Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra systémového inženýrství

Bakalářská práce

Porovnání heuristik problému obchodního cestujícího a optimalizace tras

Kristýna Bartošová

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Kristýna Bartošová

Systémové inženýrství a informatika
Informatika

Název práce

Porovnání heuristik problému obchodního cestujícího a optimalizace tras

Název anglicky

Comparison of heuristics for the travelling salesman problem and route optimization

---

Cíle práce

Cílem práce je zhodnotit vybrané heuristické metody pro řešení optimalizační úlohy známé jako problém obchodního cestujícího podle časové náročnosti výpočtu a délky nalezené cesty. Součástí cíle je také nalezení co nejkratších okružních cest pro rozvoz smluv investiční společností In Investments s. r. o. mezi partnerské společnosti. Firma si nyní volí cestu pouze na základě intuice. Výstupem práce tedy budou vlastním programem porovnané algoritmy na řešení jednookruhového okružního dopravního problému pro optimalizaci trasy rozvozu smluv a nalezení co nejoptimálnějšího řešení.

Metodika

Práce je rozdělená na teoretickou a praktickou část. Metodika první teoretické části je založená na studiu odborných informačních zdrojů a syntézou těchto nastudovaných znalostí budou popsány základní pojmy teorie grafů, výpočetní složitost algoritmů, problém obchodního cestujícího a exaktní a heuristické metody pro jeho řešení. Následovat bude charakteristika rozvozu smluv firmy In Investments s. r. o., zde bude popsán program pro optimalizaci trasy z centrály s návratem zpět na výchozí místo. V druhé praktické části budou porovnány výsledky získané vlastním programem, a to délka získané trasy a rychlost výpočtu. V programu bude implementována metoda nejbližšího souseda, hladový algoritmus a zlepšující Lin–Kernighanova metoda 2-opt. Zmíněné metody budou aplikovány na úlohy rozvozu smluv vybrané firmy, pro které bude nalezena co nejoptimálnější trasa.

Doporučený rozsah práce

30 – 40 stran

Klíčová slova

Problém obchodního cestujícího, Teorie grafů, Heuristika, Výpočetní složitost, Metoda nejbližšího souseda, Hladový algoritmus, Linova–Kernighanova metoda

---

Doporučené zdroje informací

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

KUČERA, Luděk. Kombinatorické algoritmy. Praha: SNTL – Státní nakladatelství technické literatury, 1989. ISBN 04-009-89.

LIN, Shen. Computer solutions of the traveling salesman problem. The Bell System Technical Journal. 1965, 1965(44), 2245 – 2269. Dostupné z: doi:10.1002/j.1538-7305.1965.tb04146.x

---

Předběžný termín obhajoby

2021/22 ZS – PEF

Vedoucí práce

RNDr. Petr Kučera, Ph.D.

Garantující pracoviště

Katedra systémového inženýrství

Elektronicky schváleno dne 24. 11. 2021

doc. Ing. Tomáš Šubrt, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 25. 11. 2021

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 30. 11. 2021

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Porovnání heuristik problému obchodního cestujícího a optimalizace tras" jsem vypracovala samostatně pod vedením vedoucího bakalářské 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 bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušila autorská práva třetích osob.

V Praze dne _____

Poděkování

Ráda bych touto cestou poděkovala RNDr. Petru Kučerovi, PhD. za odborné vedení, cenné rady, konzultace a čas, který mi věnoval. Zároveň děkuji společnosti In Investments s. r. o za poskytnutí nezbytných informací.

Porovnání heuristik problému obchodního cestujícího a optimalizace tras

Abstrakt

Práce se zabývá porovnáním heuristických metod pro řešení problému obchodního cestujícího z hlediska kvality nalezeného řešení a výpočetního času potřebného k výpočtu. Porovnávané metody jsou metoda nejbližšího souseda, hladový algoritmus a metoda zlepšující řešení 2-opt. Algoritmy byly implementovány ve webové aplikaci a odzkoušeny na testovacích příkladech, kterými jsou cesta po hlavních městech všech států Evropy a rozvoz smluv po partnerských společnostech firmy In Investments s. r. o. Předmětem práce je také navrhnout a vytvořit aplikaci pro rozvoz smluv firmy, kterou firma může využívat. V práci jsou popsány základní pojmy z teorie grafů, výpočetní složitost a jednotlivé metody pro řešení problému obchodního cestujícího, zejména ty použité ve vlastním programu.

Klíčová slova: Problém obchodního cestujícího, Teorie grafů, Heuristika, Výpočetní složitost, Metoda nejbližšího souseda, Hladový algoritmus, Linova–Kernighanova metoda

Comparison of heuristics for the travelling salesman problem and route optimization

Abstract

This thesis deals with the comparison of heuristic methods for solving the traveling salesman problem by the quality of the solution and the computational time required for the calculation. The compared methods are the nearest neighbor method, the greedy algorithm and the 2-opt method which improves the solution. The algorithms were implemented in a web application and tested on examples, which are a journey around the capital cities of Europe and the delivery of contracts to partner companies of company In Investments s. r. o. The aim of the thesis is also to design and create an application for the distribution of the company contracts, which can be used by the company. The thesis describes the basic concepts of graph theory, computational complexity and methods for solving the traveling salesman problem, especially those used in the program itself.

