Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra systémového inženýrství

Diplomová práce

Mravenčí kolonie a další heuristiky pro řešení problému obchodního cestujícího

Bc. Kristýna Bartošová

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Kristýna Bartošová

Informatika

Název práce

Mravenčí kolonie a další heuristiky pro řešení problému obchodního cestujícího

Název anglicky

Ant colonies and other heuristics for solving the traveling salesman problem

---

Cíle práce

Cílem práce je zhodnocení vybraných metod pro řešení těžkého optimalizačního problému známého jako problém obchodního cestujícího. Vybrané metody, kterými jsou mravenčí kolonie, heuristiky z oblasti teorie grafů a exaktní algoritmus, budou zkoumány z hlediska časové náročnosti výpočtu a délky nalezené trasy pomocí vlastního programu s implementací těchto algoritmů. Úloha by měla být dostatečně velká, aby byly znatelné rozdíly mezi metodami, ale zároveň by měla být zahrnuta i úloha, kde bude možné použít i exaktní algoritmus.

Metodika

Teoretická část diplomové práce bude založená na studiu odborných informačních zdrojů na téma problému obchodního cestujícího se zaměřením na vybrané algoritmy a témata s tím související jako je například inteligence hejna a teorie grafů, což bude sloužit jako východisko pro zpracování praktické části práce.

V praktické části bude pro zhodnocení vybraných metod na řešení problému obchodního cestujícího naimplementován vlastní program v jazyce C#, kde bude možné na různých úlohách spustit vybrané metody a spočítat čas výpočtu a délku nalezené trasy. V programu budou implementovány mravenčí kolonie, metoda nejbližšího souseda, hladový algoritmus, zlepšující Lin–Kernighanova metoda 2-opt a dále také exaktní algoritmus. Úlohou, na které budou algoritmy testovány bude okružní cesta přes všechna hlavní města v Evropě, kterých je 50, dále bude vybrána i vhodná úloha, kde bude možné v reálném čase dokončit exaktní algoritmus, aby bylo možné tento algoritmus lépe srovnat s ostatními.

Doporučený rozsah práce

60-80 stran

Klíčová slova

Problém obchodního cestujícího, Mravenčí kolonie, Inteligence hejna, Teorie grafů, Heuristiky, Metoda nejbližšího souseda, Hladový algoritmus, Lin–Kernighanova metoda, Optimalizace okružní trasy

---

Doporučené zdroje informací

COOK, William. *Po stopách obchodního cestujícího : matematika na hranicích možností*. Praha: Dokořán, 2012. ISBN 978-80-7363-412-4.

DORIGO, Marco a Thomas STÜTZLE. Ant Colony Optimization. Cambridge: MIT Press, 2004. ISBN 9780262042192;0262042193;

YING, Tan. Swarm Intelligence, Volume 1 – Principles, Current Algorithms and Methods. Institution of Engineering and Technology, 2018. ISBN 1785616277;9781785616273;

Předběžný termín obhajoby

2023/24 LS – PEF

Vedoucí práce

RNDr. Petr Kučera, Ph.D.

Garantující pracoviště

Katedra systémového inženýrství

Elektronicky schváleno dne 23. 11. 2023

doc. Ing. Tomáš Šubrt, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 23. 11. 2023

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 31. 03. 2024

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Mravenčí kolonie a další heuristiky pro řešení problému obchodního cestujícího" jsem vypracovala samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autorka uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 31.03.2024 _____

Poděkování

Ráda bych touto cestou poděkovala RNDr. Petru Kučerovi, PhD. za odborné vedení práce včetně poskytnutí cenných rad a zkušeností a Matějovi Uchytilovi za zkušenosti a podporu v celém průběhu tvorby práce i studia.

Mravenčí kolonie a další heuristiky pro řešení problému obchodního cestujícího

Abstrakt

Práce je zaměřená na porovnání algoritmů pro řešení problému obchodního cestujícího z hlediska optimality nalezeného řešení a výpočetního času. Zkoumán je exaktní algoritmus a heuristické metody, kterými jsou optimalizace mravenčími koloniemi, metoda nejbližšího souseda, hladový algoritmus a metoda zlepšující řešení 2-opt. Algoritmy jsou implementovány ve webové aplikaci a odzkoušeny na testovacích příkladech, kterými jsou čtyři varianty cesty po hlavních městech padesáti států Evropy. V práci je popsána výpočetní složitost, teorie grafů, inteligence hejna s důrazem na mravenčí kolonie a jednotlivé metody pro řešení problému obchodního cestujícího, zejména ty použité ve vlastním programu.

Klíčová slova: Problém obchodního cestujícího, Mravenčí kolonie, Inteligence hejna, Výpočetní složitost, Teorie grafů, Heuristiky, Metoda nejbližšího souseda, Hladový algoritmus, Lin–Kernighanova metoda, Optimalizace okružní trasy

Ant colonies and other heuristics for solving the traveling salesman problem

Abstract

This work focuses on comparing algorithms for solving the traveling salesman problem in terms of the optimality of the solution and computational time. The exact algorithm and heuristic methods such as the ant colony optimization, the nearest neighbor method, the greedy algorithm and the 2-opt method, which improves the solution, are investigated. The algorithms are implemented in a web application and tested on examples, which are four variants of travel across fifty European capital cities. Computational complexity, graph theory, swarm intelligence with emphasis on ant colonies, and the different methods for solving the traveling salesman problem, especially those used in the actual program, are described.

Keywords: Traveling salesman problem, Ant colony optimization, Swarm intelligence, Computational complexity, Graph theory, Heuristics methods, Nearest neighbor method, Greedy algorithm, Lin-Kernighan method

Obsah

1 Úvod

Problém obchodního cestujícího má své kouzlo v jednoduchosti definice problému a naproti tomu ve složitosti problém vyřešit. Byl již zkoumán matematiky po dlouhou dobu a stále není známé žádné efektivní řešení zaručující přesné výsledky v polynomiálním čase výpočtu.

Problém obchodního cestujícího je velmi často používán jako příklad NP-úplných úloh a patří k nejslavnějším optimalizačním úlohám. Lze ho popsat následujícím způsobem: Nalezněte nejkratší cestu pro navštívení všech měst na daném seznamu s tím, že každé město smíte navštívit pouze jednou a musíte se vrátit zpět tam, kde jste začínali. Cílem je tedy vytvořit okruh přes všechna města tak, aby cesta byla co nejkratší.

Ačkoli samotná formulace problému zní jednoduše, výpočet už tak jednoduchý není. Pokud například chcete vyjet z jednoho města a navštívit 20 dalších měst, tak možností, jaký okruh zvolíte, je $20!/2$, to je přesně 1 216 451 004 088 320 000 možných okruhů, ze kterých potřebujete vybrat ten nejkratší. Udělat přesný výpočet tak velké úlohy je náročné nejen pro člověka, ale i pro stroj. Největší úskalí tkví v tom, že s každým dalším městem, které chcete navštívit, počet přípustných řešení velmi rychle roste a to přímo faktoriálovou rychlostí. Doba potřebná k výpočtu se tedy stává neúnosná už při řádově desítkách měst. (Cook, 2012)

Taková úloha je často řešena například v logistice, ale vyřešením problému obchodního cestujícího by mělo mnohem dalekosáhlejší dopad než jen na okruh oborů, kde je tato úloha přímo řešena. Když bychom vyřešili jednu NP-úplnou úlohu, tak bychom tím našli řešení i na další takové úlohy, jako je například prolomení klíčů, které slouží pro bezpečnost komunikace přes internet.

Zdali je možné najít řešení problému obchodního cestujícího jiným způsobem než prověřením všech možných okruhů, zatím není známo. Jsou ale známé algoritmy, které fungují v přijatelném čase, nicméně k optimálnímu řešení se jen přibližují. Tyto algoritmy mohou být inspirované samotnou přírodou, například mravenci hledající nejkratší cestu při přenosu jídla do mraveniště. Inspiraci pro hledání takových algoritmů je ale možné najít i v teorii grafů, neuronových sítích a dalších odvětvích.

Součástí této práce je vytvoření programu, ve kterém budou implementovány algoritmy přibližující se řešení, které jsou inspirované chováním mravenců a teorií grafů.

Je vytvořen i algoritmus, který projde všechna řešení. Jako vzorová úloha, na které budou jednotlivá řešení testována, je cesta po hlavních městech padesáti států Evropy.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem práce je zhodnocení vybraných metod pro řešení těžkého optimalizačního problému známého jako problém obchodního cestujícího. Vybrané metody, kterými jsou mravenčí kolonie, heuristiky z oblasti teorie grafů a exaktní algoritmus, budou zkoumány z hlediska časové náročnosti výpočtu a délky nalezené trasy pomocí vlastního programu s implementací těchto algoritmů. Úloha by měla být dostatečně velká, aby byly znatelné rozdíly mezi metodami, ale zároveň by měla být zahrnuta i úloha, kde bude možné použít i exaktní algoritmus.

2.2 Metodika

Diplomová práce navazuje na bakalářskou práci zpracovanou na téma porovnání heuristik problému obchodního cestujícího a optimalizace tras, kterou dále rozšiřuje o optimalizaci mravenčími koloniemi a exaktní algoritmus.

Teoretická část diplomové práce bude založená na studiu odborných informačních zdrojů na téma problému obchodního cestujícího se zaměřením na vybrané algoritmy a témata s tím související, jako je například inteligence hejna, výpočetní složitost a teorie grafů, což bude sloužit jako východisko pro zpracování praktické části práce.

V praktické části bude upraven program naimplementovaný v rámci bakalářské práce, na kterou tato práce navazuje. Program je naimplementovaný v jazyce C# a umožnuje na různých úlohách spustit vybrané metody a spočítat čas výpočtu a délku nalezené trasy. V programu budou nově implementovány mravenčí kolonie a exaktní algoritmus. Výsledky budou srovnány i s metodou nejbližšího souseda, hladovým algoritmem a zlepšující Lin-Kernighanovou metodou 2-opt. Úlohou, na které budou algoritmy testovány, bude okružní cesta přes všechna hlavní města v Evropě, kterých je 50. Dále bude vybrána i vhodná úloha, kde bude možné v reálném čase dokončit exaktní algoritmus, aby bylo možné tento algoritmus lépe srovnat s ostatními.

3 Teoretická východiska

Problém obchodního cestujícího se dotýká různých oblastí, jako je výpočetní složitost, teorie grafů, heuristické algoritmy a pro řešení pomocí mravenčích kolonií i například teorie hejna. (Cook, 2012)

3.1 Problém obchodního cestujícího

Problém obchodního cestujícího je optimalizační problém, kde cestující potřebuje projít přes všechna města právě jednou tak, aby se vrátil do výchozího města a aby celková délka cesty byla co nejkratší. Proslavil se zejména pro svou jednoduchou, lehce pochopitelnou definici, ale naproti tomu složitému výpočtu. (Cook, 2012)

Tento problém lze formálně popsat pomocí teorie grafů jako hledání hamiltonovské kružnice v úplném neorientovaném grafu s kladně ohodnocenými hranami. Tato kružnice musí obsahovat všechna města a součet cen hran musí být co nejmenší. Počet měst musí být větší než tři. (Cook, 2012)

3.1.1 Uplatnění

Jedná se o typický problém, který je vysoce relevantní v širokém okruhu praktických úloh. Například v oblasti plánování počítačových sítí může problém obchodního cestujícího pomoci optimalizovat trasování datových paketů, minimalizovat zpoždění a maximalizovat propustnost sítě. Umístění senzorů v průmyslových prostředích často vyžaduje efektivní pokrytí dané oblasti, což lze dosáhnout pomocí algoritmů založených na problému obchodního cestujícího. (Du, Liu, Zhang, & Lu, 2021) (Lawler, Lenstra, Rinnooy, & B., 1991)

V oblasti logistiky a distribuce může být tento problém využit k plánování optimálních tras dodávek, snižování nákladů na dopravu a zkracování času doručení zboží. Při vrtání a pájení desek s plošnými spoji může být použit k minimalizaci pohybu vrtáku nebo pájky, což vede k efektivnější výrobě. Inteligentní dopravní systémy mohou využít algoritmy řešící problém obchodního cestujícího k optimalizaci tras a časů příjezdu vozidel. Problém obchodního cestujícího má také uplatnění ve směřování teleskopů, rentgenových paprsků a laserů v oblastech jako astronomie, lékařství a průmysl. (Du, Liu, Zhang, & Lu, 2021) (Lawler, Lenstra, Rinnooy, & B., 1991)

3.2 Výpočetní složitost

Složitostí úlohy se obvykle myslí nároky na čas a paměť při výpočtu algoritmu pomocí počítače. Algoritmus je přesný návod či postup, kterým lze vyřešit daný typ úlohy. Výpočetní složitost se dělí na časovou a paměťovou složitost, které se říká i prostorová složitost. Paměťová složitost vyjadřuje počet bitů potřebných k uložení dat během výpočtu. V dnešní době už paměťová složitost nehraje tak významnou roli, protože počítače mají obvykle paměť dostatečnou. Na časovou složitost je kladený důraz vzhledem ke zvyšujícím se nárokům na rychlou odezvu programu. (Savický, 2016)

Ve výpočetní složitosti tedy nejsou brána v úvahu konkrétní softwarové ani hardwarové zařízení, pomocí nichž může být algoritmus spouštěn. Kromě zkoumání složitosti algoritmu jako takového, můžeme zkoumat i složitost té dané úlohy, čímž si vytvoříme odhad složitosti algoritmu, který je pro danou úlohu nejlepší. Složitost úlohy je pak složitost nejlepšího algoritmu, který úlohu řeší. (Savický, 2016)

3.2.1 Asymptotická časová složitost

Časová složitost se měří počtem kroků algoritmu. Platí, že s rostoucím počtem kroků algoritmu roste i čas, který počítač potřebuje k výpočtu. Zároveň platí, že čím výkonnější počítač je, tím kratší dobu výpočet potrvá. Výpočetní složitost má ale obvykle na čas výpočtu podstatně větší vliv než výkon počítače, a to obzvláště při větších vstupech, protože u složitých algoritmů se může doba výpočtu prodlužovat exponenciálně. (Tesař & Mareš)

Častým způsobem, jak klasifikovat složitost algoritmů, je asymptotická časová složitost. Pro klasifikaci algoritmu zjišťujeme, jak bude růst počet kroků algoritmu v závislosti na velikosti vstupních dat. Horní asymptotická složitost je složitost v nejhorším možném případě, zapisuje se pomocí Landauovy notace funkcí jako $O(f(n))$, kde n je velikost vstupu. Zajímá nás hlavně, jak se složitost projevuje na větších vstupech, protože na malých vstupech je rychlý téměř každý algoritmus. (Tesař & Mareš)

Příklady nejčastější klasifikace:

- $O(1)$ – konstantní
- $O(n)$ – lineární
- $O(\log_2 n)$ – logaritmická
- $O(n \log_2 n)$ – lineárně-logaritmická
- $O(n^2)$ – kvadratická
- $O(n^3)$ – kubická
- $O(k^n)$ – exponenciální, pro $k > 0$, dosud nejlepší nalezený exaktní algoritmus pro řešení problému obchodního cestujícího běží v čase $O(n^2 2^n)$, n zde představuje počet vrcholů grafu neboli měst. (Cook, 2012)
- $O(n!)$ – faktoriálová, například problém obchodního cestujícího řešený hrubou silou, který má složitost $O((n-1)!/2)$. (Cook, 2012)

Mezi polynomiální charakteristiky patří například lineární, logaritmické a kvadratické, u těchto úloh můžeme případně dobu výpočtu zrychlit výkonnějším počítačem. U nepolynomiálních úloh, jako jsou exponenciální a faktoriálové, roste složitost velmi rychle, což je znázorněno v tabulce níže, kde n udává velikost vstupu. Jako jednu operaci uvažujeme jednu ns, takže se vykoná 1 miliarda operací za vteřinu, což přibližně odpovídá rychlosti běžných procesorů. Pro lepší porozumění řádů čísel v následující tabulce 10^9 sekund odpovídá přibližně 31 let, 10^{18} sekund odpovídá přibližně stáří vesmíru a 10^{78} je přibližně počet částic ve vesmíru. Z tabulky je patrné, že exponenciální a faktoriálové úlohy nejsou použitelné už pro úlohy v řádu stovek nebo i desítek jednotek na vstupu. V tabulce jsou také zahrnuty složitosti některých známých exaktních algoritmů pro řešení problému obchodního cestujícího. Můžete si všimnout, že exaktní algoritmy přestávají dávat smysl už přibližně od 20 měst. (Tesař & Mareš)

Tabulka 1 - Rychlost výpočtu, kdy jedna operace trvá jednu ns a *n* je velikost vstupu. (Tesař & Mareš)

O	n					
	10	**20**	**50**	**100**	**1 000**	**10^6**
$O(n)$	10 ns	20 ns	50 ns	100 ns	1 000 ns	106 ns
$O(\log_2 n)$	3.3 ns	4.3 ns	4.9 ns	6.6 ns	10.0 ns	19.9 ns
$O(n \log_2 n)$	33 ns	86 ns	282 ns	664 ns	10 µs	20 ms
$O(n^2)$	100 ns	400 ns	900 ns	100 µs	1 ms	1 000 s
$O(n^3)$	1 µs	8 µs	27 µs	1 ms	1 s	109 s
$O(2^n)$	1 µs	1 ms	1 s	10^{21} s	10^{292} s	$\approx \infty$
$O(n^2 2^n)$	102 µs	419 ms	10^9 s	10^{25} s	10^{298} s	$\approx \infty$
$O((n-1)!/2)$	181 µs	10^7 s	10^{53} s	10^{146} s	$10^{2\,555}$ s	$\approx \infty$
$O(n!)$	3 ms	10^9 s	10^{55} s	10^{148} s	$10^{2\,558}$ s	$\approx \infty$

3.2.2 Základní třídy složitosti

U mnohých úloh lze složitost charakterizovat jen nepřímo, to znamená například odkazem na soubor úloh podobné složitosti. K tomuto účelu byly vytvořeny třídy složitosti. Skutečnost, že můžeme úlohu zařadit do určité třídy, může pomoci odhalit společné vlastnosti s jinými úlohami a na základě toho můžeme využít známých postupů pro řešení dané úlohy. Cílem je zařadit danou úlohu do co nejnižší třídy složitosti. (Savický, 2016)

3.2.2.1 Třída P

Třída P je třídou úloh, které lze řešit v čase, který je omezen polynomem v závislosti na velikosti vstupních dat s možností využívat neomezené množství paměti. Pokud úloha spadá do této třídy, můžeme ji ve většině případů označit za efektivně řešitelnou, nicméně z praktického hlediska je třeba složitost rozlišovat i jinak než pouze na základě toho, zdali má polynomiální složitost. Například algoritmus složitosti n^{100}, kde n je velikost vstupu, nelze použít pro vstupy s již relativně malou velikostí. (Savický, 2016)

3.2.2.2 Třída NP

Třída NP obsahuje úlohy, pro které není znám žádný efektivní algoritmus, ale je možné, že nějaký takový algoritmus existuje. Správnost výsledku algoritmu je ale možné v polynomiálním čase ověřit. U těchto úloh není dokázáno, že skutečně mají vysokou výpočetní složitost, stejně tak není dokázán dolní odhad jejich složitosti. (Savický, 2016)

3.2.2.2.1 NP-úplné problémy

Úplné problémy v dané třídě patří k těm nejsložitějším, a právě problém obchodního cestujícího do této třídy spadá. NP-úplné problémy jsou takové problémy, na který jsou polynomiálně redukovatelné všechny ostatní problémy z NP. To znamená, že jakýkoli problém z NP lze v polynomiálním čase převést na problém obchodního cestujícího. (Savický, 2016)

