Modifikácie znižujúce pamäťovú náročnosť

Pamäťová náročnosť MA sa ukázala byť problematická pri rozsiahlych inštanciách, nakoľko operačná pamäť potrebná pre výpočet dosahovala často aj niekoľko desiatok, až stoviek GB pri inštanciách nad 100 000 miest. Názorný príklad pamäťovej náročnosti je ukázaný v tabuľke 3.1. V následujúcich sekciách sa pojednáva o snahe odstrániť alebo efektívne upraviť feromónovú maticu tak, aby kvadratická priestorová zložitosť algoritmu bola nahradená lineárnou [16].

Veľkosť inštancie	Feromónová matica	Obmedzená matica
100	39 KB	12,5 KB
1 000	3,8 MB	125 KB
10 000	381,5 MB	1,22 MB
100 000	37,3 GB	12,2 MB

Tabuľka 3.1: Operačná pamäť potrebná na výpočet úlohy TSP s rôznym počtom miest pri použití feromónovej matice a upravenej obmedzenej feromónovej matice [16].

3.2.1 P-ACO a PartialACO

Ako už bolo spomenuté, základná pamäťová náročnosť neupravených ACO algoritmov pre feromónovú maticu je $O(n^2)$, kde n je počet miest. Takáto zložitosť sa stáva pri inštanciách o veľkosti 10^5 miest a viac neúnosnou pre väčšinu novodobého hardvéru.

Jednou z prvých snáh o zníženie pamäťovej náročnosti je modifikácia *Populationbased ACO (P-ACO)*. Tá síce bola prv určená primárne na riešenie dynamických problémov, než rozsiahlych problémov ako takých. Ako napovedá názov, P-ACO sa pri konštrukcii riešenia zameriava výhradne na populáciu dobrých riešení a teda feromónová matica je kompletne odstránená. Až riešenia dosiahnu určitého „veku“, sú z populácie odstránené a nahradené novými. Oproti pôvodným ACO algoritmom, v ktorých sa mravce riadia feromonóvou stopou, využívajú mravce populáciu dobrých riešení na výber ďalšieho mesta [11, 16].

Ďalšou technikou, ktorá sa snažila znížiť pamäťovú náročnosť je *PartialACO*. Táto technika bola výrazne inšpirovaná predchádzajúcou modifikáciou P-ACO. Feromónová matica v tomto prípade nie je úplne odstránená, ale nahradená lokálnou pamäťou pre každého mravca, v ktorej sa ukladá najlepšie riešenie každého jedinca. Mravec pri pravdepodobnostnom výbere nasledujúceho mesta berie do úvahy feromónové stopy z najlepšieho riešenia, ktoré je uložené v jeho lokálnej pamäti. Nevytvára nové riešenie v každej iterácii, teda upravuje iba časť riešenia. Pokiaľ je počet mravcov výrazne menší, ako počet miest v inštancii, nastáva výrazné zníženie pamäťovej náročnosti. Na začiatku iterácie je každý mravec náhodne umiestnený do počiatočného mesta a vyberie si časť miest z jeho doterajšieho najlepšieho riešenia, ktoré si ponechá a budú tvoriť časť nového riešenia. Vďaka tejto modifikácii bolo možné zaznamenať prvé výsledky rozsiahlych inštancií s veľkosťou od 100 000 až po 200 000 miest. Čas potrebný na výpočet týchto inštancií sa pohyboval medzi 1 až 7,4 hodinami [3, 16].

3.2.2 Restricted Pheromone Matrix

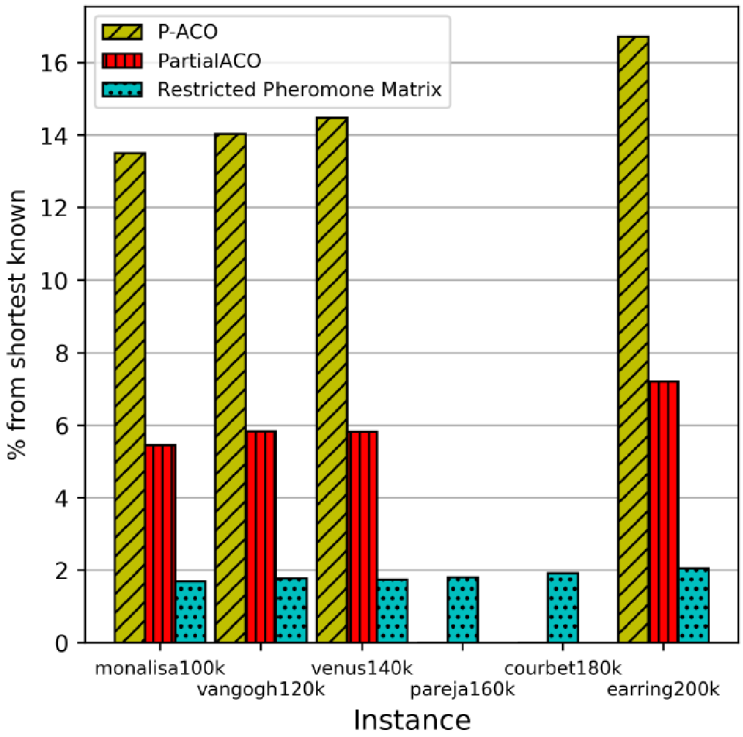
Predchádzajúce modifikácie snažiace sa kompletne odstrániť feromónovú maticu úspešne znížili pamäťovú náročnosť ACO algoritmov, no čiastočne sa odklonili od základnej myšlienky – komunikácia mravcov pomocou feromónových stôp. Postupným skúmaním sa prišlo k vylepšeniu, ktoré si zachováva pôvodnú myšlienku nepriamej komunikácie a zároveň znižuje pamäťovú náročnosť. Spomínaná modifikácia je implementovaná v tejto práci.

Obmedzená feromónová matica (angl. Restricted Pheromone Matrix) využíva novú pamäťovú štruktúru – kandidátne sety najbližších susedných miest a tým znižuje pamäťovú zložitosť inštancie z kvadratickej na lineárnu. Taktiež značne prispieva aj k zníženiu výpočetného času. Feromónové stopy sú ukladané len pre hrany idúce z aktuálneho mesta do miest, ktoré sú jeho najbližšími susedmi. V porovnaní so zložitosťou $O(n^2)$, akú dosahujú pôvodné mravčie algoritmy, obmedzená feromónová matica používa $nn_{NN}v$ reálnych čísel,

kde n je počet miest v inštancii, n_{NN} je počet najbližších miest a v predstavuje veľkosť vektoru dostupného na aktuálnom hardvéri.