Keywords: Traveling salesman problem, Graph theory, Heuristic methods, Algorithmic complexity, Nearest neighbor method, Greedy algorithm, Lin–Kernighan method

Obsah

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

HA	Hladový algoritmus
NS	Metoda nejbližšího souseda
Random	Náhodná trasa
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
MHD	Městská veřejná doprava

1 Úvod

Problém obchodního cestujícího je fascinující tím, že i přes to, že na jeho řešení pracuje velké množství matematiků, se stále neví, jestli pro něj existuje nějaké efektivní řešení, což znamená takové řešení, které dává přesné výsledky a ke svému řešení nepotřebuje velké množství času. Patří také k nejznámějším optimalizačním úlohám. Problém obchodního cestujícího zní: nalezněte nejkratší cestu pro navštívení všech míst na daném seznamu s tím, že každé místo smíte navštívit pouze jednou a musíte se vrátit zpět tam, kde jste začínali. Jedná se o problém, který je v praxi velmi často řešen, zejména v logistice, mnohdy stále bez pomoci softwaru. Ačkoli samotná formulace problému zní jednoduše, výpočet tak jednoduchý není. Pokud například budeme chtít vyjet z jednoho místa a navštívit 20 dalších míst, tak možností, jakou cestou se vydáme, je $20!/2$, což je přesně 1 216 451 004 088 320 000 možných cest, ze kterých potřebujete vybrat tu nejkratší. Spočítat takové úlohy není náročné jen pro člověka, ale i pro počítač, zejména proto, že s každým dalším místem počet přípustných řešení velmi rychle roste. Doba potřebná k výpočtu se tedy stává neúnosná už při řádově desítkách míst.[2]

Firma In Investments s. r. o. je příkladem firmy, která rozváží smlouvy do svých partnerských společností, kterých je právě 20, z toho 13 má dodací adresu ve městě sídla firmy. Do poboček, kam se firmě nevyplatí jezdit, se posílají smlouvy poštou, jedná se zejména o ty pobočky, které jsou daleko od sídla firmy. Pobočky, do kterých je třeba rozvážet, jsou pokaždé jiné a pracovníci si vždy musí volit trasu na základě intuice. Brigádník, který smlouvy rozváží, má často omezený čas, a tak nestíhá navštívit vše, co by bylo potřeba, což je možná zbytečné a nemuselo by tomu tak být, kdyby firma mohla využívat nástroj pro nalezení kratší cesty. Je totiž dobré, aby se smlouvy dostaly do partnerských společností co nejdříve. Cílem práce je tedy mimo jiné vytvořit program pro optimalizaci tras ze sídla firmy přes libovolné pobočky, který bude uživatelsky přívětivý a firma ho bude moci využívat.

Odpověď na otázku, zdali je možné najít řešení problému obchodního cestujícího i jinak než prověřením všech možností cest, neznáme. Víme ale, že existují algoritmy, které fungují v přijatelném čase, nicméně k optimálnímu řešení se jen přibližují. Pomocí programu pro optimalizaci tras firmy budou implementovány některé takové algoritmy a porovnány

jejich výsledky. Pro různé vstupy může totiž být vhodný jiný algoritmus a je těžké předem určit který.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem práce je zhodnotit vybrané heuristické metody pro řešení optimalizační úlohy známé jako problém obchodního cestujícího podle časové náročnosti výpočtu a délky nalezené cesty. Součástí cíle je také nalezení co nejkratších okružních cest pro rozvoz smluv investiční společností In Investments s. r. o. mezi partnerské společnosti. Firma si nyní volí cestu pouze na základě intuice. Výstupem práce tedy budou vlastním programem porovnané algoritmy na řešení jednookruhového okružního dopravního problému pro optimalizaci trasy rozvozu smluv a nalezení co nejlepšího řešení.

2.2 Metodika

Práce je rozdělená na teoretickou a praktickou část. Metodika první, teoretické části je založená na studiu odborných informačních zdrojů. Syntézou těchto nastudovaných znalostí budou popsány základní pojmy teorie grafů, výpočetní složitost algoritmů, problém obchodního cestujícího a exaktní a heuristické metody pro jeho řešení. Následovat bude charakteristika rozvozu smluv firmy In Investments s. r. o., zde bude popsán vhodný program pro optimalizaci trasy z centrály s návratem zpět na výchozí místo. V druhé, praktické části budou porovnány výsledky získané vlastním programem, a to délka získané trasy a rychlost výpočtu. V programu bude implementována metoda nejbližšího souseda, hladový algoritmus a zlepšující Lin–Kernighanova metoda 2-opt. Zmíněné metody budou aplikovány na úlohy rozvozu smluv vybrané firmy, pro které bude nalezena co nejlepší trasa. Další úlohou pro porovnání algoritmů bude cesta po všech padesáti státech Evropy, respektive po jejich hlavních městech.

3 Teoretická východiska

3.1 Úvod do teorie grafů

Mnoho situací v matematice, informatice a ve světě lze charakterizovat pomocí množiny bodů a spojnic mezi těmito body. Matematickou abstrakcí takových situací je graf.[12]

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 hamiltonovskou kružnici, kde jsou všechny vrcholy grafu právě jednou tak, aby součet ocenění všech hran kružnice byl minimální. V následující části tedy budou popsány základní pojmy z teorie grafů, zejména ty používané při popisování problému obchodního cestujícího.

3.1.1 Základní pojmy

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 $\boldsymbol{E(G)}$.[9]

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.[9]

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.