Právě jedno z kouzel tohoto a všech NP-úplných problémů je, že není prokázáno, že pro ně neexistuje efektivní neboli dobrý algoritmus. Tuto úplnost ukázali Karp a Cook, kteří popsali tyto problémy, o kterých se neví, jestli mají efektivní řešení, ale mají tu vlastnost, že když pro některý z nich bude nalezeno efektivní řešení, tak pro velké množství nezávisle zaměřených problémů bude také existovat efektivní algoritmus. Cookova věta přesněji tvrdí, že existují problémy, pro které když se najde efektivní algoritmus, tak zaručuje, že existuje efektivní algoritmus i pro všechny ostatní problémy ve třídě NP, což by znamenalo, že třída NP se rovná P. Takovým problémům se říká NP-úplný a problém obchodního cestujícího je právě jedním z nich. Nalezení efektivního řešení problému obchodního cestujícího tedy sahá daleko za tento problém. Zároveň jakýkoli důkaz o neřešitelnosti těchto problémů pravděpodobně přidá nebo změní základy toho, jakým způsobem tyto problémy chápeme a řešíme. (Cook, 2012)

Všeobecně se spoléhá na to, že NP se nerovná P, z teoretického hlediska pro to ale není důvod. Současné šifrovací systémy dokonce stojí na předpokladu, že pro ně efektivní algoritmus neexistuje. Lance Fortnow v roce 2009 napsal:

„Mnoho lidí se zabývá negativními důsledky, např. že P = NP by znemožnilo technicky šifrování založené na veřejném klíči. To je pravda, ale kdyby platilo P = NP, pak by celý internet vypadal jen jako zanedbatelná položka před zrodem mnohem bohatší historie.“ (Lance, 2009) (Cook, 2012)

3.2.2.3 Třídy PSPACE a NPSPACE

Tyto třídy jsou řešeny za použití polynomiální velikosti paměti, výpočet příslušných úloh tedy nemá vysoké nároky na paměť. Zařazením úlohy do této třídy je vyjádřena prostorová složitost, ale ne časová. (Savický, 2016)

3.2.2.4 Třídy EXPTIME a EXPSPACE

Jedná se o třídy, které mají exponenciální složitost a není pro ně znám polynomiální algoritmus. To znamená, že čas potřebný k nalezení řešení roste exponenciálně s velikostí vstupu. To nicméně neznamená, že neexistují lepší řešení dané úlohy, nebo že problémy v této třídě nelze vylepšit. Třída EXPTIME je spíše označením doposud nalezené nejnižší časové složitosti těchto problémů. Příkladem jsou například hry šachy, dáma nebo GO. Problém obchodního cestujícího má horší než exponenciální složitost, protože jeho složitost je faktoriálová. (Cook, 2012) (Savický, 2016)

3.2.3 Složitost problému obchodního cestujícího

Jedná se o NP-úplný problém, jehož řešení neboli výstup lze v polynomiálním čase zkontrolovat, to znamená, že dokážeme snadno spočítat, jak dlouhá je nejkratší cesta, ale nalézt tu nejkratší cestu není možné v polynomiálním čase. Při nalezení řešení za pomoci projití všech možností je časová složitost faktoriálová. S každým dalším zahrnutým městem do okruhu se čas potřebný pro výpočet zvětší mnohokrát, konkrétně když je zahrnuto do řešení patnácté město, čas potřebný pro výpočet se zvýší patnáctkrát oproti době potřebné pro čtrnáct měst. Z toho plyne, že čím více je měst na vstupu, tím vícekrát se se prodlouží výpočet při zahrnutí jednoho dalšího města. Zatím žádný efektivní algoritmus pro problém obchodního cestujícího neznáme, bylo však vyvinuto mnoho časově efektivních aproximačních neboli heuristických metod, které mají rychlejší výpočet, ale nalezené řešení nemusí být optimální. (Karp & Thatcher, 2009)

Jako dosavadní rekord byla vyřešena instance problému obchodního cestujícího s neuvěřitelnými 85 900 městy v roce 2006 výzkumným týmem ve složení: D. Applegate, R. Bixby, V. Chvátal, W. Cook, D. Espinoza, M. Goycoolea, K. Helsgaun. (Cook, 2012)

3.3 Teorie grafů

Mnoho situací nejen v matematice a informatice lze charakterizovat pomocí množiny bodů a spojnic mezi těmito body. Matematickou abstrakcí takových situací je graf. (Matoušek, 2000)

Problém obchodního cestujícího patří k úloze, kterou lze také znázornit pomocí teorie grafů, konkrétně neorientovaným úplným grafem, který je hranově ohodnocený. Předmětem úlohy je v tomto grafu najít minimální hamiltonovskou kružnici, kde jsou všechny vrcholy grafu právě jednou tak, aby součet ocenění všech hran kružnice byl minimální. (Cook, 2012)

Obyčejný neorientovaný graf je dvojice $G = (V, E)$, kde V je množina, jejíž prvky se nazývají **vrcholy** nebo uzly grafu. E je množina neuspořádaných dvojic $\{u, v\}$, kde u, v jsou dva různé prvky množiny V. Prvky množiny E nazýváme **hranami** grafu. Dva vrcholy spojené hranou jsou **sousední**. Vrchol, který náleží k hraně, je s hranou **incidentní**. Množinu vrcholů značíme $\boldsymbol{V(G)}$. Obdobně množinu hran zapisujeme jako $\boldsymbol{E(G)}$. (Kučera, 1989)

Orientovaný graf je dvojice (V, E), kde V nazýváme jako množinu vrcholů orientovaného grafu a E je množina uspořádaných dvojic vrcholů z množiny V. Prvky množiny E nazýváme orientované hrany. Neorientovaný graf můžeme chápat jako speciální případ orientovaného grafu, kdy hrana $\{v, w\}$ v neorientovaném grafu představuje orientované hrany $\{v, w\}$ a $\{w, v\}$. Úvahy platné pro orientovaný graf jsou tedy obecnější, protože zahrnují i neorientovaný přístup. (Kučera, 1989)

Stupeň vrcholu u neorientovaného grafu je počet hran, které z vrcholu vycházejí. U orientovaného grafu se počtu hran, které z vrcholu vycházejí, říká výstupní stupeň vrcholu a počtu hran, které do vrcholu vchází, říkáme vstupní stupeň vrcholu. (Hliněný, 2010)

Každá hrana grafu může být ohodnocena, této hodnotě budeme říkat **cena nebo váha**. Cena/váha může znázorňovat například vzdálenost, kapacitu propustnosti, dobu trvání, rychlost přenosu a tak dále. (Hliněný, 2010)

Podgrafem grafu G rozumíme libovolný graf H na podmnožině vrcholů $\mathrm{V(H)} \subseteq \mathrm{V(G)}$, který má za hrany libovolnou podmnožinu hran $E(G)$, pro kterou platí, že jsou všechny vrcholy hran ve $V(H)$. Podgraf značíme jako $\mathrm{H} \subseteq \mathrm{G}$. (Hliněný, 2010)

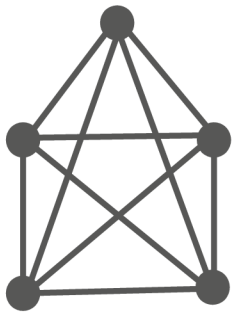
Grafy je možné vizualizovat kreslením do roviny, kdy se vrcholům přiřadí body roviny a hrany se vyjádří spojením příslušných vrcholů. U orientovaného grafu vyjadřujeme

orientaci hrany šipkou. Graf je také možné zadat výčtem vrcholů a výčtem hran. (Hliněný, 2010)

Pomocí grafů je tedy možné vizualizovat mnoho úloh z reálného světa, které se dají vyjádřit matematicky i bez použití grafů, ale díky možnosti vizuálního znázornění se grafy stávají jednoduše čitelné pro každého. Pomocí grafů lze vyjádřit návaznosti, spojení, závislosti, vztahy, toky mezi objekty (uzly/vrcholy) atd. Z toho důvodu a také pro jejich jednoduchou datovou reprezentaci, si grafy našly své místo také v informatice. (Matoušek, 2000) (Hliněný, 2010)

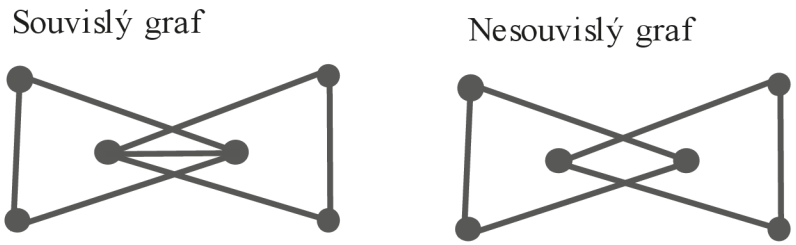
Grafy můžeme implementovat maticí sousednosti pomocí dvourozměrného pole $g[,]$, ve kterém $g[i, j] = 1$ znamená hranu mezi vrcholy i a j s cenou 1. Případně je možné použít takzvaný výčet sousedů tak, že k dvourozměrnému poli přidáme i jednorozměrné pole stupňů vrcholů. (Hliněný, 2010)

V **Úplném grafu** jsou všechny jeho dvojice vrcholů sousední neboli množina $E(G)$ obsahuje všechny kombinace vrcholů v množině $V(G)$. Úplný graf je ideální pro testování efektivity algoritmů, protože obsahuje nejvyšší možný počet hran. Procházení úplného grafu zabírá nejvíce času ze všech typů grafů. Je-li počet vrcholů v grafu $|V(G)|$ a počet hran v grafu $|E(G)|$, tak v úplném neorientovaném grafu vypočítáme počet hran jako $|E(G)| = \frac{|V(G)|*(|V(G)|-1)}{2}$. To znamená, že s každým dalším vrcholem přibyde $|V(G)| - 1$ hran. Naopak z počtu hran zjistíme počet vrcholů jako $|V(G)| = \frac{1+\sqrt{1+8*|E(G)|}}{2}$. Problém obchodního cestujícího je obvykle řešen nad úplným grafem. (Hliněný, 2010)



Obrázek 1 - Úplný graf K5, zdroj: vlastní zpracování

V **souvislém** grafu vede z každého vrcholu $V(G)$ cesta do jakéhokoli jiného vrcholu v množině $V(G)$. V souvislém grafu jsou tedy všechny vrcholy propojené cestou. Souvislost nás zajímá hlavně v grafech reprezentující nějakou síť, například internetovou, počítačovou, dopravní, nebo potrubní, protože bývá nežádoucí, aby taková síť byla přerušena. (Hliněný, 2010)



Obrázek 2 - Souvislý a nesouvislý graf, zdroj: vlastní zpracování

Sled v grafu G je taková posloupnost vrcholů $v_i = 1, 2, \ldots, n$ a hran, že $\{v_i - 1,\ v_i\}$ je hranou pro všechna $i = 1, 2, \ldots, n$. Jedná se tedy o posloupnost sousedních vrcholů za sebou, kde se mohou opakovat vrcholy i hrany. (Kučera, 1989)

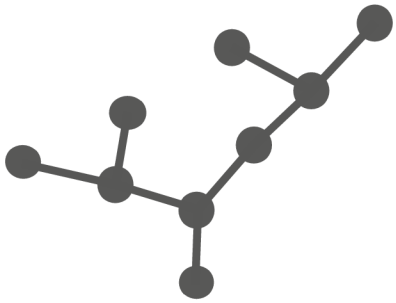
Tah je sled, ve kterém se nevyskytuje více stejných hran. **Cesta** je sled, ve kterém se nevyskytuje více stejných vrcholů. **Eulerův tah** je uzavřená cesta, která prochází každou hranou právě jednou. (Kučera, 1989)

Kružnice obsahuje $n \geq 3$ hran a $n \geq 3$ vrcholů, které jsou sousední a zároveň žádná hrana ani vrchol se v kružnici neopakují. Jedná se tedy o uzavřenou posloupnost propojených vrcholů, kde hrany vedou mezi sousedními vrcholy a také mezi prvním a posledním vrcholem. Počet hran i počet vrcholů je v kružnici totožný. (Kučera, 1989)

Hamiltonovská kružnice je uzavřená cesta v grafu, která prochází každým vrcholem právě jednou. Minimální / maximální hamiltonovská kružnice je taková hamiltonovská kružnice, jejíž součet ocenění hran je minimální / maximální. Cílem problému obchodního cestujícího je právě nalezení hamiltonovské kružnice. (Kučera, 1989)

Vzdálenost $dG(u,\ v)$ dvou vrcholů u, v v grafu G je dána sumací cen hran nejkratší cesty mezi u a v. Pro výpočet vzdálenosti je možné použít Dijkstrův algoritmus, který je efektivní a exaktní. (Hliněný, 2010)

Strom je jednoduchý souvislý graf bez kružnic. Počet hran ve stromu je vždy menší o jednu jednotku, než je počet vrcholů. Jedná se o graf, který je vhodný pro znázornění hierarchie. Úplně nejstarším použitím stromu jsou rodokmeny, jejichž původ sahá daleko před vznik teorie grafů. (Hliněný, 2010)



Obrázek 3 - Strom, zdroj: vlastní zpracování

3.4 Inteligence hejna

Inteligence hejna spolu s genetickými algoritmy a neuronovými sítěmi spadá do oboru umělé inteligence. Problém obchodního cestujícího je možné řešit všemi těmito přístupy, které je možné navzájem kombinovat. (Wirsansky, 2020) (Meshram, a další, 2019)

Genetické algoritmy jsou inspirovány evolučními procesy přírody, zejména principy dědičnosti genů a přirozeného výběru/selekce. Používají se k optimalizaci a hledání řešení problémů prostřednictvím procesů, jako je selekce, křížení a mutace. (Wirsansky, 2020)

Neuronové sítě jsou naproti tomu inspirovány strukturou a fungováním mozku. Tyto modely se snaží napodobit způsob, jakým neurony v mozku komunikují a jak se dokáží učit. Jsou používány pro úkoly, jako je rozpoznávání obrazu, zpracování přirozeného jazyka a predikce. (Meshram, a další, 2019)

Inteligence hejna se inspiruje chováním a interakcemi organismů v přírodě, jako jsou hejna ryb a ptáků anebo kolektivní organismy jako například včely a mravenci. Tento koncept vychází z pozorování skupinového chování těchto organismů a aplikuje je na problémy v informatice. Zabývá se tedy vytvářením algoritmů, které simulují chování hejna a využívají jeho principy k řešení různorodých úkolů. Tyto algoritmy si našly své využití v optimalizaci jako například hledání globálního extrému v nelineárních funkcích,

nalezení optimální cesty v sítích, shlukování dat do homogenních skupin, kooperativní chování autonomních robotů a v dalších oblastech. Inteligence hejna je založena na myšlence, že prostá interakce jednoduchých jednotek může vést k emergentnímu a komplexnímu chování celku. (Ying, 2018)

3.4.1 Základní principy inteligence hejna

Emergence je princip, který vyjadřuje, že komplexní chování hejna vychází z jednoduchých interakcí mezi jednotlivými jednotkami. Emergence obecně znamená vznik struktur složitých systémů, které není snadné odvodit z vlastností jednotlivých jednotek systému. Například chování mravenčích kolonií není snadné odvodit ze znalosti vlastností jednoho mravence. (Ying, 2018)

Sebeorganizace znamená, že hejno se samo organizuje na základě lokálních pravidel a bez centrálního řízení. Například mravenci se řídí pomocí zanechávání feromonů. (Ying, 2018)

Adaptabilita hejna je, že jednotlivé jednotky hejna jsou schopny reagovat a přizpůsobovat se změnám v prostředí. Například mravenec dokáže reagovat na intenzitu feromonů na cestě. (Ying, 2018)

3.4.2 Umělé včelí kolonie

Umělé včelí kolonie jsou inspirovány chováním včel při sběru nektaru. Včely jsou rozděleny do tří rolí, a to na dělnice, vyčkávající včely a na průzkumnice. Každá dělnice opracovává jeden zdroj jídla, ale zároveň pravidelně navštěvuje i další zdroje jídla v okolí. Pokud zjistí, že je na zdroji v okolí jídlo kvalitnější, nahradí svůj zdroj za tento nový kvalitnější. Následně se všechny dělnice sejdou, přičemž si vymění informace o kvalitě zdrojů a vyčkávající včely si vyberou pomocí ruletové selekce některý z těchto zdrojů. Dělnice si zároveň pamatují, jak dlouho již opracovávají daný zdroj a pokud tato doba překročí určitý limit, tak se z ní stane průzkumnice. Průzkumnice prohledávají prostor náhodně a hledají nové zdroje potravy. (Pilát, 2016)

3.4.3 Optimalizace hejnem částic

Optimalizace hejnem částic je inspirována chováním hejna ryb nebo ptáků. Slouží k hledání optimálního řešení spojitých optimalizačních problémů. Princip algoritmu spočívá

v tom, že každá částice v hejnu je reprezentována jako dvojice vektorů v prostoru, kde jedním vektorem je její pozice a druhým je její rychlost. Každá částice si pamatuje svoji dosud nejlepší nalezenou pozici. Hejno si zároveň pamatuje dosud nejlepší dosaženou globální pozici v celém hejnu. Částice se pohybují v prostoru a jsou přitahovány k místům, kde už našly své nejlepší řešení a také k místu nejlepší globální pozice. V každé iteraci algoritmus aktualizuje rychlosti a pozice všech částic. (Pilát, 2016) (Ying, 2018)

3.4.4 Optimalizace mravenčí kolonií

Optimalizace mravenčí kolonií simuluje chování mravenčí kolonie při hledání cesty k potravě. Mravenci komunikují pomocí feromonů a přizpůsobují svou trasu na základě informací získaných z okolí. (Ying, 2018)

3.4.4.1 Chování mravenců

Vyšší druhy zvířat obvykle používají jako své hlavní smysly zrak a sluch, ale mravenci mají zrak velmi omezeně vyvinutý a jsou dokonce druhy mravenců, kteří jsou kompletně slepí. Většina druhů mravenců namísto zraku používá specifické stopovací feromony. Tyto feromony jsou například typické pro mravence obecného (Lacius niger), který je v České republice nejrozšířenější, nebo pro mravence argentinského (Linepithema humile), který je původem z Argentiny, ale nyní je rozšířený po celém jihu Evropy. (Dorigo & Stützle, 2004)

Mravenci mají komplexní sociální chování, které zajímá lidi již od nepaměti. Jedna z nejpatrnějších forem jejich chování je vytváření cest, kterými se pohybují. Mravenčí kolonie má jednu velmi užitečnou schopnost, kterou je nalézt nejkratší cestu v prostoru. Tuto schopnost mravenci získali díky komunikaci pomocí feromonů neboli pachových stop, které za sebou zanechávají a dokáží je vnímat. Přestože mravenec, jako jednotlivec, je jednoduchý tvor, mravenčí společenství představuje vysoce organizovanou sociální strukturu. Díky této organizaci jsou mravenci, jako společenství, schopni zvládnout úkony, které by samotný mravenec nedokázal splnit. Odvětví mravenčích algoritmů se zabývá modely inspirovanými skutečným chováním mravenců a používá tyto modely jako inspiraci pro navrhování algoritmů pro řešení optimalizačních a distribučních úloh. (Dorigo & Stützle, 2004)

Mnoho aspektů chování mravenců slouží jako inspirace pro algoritmické řešení různých úloh jako například rozdělování práce, hledání potravy, třídění, kooperativní přeprava a další. Ve všech těchto případech mravenci koordinují své chování pomocí stigmergie, což je forma nepřímé komunikace řízené modifikací prostředí. Například mravenec, který hledá potravu, zanechává na cestě chemické stopy, které zvyšují pravděpodobnost, že i ostatní mravenci se rozhodnou následovat tu stejnou trasu. Stigmergie se tedy ukázala jako efektivní způsob, jak sociální hmyz může dosáhnout sebeorganizace. Pointa za mravenčími algoritmy je potom vytvoření virtuální stigmergie za účelem koordinace virtuálních agentů (zde mravenců) jako součást počítačového programu. (Dorigo & Stützle, 2004)