Tak, ako už bolo ukázané v tabuľke 3.1, význam tejto modifikácie rastie s počtom miest v inštancii – napríklad inštancia TSP obsahujúca 100 000 miest zaberá v obmedzenej feromónovej matici len 0,26% priestoru oproti klasickej feromónovej matici. Vzdialenosti medzi mestami sú uložené len pre mestá, ktoré sú súčasťou kandidátnych setov, a preto má matica vzdialenosti rovnakú zložitosť, ako obmedzená feromónová matica [16].

Výsledky experimentov z článku [16] dokazujú, že obmedzená feromónová matica produkuje riešenia, ktoré sú horšie o 1% až 2% oproti najlepším dosiahnutým výsledkom. Na obrázku 3.8 je možné vidieť porovnanie techník zaoberajúcich sa znižovaním pamäťovej náročnosti, kde je jasne ukázané, že upravená feromónová matica je doposiaľ najpresnejšia.



Obr. 3.8: Graf ukazujúci rozdiel kvality riešenia inštancií TSP s použitím P-ACO, PartialACO a obmedzenej feromónovej matice v porovnaní s najlepším známym riešením. Pre inštancie `pareja160k` a `courbet180k` nie sú dostupné riešenia pri použití techník P-ACO a PartialACO. Výsledky experimentov spolu s grafom pochádzajú z článku [16].

Kapitola 4

Vlastné modifikácie mravčích algoritmov pre rozsiahle úlohy TSP

Táto kapitola sa zaoberá vlastnými vylepšeniami mravčích algoritmov, ktoré neboli spomenuté v hlavnom článku, z ktorého práca vychádza – *Scaling Techniques for Parallel Ant Colony Optimization on Large Problem Instances* [16] a pridruženej literatúre, z ktorej autori čerpali. Mnou navrhnuté vylepšenia cielia k zníženiu pamäťovej náročnosti algoritmu a k ďalšiemu zrýchleniu výpočtu. V prvej sekcii sa pojednáva o upravenom spôsobe ukladania zoznamov najbližších miest, ktorý šetrí pamäť presunutím informácií o feromónoch do vlastnej štruktúry. Ďalej sa rozoberá paralelná práca s feromónovými stopami s využitím vektorových inštrukcií a v poslednej sekcii je popísaný vektorizovaný heuristický fallback, ktorý sa ukázal byť výhodný pre zrýchlenie výpočtu, ale nevhodný pri použití v kombinácii s lokálnym prehľadávaním.

4.1 Upravený spôsob ukladania zoznamov najbližších miest

Zoznamy najbližších susedných miest sú kľúčovým vylepšením, ktoré sa nachádza v tejto práci. Detailnejšie sú popísané v sekcii 3.1.3. V tejto práci boli na ich implemetáciu použité C++ kontajnery, ktoré nahradili jednoduché polia z jazyka C. Jednou z výhod týchto objektov sú ich metódy, ktoré pracujú s obsiahnutými dátami.

Efektívnejšie uloženie najbližších susedných miest

Pôvodná implementácia zoznamov najbližších miest alokovala pevný počet jednotlivých štruktúr tvoriacich zoznam nabližších miest. V takto alokovaných štruktúrach boli uložené masky, v ktorých boli nastavené jednotlivé bity reprezentujúce susedné mestá. Avšak nie vždy všetky masky boli použité a to najmä pri veľkých inštanciách. V najhoršom prípade budú indexy susedných miest tak rozptýlené, že v každej použitej bitovej maske bude nastavený práve jeden bit. Na toto chovanie ale cieli predspracovanie inštancie, ktoré radí mestá podľa vzdialenosti. Preto sa bitové masky využívajú efektívnejšie a je ich treba menej na uloženie rovnakého počtu miest.

Novonavrhnutý spôsob uloženia zoznamov najbližších miest využíva možnosti C++ kontajnerov, napríklad možnosť použitia iterátorov pre sekvenčené iterovanie alebo dynamické zistenie veľkosti kontajneru. Prínosom tohto vylepšenia je ušetrenie miesta tým, že posledných x štruktúr od poslednej, ktorá obsahuje platnú masku dealokuje, resp. ani nikdy nealokuje. Príklad je možné vidieť v tabuľke 4.1, kde sa za posledným platným prvkom

nenachádzajú už žiadne ďalšie prvky s neplatnými maskami. Týmto spôsobom je možné skrátiť potrebný zoznam a tak ušetriť miesto v pamäti. Taktiež sa zníži aj počet opakovaní cyklov, ktoré pracujú s týmito zoznamami.

index mesta: 0					
ivec	-1	0	-1	1	2
maska	null	[mask]	null	[mask]	[mask]

Tabuľka 4.1: Príklad možného uloženia upraveného zoznamu najbližších miest pre mesto s indexom 0. Zoznam je kratší oproti pôvodnému o prvky s prázdnymi maskami od poslednej platnej masky, ktorá má hodnotu $ivec = 2$. Obsah masiek je bližšie popísaný v 3.1.3.

Uloženie feromónov, vzdialeností a váh prechodov

Pri zmene ukladania najbližších miest bola navrhnutá aj zmena ukladania feromónových stôp, vzdialeností a váh prechodov. Táto zmena bola motivovaná predovšetkým zlepšením pamäťovej lokality jednotlivých informácií. Predtým boli informácie asociované s daným prechodom uložené v spoločnej dátovej štruktúre s maskou buď ako ukazatele na dynamicky alokované polia, alebo ako staticky alokované polia. Ani jedna z možností ale nebola dostatočne efektívna a tak bol navrhnutý nový spôsob ich ukladania.

Jednotlivé informácie boli uložené do separátnych C++ kontajnerov, každá do vlastného. Jednotlivé kontajnery obsahujú ďalšie kontajnery pre jednotlivé mestá. V týchto sú uložené jednotlivé informácie do kontajnerov o dĺžke vektoru dostupného na konkrétnej architektúre. Takýmto uložením sa avšak stratila informácia, ktoré informácie patria ku ktorému mestu. Riešenie tejto situácie spočíva v zmene spôsobu akým sa priradzuje a interpretuje hodnota `ivec` zo štruktúry objektu najbližšieho susedného mesta. Pôvodný význam hodnoty `ivec` spočíval iba v tom, aby sa odlíšili NN objekty obsahujúce masku a tie ktoré masku neobsahujú. Po novom všetky hodnoty `ivec` rôzne od -1 tvoria jednoduchú postupnosť celých čísel s krokom 1.

Toto nové usporiadanie nevyžaduje ukladať informácie o feromónoch/vzdialenosti/váhe v objektoch tvoriacich zoznam najbližších miest a tak sa zmenší dátová štruktúra minimálne o ukazatele na tieto polia. Teoreticky je to minimálne $N * N/p$, kde N je počet miest v inštancii a p je dĺžka vektoru dostupného na procesore.

Táto postupnosť hodnôt sa následne používa napríklad vo VCSS, kde sa pri načítaní váh prechodov z pamäte ako indexi použijú aktuálne mesto a aktuálna hodnota `ivec`. Príklad ako by mohla vypadať vylepšená štruktúra obsahujúca dáta o feromónoch, vzdialenostiach alebo váhe prechodov je ilustrovaná v tabuľke 4.2.