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

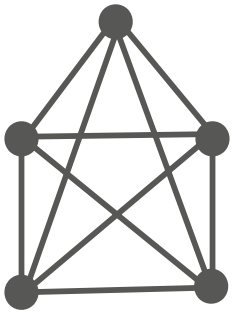
Podgrafem grafu G rozumíme libovolný graf H na podmnožině vrcholů $\mathrm{V(H)} \subseteq \mathrm{V(G)}$, který má za hrany libovolnou podmnožinu hran $E(G)$, pro kterou platí, že jsou všechny vrcholy hran ve $V(H)$. Podgraf značíme jako $\mathrm{H} \subseteq \mathrm{G}$.[7]

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.

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 i pro lidi z jiných oborů. Příkladem jsou sociální sítě, kdy z vizualizace pouhým pohledem zjistíme, kolik přátel má který člověk nebo kolik přátel dělí jednoho člověka od druhého. Dalším příkladem může být vyjádření vztahů mezi různými objekty, například jaké kusy oblečení si mohu obléct zároveň, přičemž platí, že vrcholy znázorňují druh oblečení a hrany vztahy mezi nimi. Déle lze pomocí grafů vyjádřit návaznosti, spojení, závislosti, toky mezi objekty atd. Z toho důvodu, a i pro jejich jednoduché zpracování dat, si grafy našly své místo také v informatice.

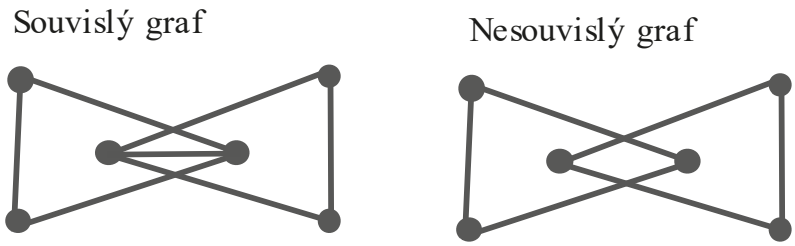
Grafy můžeme implementovat maticí sousednosti pomocí dvourozměrného pole $g[,]$, ve kterém $g[i, j] = 1$ znamená hranu mezi vrcholy i a j s cenou 1. Případně je možné použít takzvaný výčet sousedů tak, že k dvourozměrnému poli přidáme i jednorozměrné pole stupňů vrcholů.[7][12]

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 tedy 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 $|V(G)| = \frac{1+\sqrt{1+8*|E(G)|}}{2}$.



Obrázek 1 - Úplný graf K5, zdroj: autor

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.



Obrázek 2 - Souvislý a nesouvislý graf, zdroj: autor

Sled v grafu G je taková posloupnost vrcholů $vi = 1, 2, \ldots, n$ a hran, že $\{vi - 1,\ vi\}$ 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.[9]

Tah je sled, ve kterém se nevyskytuje více stejných hran.

Cesta je sled, ve kterém se nevyskytuje více stejných vrcholů.

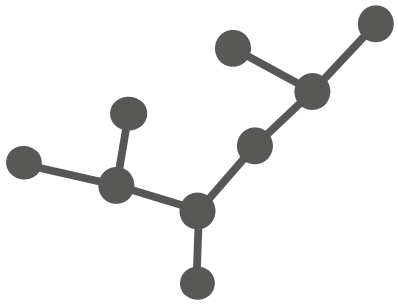
Kružnice obsahuje $n \geq 3$ hran a $n \geq 3$ vrcholů, které jsou sousední a zároveň žádná hrana ani vrchol se v kružnici neopakují. Jedná se tedy o uzavřenou posloupnost propojených vrcholů, kde hrany vedou mezi sousedními vrcholy a také mezi prvním a posledním vrcholem. Počet hran i počet vrcholů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

Eulerův tah je uzavřená cesta, která prochází každou hranou právě jednou.

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.

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.[7]

Strom je jednoduchý souvislý graf bez kružnic. Počet hran ve stromu je vždy o jedna menší, než 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.[7]