Když mravenec musí ujít velkou vzdálenost pro jídlo tam a zpět, tak intenzita feromonů na této cestě nebude tak velká, jako na cestě, kde mravenec ujde pro jídlo kratší vzdálenost. To je dáno tím, že feromony vyprchávají a mravenec stihne po kratší cestě projít tam a zpět vícekrát za stejný časový úsek. (Dorigo & Stützle, 2004)

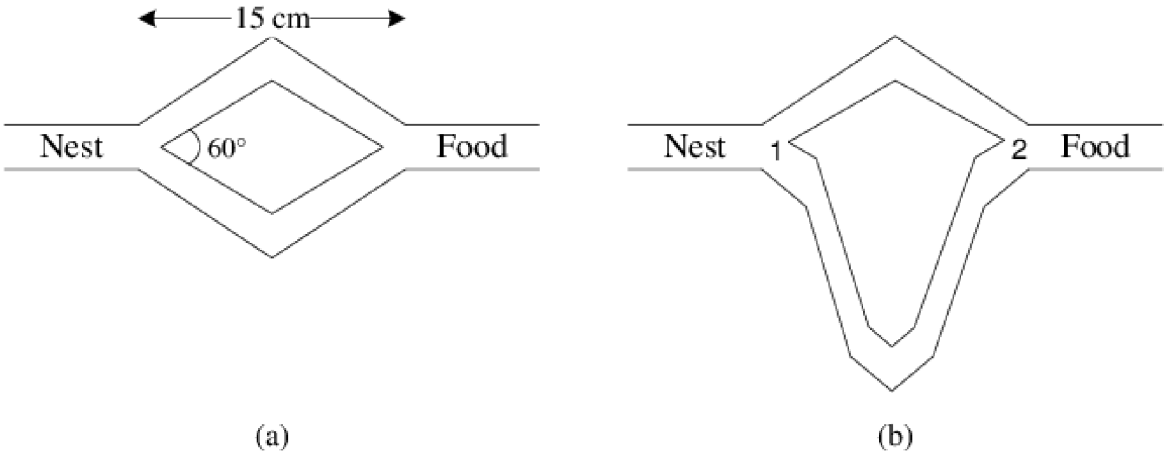
3.4.4.1.1 Experiment s dvěma mosty

Pro pozorování a porozumění chování mravenců při hledání jídla byl historicky významný experiment, který navrhl Éric Bonabeau a jeho kolegové v roce 1995. Mravenci totiž hledají potravu, kterou následně nosí do své základny. Takto tedy cestují mezi potravou a svojí základnou tam a zpět. Experiment s dvěma mosty spočívá v tom, že se mravencům ze základny k jídlu postaví dvě alternativní cesty a sleduje se jejich chování při výběru, jakou z nabízených cest si zvolí. (Dorigo & Stützle, 2004)

V prvním případě jsou mravencům nabídnuty dvě cesty, které jsou stejně dlouhé. Mravenci zprvu používají obě cesty rovnoměrně, ale vzhledem k tomu, že každý mravenec se rozhoduje i na základě náhody, jestli půjde jednou, nebo druhou cestou, časem se koncentrace feromonů u jedné z cest stane intenzivnější a mravenci začnou používat převážně jen jednu z cest i přes to, že obě jsou stejně dlouhé. (Dorigo & Stützle, 2004)

V druhém případě je mravencům nabídnuta cesta, kde je jedna z cest dvojnásobně delší než druhá. Zprvu mravenci opět používají obě cesty, ale vzhledem k tomu, že se feromony u kratší cesty rychleji hromadí a nestíhají vyprchat, tak mravenci brzy začínají převážně používat kratší cestu. Je zajímavé, že i přes to, že jedna cesta má po čase výrazně větší koncentraci feromonů, tak je malé procento mravenců, kteří se rozhodnou jít delší cestou.

Nikdy se tedy nestane, že by všichni mravenci zvolili tu nejkratší cestu. Toto chování by se dalo považovat za prozkoumávání okolí. (Dorigo & Stützle, 2004)



Obrázek 4: Experiment s dvěma mosty (Dorigo & Stützle, 2004)

Další zkoumanou situací bylo přidání kratší cesty až po půl hodině trvání experimentu. V tomto případě i přes to, že malé procento mravenců prozkoumává nové cesty, tak mravenci již nebyli schopni svou cestu změnit z delší na kratší, a tak zůstali uvězněni na delší cestě. Je to dáno tím, že feromony na delší cestě jsou již velmi koncentrované a také pomalým vyprcháváním feromonů z cesty. Vyprchání feromonů by mohlo umožnit příležitostně najít nové řešení, ale v tomto případě je příliš pomalé na to, aby mravenčí kolonie mohla zapomenou na delší cestu a začít používat kratší. (Dorigo & Stützle, 2004)

3.4.4.2 Řešení TSP mravenčími koloniemi

Optimalizace mravenčími koloniemi, anglicky Ant colony optimization algorithm zkráceně ACO, byla poprvé popsána Marcem Dorigem v roce 1992. Mravenčí kolonie se v té době ukázaly jako velmi úspěšný přístup. (Dorigo & Stützle, 2004)

Fungování algoritmu lze popsat tak, že z každého města jsou vysláni mravenci, odkud každý mravenec udělá vlastní okruh. Při rozhodování, jakou cestou se vydat, se mravenec řídí na základě intenzity feromonů na cestách, vzdálenosti sousedních měst a na základě náhody. (Dorigo & Stützle, 2004) (Soofastaei, 2022)

V moment, kdy všichni mravenci najdou svůj okruh, se aktualizují feromony na cestách, přes které mravenci prošli. Takto se postup opakuje do daného počtu iterací. Jako výstup slouží nejúspěšnější okruh, který jakýkoliv mravenec prošel. (Dorigo & Stützle, 2004) (Soofastaei, 2022)

Vzhledem k tomu, že úloha má různý charakter, nejsou všichni mravenci vysláni jen z jednoho počátečního města, ale každý mravenec je vyslán z odlišného města jako výchozí bod. Je to dáno tím, že krom feromonů se mravenec rozhoduje i na základě vzdálenosti daného města od místa, kde se nachází. Kdyby každý mravenec vycházel ze stejného města, mohlo by se stát, že některé vhodné možnosti nalezení okruhu by zůstaly neprozkoumány. Obzvlášť negativní efekt by mohl nastat v případě úlohy s odlehlými městy. (Dorigo & Stützle, 2004)

Počet vyslaných mravenců je možné udat jako x násobek počtu měst v řešení. Počet vyslaných mravenců má přímý vliv na rychlost programu. Čím více mravenců je vysláno, tím delší dobu trvá výpočet, to stejné platí i u celkového počtu iterací. Počet mravenců udává, jak často se aktualizuje intenzita feromonů na cestách. Snížíme-li počet mravenců, je možné zvýšit počet iterací při zachování stejné doby výpočtu, tímto by bylo docíleno častější aktualizace intenzity feromonů na cestách. (Dorigo & Stützle, 2004)

3.4.4.2.1 Rozhodování mravence, kam se vydá

Mravenec, který je vyslán na cestu, se právě nachází ve výchozím městě a jeho úkolem je udělat okruh přes všechna města a vrátit se. Nyní se mravenec musí rozhodnout, do jakého sousedního města se vydá. Protože ještě žádné město nenavštívil, může se vydat kamkoli. U každé z možností je známá intenzita feromonů na cestě (hraně grafu) a aproximovaná vzdálenost cesty. Aproximovaná vzdálenost se vypočítá jako konstanta vydělená skutečnou vzdáleností mezi městy. Z toho plyne, že čím vyšší je aproximovaná vzdálenost, tím je cesta ve skutečnosti kratší, tedy lepší stejně jako tomu je u hodnoty intenzity feromonů. (Dorigo & Stützle, 2004)

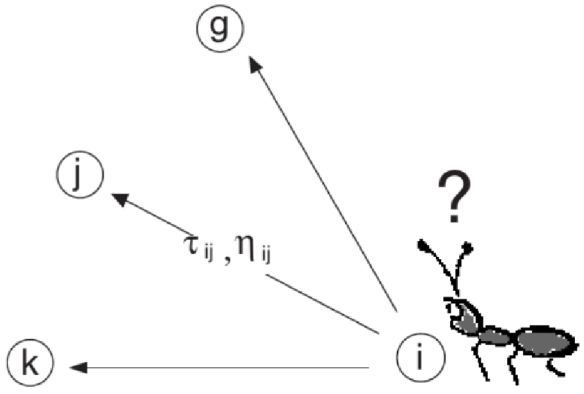
Pro výpočet pravděpodobnosti P_{ij}, že se mravenec vydá z aktuálního města i do jednoho ze sousedních měst j, vydělíme touhu mravence $\tau_{ij}^{\alpha}\eta_{ij}^{\beta}$ jít do tohoto konkrétního města j součtem všech tuh mravence jít do všech volných okolních měst. Jinými slovy pravděpodobnost, že mravenec půjde v dalším kroku do města j se rovná touze

mravence se tam přemístit vydělenou součtem všech tuh. Takto je tedy možné vypočítat pravděpodobnost pro všechna ještě nenavštívená města. Když všechny tyto pravděpodobnosti sečteme, výsledek bude roven jedné. (Blum, 2005) (Dorigo & Stützle, 2004)

$$P_{ij} = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum \tau_{im}^{\alpha}\eta_{im}^{\beta}}$$

Obrázek 5: Vzorec pro výpočet pravděpodobnosti, kam se mravenec vydá. (Dorigo & Stützle, 2004)

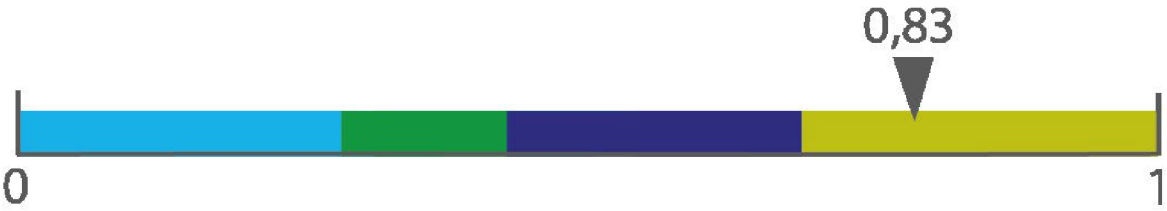
Touha mravence $\tau_{ij}^{\alpha}\eta_{ij}^{\beta}$ jít z aktuálního města i do j je násobek hodnoty intenzity feromonů τ_{ij}^{α} na cestě a hodnoty aproximované vzdálenosti η_{ij}^{β}. Empirické konstanty α a β určují, jestli se má mravenec spíše rozhodovat na základě feromonů anebo aproximované vzdálenosti. Pokud jsou oba koeficienty α a β nastaveny na hodnotu jedna, mravenec nedává při rozhodování větší přednost ani feromonům, ani vzdálenosti. Pokud je $\alpha > \beta$ rozhoduje se mravenec více na základě intenzity feromonů, a naopak pokud je $\alpha < \beta$, tak se mravenec rozhoduje spíše na základě aproximované vzdálenosti. (Dorigo & Stützle, 2004)



Obrázek 6: Mravenec se rozhoduje na základě touhy. (Dorigo & Stützle, 2004)

Pro konečné rozhodnutí mravence, kam se vydá, slouží generátor náhodných čísel. Jednotlivé pravděpodobnosti se uspořádají do číselné osy od 0 do 1. Poté je vygenerováno náhodné číslo od 0 do 1, které se přiřadí na číselnou osu. Mravenec se vydá do toho města, které je zvoleno náhodným číslem na číselné ose. (Dorigo & Stützle, 2004)

Například pokud se mravenec rozhoduje, kam se vydá mezi čtyřmi městy, tak se vypočítá pravděpodobnost pro navštívení každého města zvlášť. Každá z těchto pravděpodobností zaujímá určitý interval na ose, kdy pořadí uspořádání na ose není relevantní. Pomocí vygenerování náhodného čísla například 0,83 je určeno, do jakého města se mravenec vydá. (Dorigo & Stützle, 2004)



Obrázek 7: Rozhodnutí mravence na základě pravděpodobnosti a náhody, zdroj: vlastní zpracování

V následujících krocích se mravenec opět rozhoduje stejným způsobem, přičemž se nevrací do měst, kam se již vydal, to znamená, že pro již navštívená města není spočtena pravděpodobnost navštívení. Tímto způsobem se kroky opakují, dokud už nezbývají žádná nenavštívená města. Následně se mravenec vrátí do výchozího města a tím uzavře okruh. (Dorigo & Stützle, 2004)

3.4.4.2.2 Aktualizace feromonů

Před prvním vysláním mravenců jsou všechny feromony na všech cestách nastaveny na určitou jednotnou hladinu. Následně pokaždé po tom, co všichni mravenci projdou svůj okruh, dojde k aktualizaci intenzity feromonů na cestách mezi městy. To způsobí, že další mravenci už budou při rozhodování ovlivněni tím, kudy šli předchozí mravenci. (Mavrovouniotis, Muller, & Yang, 2017) (Dorigo & Stützle, 2004)

Vzhledem k tomu, že feromony, které za sebou mravenci v přírodě zanechávají po čase vyprchávají, tak i při aktualizaci feromonů je jako první provedeno snížení intenzity feromonů o několik procent. Čím větší je intenzita vyprchávání feromonů, tím více se rozšíří okruh cest, které mravenci prozkoumají před tím, než se upnou k jednomu řešení.

Bez vyprchávání feromonů by docházelo k tomu, že už během prvních pár opakování bude preferováno jedno řešení, kde je jednoznačně více feromonů a mravenčí kolonie toto řešení již neopustí. Spuštění algoritmu ve více opakování by v tomto případě nemělo přílišný efekt. Díky vyprchávání feromonů mají umělí mravenci lepší schopnost zapomenout na předchozí horší řešení v případě, že najdou lepší. Vyprchávání feromonů je tedy velmi důležité a jeho nastavení má značný vliv na výsledky algoritmu. U úloh, které nejsou tak komplexní jako problém obchodního cestujícího, nemá snižovaní intenzity feromonů tak velký význam, dokonce ani v přírodě pro mravence vyprchávání feromonů nemá významný vliv na optimalizaci cesty za potravou. (Dorigo & Stützle, 2004)

Po snížení hladiny intenzity feromonů dojde k přičítání feromonů na cesty, po kterých mravenci prošli. Mravenci, kteří našli kratší okruh, za sebou zanechávají více feromonů. Prvně je vypočítána délka okruhu každého mravence. Následně se vypočítá přesný počet zanechaných feromonů mravence na každé cestě (hraně grafu), který je vypočítán jako konstanta Q vydělená délkou okruhu mravence. (Mavrovouniotis, Muller, & Yang, 2017) (Dorigo & Stützle, 2004)

Modifikace aktualizace feromonů

Bylo vymyšleno několik modifikací aktualizace feromonů za účelem snahy zefektivnit algoritmus. Jedním příkladem je, že mravenec, který našel zatím doposud nejlepší řešení, bude za sebou zanechávat více feromonů než běžní mravenci. Tento přístup se nakonec ukázal jako méně efektivní, protože dochází k příliš brzkému uchýlení se k jednomu řešení, které nemusí být vzhledem k náhodě dostatečně optimální. Jinými slovy při pokusech, kdy byl algoritmus spouštěn mnohokrát za sebou, bylo mnoho případů, kdy výsledek byl znatelně horší, tzn. výsledek algoritmu byl občas dobrý a občas ne. Zdali budou mravenci dosahovat lepších, nebo horších výsledků po této úpravě, může být dané typem úlohy, významný vliv může mít například to, jak jsou v úloze rozprostřená města. (Birattari, a další, 2004)

Dalším příkladem optimalizace je určení dolní a horní hladiny intenzity feromonů, kterou není možné překročit. Tento přístup naopak rozšíří okruh, do kterého se mravenci vydají a tím se tedy zvyšuje pravděpodobnost, že i po mnohých iteracích bude nalezeno nové lepší řešení. Tato optimalizace, nazývaná jako MAX-MIN mravenčí systém, se ukázala v mnohých případech jako úspěšná. (Birattari, a další, 2004)

3.5 Exaktní algoritmy

Exaktní metody fungují pouze pro malý počet měst, mají ale přesný výsledek. Zatím není znám žádný exaktní algoritmus s výpočtem v polynomiálním čase a ani není stále potvrzeno, že takový algoritmus neexistuje. (Cook, 2012)

Mezi historicky nejdůležitější exaktní algoritmy patří ty vyvinuté týmem Dantziga. Tyto algoritmy tvoří základy pro nejvíce efektivní algoritmy, které se dnes používají. Za jeden z dnes nejlepších dostupných programů pro exaktní výpočet problému obchodního cestujícího je považován program s názvem Concorde TSP, který byl ve svých ranných fázích v roce 2003 označen jako Dantzig-Fulkerson-Johnsonův algoritmus pro velký problém obchodního cestujícího. (Cook, 2012) (Applegate, Bixby, Chvátal, & Cook, 2003)

Concorde TSP využívá lineárního programování pro hledání minima, tedy hodnoty udávající, jak krátký je nejkratší možný okruh dané úlohy. To znamená, že program dokáže určit, zdali okruh, který našel je optimální. Toto ověření není výpočetně náročné. Pro nalezení okruhu Concorde TSP využívá kombinaci různých heuristických algoritmů, které mohou sloužit jako vstup pro exaktní výpočet, který je prováděn metodami jako jsou metoda řezných rovin a metoda větví a mezí. Concorde TSP již dokáže nalézt optimální cestu pro úlohy s počtem měst v rámci tisíců, nicméně výpočet trvá roky a provádí se na velmi výkonných počítačích určených k výzkumu. Rozhodnutí, zdali řešit úlohu pomocí exaktního algoritmu, nebo pomocí heuristik, tedy závisí na velikosti a typu úlohy, nároků na čas výpočtu v kombinaci s tím, jak výkonný je počítač, na kterém bude algoritmus spouštěn. (Hoos & Stützle, 2014) (Cook, 2012) (Applegate, Bixby, Chvátal, & Cook, 2003)

3.5.1 Metoda hrubou silou

Jedná se o nejméně efektivní metodu. Metoda spočívá v tom, že vyzkouší všechny možné přípustné řešení a vybere z nich to nejkratší. (Cook, 2012)

Počet přípustných řešení si můžeme vypočítat vztahem x = $\frac{(n-1)!}{2}$, kde n je počet měst. Představme si, že máme deset měst, které potřebujeme navštívit s tím, že budeme začínat a končit ve stejném městě. Na první pozici v pořadí všech možností tedy vždy navštívíme stejné město, pak máme devět možností, jak zvolit druhé město k navštívení, osm možností, jak zvolit třetí město a tak dále, proto je v činiteli zlomku $(n - 1)!$. Zároveň je jedno, jakým

směrem okruh pojedeme, proto celý výsledek ještě vydělíme dvěma. V případě deseti měst máme tedy 181 440 možných okruhů kudy se vydat a chceme z nich vybrat ten nejkratší.