4.2 Paralelná práca s feromónovými stopami

Nezanedbateľná časť výpočtu je venovaná nie len výberu následujúceho mesta, ale aj manipulácii s feromónovými stopami. Práca s feromónmi sa delí na dve základné fázy – inicializácia feromónov na ich maximálnu hodnotu na začiatku výpočtu a neustála aktualizácia počas výpočtu.

Index mesta	Uložené hodnoty					
0	[values]	[values]	[values]	null	null	...
1	[values]	[values]	[values]	null	null	...
2	[values]	[values]	[values]	[values]	null	...
...	...	...	...	...	...	...
N - 1	[values]	[values]	[values]	[values]	null	...

Tabuľka 4.2: Príklad možného uloženia dát v novonavrhnutom spôsobe uloženia feromónov, vzdialeností alebo váh prechodov. Informácie viažuce sa k jednotlivým mestám sú uložené v pamäti za sebou. Symbol `[values]` reprezentuje kontajner obsahujúci p hodnôt, kde p je šírka vektoru dostupného na použitej architektúre. N je celkový počet miest v inštancii.

Inicializácia feromónov na začiatku výpočtu

Pri inicializácii algoritmu sa vypočíta maximálna počiatočná hodnota feromónovej stopy pre všetky hrany. Táto hodnota je vypočítaná na základe jednoducho skonštruovanej cesty grafom a jej dĺžka je použitá pre výpočet maximálnej a minimálnej počiatočnej hodnoty feromónov. Pri inicializácii sú použité 2 cykly, z ktorého jeden je vnorený. Zatiaľ čo vonkajší cyklus iteruje po jednotlivých mestách, vnorený cyklus iteruje po jednotlivých kontajneroch, využívajúc vektorové inštrukcie na ukladanie niekoľkých dát zároveň, podľa dostupnej šírky vektoru procesora. Oba cykly sú paralelizované pomocou OpenMP direktív preprocesoru.

Práca s feromónmi počas výpočtu

Počas výpočtu sa práca s feromónmi delí na 4 základné časti:

- **Aktualizácia feromónov na základe najlepšieho riešenia** – cyklus iterujúci po najlepšej ceste a pripočítavajúci pevnú hodnotu d_τ (prírastok feromónovej stopy) ku každej hrane, ktorá tvorí danú cestu, bol paralelizovaný pomocou OpenMP.
- **Vyparovanie feromónov** – vyparovanie feromónov je realizované pomocou vektorových inštrukcií, ktoré naraz spracúvajú niekoľko hodnôt. Pre iteráciu po všetkých hranách sú využité dva cykly, z toho jeden vnorený a oba sú paralelizované pomocou OpenMP.
- **Kontrola hodnoty feromónových stôp** – v tejto fáze sa znovu prejdú všetky hrany a kontroluje sa, či sú feromónové stopy v rozmedzí hodnôt $< trail\_min, trail\_max >$. Táto kontrola je vykonávaná pomocou vektorových inštrukcií a naraz je porovnávaných niekoľko hodnôt s hodnotou $trail\_min$ a $trail\_max$. Taktiež boli využité vnorené cykly, ktoré boli paralelizované pomocou OpenMP.
- **Výpočet váh prechodov** – v poslednej fáze sa aktualizujú váhy prechodov medzi jednotlivými mestami. Táto operácia využíva 2 cykly na iteráciu po všetkých hranách. Vektorové inštrukcie sú použité na spracovanie niekoľkých hodnôt zároveň.

4.3 Vektorizovaný heuristický fallback

Heuristický fallback (bližšie popísaný v podsekcii sekcie 3.1.3) je jednou z dvoch možností ako nájsť nasledujúce mesto pri prázdnom zozname nasledujúcich miest. I keď tento fallback využíva namiesto výpočtovo náročnej euklidovskej vzdialenosti vzdialenosť štvorcovú, stále

je v tejto časti výpočtu priestor na teoretické zrýchlenie. Hlavným zdrojom zrýchlenia je využitie vektorových inštrukcií. Výsledkami experimentov sa zistilo, že modifikácia priniesla značné zrýchlenie výpočtu, no taktiež k výraznému zhoršeniu kvality výsledku. V kombinácii s lokálnym prehľadávaním je modifikácia pomalšia ako pôvodná nevektorizovaná implementácia, a tak sa napriek kvalitnejším výsledkom (za použitia lokálneho prehľadávania) zatiaľ neoplatilo vektorizovanú verziu používať. S ďalším výskumom a úpravami je ale možné dosiahnuté výsledky za použitia vektorizovaného heuristického fallback-u skvalitniť. Túto modifikáciu je možno vidieť v pseudo-kóde 2.

Algoritmus 2: Pseudo-kód vektorizovaného heuristického fallback-u.

Input : current city index **cc**, number of cities in instance **N**, list of all cities coordinates **nodes[N]**, masks of visited cities **v**, width of available vector **p**

Output: Next city to visit

for *i = 0* **to** *N/p* **do**
- x1 = ⟨ nodes[cc].x, ..., nodes[cc].x ⟩;
- x2 = ⟨ nodes [i · p].x, ..., nodes[i · p + p - 1].x ⟩;
- y1 = ⟨ nodes[cc].y, ..., nodes[cc].y ⟩;
- y2 = ⟨ nodes [i · p].y, ..., nodes[i · p + p - 1].y ⟩;
- indexes = ⟨ i · p, ..., i · p + p - 1 ⟩;
- x1 = x1 - x2;
- x1 = x1 · x1;
- y1 = y1 - y2;
- y1 = y1 · y1;
- result = y1 + x1;
- result = ApplyMask(⟨ -1, ..., -1 ⟩, result, v[i]);
- max_mask = result > running_max;
- running_max = ApplyMask(result, running_max, max_mask);
- indexes_max = ApplyMask(indexes, indexes_max, max_mask);

end

return *FindIndexOfMax(running_max, indexes_max)*;

Ako prvé sa inicializujú vektory $x1$ a $y1$, ktoré sa naplnia súradnicami mesta, v ktorom sa práve mravec nachádza. Vektory $x2$ a $y2$ obsahujú súradnice ostatných miest. Pomocou vektorových inštrukcií je vykonaný výpočet podľa vzorca

$$(x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2) \tag{4.1}$$

na štvorcové vzdialenosti jednotlivých bodov. Tento výpočet sa deje súčasne na oboch vektoroch. Následne je aplikovaná maska, ktorá vylúči z výsledku už navštívené mestá. Potom sa aktuálne vypočítané vzdialenosti porovnajú s doposiaľ najlepšími nájdenými. Z tohto porovnania vzíde maska `max_mask`, ktorá je následne použitá na aktualizáciu doposiaľ najlepších hodnôt. Po skončení cyklu a výpočte všetkých hodnôt je nájdené minimum spomedzi všetkých hodnôt vektoru `running_max` a index asociovaný s touto hodnotou je vrátený ako následujúce mesto.