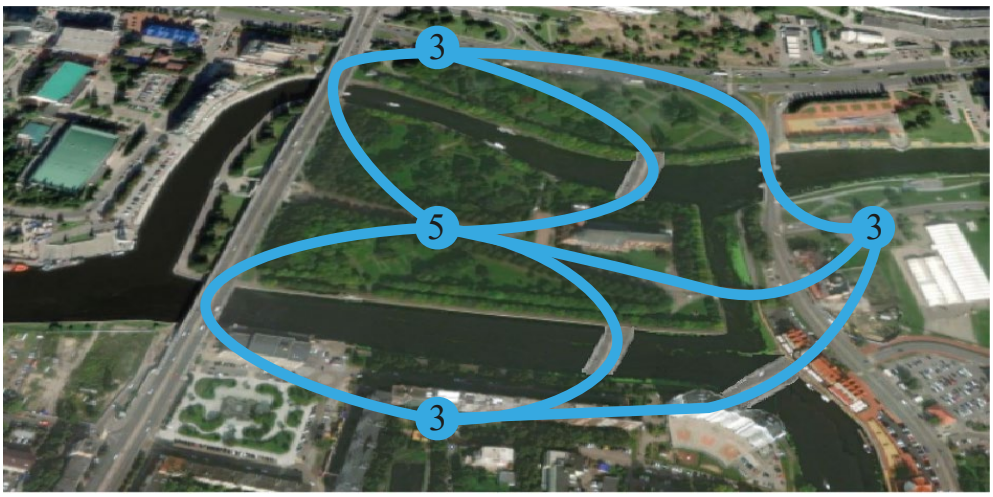
Obrázek 3 - Strom, zdroj: autor

3.1.2 Mosty v Královci

Jeden z největších matematiků Leonhard Euler vyřešil hádanku, která údajně dlouho trápila obyvatele tehdejšího města Královce (dnešní Kaliningrad) ve východním Prusku a položil tím základy pro obor teorie grafů. Tehdy bylo na řece ve městě sedm mostů. Říká se, že obyvatelé hledali způsob, jak všech sedm mostů projít během jedné procházky tak, aby se na žádný most nemuseli vracet. Euler k úloze přistupoval jako ke grafu, kdy procházka Královcem představuje cestu od vrcholu k vrcholu po hranách příslušného grafu. Úlohu si je také možné představit jako: nakresli celý daný graf jedním tahem, kdy každou hranu smíte nakreslit právě jednou. Euler si všiml, že v jakékoli cestě mezi dvěma různými vrcholy je stupeň prvního a posledního vrcholu lichý, a zároveň zbývající vrcholy jsou vždy sudého stupně. A pokud cesta začíná a končí ve stejném bodě, musí mít všechny vrcholy na

cestě sudý stupeň. To znamená, že abychom mohli projít všechny hrany grafu právě jednou, musí mít graf všechny vrcholy sudého stupně, nebo právě dva vrcholy stupně lichého. Má-li graf dva liché stupně, musíme začít svou cestu v některém z lichých vrcholů, abychom prošli všechny hrany právě jednou.[2]

Graf představující mosty přes řeku v Královci měl všechny své čtyři vrcholy lichého stupně. Pro obyvatele Královce to nebyla dobrá zpráva, znamenalo to, že jejich kýžená procházka přes každý most právě jednou neexistuje.[2]



Obrázek 4 - Mosty v královci, zdroj: mapy.cz a autor

Z příkladu mostů v Královci vznikl pojem Eulerův tah, který se liší od hamiltonovské kružnice tím, že prochází každou hranou právě jednou, zatímco v hamiltonovské kružnici se prochází právě jednou každým vrcholem. Určit, jestli graf obsahuje hamiltonovskou kružnici, tedy jestli je graf hamiltonovský, není tak jednoduché jako v případě Eulerova tahu. Mnoho matematiků se snažilo formulovat přesné podmínky pro rozhodnutí, zdali je graf hamiltonovský, ale setkali se s neúspěchem. Tento problém je dalším příkladem NP-úplného problému. To znamená, že jeho složitost je na stejné úrovni, jako celý problém obchodního cestujícího, a to i přes to, že odpověď na otázku je pouze ano nebo ne. Dirac přišel alespoň s podmínkou, že každý graf na $n \geq 3$ vrcholech s minimálním stupněm $\geq \frac{n}{2}$ je hamiltonovský.[2][7]

3.2 Výpočetní složitost

Složitostí algoritmické úlohy se obvykle myslí její nároky na čas a na paměť při řešení pomocí počítače. Algoritmus je přesný návod či postup, kterým lze vyřešit daný typ úlohy.

Paměťová neboli prostorová složitost vyjadřuje počet bitů potřebných k uložení dat během výpočtu. V dnešní době už v podstatě paměťová složitost nehraje roli, protože počítače mají obvykle paměť dostatečnou.

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.

Ve výpočetní složitosti nejsou tedy brány v úvahu konkrétní softwarové ani hardwarové zařízení, pomocí kterých 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. Jinými slovy složitost úlohy je složitost nejlepšího algoritmu, který úlohu řeší.

U mnohých úloh lze složitost charakterizovat jen nepřímo, to znamená například odkazem na třídu úloh podobné složitosti. K tomuto účelu byly vytvořeny třídy složitosti. Vědomí toho, že můžeme úlohu zařadit, nám může pomoci odhalit podobnosti s ostatními úlohami a na základě toho můžeme využít známých postupů pro řešení dané úlohy. Snahou je zařadit úlohu do co nejnižší třídy.[15]

3.2.1 Základní třídy složitosti

Třída P je třídou úloh, které lze řešit v čase, který je omezen polynomem v závislosti na velikosti vstupních dat, s možností využít neomezené množství paměti. Pokud algoritmus nebo úloha spadá do této třídy, můžeme je ve většině případů prohlásit za efektivně řešitelný. Nicméně z praktického hlediska je třeba složitost rozlišovat i jinak než pouze na základě toho, zdali mají polynomiální složitost. Například algoritmus složitosti n^{100}, kde n je velikost vstupu, nelze použít pro vstupy již relativně malé velikosti.[15]

Třída NP obsahuje úlohy, pro které nemusí být znám efektivní algoritmus, ale je možné, že nějaký takový algoritmus existuje. Můžeme však správnost výsledku v polynomiálním čase ověřit. U těchto úloh není dokázáno, že mají vysokou výpočetní složitost, stejně tak není dokázán dolní odhad jejich složitosti. Mezi NP-úplné problémy patří právě problém obchodního cestujícího. **NP-úplné** problémy jsou takové problémy, na které jsou polynomiálně redukovatelné všechny ostatní problémy z NP, takže úplné problémy v dané třídě jsou ty nejsložitější. Dalším klasickým reprezentantem této třídy je úloha zjistit, jestli daný graf obsahuje podgraf, který je úplným grafem o dané minimální velikosti.

Třídy PSPACE a NPSPACE jsou řešeny za použití polynomiální velikosti paměti.