Pro vytvoření představy o rychlosti tohoto postupu by vyřešení problému o 33 městech trvalo 28 bilionů let, což je dvakrát déle, než je stáří vesmíru, a to jen v případě, kdy by prověření jedné cesty zabralo jen jednu aritmetickou operaci a k výpočtu bychom použili počítač IBM Roadrunner se 129 600 procesory s výkonem 1 457 bilionů aritmetických operací za sekundu. (Cook, 2012)

3.5.2 Metoda řezných rovin

Jedná se o metodu inspirovanou celočíselným lineárním programováním, vyvinutou vědeckým týmem v zastoupení Dantzig, Fulkerson a Johnson. Jejich metoda nemá za cíl řešit celou úlohu najednou, namísto toho generuje a počítá části podle toho, jak jsou při řešení potřeba. Tento přístup má přínos nejen pro problém obchodního cestujícího. Později se tato metoda začala kombinovat i s metodou větví a mezí. (Applegate, Bixby, Chvátal, & Cook, 2003) (Hladík, 2023) (Cook, 2012)

3.5.3 Metoda větví a mezí

Tato metoda vyvinutá ve spolupráci s Dantzigem také využívá znalostí z lineárního celočíselného programování. Funguje na principu rozdělení úlohy na dvě jednodušší podúlohy, čemuž se říká větvení. Pokud jsou jednotlivé podúlohy stále příliš obtížné k řešení, tak se dále větví na jednodušší podúlohy. Vzniká tak strom řešení. Podúloha nikdy nesmí být složitější než její rodič. Pomocí podúloh se získá řešení hlavní úlohy. Aby se zbytečně neprocházely větve výpočtu, které prokazatelně neobsahují optimální řešení, tak během větvení dochází k prořezávání podúloh za účelem snížení exponenciální obtížnosti celé úlohy. (Applegate, Bixby, Chvátal, & Cook, 2003) (Hladík, 2023) (Cook, 2012)

Jedná se o víceúčelový přístup, nejprve byl formulován právě v kontextu problému obchodního cestujícího. Metoda větví a mezí společně s metodou řezných rovin dokážou řešit úlohy s tisíci městy, což je velmi dobrý výsledek. V moderních programech pro problém obchodního cestujícího se právě tato kombinace používá. (Applegate, Bixby, Chvátal, & Cook, 2003) (Hladík, 2023) (Cook, 2012)

3.6 Heuristiky založené na teorii grafů

Heuristický algoritmus je implementován pro specifický problém pomocí setu pravidel nebo strategií. Strategie může vycházet například z nějaké intuitivní myšlenky, náhody nebo zkušenosti. Změnou v těchto pravidlech vznikne další heuristika, je tedy možné heuristiky různě modifikovat. V porovnání s tradičními algoritmy heuristiky negarantují optimální řešení, ale mohou nabídnout uspokojivé řešení v krátkém čase. Zároveň jsou velmi adaptivní na druh problému a používány skoro ve všech vědeckých odvětvích a inženýrských aplikacích. (Du, Liu, Zhang, & Lu, 2021)

Protože tradiční optimalizační algoritmy problému obchodního cestujícího nejsou efektivní, jsou k řešení často používané heuristické nebo sofistikované metaheuristické algoritmy pro optimalizaci. Problém obchodního cestujícího je také často používaný jako benchmark neboli testovací úloha pro srovnání efektivity těchto algoritmů. Výstupem je buď nové řešení problému anebo vylepšení už hotového řešení. Heuristiky jsou na rozdíl od exaktních algoritmů vždy rychlé a robustní, což v tomto kontextu znamená, že nejsou tak citlivé na velikost vstupu. (Du, Liu, Zhang, & Lu, 2021)

3.6.1 Metoda nejbližšího souseda

Metoda nejbližšího souseda má asymptotickou časovou složitost $O(n^2)$.

Nalezený výsledek je vždy menší než $(1 + \frac{1}{2} \log n)$násobek délky optimální cesty. Například máme-li 50 měst, tak nalezená cesta metodou nejbližšího souseda bude maximálně čtyřikrát delší než optimální. (Cook, 2012)

Vstupem je matice cen hran, kde indexy matice udávající řádek a sloupec znázorňují vrcholy. Výstupem je seznam seřazených vrcholů. (Cook, 2012)

Kdyby se obchodní cestující řídil metodou nejbližšího souseda, vydal by se vždy z místa, kde se nachází, do dalšího města, které mu je nejblíž a zároveň ho ještě nenavštívil. Takto by se tedy posouval vždy k nejbližšímu městu, dokud by mu nezbývalo jen výchozí město, do kterého by se vrátil. Je zde nebezpečí, že by vzdálenost mezi výchozím a posledním městem byla zbytečně veliká. Metoda nejbližšího souseda totiž běžně nepodává dobré výsledky, pokud se v seznamu vyskytují města, které jsou od ostatních nepřiměřeně daleko. Metoda může dosahovat různých výsledků podle toho, jaké město zvolíme jako

výchozí. Je tedy potřeba vyzkoušet všech n možností, kde n je počet měst a okruh vždy začínáme v jiném městě. Z těchto možností potom vybereme tu nejkratší. (Cook, 2012)

3.6.2 Hladový algoritmus

Hladový algoritmus má asymptotickou časovou složitost $O(n \log_2 n)$.

Nalezený výsledek je vždy menší než $(\frac{1}{2} + \frac{1}{2} \log n)$násobek délky optimální cesty, to je o něco lepší než u metody nejbližšího souseda. (Cook, 2012)

Vstupem je matice nebo množina hran a výstupem je množina hran. Hladový algoritmus na rozdíl od metody nejbližšího souseda staví cestu po částech. (Cook, 2012)

Algoritmus je pojmenovaný hladový, protože se snaží vždy nenasytně sníst neboli pojmout hrany, které mají nejlepší ceny. (Cook, 2012)

Postupujeme tak, že všechny ohodnocené hrany v $E(G)$ seřadíme podle cen vzestupně. Postupně si bereme hrany popořadě od nejmenší a pokoušíme se je zařadit do řešení. Hranu můžeme zařadit do řešení jen pokud přidáním hrany nebudou jeho vrcholy většího stupně než 2. Zároveň přidáním hrany nesmí vzniknout kružnice, vyjma případu, kdy už máme v řešení o jedna méně hran, než je celkový počet vrcholů. V moment, kdy jsme zařadili n hran, kde n je počet vrcholů, skončíme. (Cook, 2012)

Na začátku výpočtu vypadá metoda slibně, ale ke konci může být nucena zařadit i některé zbytečně dlouhé hrany. (Cook, 2012)

3.6.3 Lin-Kerninghanovy metody zlepšující řešení

Tyto metody mají jako vstup už hotové řešení spočítané nějakou heuristickou metodou anebo náhodně vybraný okruh. Výstupem je také hamiltonovská kružnice, ale se stejnou nebo lepší celkovou délkou cesty než vstup. Cílem těchto metod je tedy navržené řešení zlepšit. Zlepšení se provádí pomocí výměny hran za nové, které mají v součtu cen hran menší hodnotu. Výchozí hrany, které se mají nahrazovat a alternativní hrany, kterými se můžou nahradit výchozí hrany, jsou vybírány tak, aby se po výměně nenarušila souvislost grafu a graf stále tvořil hamiltonovskou kružnici. (Keld, 2006)

Lin-Kerninghanův algoritmus funguje na takzvaných r-opt výměnách hran, r zde určuje počet vyměňovaných hran. Lin a Kerninghan vynalezli algoritmus, který si v průběhu volí, kolik hran bude zaměňovat podle toho, co je zrovna výhodné. Jedná se o metodu, která

může najít optimální řešení do 145 vrcholů v čase pod $30n^3$ μsekund, kde n je počet vrcholů. Touto metodou jsme tedy schopní dosahovat dobrých výsledků v krátkém čase. Za posledních 50 let se původní metoda stále vyvíjela a zlepšovala, nyní umožňuje najít dobré řešení pro úlohy s více než deseti miliony body. (Du, Liu, Zhang, & Lu, 2021) (Keld, 2006) (Lin, 1965)

Metodu výměn nejde použít na úloze o třech vrcholech, protože takový graf nemá k dispozici žádné hrany k výměně, vylepšení takového okruhu ani nedává smysl. Maximální možný počet záměn je omezený počtem vrcholů, platí zde vztah $r \leq \frac{(n\text{-}1)}{2}$, kdy n je liché číslo představující počet vrcholů, nebo $r \leq \frac{n}{2}$ pro sudá n, platí, že $r \epsilon N$. (Misevicius, 2011)

Algoritmus prochází a porovnává kombinace hran tak dlouho dokola, dokud nachází hrany k výměně.

Čím více se rozhodneme vyměnit hran, tím více máme možností, kde všude hrany vyměňovat. Při výměně tří a více hran je i více voleb, jaké hrany nabídnout za hrany původní. Platí, že pro r hran k výměně máme na výběr $(r-1)!\,2^{r-1}$ alternativních možností pro výměnu za původní hrany. Ze vztahu je zřejmé, že použití r-opt má své hranice pro hodnotu r, a to poměrně nízko. Obtížnost tedy rychle stoupá s rostoucím počtem vyměňovaných hran. Výpočetní složitost činí $O(n^r)$. Prakticky je využitelný spíše jen 2-opt a 3-opt, to jsou algoritmy, které vyměňují dvě nebo tři hrany. (Misevicius, 2011)

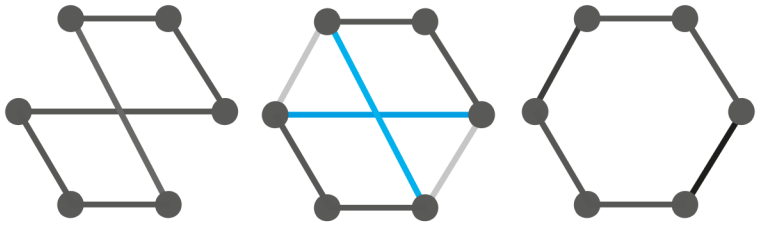
3.6.3.1 Metoda 2-opt

Misevicius (2011) uvedl, že metoda metoda 2-opt má asymptotickou časovou složitost $O(n^2)$.

Tato vylepšující metoda, jak název napovídá, vždy vyměňuje dvě hrany za jiné dvě hrany. 2-opt můžeme použít na graf už od čtyř vrcholů. Pro výpočet budeme postupovat výběrem dvojice hran a změříme je, změříme i dvojici hran, která se nabízí pro výměnu a pokud je nová dvojice kratší, hrany vyměníme. (Keld, 2006)

Na obrázku je znázorněný příklad výměny dvou hran. Nalevo je původní graf. Modře jsou zvýrazněné hrany původního grafu, nad kterými se uvažuje výměna. Šedě jsou

označeny hrany, které jsou alternativní k vybraným hranám. Protože šedé hrany jsou zde v součtu kratší než modré, došlo k jejich výměně a napravo je už optimalizovaný graf.

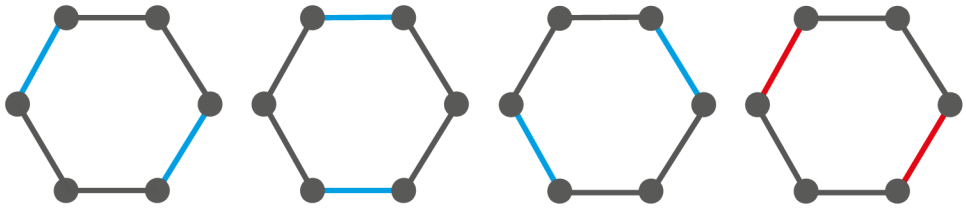


Obrázek 8 - Princip metody 2-opt, zdroj: vlastní zpracování

Počet možností pro výběr dvou hran v grafu můžeme spočítat jako $x = \frac{n(n-3)}{2}$, kde uvažujeme n jako počet měst. U sudého i lichého počtu vrcholů (tedy i hran) je vzorec pro výpočet počtu možností totožný. (Misevicius, 2011) (Cook, 2012)

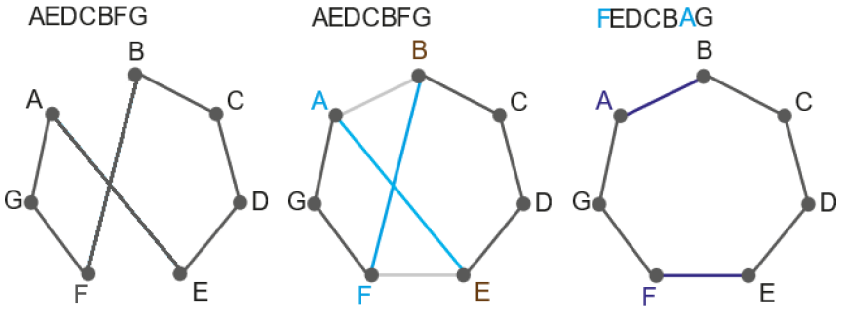
Dvě hrany pro optimalizaci spolu nesmí sousedit, pro takové dvě hrany není možné nalézt jiné dvě alternativní hrany. Následně vybíráme dvojice tak, že vybereme všechny dvojice hran, které jsou od sebe vzdálené jednu hranu, pak všechny dvojice, které jsou od sebe vzdálené dvě hrany a takto přidáváme o jednu hranu, dokud počet hran mezi vrcholy nedosáhne maximálnímu možnému počtu hran pro daný graf. Maximální počet hran mezi dvěma vybranými hranami pro 2-opt je možné u grafu s lichým počtem vrcholů vyjádřit jako $\frac{n-3}{2}$ a u grafu se sudým počtem vrcholů jako $\frac{n-2}{2}$. Počet dvojic pro každý možný počet hran odpovídá počtu měst, například u grafu s šesti vrcholy bude počet dvojic vzdálených od sebe o jednu hranu roven šesti. U grafu se sudým počtem vrcholů je třeba při výběru dvojic se svou maximální vzdáleností vybrat jen polovinu možností, protože zde nastane situace, kdy vybíráme hrany, které mají mezi sebou na obou stranách stejný počet vrcholů. Při zakreslení grafu do pravidelného n-úhelníku by se jednalo o hrany, které jsou naproti sobě, jak je znázorněno na obrázku níže. Například u grafu s šesti vrcholy je totožné, jestli vybereme první a čtvrtou hranu anebo čtvrtou a první. U grafu s lichým počtem vrcholů k této symetrii nedochází. To znamená, že pro úlohy se sudým počtem měst je úloha nepatrně méně náročná, ale jedná se o zanedbatelné přilepšení. Na obrázku níže je znázorněný výběr hran v grafu se sudým počtem měst, kdy jsou ve výběru jen varianty s dvojicemi vzdálenými od sebe maximálním možným počtem hran dle vzorce. Červeně je označena varianta, kterou již není třeba vybrat pro výpočet, protože tato kombinace již byla vybrána. U úlohy s šesti

městy je tedy vybráno šest dvojic, které jsou od sebe vzdálené jednu hranu a tři dvojice, které jsou od sebe vzdálené tři hrany, to je v součtu devět. U úlohy se sedmi městy by bylo vybráno sedm dvojic vzdálených od sebe jednu hranu a sedm dvojic od sebe vzdálených dvě hrany, to je v součtu čtrnáct. (Misevicius, 2011) (Cook, 2012)



Obrázek 9 - Výběr hran v sudém grafu u 2-opt, zdroj: vlastní zpracování

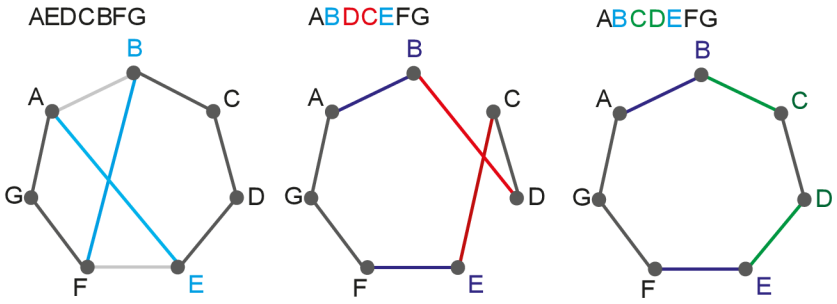
Pro výměnu hran je možné prohodit vrcholy v posloupnosti vrcholů tvořící cestu. K prohození volíme libovolné dva vrcholy, které nejsou incidentní s původními hranami, ani s alternativními neboli novými hranami. Máme tedy vždy dvě možnosti na výběr. Obrázek znázorňuje výměnu dvou vrcholů tak, aby vznikla cesta s prohozenými hranami. Na příkladu z obrázku je možné prohodit vrchol A s F, nebo vrchol B s E, ale není možné navzájem prohodit vrcholy F s B, nebo třeba F s E. (Misevicius, 2011)



Obrázek 10 - Prohození vrcholů u 2-opt, zdroj: vlastní zpracování

Pokud se chystáme prohodit vrcholy, které jsou od sebe vzdálené tři nebo více hran, je třeba v úseku mezi těmito vrcholy otočit pořadí vrcholů. Je to potřeba proto, aby vyměněné hrany navazovaly na hrany, na které původně navazovaly. Na obrázku je znázorněna situace, kdy je třeba změnit pořadí vrcholů v určitém úseku. Původně byla cesta vrcholů v pořadí AEDCBFG, po výměně vrcholů je cesta ABCDEFG, to znamená,

že došlo k prohození vrcholů E s B, následně bylo otočené pořadí vrcholů v úseku mezi B a E, což jsou v tomto případě jen dva vrcholy D a C. (Misevicius, 2011)



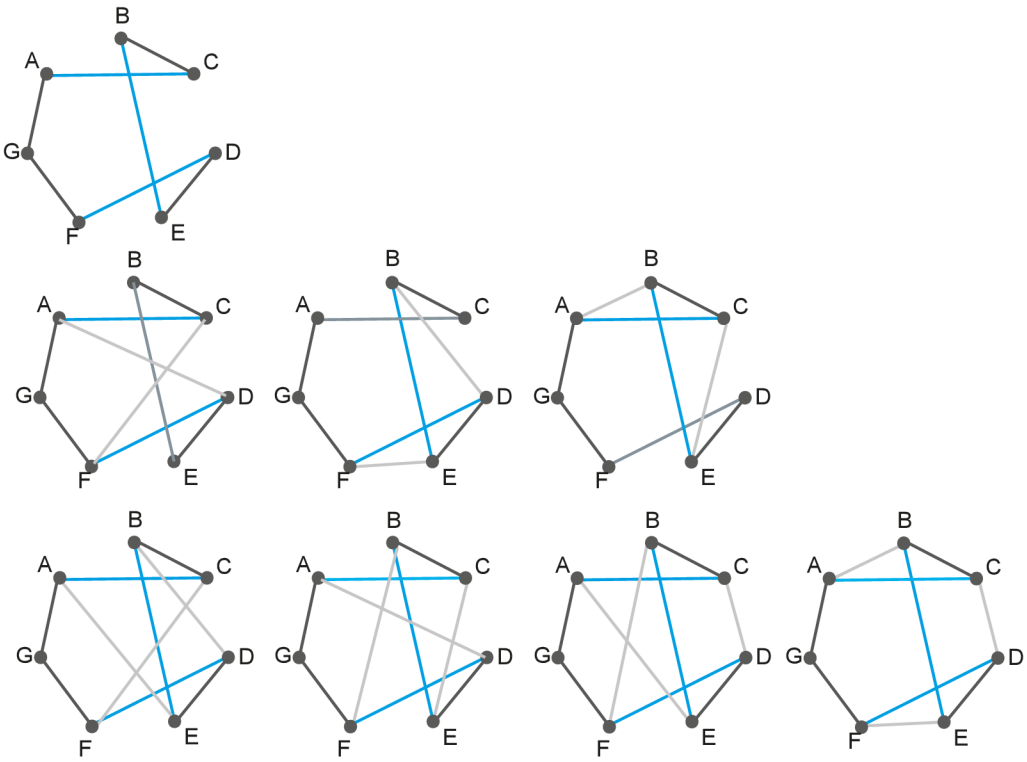
Obrázek 11 - Změna pořadí vrcholů u 2-opt, zdroj: vlastní zpracování

Algoritmus iteruje, dokud už nemá co vylepšovat. Po tom, co tedy projde všechny dvojice a prohodí nějaké vrcholy, tak na vylepšeném grafu projde zase všechny dvojice a když už v dané iteraci nevymění žádné hrany, tak se ukončí. Téměř vždy nalezne lepší řešení než původní, je rychlý a efektivní. (Cook, 2012)

3.6.3.2 Metoda 3-opt

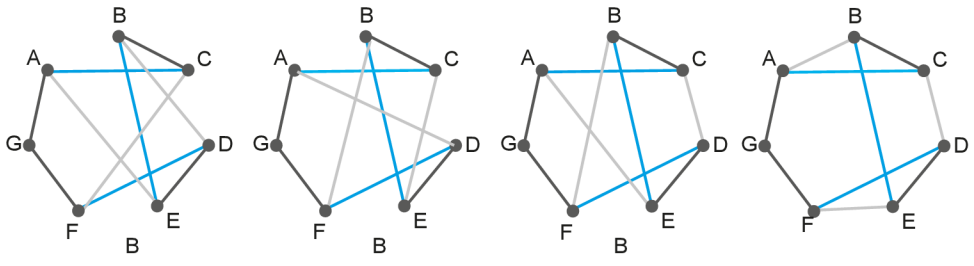
Misevicius (2011) uvedl, že metoda 3-opt má asymptotickou časovou složitost $O(n^3)$.