Kapitola 5

Experimentálne výsledky

Táto kapitola je zameraná na testovanie upravenej implementácie MAX-MIN Ant systému a porovnanie výsledkov s pôvodnou verziou mravčích algoritmov od Doriga[1], ktorá je priložená na CD. Zároveň sa budú výsledky upravenej implementácie porovnávať s výsledkami z článku *Scaling Techniques for Parallel Ant Colony Optimization on Large Problem Instances* [16] a doposiaľ najlepšími nájdenými riešeniami[2] taktiež priloženými na CD. Testované budú rozsiahle inštancie problému obchodného cestujúceho (5 000 až 100 000 miest). Na záver sa výsledky zhodnotia súhrnne a vymenujú sa výhody/nevýhody daných modifikácií a prístupov. Názvy inštancií obsahujú číslo, ktoré reprezentuje počet miest v danej inštancii (napr. `pla7397` obsahuje 7 397 miest).

Výsledky naznačujú, že upravená verzia MAX-MIN Ant systému je vždy rýchlejšia, ako pôvodné algoritmy. Pôvodné mravčie algoritmy AS a MMAS sa v experimentoch nenachádzajú vôbec, a to z dôvodu ich príliš pomalého behu na väčších inštanciách. Na porovnanie upravenej implementácie MMAS bol teda vybraný pôvodný ACS, nakoľko je z pôvodných algoritmov najrýchlejší.

5.1 Experiment č. 1 – Porovnanie s ACS

V tomto experimente bola porovnávaná pôvodná implementácia ACS s upravenou implementáciou MMAS. Každá inštancia bola podrobená 10 behom, z ktorých bol spriemerovaný čas potrebný na výpočet 1 000 iterácií. Nastavenia parametrov boli následovné:

- alfa – 1,
- beta – 2,
- počet mravcov – 36,
- počet najbližších susedných miest – 32,
- miera vyparovania feromónov – 0.02,
- bez lokálneho prehľadávania (local-search = 0).

Ako je možné vidieť vo výsledkoch v tabuľke 5.1, upravená implementácia dosahuje značného zrýchlenia v porovnaní s pôvodou implementáciou ACS. Zrýchlenie sa líši aj medzi

[1] Dostupné na http://www.aco-metaheuristic.org/aco-code/public-software.html
[2] Dostupné na http://elib.zib.de/pub/mp-testdata/tsp/tsplib/stsp-sol.html

Inštancia	ACS[s]	BP HF[s]	Zrýchlenie	BP PMF[s]	Zrýchlenie
lin318	0,0019	0,0003	6,5338	0,0005	3,8297
pcb442	0,0029	0,0008	3,5960	0,0014	1,9986
rat783	0,0058	0,0007	7,7617	0,0017	3,3264
pr1002	0,0064	0,0010	6,7440	0,0023	2,7715
fl1577	0,0089	0,0016	5,5261	0,0038	2,3358
pr2392	0,0197	0,0024	8,2790	0,0075	2,6448
fl3795	0,0349	0,0039	8,8683	0,0142	2,4475
rl5934	0,0814	0,0077	10,5125	0,0263	3,0936
pla7397	0,0908	0,0183	4,9617	0,0487	1,8630

Tabuľka 5.1: Číslo v názve inštancie reprezentuje počet miest v danej inštancii. Stĺpec ACS vyjadruje čas potrebný na výpočet jednej iterácie neupraveným algoritmom ACS. Stĺpec BP HF vyjadruje čas potrebný na jednu iteráciu upraveným algoritmom MMAS (modifikáciami z 3.1.3, 3.2.2, 4.1 a 4.2) s heuristickou fallback metódou. Zrýchlenie udáva zaokrúhlene o koľko percent je upravené riešenie MMAS rýchlejšie ako pôvodná implementácia ACS. Stĺpec BP PMF používa ako fallback metódu feromónovú mapu.

jednotlivými fallback metódami. Heuristický fallback (HF) dosahuje väčšieho zrýchlenia ako fallback pomocou feromónovej mapy (PMF), pretože s každou novou hranou pridanou do feromónovej mapy sa predlžuje čas potrebný na získanie feromónovej stopy konkrétnej hrany.

5.2 Experiment č. 2 – Porovnanie s ACS s použitím lokálneho prehľadávania

Inštancia	ACS[s]	BP HF[s]	Zrýchlenie	BP PMF[s]	Zrýchlenie
lin318	0,0063	0,0043	1,4449	0,0006	10,1472
pcb442	0,0062	0,0014	4,3277	0,0010	6,1249
rat783	0,0127	0,0007	17,0027	0,0021	5,9995
pr1002	0,1808	0,0039	45,8934	0,0037	49,0825
fl1577	0,0186	0,0123	1,5028	0,0097	1,9225
pr2392	0,0418	0,0150	2,7835	0,0128	3,2686
fl3795	0,0515	0,0420	1,2262	0,0408	1,2632
rl5934	0,1367	0,0584	2,3388	0,0716	1,9089
pla7397	0,1223	0,1005	1,2169	0,1199	1,0196

Tabuľka 5.2: Číslo v názve inštancie reprezentuje počet miest v danej inštancii. Stĺpec ACS vyjadruje čas potrebný na výpočet jednej iterácie neupraveným algoritmom ACS s použitím lokálneho prehľadávania. Stĺpec BP HF vyjadruje čas potrebný na jednu iteráciu upraveným algoritmom MMAS s heuristickou fallback metódou a lokálnym prehľadávaním. Zrýchlenie udáva zaokrúhlene o koľko percent je upravené riešenie MMAS rýchlejšie ako pôvodná implementácia ACS. Stĺpec BP PMF používa ako fallback metódu feromónovú mapu spolu s lokálnym prehľadávaním.

V tomto experimente bola porovnávaná pôvodná implementácia ACS s upravenou implementáciou MMAS avšak u oboch algoritmov bolo spustené aj lokálne prehľadávanie. Každá inštancia bola podrobená 10 behom, z ktorých bol spriemerovaný čas potrebný na výpočet 1 000 iterácií. Nastavenia parametrov boli následovné:

- alfa – 1,
- beta – 2,
- počet mravcov – 36,
- počet najbližších susedných miest – 32,
- miera vyparovania feromónov – 0.02,
- s lokálnym prehľadávaním (local-search = 3).

Z tohto experimentu vyplynulo, že lokálne prehľadávanie je výrazným spomalením výpočtu aj napriek jeho paralelizácii presunutím do vlákna mravca. No napriek tomu je upravená implemetácia MMAS rýchlejšia ako pôvodné ACS. Avšak pri jeho použití boli sledované presnejšie výsledky ako bez jeho použitia, ako je ukázané v 5.3.