Třídy EXPTIME a EXPSPACE mají exponenciální složitost a není pro ně polynomiální algoritmus. Jsou to prokazatelně nezvladatelné problémy. Příkladem jsou například hry šachy, dáma nebo GO.[15]

3.2.2 Asymptotická časová složitost

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.[16]

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 algoritmus pro řešení problému obchodního cestujícího běží v čase $O(n^2 2^n)$, n zde představuje počet vrcholů grafu neboli měst.

- $O(n!)$ – faktoriálová, například problém obchodního cestujícího řešený hrubou silou, který má složitost $O((n-1)!/2)$.

Mezi polynomiální charakteristiky patří například lineární, logaritmické a kvadratické, to znamená, že tyto charakteristiky spadají do třídy P, 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, to znamená, že se vykoná 1 miliarda operací za vteřinu, což přibližně odpovídá rychlosti běžných procesorů. 10^9 sekund je cca 31 let a 10^{18} s je cca stáří 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 cca od 20 měst.[16]

Tabulka 1 - Rychlost výpočtu, kdy uvažujeme, že jedna operace trvá jednu ns a n je velikost vstupu.[16]

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	10^8 s			
$O((n-1)!/2)$	181 µs	10^8 s				
$O(n!)$	3 ms	10^9 s	10^{23} s	10^{149} s	10^{2558} s	$\approx \infty$

3.3 Problém obchodního cestujícího

Problém obchodního cestujícího je optimalizační problém, kde cestující potřebuje projít přes všechny města právě jednou tak, aby se vrátil do výchozího města a aby celková délka cesty byla co nejkratší.

Pomocí teorie grafů můžeme problém popsat jako hledání hamiltonovské kružnice na úplném neorientovaném hranově kladně ohodnoceném grafu, kde $|V(G)| \geq 3$ tak, aby součet všech cen hran hamiltonovské kružnice byl minimální.

Na popularitě získal zejména díky své jednoduché definici, ale složitému výpočtu.

3.3.1 Uplatnění

Jedná se o typický problém, který je vysoce relevantní v širokém okruhu praktických problémů, jako je například plánování počítačové sítě, umístění senzorů, logistika a distribuce, vrtání a pájení desek s plošnými spoji, inteligentní doprava pomocí mapového systému a GPS, mapování genomu, směřování teleskopů, rentgenových paprsků a laserů, testování procesorů nebo také minimalizace odpadu při tapetování a tak dále.[3][2]

3.3.2 Složitost

Jedná se o NP-úplný problém, jehož řešení neboli výstup lze v polynomiálním čase zkontrolovat. Jedno z kouzel tohoto a všech NP-úplných problémů je, že není prokázáno, že pro ně neexistuje efektivní neboli dobrý algoritmus. Tuto úplnost ukázal Karp a Cook, kteří popsali 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.[14][2]

Všeobecně se počítá s tím, že NP se nerovná P, ale z teoretického hlediska pro to není důvod. Dokonce současné šifrovací systémy 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.“[2][4]

Zatím žádný efektivní algoritmus pro problém obchodního cestujícího neznáme, bylo však vyvinuto mnoho časově efektivních aproximačních neboli heuristických metod. 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.[2]

3.4 Exaktní algoritmy

Exaktní metody fungují pouze pro malý počet měst, mají ale přesný výsledek. Zatím není znám efektivní exaktní algoritmus a ani není potvrzeno, že takový algoritmus neexistuje.

Mezi historicky nejdůležitější exaktní algoritmy patří ty vyvinuté týmem Dantziga. Tyto algoritmy tvoří základy pro nejvíce efektivní algoritmy, které se dnes používají. CONCORDE, který se dnes považuje za nejlepší dostupný exaktní algoritmus, byl ve svých ranných fázích zveřejněn s titulkem:[10]

„Implementace Dantzig-Fulkerson-Johnson algoritmu pro velký problém obchodního cestujícího“[1]

3.4.1 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.

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. Trvalo by to dvakrát delší dobu, než je stáří našeho 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.[2]

3.4.2 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 obchodního cestujícího. Později se tato metoda začala kombinovat i s metodou větví a mezí.[6][2]

3.4.3 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.[6][2]

Jedná se o víceúčelový nástroj, nejprve byl formulován v kontextu obchodního cestujícího. Metoda větví a mezí společně s metodou řezných rovin dokážou řešit úlohy s tisíci měst, což je velmi dobrý výsledek. V moderních programech pro problém obchodního cestujícího se právě tato kombinace používá.

3.5 Heuristiky

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 pravidel 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.[3]

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.[3]

Heuristik je mnoho, následně jsou popsány heuristiky používané v praktické části této práce.

3.5.1 Metoda nejbližšího souseda

Metoda 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.[2]

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

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 dosahuje jiných výsledků podle toho, jaké město zvolíme jako výchozí. Je tedy potřeba vyzkoušet všech n možností, kde n je počet měst, kdy vždy začínáme okruh v jiném městě. Z těchto možností potom vybereme tu nejkratší.[2]

3.5.2 Hladový algoritmus

Metoda 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.[2]

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

Říká se mu hladový, protože se snaží vždy nenasytně sníst neboli pojmout hrany, které mají nejlepší ceny.

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.