Tato metoda funguje obdobně jako 2-opt. Namísto dvou hran, ale vždy vyměňujeme tři hrany. Pokud vybereme tři hrany v grafu, máme vždy osm možností, jak vybrat alternativní tři hrany, kterými se případně nahradí původní hrany tak, aby výsledný graf byl po záměně hran souvislý a tvořil hamiltonovskou kružnici. 3-opt tedy v sobě obsahuje i všechny možné výměny, které se nabízejí ve 2-opt, spouštět 3-opt zároveň s 2-opt by nepřineslo lepší výsledky oproti samotnému 3-opt. Na obrázku je znázorněno všech osm možností, respektive alternativ, které mohou tvořit okruh namísto výchozího původního okruhu pro jednu trojici hran. Mezi osm variant se započítávají i výchozí vybrané tři hrany. Výpočet 3-opt je tedy znatelněji náročnější oproti výpočtu 2-opt, kde pro každou dvojici byly jen dvě možnosti. (Misevicius, 2011) (Cook, 2012)



Obrázek 12 - Možnosti výměn hran ve 3-opt, zdroj: vlastní zpracování

Aby byl algoritmus 3-opt rychlejší, je například možné ze zmiňovaných osmi variant vybrat jen ty, kde se zaměňují všechny tři hrany. Tímto způsobem vždy zmenšíme počet alternativních možností k výměně za původní hrany na polovinu. U 3-opt je to sice jen z osmi na čtyři, ale například u 4-opt je to ze 48 na 25. U takto modifikovaného algoritmu se nedá říct, že výsledek je r-optimální, ale takováto nebo podobné modifikace mohou velmi urychlit čas výpočtu. Je dokonce možné složitost $O(n^r)$ snížit až na $O(n)$. (Misevicius, 2011)



Obrázek 13 - Výběr všech tří hran pro 3-opt, zdroj: vlastní zpracování

4 Vlastní práce

Pro porovnání algoritmů je naprogramovaný program ve formě webové aplikace v jazyce C# s uživatelským rozhraním. Tento program je rozšířením programu implementovaného v rámci bakalářské práce zpracované na téma porovnání heuristik problému obchodního cestujícího a optimalizace tras, která je dále rozšířena o metodu mravenčích kolonií a o exaktní algoritmus. Všechny implementované algoritmy jsou metoda nejbližšího souseda, hladový algoritmus, optimalizace mravenčími koloniemi, exaktní algoritmus, náhodně vygenerovaný okruh a metoda zlepšující řešení 2-opt. Program je přizpůsoben pro úlohu okružní cesty po hlavních městech všech států Evropy, kterých je padesát.

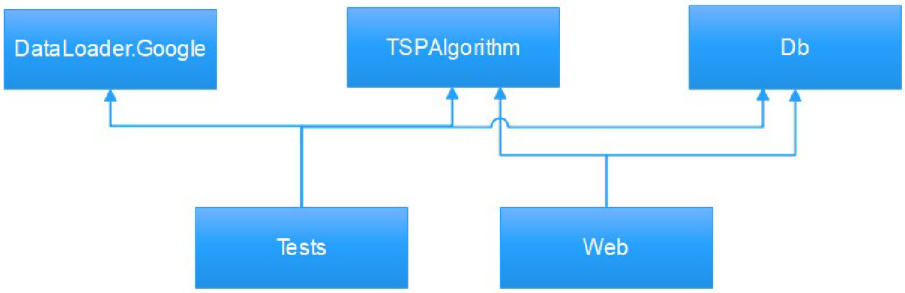
4.1 Implementace programu

Program je implementován jako webová aplikace pro snadnou dostupnost pro uživatele a jednoduchost ovládání. Je vytvořen pomocí jazyka C#. Finální aplikace se skládá z několika knihoven na platformě .NET Standard a webové aplikace typu ASP.NET Core Web App. Architektura webové aplikace je tvořena pomocí návrhového vzoru MVC (Model/View/Controller), pro tvorbu HTML šablon je použit šablonový systém Razor. Data jsou uchovávána v souboru ve formátu JSON. (Anderson, Brock, & Larkin, 2011)

4.1.1 Architektura aplikace

Diagram znázorňuje závislosti mezi jednotlivými komponentami celé aplikace nazvané Optiroute. Knihovna DataLoader.Google obsahuje metody pro volání Google API. V knihovně TSPAlgorthm jsou implementovány algoritmy pro řešení problému obchodního cestujícího. Knihovna Db obsahuje datový model pro uložení informací o adresách a cestách mezi nimi a jejich vzdálenostech. Projekt Tests obsahuje testy pro volání Google API, také je zde metoda pro vygenerování vzdáleností mezi všemi městy do formátu JSON za využití struktury tříd a kolekcí v projektu Db. Projekt Web je webová aplikace spustitelná na serveru, obsahuje všechny nezbytné struktury a soubory pro fungování webové aplikace. Pro vypočítání trasy jsou naimplementovány různé algoritmy (například mravenčí kolonie)

z TSPAlgorthm. Web také využívá knihovnu Db pro práci s datovým modelem, například načítání pojmenování města/adresy dle ID místa.



Obrázek 14 - Diagram závislostí v projektu, zdroj: vlastní zpracování

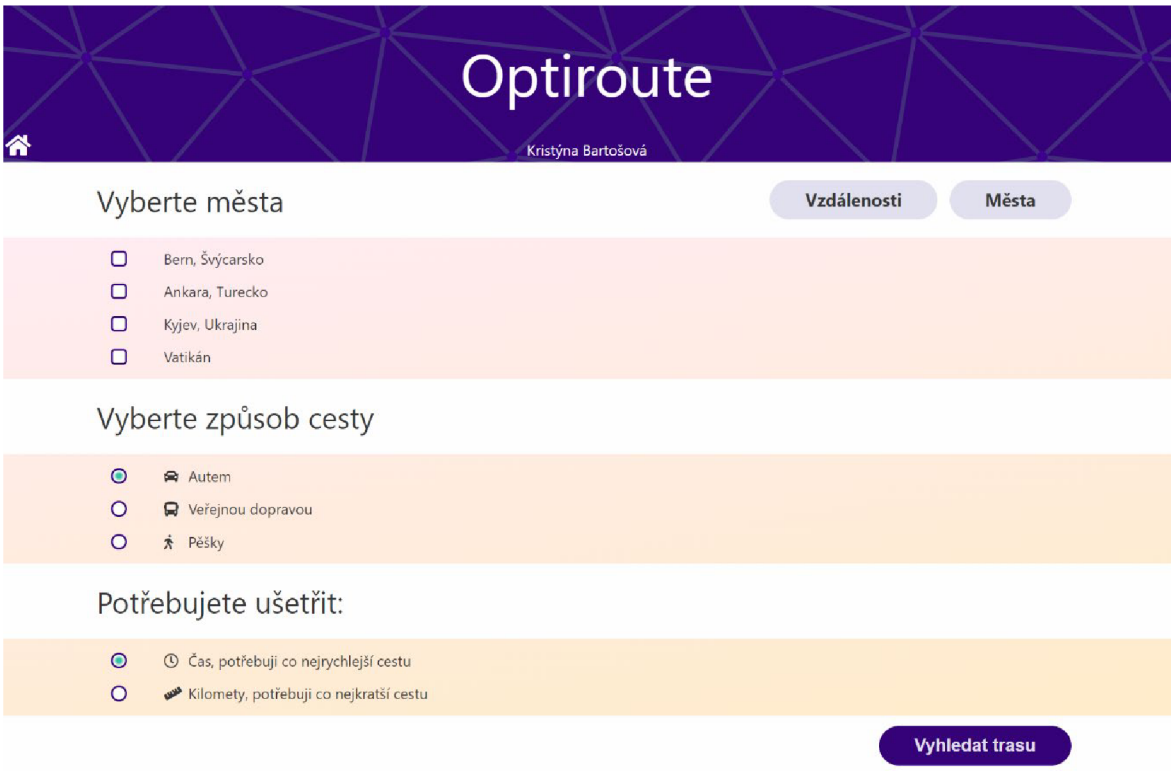
4.1.2 Google API

Pro získání vzdáleností mezi městy je využíváno API od Google, konkrétně Geocoding API pro získání id adresy a Distance Matrix API pro získání všech vzdáleností měst mezi sebou. Jednotlivé vzdálenosti jsou získány v různých režimech dopravy a to autem, veřejnou hromadnou dopravou a pěšky. U varianty veřejnou hromadnou dopravou nebylo nalezeno spojení pro každé město, na úloze o všech padesáti státech Evropy ji tedy nelze použít. Vzdálenosti jsou měřeny jak v metrických, tak v časových jednotkách. K získání potřebných vzdáleností a časů bylo použito API z toho důvodu, že pro padesát měst by muselo být ručně prováděno 7 350 měření, vhledem k tomu, že je každá trasa měřena v časových i metrických jednotkách a je zde počítáno se třemi způsoby dopravy. Další výhodou je, že je díky využití API možné údaje jednoduše upravit a aktualizovat. Déle by díky API bylo případně možné aplikaci rozšířit tak, aby hledaná trasa vždy odpovídala aktuální dopravní situaci. (Google, 2023)

4.1.3 Uživatelské rozhraní aplikace

Aplikace byla pojmenována Optiroute. Uživatel si pomocí grafického uživatelského rozhraní webové aplikace volí jednotlivá města pomocí zaškrtávacích polí. Následně si je možné přepínači zvolit způsob dopravy, zdali se jedná o jízdu autem, chůzi pěšky anebo přepravu veřejnou dopravní obsluhou. Zdali uživatel potřebuje najít co nejrychlejší trasu anebo co nejkratší trasu, si volí také za pomoci přepínačů.

Na obrázku níže je screenshot webové aplikace, počet měst je zde pro ilustraci menší než ve skutečnosti.

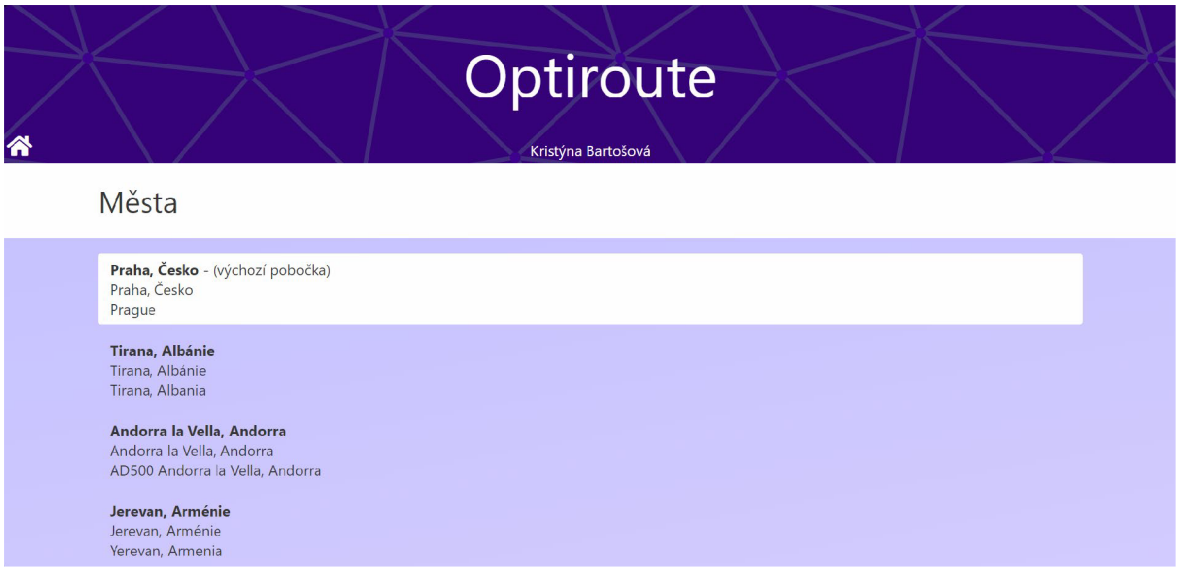


Obrázek 15 - Rozhraní titulní stránky aplikace, zdroj: vlastní zpracování

Po tom, co si uživatel vše navolí, tak si může nechat vyhledat trasu, kdy se provedou všechny algoritmy pro výpočet optimální trasy. Na další stránce se zobrazí nejlepší nalezená trasa vybraných poboček a její délka. Pro zobrazení podrobných výsledků různých algoritmů si je možné rozkliknout detailní výstup tlačítkem s popiskem Zobrazit více. Na obrázku je stránka s nejkratší nalezenou trasou.

Obrázek 16 - Rozhraní stránky s nalezenou trasou, zdroj: vlastní zpracování

Pro více informací si je možné zobrazit seznam všech měst, kde jsou zobrazené přesné adresy daných míst, v tomto případě padesáti evropských měst.



Obrázek 17 - Seznam měst ve webové aplikaci, zdroj: vlastní zpracování

Pro přehled, jak daleko jsou jednotlivá města od sebe vzdálená, si je možné zobrazit tabulku, kde je zobrazena jak metrická, tak časová vzdálenost mezi všemi městy, a to všemi dopravními prostředky. Na úloze o padesáti měst se jedná o tři tabulky, kdy každá z nich obsahuje 1 275 řádků.

Z místa	Do místa	Čas (min)	Vzdálenost (m)
Praha, Česko	Tirana, Albánie	17 h 17 min	1620,4 km
Praha, Česko	Andorra la Vella, Andorra	17 h 14 min	1691,3 km
Praha, Česko	Jerevan, Arménie	1 d 15 h 25 min	3602,1 km
Praha, Česko	Baku, Ázerbájdžán	1 d 20 h 41 min	3548,9 km
Praha, Česko	Bruselas, Belgie	9 h 13 min	899,0 km
Praha, Česko	Minsk, Bělorusko	12 h 46 min	1218,4 km
Praha, Česko	Sarajevo, Bosna a Hercegovina	11 h 5 min	1040,4 km
Praha, Česko	Sofie, Bulharsko	12 h 47 min	1283,7 km
Praha, Česko	Podgorica, Černá Hora	15 h 21 min	1330,9 km

Obrázek 18 – Tabulka vzdáleností mezi městy v aplikaci, zdroj: vlastní zpracování

4.2 Implementace algoritmů pro řešení problému obchodního cestujícího

V programu jsou implementovány heuristické algoritmy, a to metoda nejbližšího souseda, hladový algoritmus a mravenčí kolonie. Cesta získaná těmito heuristikami se následně ještě vylepšuje pomocí algoritmu 2-opt. Dále je součástí programu exaktní algoritmus, který projde všechny možnosti. Také byl implementován algoritmus pro náhodně vybranou cestu, který je také následně vylepšen pomocí metody 2-opt.

Všechny algoritmy mají na vstupu matici reprezentovanou jako dvourozměrné celočíselné pole, kde jednotlivé indexy reprezentují místa a hodnota na zvolených indexech reprezentuje vzdálenost míst mezi sebou. Matice je symetrická, to znamená, že na pozici $[i, j]$ je stejná hodnota jako na pozici $[j, i]$, reprezentuje tedy neorientovaný graf. Dále je na vstupu kolekce, která má na svých indexech uložené jednoznačné identifikátory (ID) míst tak, jak jsou uložené v souboru JSON. Tato kolekce nám umožňuje zjistit podle hodnoty indexu, reprezentující nějaké město, ID tohoto města a díky tomu například vypsat uživateli názvy měst v nalezené cestě. Dalším vstupním údajem je ID města, které je výchozí, což je v tomto případě nastavené na Prahu.

Zlepšující metoda 2-opt má ještě navíc na vstupu trasu pro zlepšení, tato trasa je reprezentována posloupností jednotlivých ID míst.

Výstupem je vždy posloupnost ID míst, celková délka trasy a čas výpočtu algoritmu. Čas výpočtu je počítán včetně veškerých čtení a zápisů do kolekcí, počítání délek, přeskupování pořadí ID míst, tak aby se vždy začínalo ve výchozí pobočce a tak dále. U optimalizačních algoritmů není do celkového času výpočtu počítán výpočet vstupní cesty k vylepšení.

4.2.1 Implementace exaktního algoritmu

Na začátku je deklarovaný list obsahující seznam měst, nad kterým se úloha počítá. Dále je deklarovaná proměnná pro uchování posloupnosti měst, které zatím tvoří nejkratší okruh a číselná proměnná pro uchovávání její vzdálenosti.

Pomocí rekurzivního volání jsou postupně prohledány všechny existující okruhy v grafu a pokud je nově nalezená cesta kratší než doposud nalezená trasa, tak se uloží včetně její délky. Algoritmus skončí pro projití všech možností.

```
List<int> placeIDs = matrixRoute.placeIDs.Where(d => !(d ==
matrixRoute.DefaultPlaceID)).ToList();
List<int> currentBestRoute = new List<int>();
int currentBestlengthOfRoute = int.MaxValue;
CalculateAllVariants(new List<int>(), placeIDs);
void CalculateAllVariants(List<int> currentPartialRoute, List<int>
placeIDsLeft)
{
	if (placeIDsLeft.Count == 0)
	{
		var currentRoute = new List<int>();
		currentRoute.Add(matrixRoute.DefaultPlaceID);
		currentRoute.AddRange(currentPartialRoute);
		currentRoute.Add(matrixRoute.DefaultPlaceID);
		var length = CalculateRouteLength(matrixRoute, currentRoute);
		if (currentBestlengthOfRoute > length)
		{
			currentBestlengthOfRoute = length;
			currentBestRoute = currentRoute;
		}
		return;
	}
	for (int i = 0; i < placeIDsLeft.Count; i++)
	{
		var newCurrentPartialRoute = currentPartialRoute.ToList();
		newCurrentPartialRoute.Add(placeIDsLeft[i]);
		var newPlaceIDsLeft = placeIDsLeft.ToList();
		newPlaceIDsLeft.RemoveAt(i);
		CalculateAllVariants(newCurrentPartialRoute, newPlaceIDsLeft);
	}
}
```

Obrázek 19 - Implementace exaktního algoritmu, zdroj: vlastní zpracování

4.2.2 Implementace optimalizace mravenčími koloniemi

V algoritmu mravenčích kolonií jsou na začátku deklarovány matice aproximace a matice feromonů. Do matice feromonů je nastavena výchozí hladina na 300. Pro naplnění matice aproximace je použit výpočet, kdy se každá vzdálenost mezi městy vydělí hodnotou 1 500 000. Tato hodnota byla vybrána tak, aby se průměr výsledku přibližoval tomu, jakou hladinu má většinou po dobu výpočtu hladina feromonů. Matice feromonů se nadále v algoritmu průběžně aktualizuje, ale matice aproximace se již nemění. Dále je vytvořená proměnná pro ukládání doposud nejlepší nalezené cesty. Ve zdrojovém kódu níže je implementována iterace, která má počet opakování nastaven na 2 000, toto číslo uvádí kolikrát jsou mravenci vysláni a také kolikrát dojde k aktualizaci matice feromonů. Počet mravenců je nastavený jako dvojnásobek počtu měst. Mravenci jsou po jednom vysláni metodou `SendTheAnt`, aby každý z nich udělal okruh. Z každého města začínají svou trasu dva mravenci. Pro zrychlení výpočtu jsou mravenci vysíláni paralelně, to znamená, že je vysláno několik mravenců najednou a počítač, na kterém je algoritmus spuštěn, využívá pro výpočet všechny dostupná vlákna procesoru. Tato úprava zrychlila výpočet přibližně o třetinu, nicméně zrychlení je závislé na počítači, na kterém je algoritmus spouštěn.