5.3 Experiment č. 3 – Porovnanie riešení nájdených v tejto práci s najlepšími známymi riešeniami daných TSP

Inštancia	**Najlepšie známe riešenie**	**HF LS=3**	**Rozdiel [%]**	**PMF LS=3**	**Rozdiel [%]**
lin318	42 029,00	42 479,00	1	44 079,00	5
pcb442	50 778,00	51 797,00	2	54 503,00	7
rat783	8 806,00	9 261,00	5	9 688,00	10
pr1002	259 045,00	278 462,00	7	298 035,00	15
fl1577	22 249,00	23 844,00	7	24 171,00	9
pr2392	378 032,00	423 778,00	12	445 989,00	18
fl3795	28 772,00	32 073,00	11	32 482,00	13
rl5934	556 050,00	622 429,00	12	639 433,00	15
pla7397	23 260 728,00	27 024 362,00	16	27 841 331,00	20

Tabuľka 5.3: Číslo v názve inštancie reprezentuje počet miest v danej inštancii. V tabuľke sú porovnané jednotlivé inštancie a ich doposiaľ najlepšie nájdené riešenia s riešeniami, ktoré našla upravená implementácia MMAS (s modifikáciami z 3.1.3, 3.2.2, 4.1 a 4.2). Rozdiel v riešeniach je vyjadrený v percentách o koľko je horšie nájedené riešenie oproti najlepšiemu známemu riešeniu. Najlepšie nájdené výsledky možno pozorovať pri použití heuristického fallbacku.

V prvej časti experimentu, ktorého výsledky sú v tabuľke 5.3, bola použitá pôvodná implementácia ACS a upravená implementácia MMAS s následujúcimi parametrami:

- alfa – 1,
- beta – 2,

- počet mravcov – 36,
- počet najbližších susedných miest – 32,
- miera vyparovania feromónov – 0.02,
- s lokálnym prehľadávaním (local-search = 3).

Ako je možné pozorovať z výsledkov, najmenšie rozdiely oproti najlepšiemu známemu riešeniu dosahuje heuristický fallback. Možno sledovať zvyšujúci sa trend rozdielu medzi najlepším nájdeným riešením v experimentoch a doposiaľ najlepším známym riešením spojený s rastúcim počtom miest v inštancii.

Inštancia	**Najlepšie známe riešenie**	**HF LS=0**	**Rozdiel [%]**	**PMF LS=0**	**Rozdiel [%]**
lin318	42 029,00	43 440,00	3	46 893,00	12
pcb442	50 778,00	58 453,00	15	53 835,00	06
rat783	8 806,00	9 482,00	8	10 302,00	17
pr1002	259 045,00	286 307,00	11	308 788,00	19
fl1577	22 249,00	24 802,00	11	26 483,00	19
pr2392	378 032,00	441 103,00	17	487 387,00	29
fl3795	28 772,00	34 336,00	19	39 159,00	36
rl5934	556 050,00	636 379,00	14	674 845,00	21
pla7397	23 260 728,00	27 579 037,00	19	30 547 100,00	31

Tabuľka 5.4: Číslo v názve inštancie reprezentuje počet miest v danej inštancii. V tabuľke sú porovnané jednotlivé inštancie a ich doposiaľ najlepšie nájdené riešenia s riešeniami, ktoré našla upravená implementácia MMAS (s modifikáciami z 3.1.3, 3.2.2, 4.1 a 4.2). Rozdiel v riešeniach je vyjadrený v percentách o koľko je horšie nájedené riešenie oproti najlepšiemu známemu riešeniu. Výsledky nájdené s použitím heuristického fallbacku (HF LS=0) sú kvalitnejšie, ako pri použití feromónovej mapy ako fallbacku (PMF LS=0).

V druhej časti experimentu, ktorého výsledky sú v tabuľke 5.4, bol spúšťaný rovnako ako predošlý experiment, avšak **bez lokálneho prehľadávania.** Vo výsledkoch je možné vidieť prínos lokálneho prehľadávania pre zvyšovanie kvality nájdeného riešenia. Opäť je možné pozorovať rozdiel medzi heuristickým fallbackom a feromónovou mapou týkajúci sa kvality riešenia.

5.4 Experiment č. 4 – Inštancia s 100 000 mestami

V tomto experimente bola použitá upravená implementácia MMAS algoritmu s modifikáciami z 3.1.3, 3.2.2, 4.1, 4.2 na vyriešenie inštancie `mona-lisa100k`, ktorá obsahuje 100 000 miest. Spustená bola s následujúcimi parametrami:

- alfa – 1,
- beta – 2,
- počet mravcov – 36,
- počet najbližších susedných miest – 32,

Riešenie	Rozdiel [%]	Čas [s]	Jedna iterácia [s]
6 806 548	18	2 133,64	2,13364
6 804 549	18	2 143,04	2,14304
6 807 890	18	2 131,58	2,13158
6 799 012	18	2 137,86	2,13786
6 907 598	19	2 133,99	2,13399
6 884 134	19	2 147,22	2,14722
6 808 843	18	2 128,09	2,12809
6 795 734	18	2 125,94	2,12594
6 881 188	19	2 136,35	2,13635
6 804 621	18	2 137,28	2,13728
6 876 794	19	2 132,48	2,13248

Tabuľka 5.5: Nájdené riešenia boli porovnávané s najlepším známym riešením, ktoré má hodnotu 5 757 191. Rozdiel v riešeniach je vyjadrený v percentách o koľko je horšie nájedené riešenie oproti najlepšiemu známemu riešeniu. Čas jednej iterácie bol spriemerovaný.

- miera vyparovania feromónov – 0,
- bez lokálneho prehľadávania (local-search = 0).

Výsledky 10 behov sú vidieť v tabuľke 5.5. Každý beh pozostával z 1 000 iterácií. Priemerne bolo potreba 2 135 sekúnd, čo je približne 36 minút. V priemere bolo nájdené riešenie 1,18-krát horšie, ako najlepšie známe riešenie. Odchýlka teda predstavuje 18%.

5.5 Experiment č. 5 – Inštancia s 200 000 mestami

V tomto experimente bola počítaná doposiaľ najrozsiahlejšia inštancia tejto práce, a to s počtom až 200 000 miest. Pre výpočet bol použitý upravený MMAS algoritmus s modifikáciami z kapitol 3.1.3, 3.2.2, 4.1 a 4.2. Výpočet prebiehal na clusteri Barbora, ktorý patrí pod IT4Innovations a poskytuje Intel procesory s inštrukčným rozšírením `AVX-512`. Vďaka udelenému prístupu na superpočítač bolo možné vypočítať v rozumnom čase aj takto rozsiahlu inštanciu.