Na začátku výpočtu vypadá metoda slibně, ale ke konci může být nucena zařadit i některé zbytečně dlouhé hrany.[2]

3.5.3 Lin-Kerninghanovy metody zlepšující řešení

Tyto metody mají jako vstup už hotové řešení spočítané nějakou heuristickou metodou anebo náhodně vybranou cestu. 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.[5]

Lin-Kerninghanův algoritmus funguje na takzvaných r-opt výměnách hran, r zde určuje počet vyměňovaných hran. Lin a Kerninghana vymysleli 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.[5][11][2]

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 $r \epsilon N$.

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.[13]

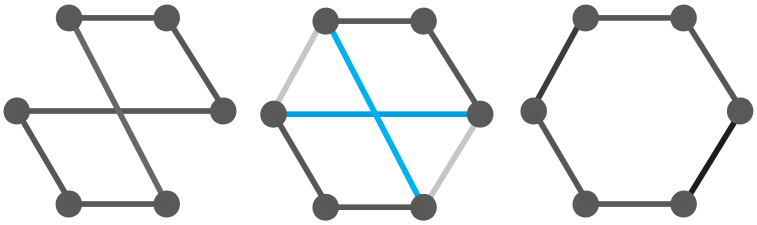
3.5.3.1 Metoda 2-opt

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.[5]

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

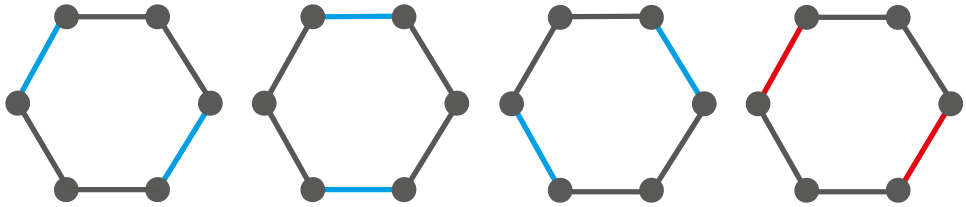
označeny hrany, které jsou alternativní k vybraným hranám. Protože šedé hrany jsou zde v součtu kratší než modré, došlo k jejich výměně a napravo je už optimalizovaný graf.



Obrázek 5 - Princip metody 2-opt, zdroj: autor

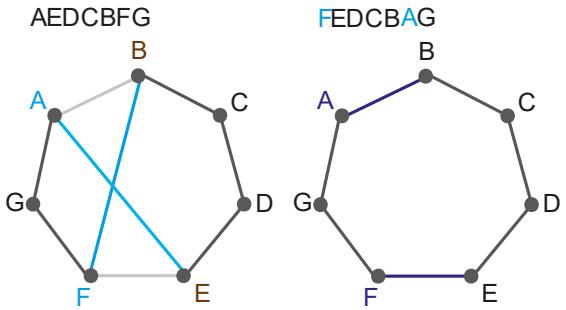
Počet možností pro výběr dvou hran v grafu můžeme spočítat jako $x = \frac{n(n-3)}{2}$. Uvažujeme n jako počet měst. U sudého i lichého počtu vrcholů (tedy i hran) je vzorec pro výpočet počtu možností totožný.

Dvě hrany pro optimalizaci spolu nesmí sousedit, pro takové dvě hrany není možné nalézt jiné dvě alternativní hrany. Pro zjištění všech možných dvojit hran je možné si spočítat jaký je maximální počet hran mezi dvěma vybranými hranami v grafu s tím, že se počítá ten menší počet hran mezi nimi. 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 pokračujeme, dokud nevybereme všechny dvojice s maximálním počtem hran mezi dvěma hranami pro ten graf. Maximální počet hran mezi dvěma vybranými hranami pro 2-opt se u grafu s lichým počtem vrcholů spočítá jako $\frac{n-3}{2}$ a u grafu se sudým počtem vrcholů jako $\frac{n-2}{2}$. 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. 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 takové symetrii nedochází.



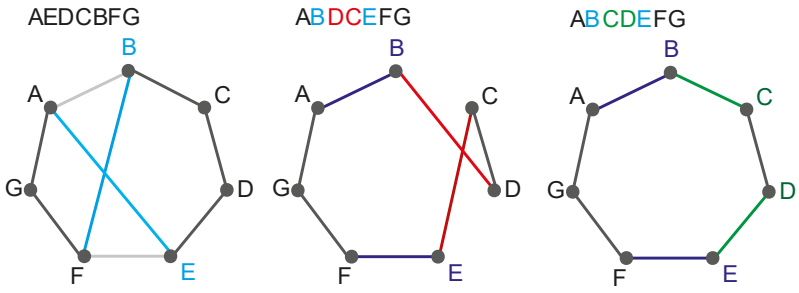
Obrázek 6 - Výběr hran v sudém grafu u 2-opt, zdroj: autor

Pro výměnu hran stačí 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.



Obrázek 7 - Prohození vrcholů u 2-opt, zdroj: autor

Pokud je vzdálenost vybraných hran od sebe dlouhá tři a více hran s tím, že počítáme tu cestu, která nezahrnuje zbylé dva vrcholy incidentní s původními hranami k výměně, je třeba v tomto úseku po výměně hran 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 v pořadí vrcholů AEDCBFG, po výměně vrcholů by cesta měla být ABCDEFG.