Cesta/okruh každého mravence je uložena pro pozdější použití k aktualizaci feromonů. V případě, že vyslaný mravenec šel cestou, jejíž celková délka je menší, než je doposud nalezená nejkratší cesta, je tato cesta uložena jako nejlepší, dokud další mravenec nenajde lepší řešení. Po tom, co každý mravenec udělal svůj okruh, se zavolá metoda `RecalculateFheromons` pro aktualizaci feromonů.

```
//Počet iterací.
int numberOfIterations = 2000;
for (int x = 0; x < numberOfIterations; x++)
{
	//Všechny cesty, kudy mravenec šel.
	var antPaths = new ConcurrentBag<List<int>>();
	Parallel.For(0, NumberOfCities, (int i) =>
	{
		Parallel.For(0, 2, (int iteration) =>
		{
			//Pošle se mravenec a jeho cesta se uloží.
			List<int> antPath = SendTheAnt(matrixRoute, feromonMatrix,
			approximationMatrix, i);
			antPaths.Add(antPath);
			//Uložení cesty, pokud je lepší než dosud nejlepší.
			int lenghtOfAntRoute = GetFullRouteLength(antPath,
			matrixRoute.matrixRoute);
			if (currentBestlengthOfRoute > lenghtOfAntRoute)
			{
				lock (lockObject)
				{
				if (currentBestlengthOfRoute > lenghtOfAntRoute)
					{
					  currentBestRoute = antPath;
					  currentBestlengthOfRoute = lenghtOfAntRoute;
					}
				}
			}
		}
	});
});
//Přepočítání feromonů
RecalculateFheromons(matrixRoute, feromonMatrix, antPaths.ToList(),
currentBestlengthOfRoute);
```

Obrázek 20 - Implementace mravenčích kolonií, zdroj: vlastní zpracování

Metoda `SendTheAnt` vrací list obsahující posloupnost měst za sebou, která znázorňuje okruh, kudy mravenec šel. Přes vstupní parametr je poslané výchozí město, odkud je mravenec vyslán. List, ve kterém se tvoří okruh, je tedy nejdříve naplněný tímto městem.

Pro výpočet, kam se mravenec vydá dál, je potřeba znát hodnotu všech tuh do okolních měst, k tomu slouží cyklus, který vypočítá touhy pro všechna okolní neobsazená města a sečte je. Touha mravence jít do konkrétního sousedního města je implementována jako násobek intenzity feromonů na cestě mezi městy a aproximované vzdálenosti na druhou. To znamená, že parametr β je větší než α, tím pádem se mravenec v tomto algoritmu spíše rozhoduje na základě vzdáleností měst než na základě intenzity feromonů. Nastavením β na 2 a α na 1 dosahují mravenčí kolonie nejlepších výsledků v případě této úlohy.

Abychom znali pravděpodobnost navštívení každého ještě nenavštíveného sousedního města, je provedený opět cyklus, který získá pravděpodobnost vydělením touhy mravence jít do města se součtem všech tuh.

Následně je vygenerováno pseudonáhodné číslo v intervalu od 0 do 1. Také je stanovená úroveň, která začíná na hodnotě 0. Postupně je k hladině přičítána pravděpodobnost navštívení měst. Jakmile je hladina větší než pseudonáhodně vygenerované číslo, přidá se do listu tvořící okruh mravence město, kterému náleží poslední přičtená touha. Z tohoto přidaného města do listu se mravenec opět rozhoduje, kam půjde dál tím stejným způsobem. Celý proces se opakuje a postupně je takto vytvořený celý okruh, kudy mravenec jde a metoda vrátí výsledek.

Metoda `RecalculateFheromons` se volá v každé iteraci programu. Na vstupu má počáteční matici intenzity feromonů, seznam všech cest mravenců, kterých je v případě cesty přes státy Evropy 100, protože států je 50 a v každé iteraci jsou poslány dvě sady mravenců. Na výstupu metoda vrací aktualizovanou matici feromonů. Matice je reprezentovaná jako dvojrozměrné pole.

Hned zprvu metody `RecalculateFheromons` je zavolaná metoda `EvaporationOfFeromons`, která sníží hladinu feromonů a simuluje tím vyprchávání feromonů na cestách, jako tomu je u skutečných mravenců. Je deklarována číselná proměnná nazvaná `feromonEvaporation`, která určuje kolik procent feromonů po vyprchání zůstane z původní hodnoty. Tato proměnná je nastavená na 0,36, tedy 36 %. To znamená, že větší část původních feromonů vyprchá, než zůstane. Tato hladina umožňuje mravencům zapomenout na první nalezené řešení v první nebo v prvních iteracích a mravenčí kolonie tedy dokáží ještě v průběhu výpočtu svou cestu vylepšit. Pro vylepšení algoritmu je nastaveno, že hladina feromonů u žádné cesty nemůže být nižší než 0,00001, což je nízká hodnota, protože u cest, které jsou velmi používané se hodnota intenzity feromonů dokáže vyšplhat až k tisícům. Nastavení spodní hranice intenzity feromonů může pomoci rozšířit okruh prohledání grafu, nicméně při nastavené hladině na vyšší hodnotu má algoritmus na této úloze horší výsledky. Nastavení této hladiny slouží k tomu, aby se po mnohých iteracích hladina feromonů u nepoužívaných cest nedostala na tak malé číslo, že ho program začne považovat za nulu. To by mohlo způsobit komplikace u výpočtu pravděpodobnosti,

do jakého města se mravenec vydá, protože suma všech tuh by se mohla v některých případech rovnat nule a dělení nulou není povoleno.

Po vyprchání feromonů program pokračuje v těle metody `RecalculateFheromons`, kde načte jeden z okruhů mravenců a celkovou délku tohoto okruhu. Celková délka okruhu má hlavní vliv na následnou produkci feromonů na cestách tohoto okruhu. Produkce feromonů je vypočtena jako podíl konstanty Q a délky okruhu, kde konstanta Q je nastavena na hodnotu 5 000 000.

U produkce feromonů jsou implementována zlepšení. Výsledky produkce feromonů si byly často podobné, například u jednoho mravence se jednalo o produkci 2,01 a u dalšího 2,05 atd. Pro zvětšení rozdílů mezi úspěšnými a neúspěšnými mravenci je naimplementované zlepšení, kdy produkce feromonů je umocněna na třetí. Dále je násobně zvětšená produkce feromonů u mravenců, kteří našli v pozdější fázi algoritmu lepší řešení, než bylo do té doby známé, tento krok by mohl pomoci mravencům se rychleji adaptovat na nové lepší nalezené řešení.

Následně je pomocí for cyklu přičtena na každou spojnici dvou měst, kudy mravenec šel, vypočtená hodnota produkce feromonů mravence. Poté se načte další okruh mravence. Opět se spočítá produkce feromonů, která se do matice feromonů přičte na místo cesty, kudy mravenec šel. Až se do matice feromonů přičtou feromony od všech mravenců, je metoda ukončena.

4.2.3 Implementace metody nejbližšího souseda

Algoritmus pomocí rekurzivního volání postupně najde potencionální nejlepší trasu z každého místa zaslaného na vstupu. Jednotlivé nalezené trasy jsou přidány do kolekce, kde jsou uloženy v podobě posloupnosti měst reprezentovanou indexy ze vstupní matice a dále je zde rovnou uložena délka cesty, která se sčítá již v rámci zařazování jednotlivých míst do výsledné cesty.

V každé iteraci for cyklu se deklaruje zásobník, do kterého se vždy zařadí pobočka, která je pro tu danou iteraci výchozí. Dále je zavolána metoda `FindNextCity`, které se jako parametr předá již deklarovaný zásobník a další parametry potřebné k výpočtu. V momentě, kdy jsou všechny cesty spočítány, se vybere ta nejkratší, která je řešením metody nejbližšího souseda. Přesné řešení je vidět v C# kódu níže.

```
public override RouteTSPResult ComputeTSPRoute(MatrixRoute matrixRoute)
{
	Stopwatch stopwatch = new Stopwatch();
	stopwatch.Start();
	List<NSRoute> variousNSRoutes = new List<NSRoute>();

	for (int i = 0; i < matrixRoute.placeIDs.Count(); i++)
	{
		int routeLength = 0;
		Stack<int> indexRoute = new Stack<int>(matrixRoute.placeIDs.Count());

		indexRoute.Push(i);
		FindNextCity(indexRoute, routeLength, variousNSRoutes, matrixRoute);
	}

	var finalLenght = variousNSRoutes.Min(d => d.NSRouteLenght);
	var finalNSRoute = variousNSRoutes.FirstOrDefault(d => d.NSRouteLenght ==
	finalLenght).IndexNSRoute;
	stopwatch.Stop();
	return new RouteTSPResult(){/*-----*/};
}
public class NSRoute
{
	public List<int> IndexNSRoute { get; set; } = new List<int>();
	public int NSRouteLenght { get; set; }
}
```

Obrázek 21 - Implementace metody nejbližšího souseda, cyklus pro vytvoření cesty z každé pobočky, zdroj: vlastní zpracování

Metoda `FindNextCity` již nalézá trasu pomocí metody nejbližšího souseda. Nejprve je nalezena množina měst, které ještě nejsou součástí doposud nalezené cesty a zároveň kam je cesta z posledního navštíveného místa nejkratší. Každé místo z množiny nejbližších měst je postupně zařazeno do řešení. Pro zjištění následujícího nejbližšího města je metoda opět rekurzivně zavolána. Pokud trasa již obsahuje všechna města, uloží se do kolekce s nalezenými trasami i s hodnotou své délky. Níže je ukázka rekurzivní metody.

```
private void FindNextCity(Stack<int> indexRoute, int routeLength, List<NSRoute>
variousNSRoutes, MatrixRoute matrixRoute)
{
	List<int> minIndexes = new List<int>();
	int minLength = int.MaxValue;

	for (int j = 0; j < matrixRoute.placeIDs.Count(); j++)
	{
		if (indexRoute.Any(d => d == j)) continue;
		int length = matrixRoute.matrixRoute[indexRoute.Peek(),j];

		if (length < minLength)
		{
			minLength = length;
			minIndexes = new List<int>() { j };
		}
		else if (length == minLength) minIndexes.Add(j);
	}

	foreach (var a in minIndexes)
	{
		indexRoute.Push(a);

		if (indexRoute.Count() == matrixRoute.placeIDs.Count())
		{
			variousNSRoutes.Add(new NSRoute() { IndexNSRoute =
			indexRoute.ToList(), NSRouteLenght = routeLength + minLength +
			matrixRoute.matrixRoute[indexRoute.Peek(), indexRoute.Last()]
			});
		}
		else FindNextCity(indexRoute, routeLength + minLength, variousNSRoutes,
		matrixRoute);

		indexRoute.Pop();
	}
}
```

Obrázek 22 - Rekurzivní metoda metody nejbližšího souseda, zdroj: vlastní zpracování

4.2.4 Implementace hladového algoritmu

Pro hladový algoritmus potřebujeme seřazený seznam vzdáleností mezi všemi městy a tyto cesty se vzdálenostmi postupně přidáváme do řešení. Na začátku algoritmu je tedy ze vstupní matice vytvořen seznam cest, kde je ke každé cestě uložena vzdálenost a obě místa, mezi kterýma byla daná vzdálenost naměřena. Tento seznam je následně seřazen vzestupně podle vzdálenosti. V metodě je také deklarována kolekce, do které se v průběhu zařazování cest do řešení zaznamenávají stupně vrcholů neboli měst. Kontrolou stupňů vrcholů se zabrání tomu, aby se z jednoho města vybraly více než dvě cesty, kružnice má totiž všechny vrcholy právě druhého stupně. Aby se zabránilo tvoření více menších

kružnic namísto jedné přes všechny vrcholy, je zde vytvořena kolekce, kde se zaznamenávají skupiny vrcholů. Tato kolekce reprezentuje komponenty souvislosti grafu při tvoření řešení. V rámci while cyklu je vždy vybrána následující nejkratší cesta z kolekce seřazených cest. Zkontroluje se, jestli by zařazením cesty do řešení nevznikly vrcholy většího stupně než dva, nebo jestli by nevznikla kružnice délky menší, než je počet vrcholů na vstupu. Pokud jsou podmínky splněny, je cesta zařazena do řešení. Ze seznamu cest je následně potřeba vytvořit seřazenou posloupnost měst tvořící výslednou kružnici. Níže je část kódu metody hladového algoritmu.

```
List<RouteIndex> orderedIndexRoutes = indexRoutes.OrderBy(d =>
d.RouteLentght).ToList();
int[] citiesGoups = new int[matrixRoute.placeIDs.Count()];
int[] citiesDeg = new int[matrixRoute.placeIDs.Count()];
List<RouteIndex> greedyRoutes = new List<RouteIndex>();
int counter = 0;
while (greedyRoutes.Count < matrixRoute.placeIDs.Count())
{
       int fromRouteIndex = orderedIndexRoutes[counter].FromIndex;
       int toRouteIndex = orderedIndexRoutes[counter].ToIndex;
       int fromGroup = citiesGoups[fromRouteIndex];
       int toGroup = citiesGoups[toRouteIndex];
       if(/*Vrátí true, když do řešení potřebujeme zařadit poslední cestu.*/) {}
       else if (((fromGroup == toGroup && fromGroup != 0) && greedyRoutes.Count !=
       matrixRoute.placeIDs.Count() - 1) || (citiesDeg[fromRouteIndex] + 1 > 2 ||
       citiesDeg[toRouteIndex] + 1 > 2))
        { counter++; continue; }
        greedyRoutes.Add(orderedIndexRoutes[counter]);
        citiesDeg[fromRouteIndex]++;
        citiesDeg[toRouteIndex]++;
        //Výpočet skupiny míst
      counter++;
}
```

Obrázek 23 - Implementace hladového algoritmu, zdroj: vlastní zpracování

4.2.5 Implementace algoritmu 2-opt

Pro optimalizaci pomocí algoritmu 2-opt je třeba určit, jaké všechny kombinace výběru dvou hran mohou nastat. Jako definici jedné kombinace je vytvořena třída, která má definované čtyři vrcholy reprezentující dvě hrany v grafu. V instanci třídy nejsou vrcholy definovány pomocí ID míst, ale jsou reprezentovány indexy v kolekci posloupností vrcholů určující trasu. Možností, jak vybrat dvě hrany, máme v grafu mnoho, proto byla vytvořena kolekce těchto instancí tříd. Níže je ukázána implementace této třídy a kolekce.

```
List<TwoRoutes> cobinationsOFTwoRoutes = new List<TwoRoutes>();

private class TwoRoutes
{
	public int FromRoute1 { get; set; }
	public int ToRoute1 { get; set; }
	public int FromRoute2 { get; set; }
	public int ToRoute2 { get; set; }
}
```

Obrázek 24 - Implementace kolekce kombinací výběru dvou hran pro 2-opt, zdroj: vlastní zpracování

Možné kombinace toho, jak vybrat dvě hrany, na kterých je možné provést výměnu za jiné dvě hrany, se odvíjí od počtu vrcholů. V programu jsou kombinace nalézány pomocí for cyklu, který iteruje tolikrát, kolik je maximální počet hran mezi libovolnýma dvěma hranama v grafu s tím, že vždy vybereme ten menší počet. V tomto for cyklu je vnořený for cyklus, kde ve většině případů iteruje nad každým vrcholem (přesněji indexem vrcholu), k tomuto vrcholu se pro utvoření hrany vybere vždy následující vrchol, což je vrchol s indexem o jedna větší. Jako druhá hrana k té první je vybrána taková hrana, která je vzdálena od té první hrany o tolik hran, kolikrát již proběhl první cyklus. Vybrané hrany se uloží do zmíněné kolekce kombinací výběru dvou hran. Níže je uvedena přesná implementace.

```
int maxSkipRoute = Convert.ToInt32(Math.Floor((matrixRoute.placeIDs.Count() - 2) /
2d));
if (matrixRoute.placeIDs.Count() == 3) maxSkipRoute = 0;

for (int i = 1; i <= maxSkipRoute; i++)
{
	int combinationNumber = matrixRoute.placeIDs.Count();
	if (combinationNumber % 2 == 0 && i == maxSkipRoute) combinationNumber =
	combinationNumber / 2;
	for (int j = 0; j < combinationNumber; j++)
	{
		cobinationsOFTwoRoutes.Add(new TwoRoutes()
		{
			FromRoute1 = j,
			ToRoute1 = (j + 1) % matrixRoute.placeIDs.Count(),
			FromRoute2 = (j + 1 + i) % matrixRoute.placeIDs.Count(),
			ToRoute2 = (j + 1 + i + 1) % matrixRoute.placeIDs.Count()
		});
	}
}
```

Obrázek 25 - Implementace zjišťování všech možností výběru dvou hran pro 2-opt, zdroj: vlastní zpracování

Zbývá projít uložené kombinace na konkrétní cestě. Tato cesta k optimalizaci musí být tvořena posloupností indexů pro získání vzdálenosti mezi dvěma vrcholy v matici vzdáleností. Algoritmus 2-opt zde prochází a prohazuje všechny možné dvojice hran, tak dlouho, dokud už není co vylepšovat, to je zde rozpoznáno tak, že v poslední iteraci přes všechny kombinace se nenašly žádné dvě hrany k prohození. Jestli se mají dvě hrany prohodit se zjišťuje tak, že se sečte vzdálenost vybraných hran a vzdálenost hran k nim alternativních. Pokud mají alternativní hrany dohromady kratší vzdálenost, dojde k jejich prohození. Takto se tedy cesta vylepšuje tak dlouho, dokud prohazováním dvou hran již řešení nelze vylepšit. Déle je konkrétní implementace.

```
List<int> indexRoute = GetIndexRouteFromIdsRoute(inputRoute.AddressisRoute,
matrixRoute.placeIDs).ToList();

bool processing = true;
while(processing)
{
	processing = false;
	foreach (var a in cobinationsOFTwoRoutes)
	{
		int oldTwoRoutesLength =
		matrixRoute.matrixRoute[indexRoute[a.FromRoute1],
		indexRoute[a.ToRoute1]] +
		matrixRoute.matrixRoute[indexRoute[a.FromRoute2],
		indexRoute[a.ToRoute2]];
		int newTwoRotesLength =
		matrixRoute.matrixRoute[indexRoute[a.FromRoute1],
		indexRoute[a.FromRoute2]] +
		matrixRoute.matrixRoute[indexRoute[a.ToRoute1],
		indexRoute[a.ToRoute2]];
		if (newTwoRotesLength < oldTwoRoutesLength)
		{
			int xchangeFromRoute2 = indexRoute[a.FromRoute2];
			int xchangeToRoute1 = indexRoute[a.ToRoute1];
			indexRoute[a.ToRoute1] = xchangeFromRoute2;
			indexRoute[a.FromRoute2] = xchangeToRoute1;
			int skippedCities = ((a.FromRoute2 + indexRoute.Count() -
			a.ToRoute1) % indexRoute.Count()) - 1;
			if (skippedCities >= 2)indexRoute = ReversePartial(indexRoute,
			a.ToRoute1 + 1, skippedCities);
			processing = true;
		}
	}
}
```

Obrázek 26 - Implementace prohazování dvou hran metodou 2-opt, zdroj: vlastní zpracování

5 Výsledky a diskuse

Jednotlivé algoritmy, kterými jsou náhodně vygenerovaná cesta (Označená jako Random.), hladový algoritmus (zkráceně HA), metoda nejbližšího souseda (NS), mravenčí kolonie (ACO) a metoda zlepšující řešení 2-opt, jsou spouštěny na úloze okružní cesty po hlavních městech všech evropských států, kterých je padesát. Cesta začíná a končí v České republice v Praze. Metoda 2-opt optimalizuje řešení nalezená předchozími algoritmy. Cesta je zkoumaná ve čtyřech variantách, kterými jsou jízda autem, nebo chůze pěšky, obě možnosti jsou vyzkoušeny jak v případě, kdy má být cesta co nejkratší, tak co nejrychlejší.