Riešenie bolo nájdené za 8 154,64 sekúnd, čo predstavuje 2,265 hodiny. Za tento čas prebehlo 1 000 iterácií a výsledná cesta mala dĺžku 9 700 478. Tento výsledok sa od najlepšieho známeho riešenia líši iba o 18%, čo môže byť brané ako uspokojujúci výsledok, vzhľadom na veľkosť inštancie a čas výpočtu.

5.6 Experiment č. 6 – Úspešnosť a výpočtový čas pri rôznom počte mravcov

V tomto experimente bola použitá upravená implementácia MMAS algoritmu na vyriešenie inštancie `pla7397`, ktorá obsahuje 7 397 miest. Spustená bola s následujúcimi parametrami:

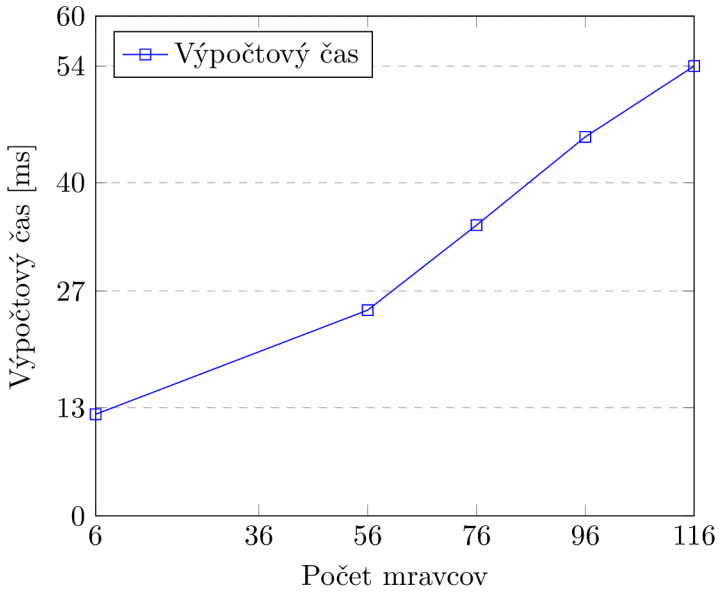
- alfa – 1,
- beta – 2,

Počet mravcov	Chyba riešenia [%]	Trvanie jednej iterácie [ms]
6	19	12,2
56	18	27,4
76	18	34,9
96	19	45,5
116	18	54,0

Tabuľka 5.6: Výsledky inštancie `pla7397` pri rôznych počtoch mravcov.Chyba v riešeniach je vyjadrená v percentách o koľko je horšie nájedené riešenie oproti najlepšiemu známemu riešeniu. Trvanie jednej iterácie bolo spriemerované z 10 behov, kde každý beh obsahoval 1 000 iterácií.

- počet mravcov – 6, 56, 76, 96, 116,
- počet najbližších susedných miest – 32,
- miera vyparovania feromónov – 0,02,
- bez lokálneho prehľadávania (local-search = 0).

Výsledky 10 behov sú vidieť v tabuľke 5.6. Prehľadné znázornenie výpočtového času možno vidieť v grafe 5.1. Každý beh pozostával z 1 000 iterácií. Priemerná doba na nájdenie výsledku narastala spolu s narastajúcim počtom mravcov. Narozdiel od domnienky, že viac mravcov urýchli a skvalitní riešenie, odchýlka od najlepšieho známeho riešenia sa v zásade skoro nemení. Z tohto experimentu vyplýva, že najefektívnejšie bude používať menší počet mravcov.

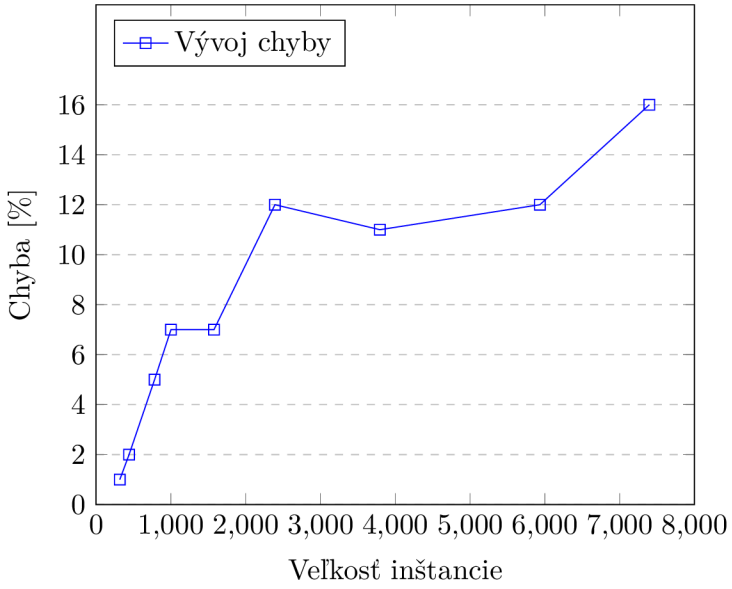


Obr. 5.1: Graf zobrazujúci vývoj chyby vo vzťahu k veľkosti inštancie. Os x zobrazuje počet miest v inštancii. Os y predstavuje percentuálnu odchýlku od najlepšieho známeho riešenia. Inštancia so 100 000 mestami mala chybu 18%, no z dôvodu veľkej priepasti medzi počtom miest do grafu nebola pridaná.

5.7 Diskusia k výsledkom

Táto kapitola sa venovala experimentovaniu s upraveným systémom MMAS a porovnávaniu výsledkov s neupravenou verziou ACS a doposiaľ najlepšími nájdenými výsledkami.

Z výsledkov je možné odovodiť vzťah medzi veľkostou inštancie a dosahovanou kvalitou riešenia. S rastúcim počtom miest rastie aj rozdiel nájdeného riešenia voči optimálnemu riešeniu. Tento vývoj možno sledovať na grafe 5.2. V najlepších prípadoch na najmenších inštanciách bol rozdiel v nájdenom riešením a najlepšom známom riešení v nízkych rádoch jednotiek percent. V najextrémenjších prípadoch dosahovala chyba až 36% pri použití feromónovej mapy ako fallbacku a pri absencii lokálneho prehľadávania na inštancii `fl3795` (experiment 5.3). S použitím heuristického fallbacku a lokálneho prehľadávania bolo možné v danej inštancii chybu zmenšiť na 11%. Pri inštancii `lin318`, ktorá obsahuje 318 miest chyba riešenia dosahuje len 1%, naopak pri inštancii `pla7397`, ktorá obsahuje 7 397 miest dosahovala najnižšia chyba 16%.



Obr. 5.2: Graf zobrazujúci vývoj chyby vo vzťahu k veľkosti inštancie. Os x zobrazuje počet miest v inštancii. Os y predstavuje percentuálnu odchýlku od najlepšieho známeho riešenia. Inštancia so 100 000 mestami mala chybu 18%, no z dôvodu veľkej priepasti medzi počtom miest do grafu nebola pridaná.