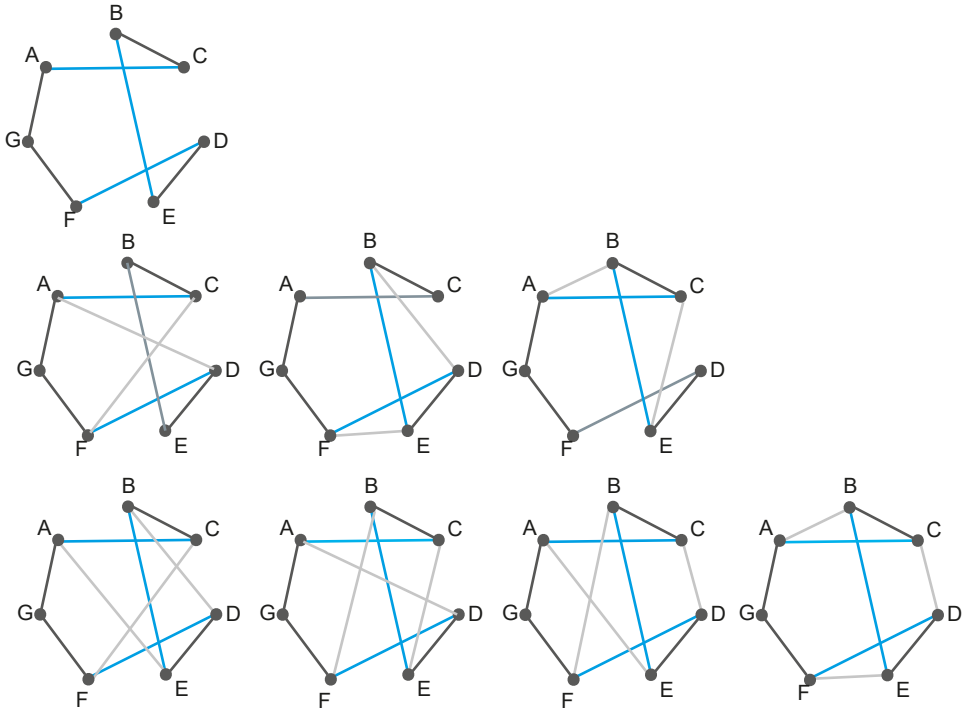
Obrázek 8 - Změna pořadí vrcholů u 2-opt, zdroj: autor

Protože algoritmus iteruje, dokud už nemá co vylepšovat, tak i v případě, že se nezmění pořadí vrcholů je-li to třeba, nalezne téměř vždy, při cestě do 20 vrcholů, lepší řešení než původní. Hrozí zde ale v takovém případě zacyklení. To do jisté míry ukazuje přizpůsobivost algoritmu, který řešení neustále vylepšuje i přes to, že vznikají události, které cestu výrazně zhoršují.

3.5.3.2 Metoda 3-opt

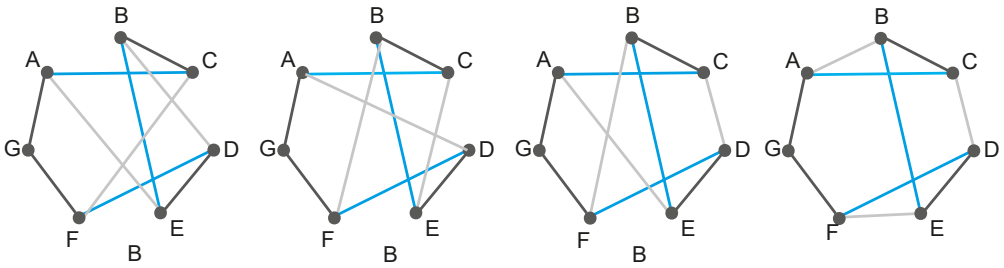
3-opt má asymptotickou časovou složitost $O(n^3)$.

Tato metoda funguje obdobně jako 2-opt. Namísto dvou hran zde 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. Na obrázku je znázorněno všech osm možností, počítají se i výchozí vybrané tři hrany.



Obrázek 9 - Možnosti výměn hran ve 3-opt, zdroj: autor

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 $O(n)$.[13]



Obrázek 10 - Výběr tří hran pro 3-opt, zdroj: autor

5 Vlastní práce

5.1 In Investments s. r. o. a rozvoz smluv

5.1.1 Popis firmy

Firma In Investments s. r. o. je investičním zprostředkovatelem. Zaměřuje se na služby pro investiční poradce, kteří svým klientům sestavují investiční strategie. Následně je podle strategie a potřeb klienta sjednána smlouva, většinou se jedná o investiční smlouvu anebo jinou finanční službu jako například penzijní spoření. Dosud firma zpracovala přibližně 53 500 smluv. In Investments s. r. o. zprostředkovává pro klienty svých poradců tyto smlouvy od různých partnerských společností. Těchto partnerských společností je cca dvacet.[8]

Příklady partnerských společností jsou Amundi Czech Republic, investiční společnost, a.s., AVANT investiční společnost, a.s., Conseq Investment Management, a.s. a Conseq penzijní společnost, a.s., European Investment Centre o.c.p., a.s. (EIC), Generali investiční společnost, a.s., Hello bank!, J&T Banka, a.s., Moventum a.s., Allianz penzijní společnost, a.s., ČSOB Penzijní společnost, a. s. Česká spořitelna — penzijní společnost, a.s., KB Penzijní společnost, a.s..

Celý seznam poboček a jejich adres je uvedený v příloze.