Kvalita algoritmů je posouzena na základě délky okruhu nalezeného řešení a rychlosti výpočtu.

Mravenčí kolonie jsou specifické tím, že při každém spuštění dosahují odlišných výsledků. Pro posouzení, jak často se mravenčí kolonie uchýlily k lepšímu nebo horšímu suboptimálnímu řešení, jsou spuštěny alespoň patnáctkrát. Při porovnávání s ostatními algoritmy je z těchto patnácti spuštění vybrán ten nejúspěšnější.

Exaktní algoritmus není možné spustit na úloze o padesáti městech, protože by výpočet mohl trvat přibližně $9{,}6\times10^{45}$ let. Je zde zkoumáno, jak dlouho výpočet reálně trvá vzhledem k množství měst zařazených do úlohy. U úloh spočítané exaktním algoritmem jsou jeho výsledky porovnány s výsledky heuristických metod.

5.1 Cesta autem po Evropě

Algoritmy jsou zkoumány na okruhu po hlavních městech padesáti států Evropy, kde je hledaný co nejrychlejší a co nejkratší okruh pěšky nebo jízdou autem.

5.1.1 Co nejkratší cesta autem po Evropě

Tabulka níže je výstupem webové aplikace, je v ní zaznamenáno, jak dlouhý okruh každý v algoritmů našel a jak dlouho trval jeho výpočet.

Před optimalizací pomocí metody 2-opt našla nejkratší cestu jednoznačně metoda mravenčích kolonií (ACO). O přibližně 7 000 km delší okruh našla metoda nejbližšího souseda (NS), hladový algoritmus (HA) nalezl okruh ještě o dalších cca 4 500 km delší. Mravenčí kolonie nalezly trasu lepší přibližně o jednu šestinu než metoda nejbližšího souseda.

Mravenčím koloniím výpočet trval více než půl minuty, což už není vhodná doba výpočtu pro systémy, kdy je žádoucí znát výsledek téměř hned. Pro takovou situaci by bylo možné snížit počet iterací a mravenců, nicméně to by mělo za následek o něco horší výsledky. Je možné i zvýšit počet mravenců nebo iterací, ale na této úloze při této konfiguraci mravenci už své nalezené řešení při více iterací nedokáží opustit a nalezený okruh není lepší, nebo jen nepatrně. Metoda nejbližšího souseda je v porovnání s hladovým algoritmem pomalá, ale obě metody mají výpočet téměř okamžitý.

Náhodně vygenerovaný okruh dosahuje jednoznačně nejhorších výsledků v délce okruhu.

Zlepšující metoda 2-opt vylepšila všechny trasy na podobnou hladinu a to téměř nezávisle na kvalitě vstupního řešení. Například u vylepšení náhodného okruhu je výsledek lepší než u vylepšení výstupu hladového algoritmu. Výpočet trval v každém případě méně než jednu milisekundu.

Kombinace metody nejbližšího souseda s 2-opt nalezla o necelý kilometr kratší okruh než samotné mravenčí kolonie a výpočet metody nejbližšího souseda trval v porovnání s mravenčím koloniemi tisícinu času. Síla této kombinace může být dána tím, že metoda nejbližšího souseda má obecně slabinu v tom, že prochází graf po nejbližších uzlech a v posledních fázích výpočtu zbydou jen ty uzly, které jsou nejvíce vzdálené od všech ostatních, tyto odlehlé uzly jsou tedy přidány do řešení až v tento moment. Tímto způsobem často u metody nejbližšího souseda dochází ke křížení cest. Zlepšující metoda 2-opt naopak ve svém principu fungování nachází a odstraňuje křížící se cesty a tím může nedostatky metody nejbližšího souseda částečně kompenzovat. Je možné se domnívat, že při následném použití metody 3-opt nebo i k-opt by výsledky byly ještě lepší.

Tabulka 2 - Výsledky algoritmů na úloze nejkratší cesty po Evropě autem, zdroj: vlastní zpracování

	Délka (m)	Výpočet ms
Random	104 446 584	< 1
2-opt: Random	35 893 559	< 1
HA	47 045 208	< 1
2-opt: HA	36 135 043	< 1
NS	42 717 561	33
2-opt: NS	35 579 780	< 1
ACO	35 656 403	33 691
2-opt: ACO	**34 946 004**	< 1

Úplně nejlepšího řešení dosáhla kombinace metody 2-opt a mravenčích kolonií i přes to, že zlepšení metodou 2-opt není tak výrazné jako u ostatních metod. Nalezený okruh po Evropě je dlouhý 34 946 km a vede v tomto pořadí:

Praha, Česko → Vaduz, Lichtenštejnsko → Bern, Švýcarsko → Lucemburk, Lucembursko → Bruselas, Belgie → Amsterdam, Nizozemsko → Dublin, Irsko → Londýn, Velká Británie → Paříž, Francie → Lisabon, Portugalsko → Madrid, Španělsko → Andorra la Vella, Andorra → Monaco-Ville, Monako → San Marino, San Marino → Vatikán → Řím, Itálie → Valletta, Malta → Atény, Řecko → Nikósie, Kypr → Ankara, Turecko → Tbilisi, Gruzie → Jerevan, Arménie → Baku, Ázerbájdžán → Nur-Sultan, Kazachstán → Moskva, Rusko → Kyjev, Ukrajina → Kišiněv, Moldavsko → Bukurešť, Rumunsko → Sofie, Bulharsko → Skopje, Severní Makedonie → Tirana, Albánie → Podgorica, Černá Hora → Sarajevo, Bosna a Hercegovina → Bělehrad, Srbsko → Záhřeb, Chorvatsko → Lublaň, Slovinsko → Budapešť, Maďarsko → Bratislava, Slovensko → Vídeň, Rakousko → Varšava, Polsko → Minsk, Bělorusko → Vilnius, Litva → Riga, Lotyšsko → Tallinn, Estonsko → Helsinki, Finsko → Stockholm, Švédsko → Oslo, Norsko → Reykjavík, Island → Kodaň, Dánsko → Berlín, Německo → Praha, Česko

Algoritmus mravenčích kolonií sice nedosahuje výrazně lepších výsledků jeli spuštěný ve více iteracích, ale pro dosažení nejlepšího výsledku bylo potřeba mravenčí kolonie spustit vícekrát, protože pokaždé úlohu vyřeší odlišně. Tabulka níže obsahuje výsledky patnácti instancí mravenčích kolonií. Nejhorší nalezená trasa je o necelých tisíc kilometrů delší než ta nejlepší, což není příliš velké variační rozpětí, ale dostatečně velké na to, aby mělo význam mravenčí kolonie spustit vícekrát.

Tabulka 3 - Výsledky mravenčích kolonií nejkratší cesty autem, zdroj: vlastní zpracování

	Délka (m)	Výpočet ms
ACO	36 530 698	34 391
ACO	36 308 055	31 503
ACO	36 039 103	29 846
ACO	36 145 807	30 733
ACO	35 656 403	33 691
ACO	35 939 485	30 177
ACO	35 960 746	30 880
ACO	36 262 463	29 588
ACO	36 396 456	35 925
ACO	35 961 619	38 504
ACO	35 818 066	33 555
ACO	36 171 170	32 689
ACO	35 773 177	37 427
ACO	36 263 649	32 358
ACO	36 300 809	30 801

5.1.2 Co nejrychlejší cesta autem po Evropě

V případě porovnání metod před optimalizací 2-opt dosahuje u co nejrychlejší cesty autem po Evropě nejlepších výsledků opět algoritmus mravenčích kolonií, který našel okruh o 40,85 hodin rychlejší než metoda nejbližšího souseda a o 78,7 hodin rychlejší než hladový algoritmus.

Přestože mravenčí kolonie dosáhly lepších výsledků než metoda nejbližšího souseda, po optimalizaci 2-opt je nalezeno lepší řešení u metody nejbližšího souseda o přibližně hodinu a půl. I v tomto případě se ukazuje kombinace metody nejbližšího souseda s optimalizací 2-opt jako úspěšná kombinace.

Doba výpočtu je srovnatelná s předchozím příkladem co nejkratší cesty autem po Evropě, nyní je jen nepatrně rychlejší.

Náhodně vygenerovaný okruh našel přibližně dvakrát delší cestu než ostatní algoritmy.

Tabulka 4 - Výsledky algoritmů na úloze nejrychlejší cesty po Evropě autem, zdroj: vlastní zpracování

	Délka (min)	Výpočet ms
Random	75 483	< 1
2-opt: Random	**32 881**	< 1
HA	38 194	< 1
2-opt: HA	33 594	< 1
NS	35 923	31
2-opt: NS	32 972	< 1
ACO	33 472	33 576
2-opt: ACO	33 065	< 1

Nejlepší trasa byla nalezena pomocí kombinace náhodně vygenerované trasy a optimalizace 2-opt. Tím se ukazuje, že u metody 2-opt skutečně výrazně nezáleží na tom, jak optimální je cesta na vstupu. Nejhorších výsledků optimalizace 2-opt dosáhla u hladového algoritmu. Nejrychlejší nalezený okruh trvá projet autem dvacet dva dní, dvacet hodin a jednu minutu, vede přes města v tomto pořadí:

Praha, Česko → Varšava, Polsko → Kyjev, Ukrajina → Kišiněv, Moldavsko → Bukurešť, Rumunsko → Sofie, Bulharsko → Ankara, Turecko → Nikósie, Kypr → Jerevan, Arménie → Tbilisi, Gruzie → Baku, Ázerbájdžán → Nur-Sultan, Kazachstán → Moskva, Rusko → Minsk, Bělorusko → Vilnius, Litva → Riga, Lotyšsko → Tallinn, Estonsko → Helsinki, Finsko → Stockholm, Švédsko → Oslo, Norsko → Reykjavík, Island → Kodaň, Dánsko → Berlín, Německo → Vaduz, Lichtenštejnsko → Bern, Švýcarsko → Lucemburk, Lucembursko → Bruselas, Belgie → Amsterdam, Nizozemsko → Londýn, Velká Británie → Dublin, Irsko → Paříž, Francie → Lisabon, Portugalsko → Madrid, Španělsko → Andorra la Vella, Andorra → Monaco-Ville, Monako → Vatikán → Řím, Itálie → Valletta, Malta → San Marino, San Marino → Lublaň, Slovinsko → Záhřeb, Chorvatsko → Sarajevo, Bosna a Hercegovina → Podgorica, Černá Hora → Tirana, Albánie → Atény, Řecko → Skopje, Severní Makedonie → Bělehrad, Srbsko → Budapešť, Maďarsko → Bratislava, Slovensko → Vídeň, Rakousko → Praha, Česko

Mravenčí kolonie byly opět spouštěny patnáctkrát s tím, že pro porovnání s ostatními algoritmy byla vybrána nejúspěšnější varianta. Rozdíl mezi nejúspěšnějším a nejméně úspěšným řešením je šest hodin. I nejhorší výsledek je lepší, než trasa nalezená metodou nejbližšího souseda a hladovým algoritmem. Vzhledem k různým výsledkům mravenčích kolonií se vyplatí, pokud to čas dovolí, algoritmus spustit vícekrát a vybrat to nejlepší řešení. Doba výpočtu je také v každém případě jiná, rozdíl mezi nejrychlejší a nejpomalejší dobou výpočtu je osm vteřin, kdy výpočet průměrně trvá přibližně třicet tři vteřin. Je možné se domnívat, že tento rozdíl je způsoben logikou přepočítávání feromonů, kde dochází k mocnění na třetí, což by u velmi velkých čísel anebo u velmi malých čísel s pohyblivou desetinnou čárkou mohlo být pomalejší.

Tabulka 5 - Výsledky mravenčích kolonií nejrychlejší cesty autem, zdroj: vlastní zpracování

	Délka (min)	Výpočet ms
ACO	33 803	36 087
ACO	33 849	30 924
ACO	33 604	31 349
ACO	33 836	33 034
ACO	33 782	31 457
ACO	33 492	35 494
ACO	33 523	38 766
ACO	33 553	35 178
ACO	33 779	33 340
ACO	33 510	33 793
ACO	33 472	33 576
ACO	33 820	31 800
ACO	33 595	32 495
ACO	33 790	35 736
ACO	33 501	31 167

5.1.3 Co nejkratší cesta pěšky po Evropě

Mravenčí kolonie mají i na úloze nejkratší cesty pěšky kolem Evropy nejlepší výsledky, jsou kratší o více než 5 000 km oproti metodě nejbližšího souseda a hladovému algoritmu. Metoda nejbližšího souseda a hladový algoritmus mezi sebou nemají výrazný rozdíl, tento rozdíl je necelých 1 000 km. Řešení mravenčími koloniemi je přibližně o 14 % lepší než řešení metody nejbližšího souseda.

Trvání výpočtu je nepatrně delší než na úloze s jízdou autem.

Kombinace 2-opt s metodou nejbližšího souseda nedosáhla lepších výsledků než kombinace hladového algoritmu s 2-opt, zatímco u jízdy autem měla výsledky lepší. Pro tuto úlohu se tedy kombinace metody nejbližšího souseda s 2-opt neprokázala jako tak silná dvojice.

Tabulka 6 - Výsledky algoritmů na úloze nejkratší cesty pěšky po Evropě, zdroj: vlastní zpracování

	Délka (m)	Výpočet ms
Random	108 973 495	< 1
2-opt: Random	35 030 673	< 1
HA	40 942 329	< 1
2-opt: HA	34 484 864	< 1
NS	40 051 177	39
2-opt: NS	34 898 107	< 1
ACO	34 459 288	36 399
2-opt: ACO	**33 986 335**	< 1

Mravenčí kolonie společně s optimalizací 2-opt nalezly nejkratší řešení, které je dlouhé 33 986,3 km. Tento okruh pěšky je v následujícím pořadí:

Praha, Česko → Berlín, Německo → Kodaň, Dánsko → Reykjavík, Island → Oslo, Norsko → Stockholm, Švédsko → Helsinki, Finsko → Tallinn, Estonsko → Riga, Lotyšsko → Vilnius, Litva → Minsk, Bělorusko → Moskva, Rusko → Nur-Sultan, Kazachstán → Tbilisi, Gruzie → Baku, Ázerbájdžán → Jerevan, Arménie → Ankara, Turecko → Nikósie, Kypr → Atény, Řecko → Valletta, Malta → Řím, Itálie → Vatikán → San Marino, San Marino → Monaco-Ville, Monako → Andorra la Vella, Andorra → Madrid, Španělsko → Lisabon, Portugalsko → Paříž, Francie → Londýn, Velká Británie → Dublin, Irsko → Amsterdam, Nizozemsko → Bruselas, Belgie → Lucemburk, Lucembursko → Bern, Švýcarsko → Vaduz, Lichtenštejnsko → Lublaň, Slovinsko → Záhřeb, Chorvatsko → Vídeň, Rakousko → Bratislava, Slovensko → Budapešť, Maďarsko → Bělehrad, Srbsko → Sarajevo, Bosna a Hercegovina → Podgorica, Černá Hora → Tirana, Albánie → Skopje, Severní Makedonie → Sofie, Bulharsko → Bukurešť, Rumunsko → Kišiněv, Moldavsko → Kyjev, Ukrajina → Varšava, Polsko → Praha, Česko

Ve všech patnácti spuštění mravenčích kolonií bylo docíleno lepších výsledků než u metody nejbližšího souseda a hladového algoritmu. Poměrně často bylo nalezeno stejné řešení, například výsledek 34 730 800 byl nalezen čtyřikrát a pokaždé měl jinou délku výpočtu, to může znamenat, že ke stejnému výsledku bylo pokaždé docíleno jinými způsoby. Obdobně výsledek 34 728 337 je zaznamenaný třikrát. Ostatní výsledky se již neopakují. V předposledním měření byla nalezena trasa 35 377 670 metrů dlouhá, což je výrazně delší okruh než ve všech ostatních měřeních. Rozdíl mezi nejkratším okruhem a druhým nejdelším okruhem je 328 km, ale rozdíl mezi nejdelším a druhým nejdelším okruhem je 591 km. V úloze nejkratší cesty po Evropě pěšky tedy často dochází k naměření stejných hodnot vícekrát za sebou a zároveň se zde občas vyskytují nepoměrně velká měření.

Doba výpočtu je nepatrně delší než u úloh s jízdou autem a není závislá na kvalitě nalezeného řešení.

Tabulka 7 - Výsledky mravenčích kolonií nejkratší cesty pěšky, zdroj: vlastní zpracování

	Délka (m)	Výpočet ms
ACO	34 786 943	32 113
ACO	34 524 307	31 989
ACO	34 680 811	42 525
ACO	34 730 800	31 129
ACO	34 732 341	30 831
ACO	34 728 337	30 696
ACO	34 730 800	35 060
ACO	34 459 288	36 399
ACO	34 730 800	34 985
ACO	34 564 251	33 063
ACO	34 728 337	36 545
ACO	34 728 337	31 612
ACO	34 730 800	34 575
ACO	35 377 670	35 419
ACO	34 645 253	31 159

5.1.4 Co nejrychlejší cesta pěšky po Evropě

Před optimalizací 2-opt byla nalezena nejrychlejší trasa pomocí mravenčích kolonií, které nalezly cestu téměř o čtyřicet dní rychlejší, než hladový algoritmus, což je o čtrnáct procent méně. Oproti předchozím variantám cesty kolem Evropy, kdy metoda nejbližšího souseda našla lepší výsledek než hladový algoritmus, je zde situace opačná, hladový algoritmus našel cestu o dvanáct dní kratší než metoda nejbližšího souseda.

Náhodně nalezená trasa je více než třikrát delší oproti ostatním výsledkům a byla optimalizací 2-opt zlepšena o sedmdesát procent.

Metoda 2-opt vylepšila všechny trasy tak, že čím kratší byla trasa na vstupu, tak tím kratší je i trasa na výstupu. Tento vztah nicméně nebyl potvrzen v předchozích úlohách cesty

kolem Evropy. Kombinace metody 2-opt s hladovým algoritmem nebo metodou nejbližšího souseda dosahuje lepších výsledků než mravenčí kolonie bez optimalizace.

Doba výpočtu je srovnatelná s úlohou co nejkratší cesty pěšky, nicméně je velmi podobná u všech úloh, kde je stejný počet měst. Nejdéle trvá v každém případě výpočet mravenčími koloniemi.

Tabulka 8 - Výsledky algoritmů na úloze nejrychlejší cesty pěšky po Evropě, zdroj: vlastní zpracování

	Délka (min)	Výpočet ms
Random	1 250 283	< 1
2-opt: Random	364 541	< 1
HA	411 764	< 1
2-opt: HA	352 218	< 1
NS	429 266	39
2-opt: NS	352 471	< 1
ACO	354 606	34 607
2-opt: ACO	**351 793**	< 1

Nejrychlejší okruh byl nalezen za použití kombinace mravenčích kolonií a optimalizace 2-opt, která původní trasu vylepšila o další dva dny s tím, že celkové trvání projití optimalizovaného okruhu pěšky trvá 244 dní, 7 hodin a 13 minut bez přestávek. Konečný optimalizovaný okruh vede postupně těmito městy:

Praha, Česko → Varšava, Polsko → Vilnius, Litva → Minsk, Bělorusko → Kyjev, Ukrajina → Kišiněv, Moldavsko → Bukurešť, Rumunsko → Sofie, Bulharsko → Skopje, Severní Makedonie → Tirana, Albánie → Podgorica, Černá Hora → Sarajevo, Bosna a Hercegovina → Bělehrad, Srbsko → Budapešť, Maďarsko → Bratislava, Slovensko → Vídeň, Rakousko → Záhřeb, Chorvatsko → Lublaň, Slovinsko → San Marino, San Marino → Řím, Itálie → Vatikán → Monaco-Ville, Monako → Valletta, Malta → Madrid, Španělsko → Lisabon, Portugalsko → Andorra la Vella, Andorra → Paříž, Francie → Londýn, Velká Británie → Dublin, Irsko

→ Amsterdam, Nizozemsko → Bruselas, Belgie → Lucemburk, Lucembursko → Bern, Švýcarsko → Vaduz, Lichtenštejnsko → Atény, Řecko → Nikósie, Kypr → Ankara, Turecko → Jerevan, Arménie → Tbilisi, Gruzie → Baku, Ázerbájdžán → Nur-Sultan, Kazachstán → Moskva, Rusko → Riga, Lotyšsko → Helsinki, Finsko → Tallinn, Estonsko → Stockholm, Švédsko → Oslo, Norsko → Reykjavík, Island → Kodaň, Dánsko → Berlín, Německo → Praha, Česko

Ve výsledcích všech patnácti výpočtů mravenčími koloniemi se nevyskytuje žádné měření, které by bylo výrazně nižší nebo vyšší než zbylé měření. Absolutní rozdíl mezi nejnižší a nejvyšší naměřenou hodnotou je 3 840 minut, což odpovídá 2,7 dním. Nejlepší výsledek je oproti nejhoršímu lepší o jedno procento, z toho plyne, že mezi měřeními nejsou výrazné rozdíly. Hodnota 354 709 byla naměřena dvakrát, je tedy možné, že k tomuto řešení mravenčí kolonie dojdou častěji než k jiným, toto jedno řešení má lepší výsledek, než je průměr.