Pri zhodnotení časových výsledkov možno usúdiť, že upravený algoritmus MMAS je rýchlejší ako pôvodná implementácia ACS. Pri počítaní s prihliadnutím na čas je vždy rýchlejší heuristický fallback ako fallback s použitím feromónovej mapy. Motiváciou pre používanie feromónovej mapy vo fallback-u bola domnienka, že malý počet feromónových stôp v algoritme spôsobí jeho horšie výsledky. Avšak, ako dokázali experimenty, algoritmus aj s obmedzenou feromónovou maticou bol schopný dosahovať rovnaké, niekedy aj lepšie výsledky ako pri použití feromónovej mapy vo fallback-u. Zároveň réžia spojená so získavaním dát z feromónovej mapy spôsobila celkové spomalenie výpočtu.

Heuristický fallback (HF), ako už bolo spomenuté, je rýchlejší ako fallback s pomocou feromónovej mapy. To je spôsobené v prvom rade réžiou súvisiacou s vyhľadávaním hrán v mape. Ďalšou výhodou HF je používanie jednoduchej štvorcovej vzdialenosti medzi

mestami. Táto vzdialenosť je počítaná pre každé jedno nenavštívené mesto aby sa našlo najbližšie mesto. Na druhej strane, feromónový fallback používa pre meranie vzdialenosti medzi mestami výpočtovo náročnejšiu euklidovskú vzdialenosť. Navyše počíta váhu prechodu podľa vzorca 2.3, čo je dosť náročné na procesorový čas.

S narastajúcim počtom mravcov rastie aj výpočtový čas čo je možné vidieť v experimente 5.6. Podľa zistených výsledkov sa riadili ostatné experimenty a preto používali menšie množstvo mravcov pre výpočet, keďže väčší počet mravcov neprispel k skvalitneniu výsledku.

Celkové hodnotenie vylepšenej implementácie v súvislosti s rozsiahlymi inštanciami možno odvodiť z experimentu 5.4, v ktorom sa úspešnosť výsledku (v porovnaní s najlepším známym riešením) v inštancii so 100 000 mestami pohybovala okolo 83%. Priemerný čas na výpočet tejto rozsiahlej inštancie bol pritom v priemere iba 36 minút. V podobnom duchu bol realizovaný aj experiment s inštanciou o rozsahu 200 000 miest. Výsledok tohto pokusu je riešenie horšie len o 18% oproti najlepšiemu známemu riešeniu. Celý tento výpočet trval približne 2,3 hodiny. Riešenie týchto inštancií s použitím pôvodných neupravených algoritmov neexistuje.

Kapitola 6

Záver

Hlavným cieľom tejto práce bolo vytvoriť systém schopný rýchlo a efektívne riešiť rozsiahle úlohy TSP bez výraznej straty na kvalite nájdeného riešenia. Prvou úlohou bolo implementovať vylepšenia z článku *Scaling Techniques for Parallel Ant Colony Optimization on Large Problem Instances* [16], čo bolo aj úspešne splnené. Počas realizácie práce boli preštudované rôzne metódy a modifikácie prispievajúce k optimalizácii mravčích algoritmov. Následne boli navrhnuté vlastné vylepšenia, ktoré znížili pamäťovú náročnosť a tým aj mierne znížili výpočtový čas. Navrhnuté vylepšenie – vektorizácia heuristického fallbacku v konečnom dôsledku nedosahovala želané výsledky a tak bola z implementácie odstránená a zostala iba v teoretickej rovine ako pseudo-kód.

Ďalším cieľom tejto práce bol výskum a experimentovanie s upravenou implementáciou mravčieho algoritmu MMAS – overenie, či upravená verzia dosahuje lepší výpočtový čas a je pamäťovo menej náročná. Výsledky boli následne porovnávané s pôvodnou implementáciou algoritmu ACS, nakoľko ostatné mravčie algoritmy sú výrazne pomalšie a ich výpočtový čas sa pri riešení rozsiahlych úloh TSP predpokladane pohybuje v desiatkach hodín napriek využitiu výkonného hardvéru. Ako dokazujú výsledky kapitoly 5, upravená implementácia MMAS je vo všetkých prípadoch značne rýchlejšia ako pôvodné ACS. Rozdiel riešenia oproti najlepším známym výsledkom sa pri menších inštanciách (približne do 2 000 miest) pohybovala okolo 5%. S narastajúcim počtom miest v inštancii rástla aj odchýlka od doposiaľ najlepšieho nájdeného riešenia, pohybujúca sa okolo 10% až 15%, v extrémnych prípadoch až 35%, a to pri nevhodnom nastavení parametrov a fallback metódy. Pamäťová náročnosť bola znížená implementáciou vylepšení zo sekcie 4.2 o upravenom spôsobe ukladania zoznamu najbližších miest a sekcie 3.2.2 o obmedzenej feromónovej matici.

Celkovo sa vylepšená implementácia MMAS ukázala ako prínosná pre riešenie rozsiahlych úloh TSP. V rámci rozšírení tejto práce a možnej následnej práce by bolo možné vytvoriť grafické užívateľské rozhranie na výukové účely. Ďalej by stálo za pozornosť preskúmať možskúmala by snosti vektorizácie heuristického fallback-u. Navrhnutý pseudo-kód 2 tejto modifikácie by mohol byť zavedený do implementácie a skúmala by sa jeho efektivita. Nakoľko upravená implementácia obsahuje odchýlky od najlepšieho možného riešenia, možné ďalšie úpravy a vylepšenia by prispeli k skvalitneniu dosahovaných výsledkov.

Literatúra

[1] Bidlo, M. *Mravenčí algoritmy: Prezentácia z predmetu Aplikované evoluční algoritmy (EVO)*. Božetěchova 1/2, 612 00 Brno-Královo Pole: FIT VUT v Brně, 2020.

[2] Cecilia, J. M., García, J. M., Nisbet, A., Amos, M. a Ujaldón, M. Enhancing data parallelism for Ant Colony Optimization on GPUs. *Journal of Parallel and Distributed Computing*. 2013, zv. 73, č. 1, s. 42 – 51. DOI: https://doi.org/10.1016/j.jpdc.2012.01.002. ISSN 0743-7315. Metaheuristics on GPUs. Dostupné z: http://www.sciencedirect.com/science/article/pii/S0743731512000032.

[3] Chitty, D. Applying ACO To Large Scale TSP Instances. *ArXiv.org*. Ithaca: Cornell University Library, arXiv.org. 2017, zv. 650. ISSN 21945357. Dostupné z: http://search.proquest.com/docview/2076629970/.

[4] Deneubourg, J.-L., Aron, S., Goss, S. a Pasteels, J. The Self-Organizing Exploratory Pattern of the Argentine Ant. *J. Insect Behav.* Marec 1990, zv. 3, s. 159. DOI: 10.1007/BF01417909.