Ve všech měřeních dosáhly mravenčí kolonie lepších výsledků než metoda nejbližšího souseda a hladový algoritmus před optimalizací pomocí 2-opt.

Tabulka 9 - Výsledky mravenčích kolonií nejrychlejší cesty pěšky, zdroj: vlastní zpracování

	Délka (min)	Výpočet ms
ACO	354 606	34 607
ACO	357 677	30 701
ACO	355 019	30 515
ACO	356 098	30 833
ACO	356 090	30 455
ACO	357 733	35 427
ACO	355 921	36 296
ACO	358 142	33 928
ACO	354 709	30 695
ACO	354 608	34 173
ACO	354 289	35 505
ACO	357 257	36 036
ACO	358 446	31 297
ACO	355 170	31 755
ACO	354 709	33 351

5.2 Řešení s exaktním algoritmem

Řešení pomocí exaktního algoritmu je porovnáno s výsledky heuristických metod s ohledem na rychlost výpočtu a na to, jestli ostatní metody také dokázaly pro danou úlohu najít nejlepší řešení, nebo zdali selhaly.

První úloha je o velkosti deseti měst, další úlohy jsou vždy o jedno město rozšířené. Pro zpřehlednění je zde vyřazená náhodně vygenerovaná trasa. Všechny úlohy jsou zkoušené na variantě co nejrychlejší cesty autem.

5.2.1 Úloha o deseti městech

Na úloze o velikosti deseti měst všechny metody kromě hladového algoritmu našly optimální okruh, ale po jeho optimalizaci 2-opt je i tento okruh optimální. Nejpomalejší výpočet mají mravenčí kolonie, a to přes pět vteřin. Exaktnímu algoritmu výpočet nezabral ani půl vteřiny. Mravenčí kolonie mají stanovený přesný počet opakování nezávisle na velikosti úlohy, tedy už při malém počtu měst je jejich výpočet poměrně pomalý, zde dokonce o mnoho pomalejší než u exaktního algoritmu.

Tabulka 10 – Úloha o deseti městech s exaktním algoritmem, zdroj: vlastní zpracování

	Délka (min)	Výpočet ms
HA	9 793	< 1
2-opt: HA	8 437	< 1
NS	8 437	< 1
2-opt: NS	8 437	< 1
ACO	8 437	5 442
2-opt: ACO	8 437	< 1
EX	8 437	357
2-opt: EX	8 437	< 1

5.2.2 Úloha o jedenácti městech

Při výpočtu nad jedenácti městy už je čas potřebný pro výpočet podobný u mravenčích kolonií i exaktního algoritmu. Exaktní algoritmus potřeboval k výpočtu tři a půl vteřiny a mravenčí kolonie pět vteřin. Hladový algoritmus ani metoda nejbližšího souseda nedokázaly najít optimální řešení. U metody nejbližšího souseda ani optimalizace 2-opt řešení nevylepšila natolik, aby řešení bylo optimální. Mravenčí kolonie dokázaly najít optimální řešení stejně jako hladový algoritmus v kombinaci s vylepšující metodou 2-opt.

Na úloze o jedenácti městech je exaktní algoritmus stále dobře použitelný a zaručuje jako jediný optimální výsledek.

Tabulka 11 - Úloha o jedenácti městech s exaktním algoritmem, zdroj: vlastní zpracování

	Délka (min)	Výpočet ms
HA	10 606	< 1
2-opt: HA	**9 108**	< 1
NS	10 606	< 1
2-opt: NS	9 175	< 1
ACO	**9 108**	4 992
2-opt: ACO	**9 108**	< 1
EX	**9 108**	3 465
2-opt: EX	**9 108**	< 1

5.2.3 Úloha o dvanácti městech

Metoda nejbližšího souseda i hladový algoritmus na úloze o dvanácti městech nenašel optimální řešení. Optimalizace 2-opt v kombinaci s hladovým algoritmem ale optimální řešení našly stejně jako u úlohy s jedenácti městy.

Mravenčí kolonie i tomto případě našly optimální řešení a jejich výpočet je devětkrát rychlejší než v případě exaktního algoritmu, který pro svůj výpočet vyžaduje 42 vteřin, což je dvanáctkrát delší doba než u úlohy o jedenácti městech.

Tabulka 12 - Úloha o dvanácti městech s exaktním algoritmem, zdroj: vlastní zpracování

	Délka (min)	Výpočet ms
HA	10 060	< 1
2-opt: HA	**9 899**	< 1
NS	11 533	< 1
2-opt: NS	9 966	< 1
ACO	**9 899**	4 643
2-opt: ACO	**9 899**	< 1
EX	**9 899**	42 827
2-opt: EX	**9 899**	< 1

5.2.4 Úloha o třinácti městech

Metoda nejbližšího souseda ani hladový algoritmus i v kombinaci s optimalizací 2-opt nenalezly optimální řešení. Mravenčí kolonie jsou jediné z heuristických metod, které optimální řešení nalezly a doba potřebná pro výpočet s přidáním města nestoupla.

Exaktní algoritmus svůj výpočet prováděl téměř osm a půl minuty, to je přibližně dvanáctkrát déle, než u úlohy s velikostí menší o jedno město.

Tabulka 13 - Úloha o třinácti městech s exaktním algoritmem, zdroj: vlastní zpracování

	Délka (min)	Výpočet ms
HA	10 258	< 1
2-opt: HA	10 098	< 1
NS	10 095	< 1
2-opt: NS	10 095	< 1
ACO	**10 001**	4 432
2-opt: ACO	**10 001**	< 1
EX	**10 001**	505 081
2-opt: EX	**10 001**	< 1

5.2.5 Úloha o čtrnácti městech

Výpočet exaktním algoritmem trval 116 minut, tedy téměř dvě hodiny, to je skoro čtrnáctkrát více než v úloze o třinácti městech, kde bylo jen o jedno město méně.

Mravenčí kolonie opět dokázaly najít optimální řešení a jsou mezi heuristikami jediné. Jejich výpočet trval pět vteřin.

Tabulka 14 -Úloha o čtrnácti městech s exaktním algoritmem, zdroj: vlastní zpracování

	Délka (min)	Výpočet ms
HA	10 289	< 1
2-opt: HA	10 128	< 1
NS	10 125	< 1
2-opt: NS	10 125	< 1
ACO	**10 031**	4 997
2-opt: ACO	**10 031**	< 1
EX	**10 031**	6 954 653
2-opt: EX	**10 031**	< 1

5.2.6 Úloha o patnácti městech

U patnácti měst již výpočet pomocí exaktního algoritmu trval přes 27 hodin, to je čtrnáctkrát pomalejší výpočet oproti úloze o čtrnácti městech. Je zde patrná faktoriálová obtížnost výpočtu, kdy s každým dalším městem je čas potřebný na výpočet delší přibližně tolikrát, kolik je počet měst v řešení.

Mravenčím koloniím se i zde povedlo najít optimální řešení, a to opět jako jediným z heuristik s tím, že ani jakákoliv kombinace zbylých heuristik s metodou 2-opt nedosáhla optimálního výsledku. Výpočet trval mravenčím koloniím jen 7 sekund, což je oproti exaktnímu algoritmu významně kratší doba.

Metoda 2-opt nezlepšila řešení nalezené metodou nejbližšího souseda, bylo tomu tak i u úlohy o třinácti a čtrnácti městech.

K vylepšení mravenčích kolonií pomocí metody 2-opt dojde poprvé na úloze o velikosti dvaceti měst.

Tabulka 15 - Úloha o patnácti městech s exaktním algoritmem, zdroj: vlastní zpracování

	Délka (min)	Výpočet ms
HA	10 504	1
2-opt: HA	10 343	< 1
NS	10 340	2
2-opt: NS	10 340	< 1
ACO	**10 246**	7 025
2-opt: ACO	**10 246**	< 1
EX	**10 246**	97 570 319
2-opt: EX	**10 246**	< 1

Trasa, kterou našly mravenčí kolonie a exaktní algoritmus je dlouhá 7 dní, 2 hodiny a 46 minut a vede v daném pořadí přes tyto města:

Praha, Česko → Sarajevo, Bosna a Hercegovina → Podgorica, Černá Hora → Tirana, Albánie → Sofie, Bulharsko → Jerevan, Arménie → Tbilisi, Gruzie → Baku, Ázerbájdžán → Minsk, Bělorusko → Tallinn, Estonsko → Helsinki, Finsko → Kodaň, Dánsko → Bruselas, Belgie → Paříž, Francie → Andorra la Vella, Andorra → Praha, Česko

6 Závěr

Metoda mravenčích kolonií nalézá řešení, které je obvykle lepší o více než deset procent, než výsledky metody nejbližšího souseda a hladového algoritmu.

U úlohy s padesáti městy je kombinace metody 2-opt s metodou nejbližšího souseda nebo hladového algoritmu úspěšná srovnatelně jako optimalizace mravenčími koloniemi, nicméně mravenčí kolonie mají výrazně vyšší požadavky na dobu výpočtu. U menšího počtu měst mravenčí kolonie nalézají optimální řešení, zatímco zbylé heuristiky nikoliv, to bylo potvrzeno na úloze o třinácti, čtrnácti a patnácti městech.

Mravenčím koloniím na úloze o padesáti městech trval výpočet přibližně půl minuty. Počet iterací je nastaven na 2 000 a počet mravenců na 100, při zvýšení počtu mravenců nebo iterací algoritmus nedosahuje lepších výsledků. Pro získání kvalitního výsledku je vhodné mravenčí kolonie spustit opakovaně, protože mravenci se rozhodují i na základě náhody a dosahují tedy téměř pokaždé odlišných výsledků. Tento fakt dobu výpočtu ještě více prodlužuje. Doba výpočtu algoritmu mravenčích kolonií je závislá na tom, jaký je nastaven počet iterací a mravenců. Při snížení počtu iterací nebo mravenců by výsledky byly méně kvalitní, ale je možné jejich snížení, pokud by bylo potřeba výpočet zrychlit. V takovém případě by bylo vhodné upravit další parametry, které mají vliv na kvalitu řešení, jako je množství feromonů zanechávané na cestách nebo případně koeficient α nebo β. Dále by bylo vhodné přizpůsobit optimalizace, jako je minimální hladina feromonů a zanechávání většího počtu feromonů u nejúspěšnějších mravenců.

Zvolení vhodného algoritmu závisí na požadavcích. V případě požadavku na co nejrychlejší výpočet by mohla být vhodná kombinace hladového algoritmu s optimalizací 2-opt, protože výpočet i na úloze o padesáti městech trval méně než jednu milisekundu a zároveň výsledek u jedné úlohy (nejrychlejší cesty pěšky kolem Evropy) dosahoval výsledků lepších, než samotné mravenčí kolonie bez optimalizace 2-opt. Hladový algoritmus je zároveň i nejjednodušší na implementaci. Kombinace metody nejbližšího souseda a optimalizace 2-opt dosahovala ještě lepších výsledků a stále byla velmi rychlá.

V případě požadavku na co nejkvalitnější řešení, kde není důležité znát výsledek okamžitě, by mravenčí kolonie byly vhodnou volbou, dokonce u menšího počtu měst do patnácti instancí vždy nalezly stejné optimální řešení jako exaktní algoritmus. Vzhledem k tomu, že teprve u úlohy s dvaceti městy bylo řešení poprvé vylepšeno pomocí 2-opt,

je možné se domnívat, že do velikosti úlohy o devatenácti městech samotné mravenčí kolonie naleznou optimální řešení. V kombinaci s optimalizací 2-opt byly mravenčí kolonie jednoznačně nejúspěšnější, tato kombinace pravděpodobně nalezne optimální řešení i u úlohy s více než dvaceti městy.

Exaktní algoritmus je vhodný pro výpočet na úlohách do velikosti jedenácti měst, kdy výpočet trvá maximálně tři a půl vteřiny.

7 Seznam použitých zdrojů

Anderson, R., Brock, D., & Larkin, K. (2011). *Introduction to Razor Pages in ASP.NET Core*. Microsoft, Dostupné z: learn.microsoft.com.

Applegate, D., Bixby, R., Chvátal, V., & Cook, W. (7 2003). Implementing the Dantzig-Fulkerson-Johnson algorithm for large traveling salesman problems. Mathematical Programming. stránky 153, Dostupné z DOI:10.1007/s10107-003-0440 4.

Birattari, M., Dorigo, M., Blum, C., Gambardella, M., Mondada, F., & Stützle, T. (2004). *Ant Colony Optimization and Swarm Intelligence*. Springer Berlin Heidelberg, ISBN: 9783662194386.

Blum, C. (12 2005). Ant colony optimization: Introduction and recent trends. *Physics of life reviews*, stránky 373, Dostupné z DOI: 10.1016/j.plrev.2005.10.001.

Cook, J. W. (2012). *Po stopách obchodního cestujícího: matematika na hranicích možností.* Praha: Dokořán. ISBN 978-80-7363-412-4.

Dorigo, M., & Stützle, T. (2004). *Ant Colony Optimization.* MIT Press, ISBN: 9780262042192.

Du, P., Liu, N., Zhang, H., & Lu, J. (10 2021). An Improved Ant Colony Optimization Based on an Adaptive Heuristic Factor for the Traveling Salesman Problem. *Journal of advanced transportation*, stránky 16, Dostupné z DOI:10.1155/2021/6642009.

Google. (2023). *Google Maps Platform.* Mountain View, California, USA: Dostupné z: https://developers.google.com/maps/documentation.

Hladík, M. (5 2023). Celočíselné Programování: text k přednášce [online]. stránky 111, Dostupné z: https://kam.mff.cuni.cz/~hladik/CP/text_cp.pdf.

Hliněný, P. (3 2010). Základy teorie grafů pro (nejen) informatiky. *Masarykova univerzita. Fakulta Informatiky*, stránky 130, Dostupné z: https://is.muni.cz/do/1499/el/estud/fi/js10/grafy/Grafy-text10.pdf.

Hoos, H., & Stützle, T. (10 2014). On the empirical scaling of run-time for finding optimal solutions to the travelling salesman problem. *European journal of operational research*, stránky 94, Dostupné z DOI: 10.1016/j.ejor.2014.03.042.

Karp, R. R., & Thatcher, M. (11 2009). Reducibility Among Combinatorial Problems. *50 Years of Integer Programming 1958-2008*, stránky 241, Dostupné z DOI: 10.2307/2271828.

Keld, H. (11 2006). An Effective Implementation of K-opt Moves for the Lin Kernighan TSP Heuristic. Computer Science, Roskilde University. *Computer Science Roskilde University* , stránky 99, ISSN: 0109-9779.

Kučera, L. (9 1989). Kombinatorické algoritmy. *Státní nakladatelství technické literatury (SNTL)*, stránky 283, ISBN 04-009-89.

Lance, F. (9 2009). The status of the P versus NP problem. Communications of the ACM. *Communications of the ACM*, stránky 86, Dostupné z DOI: 10.1145/1562164.1562186.

Lawler, E., Lenstra, K., Rinnooy, K., & B., S. (1991). *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* ISBN: 978-0-471-90413-7.

Lin, S. (12 1965). Computer solutions of the traveling salesman problem. *The Bell System Technical Journal*, stránky 25, Dostupné z: DOI:10.1002/j.1538-7305.1965.tb04146.x.

Matoušek, J. (2000). *Kapitoly z diskrétní matematiky. Vyd. 2.* Praha: Karolinum: ISBN: 80-246-0084-6.

Mavrovouniotis, M., Muller, F. M., & Yang, S. (7 2017). Ant Colony Optimization With Local Search for Dynamic Traveling Salesman Problems. *IEEE transactions on cybernetics*, stránky 1756, Dostupné z DOI: 10.1109/TCYB.2016.2556742.

Meshram, S. G., Ghorbani, M. A., Deo, R. C., Kashani, M. H., Meshram, C., & Karimi, V. (5 2019). New Approach for Sediment Yield Forecasting with a Two-Phase Feedforward Neuron Network-Particle Swarm Optimization Model Integrated with the Gravitational Search Algorithm. *Water resources management*, stránky 2356, Dostupné z DOI: 10.1007/s11269-019-02265-0.

Misevicius, A. (11 2011). Combining 2-OPT, 3-OPT and 4-OPT with K-SWAP-KICK perturbations for the traveling salesman problem. *17th International Conference on Information and Software Technologies* (str. 6). Kaunas: Kaunas University of Technology,
Dostupné z: https://isd.ktu.lt/it2011/material/Proceedings/1_AI_5.pdf.

Pilát, M. (2016). *Hejna a kolonie.* Praha: Dostupné z: https://martinpilat.com/cs/prirodou-inspirovane-algoritmy/hejna-kolonie.

Savický, P. (2016). *Výpočetní složitost I: pro obor informatika na FF UK [online].* Dostupné z: http://www.cs.cas.cz/~savicky/vyuka/vypsl/vypsl2016zs1.pdf.

Soofastaei, A. (2022). *The Application of Ant Colony Optimization.* Londýn: IntechOpen, ISBN: 9781839681776.

Tesař, K., & Mareš, M. (nedatováno). *Složitost [online].* Dostupné z: https://ksp.mff.cuni.cz/kucharky/programatorske-kucharky/01-slozitost.pdf.

Wirsansky, E. (2020). *Hands-On Genetic Algorithms with Python : Applying Genetic Algorithms to Solve Real-World Deep Learning and Artificial Intelligence Problems.* Packt Publishing, Limited, ISBN: 9781838557744.

Ying, T. (2018). *Swarm Intelligence, Volume 1 - Principles, Current Algorithms and Methods.* Institution of Engineering and Technology, ISBN: 1785616277, 9781785616273.

8 Seznam obrázků, tabulek a zkratek

8.1 Seznam obrázků

8.2 Seznam tabulek

8.3 Seznam použitých zkratek

TSP	Problém obchodního cestujícího (Traveling salesman problem)
ACO	Optimalizace mravenčími koloniemi (Ant colony optimalization)
EX	Exaktní algoritmus
HA	Hladový algoritmus
NS	Metoda nejbližšího souseda
Random	Náhodná trasa
ACO: 2-opt	Optimalizace mravenčích kolonií pomocí metody 2-opt
EX: 2-opt	Optimalizace exaktního algoritmu pomocí metody 2-opt
HA: 2-opt	Optimalizace trasy získané hladovým algoritmem pomocí metody 2-opt
NS: 2-opt	Optimalizace trasy získané metodou nejbližšího souseda pomocí metody 2-opt
Random: 2-opt	Optimalizace náhodné trasy pomocí metody 2-opt