[5] Dorigo, M. *Optimization, Learning and Natural Algorithms*. Milano, 1992. Dizertačná práca. Politecnico di Milano. Dostupné z: https://ci.nii.ac.jp/naid/10000136323/en/.

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

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

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

[9] Engelbrecht, A. P. *Computational Intelligence: an introduction*. Chichester: John Wiley & Sons, 2008. ISBN 978-0-470-03561-0.

[10] Gambardella, L. M. a Dorigo, M. Ant-Q: A Reinforcement Learning approach to the traveling salesman problem. In: Prieditis, A. a Russell, S., ed. *Machine Learning Proceedings 1995*. San Francisco (CA): Morgan Kaufmann, 1995, s. 252 – 260. ISBN 978-1-55860-377-6. Dostupné z: http://www.sciencedirect.com/science/article/pii/B9781558603776500396.

[11] Guntsch, M. a Middendorf, M. A population based approach for ACO. In:. 2002, sv. 2279, s. 72–81. ISBN 3540434321.

[12] Lloyd, H. a Amos, M. A highly parallelized and vectorized implementation of Max-Min Ant System on Intel® Xeon Phi™. In: *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. Dec 2016, s. 1–6. DOI: 10.1109/SSCI.2016.7850085.

[13] Lloyd, H. a Amos, M. Analysis of Independent Roulette Selection in Parallel Ant Colony Optimization. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. New York, NY, USA: Association for Computing Machinery, 2017, s. 19–26. GECCO '17. DOI: 10.1145/3071178.3071308. ISBN 9781450349208. Dostupné z: https://doi.org/10.1145/3071178.3071308.

[14] Neumann, F. a Witt, C. *Bioinspired computation in combinatorial optimization: algorithms and their computational complexity*. Springer, 2010. ISBN 978-3-642-16543-6.

[15] Peake, J., Amos, M., Yiapanis, P. a Lloyd, H. Vectorized Candidate Set Selection for Parallel Ant Colony Optimization. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. New York, NY, USA: Association for Computing Machinery, 2018, s. 1300–1306. GECCO '18. DOI: 10.1145/3205651.3208274. ISBN 9781450357647. Dostupné z: https://doi.org/10.1145/3205651.3208274.

[16] Peake, J., Amos, M., Yiapanis, P. a Lloyd, H. Scaling Techniques for Parallel Ant Colony Optimization on Large Problem Instances. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. New York, NY, USA: Association for Computing Machinery, 2019, s. 47–54. GECCO '19. ISBN 9781450361118. Dostupné z: https://doi.org/10.1145/3321707.3321832.

[17] Pourtakdoust, S. H. a Nobahari, H. An Extension of Ant Colony System to Continuous Optimization Problems. In: *Ant Colony Optimization and Swarm Intelligence*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, s. 294–301. ISBN 978-3-540-28646-2.

[18] Stützle, T. a Hoos, H. MAX-MIN Ant System. *Future Generation Computer Systems-The International Journal Of Escience*. ELSEVIER SCIENCE BV. 2000, zv. 16, č. 8, s. 889–914. ISSN 0167-739X.

[19] Stützle, T. a Hoos, H. Improving the Ant System: A Detailed Report on the MAX-MIN Ant System. Január 1999.

Príloha A

Návod k použitiu experimentálneho SW

Preklad programu je možný pomocou prekladača `Intel C++ Compiler 19.1.0.166 20191121` a programu `GNU Make 3.82`. Pre beh programu je potrebný procesor podporujúci inštrukčnú sadu `AVX-512` (testované na procesore Intel Cascade Lake 6240, 18-core, 2.6 GHz).

Vstupné parametre programu

Vstupné parametre programu sa zadávajú do hlavičkového súrobu param.h. Prednastavené hodnoty sú odporúčané literatúrou.

- `#define QUIET 1` - parameter určujúci množstvo výpisov
- `#define MMAS TRUE` - tento parameter sa nemení, definuje použitie MMAS algoritmu
- `#define N_ANTS 36` - počet mravcov, ktorý budú hľadať riešenie
- `#define TRIES 1` - počet opakovaných spustení algoritmu
- `#define ANT_FILE „usa13509.tsp"` - cesta k súboru obsahujúcemu inštanciu TSP, ktorá bude riešená po spustení algoritmu
- `#define LS_FLAG 3` - úroveň lokálneho prehľadávania (local-search algoritmu), je definovaná buď ako LS_FLAG 0, čo znamená žiadne lokálne prehľadávanie alebo LS_FLAG 3 čo znamená aktivované lokálne prehľadávanie
- `#define NN_ANTS 32` - dĺžka zoznamu najbližších miest
- `#define MAX_TIME_ANTS 10000` - maximálny čas behu programu
- `#define ITERATIONS 1000` - maximálny počet iterácií, po ktorých program skončí
- `#define DIMENSION 13509` - počet miest v inštancii zadanej v parametri ANT_FILE
- `#define APLHA 1` - hodnota ALPHA, ktorá určuje dôležitosť feromónovej stopy pri rozhodovaní
- `#define BETA 2` - hodnota BETA, ktorá určuje viditeľnosť mesta

- `#define RHO 0.02f` - koeficiet RHO, určujúci vyparovanie feromónových stôp
- `#define PMF` - ak je definované toto makro, použije sa metóda Pheromone Map Fallback na výber ďalšieho mesta pri prázdnom zozname susedných miest inak je použitá metóda Heuristic Fallback

Práca s programom

Príkaz odstráni všetky objektové súbory a spustiteľný program

```
$ make clean
```

Po rozbalení je program po preklade pripravený na beh s inštanciou definovanou v súbore param.h. Spustenie je bez parametrov.

```
$ make
$ ./acotsp
```

Ukážkové spustenie programu

Ukážkové spustenie programu pre potreby bakalárskej práce je možné po zakomentovaní nasledujúcich riadkov v súbore `param.h`

```
//#define ANT_FILE üsa13509.tsp"
//#define LS_FLAG 3
//#define DIMENSION 13509
```

a následným spustením skriptu `show.sh`, ktorý spustí ukážku behu na inštancii `lin318.tsp` s rôznymi nastaveniami.

Príloha B

Obsah priloženého CD

V priloženom CD sa nachádza:

- `xramos00/technicka_sprava.pdf` Text bakalárskej práce.
- `xramos00/README.txt` Popis programu.
- `xramos00/tours.txt` Súbor s doposiaľ najlepšími nájdenými výsledkami rôznych inštancií.
- `xramos00/source_files/ACOTSP-1.03/` Pôvodná implementácia ACO algoritmov.
- `xramos00/source_files/BP-ACOTSP/` Upravená implementácia MAX-MIN AS.
- `xramos00/BPlatex` Zdrojové súbory k textu bakalárskej práce.