5.1.2 Rozvoz smluv

Jednotlivé smlouvy zpracovávají pracovníci v In Investments s. r. o., zpracované smlouvy je třeba doručit na kontaktní adresy partnerských společností. Je důležité, aby byly smlouvy doručeny brzy, protože trh se vyvíjí a ceny investic se mohou během zpracování změnit, což může ovlivnit výnos pro klienta. Rozvoz zpracovaných smluv mezi partnerské společnosti zajišťuje brigádník dvakrát v týdnu. Brigádník vždy jezdí jinou trasu podle toho, od jakých partnerských společností se smlouvy zpracovaly. Z dvaceti partnerských společností má jen 13 dodací adresu v Praze, kde sídlí i In Investments s. r. o. Do společností s adresou mimo Prahu jsou smlouvy posílány poštou. Brigádník rozváží smlouvy v odpoledních hodinách v omezeném čase a většina partnerských společností má otevírací dobu do pěti hodin odpoledne. Je tedy důležité trasy optimalizovat, aby se zvýšila

pravděpodobnost, že bude možné všechny smlouvy v ten den rozvést. Pracovníci obvykle některé smlouvy k rozvozu ani nepředají, protože je nenapadá dobrá trasa a některé pobočky se pak zdají jako zbytečná zajížďka a pošlou smlouvy raději poštou. Tato rozhodnutí dělají vždy intuitivně.

Při dodání smlouvy partnerské společnosti vydá partnerská společnost předávací protokol, který je třeba dopravit opět do výchozí pobočky In Investments s. r. o. Brigádník ale často předávací protokoly předá až svůj následující pracovní den. Vyzvedne tedy vždy jako první smlouvy v sídle firmy a rozveze je po partnerských společnostech a pak se vrací zpět do sídla firmy, nebo svou trasu ukončí u sebe doma.

Případ, kdy brigádník zahájí svou trasu v sídle firmy a pak se tam i vrátí, je typický problém obchodního cestujícího. Případ, kdy brigádník zahájí svou trasu v sídle firmy a ukončí ji doma, není klasický problém obchodního cestujícího, jedná se o jeho modifikaci. Tuto úlohy lze na problém obchodního cestujícího převést tím způsobem, že mezi výchozím a konečným místem budeme uvažovat nulovou vzdálenost a cestu mezi těmito body vždy zařadíme do řešení. Tato trochu pozměněná úloha zahrnuje celou složitost problému obchodního cestujícího a řeší se stejnými postupy.

5.1.3 Popis vhodného programu pro firmu

Pro zefektivnění rozvozu smluv firmy In Investments s. r. o. by byl vhodný program spustitelný jak na mobilním zařízení, tak na stolním počítači. Bylo by dobré, aby zde byl seznam všech poboček, ze kterých si pracovníci budou moci vybrat pobočky, kam je třeba daný den rozvést smlouvy. Podle zvoleného výběru by se měla zobrazit optimalizovaná trasa mezi pobočkami s výchozím místem v sídle firmy. Dále by zde měla být možnost zadat, že si brigádník přeje zakončit svou cestu doma.

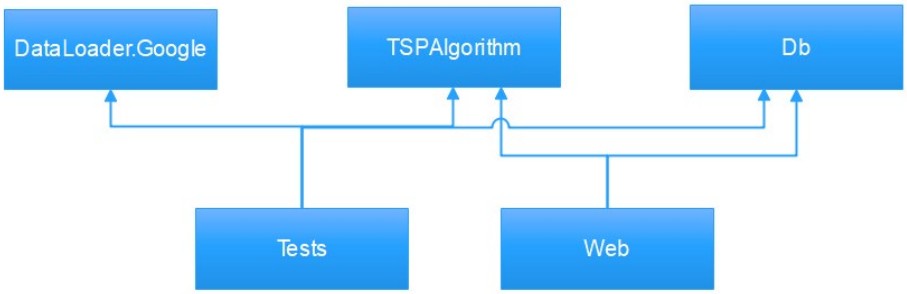
5.2 Implementace programu

Program je implementován jako webová aplikace, protože webová aplikace nejlépe splňuje požadavky popsané v předchozí kapitole. Pro svoji jednoduchost, rozšířenost a dostupnost dokumentace byl zvolen programovací jazyk 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 šablonovací systém Razor. Data jsou uchovávána v souboru ve formátu JSON.

5.2.1 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 unit testy volání Google API, také zde je metoda pro vygenerování vzdáleností mezi všemi pobočkami 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 aplikace. Pro vypočítání trasy volá různé heuristiky z TSPAlgorthm. Web také využívá knihovnu Db pro práci s datovým modelem, například načítání názvu pobočky dle ID místa.



Obrázek 11 - Diagram závislostí v projektu, zdroj: autor

5.2.2 Google API

Pro získání vzdáleností mezi všemi dvaceti pobočkami je využíváno API od Google, konkrétně Geocoding API pro získání id adresy a Distance Matrix API pro získání všech vzdáleností míst mezi sebou. Jednotlivé vzdálenosti jsou získány v různých režimech dopravy a to autem, veřejnou hromadnou dopravou a pěšky. Vzdálenosti jsou měřeny jak v metrických, tak v časových jednotkách. K získání potřebných vzdáleností a časů bylo použito API z toho důvodu, že pro dvacet poboček by muselo být ručně prováděno 1 140

měření. Další výhodou je, že je díky využití API je možné údaje jednoduše aktualizovat či upravit například při změně adresy, smazání či přidání pobočky. Déle by díky API bylo případně možné aplikaci rozšířit tak, aby hledaná trasa vždy odpovídala aktuální dopravní situaci.

5.3 Implementace Algoritmů pro řešení problému obchodního cestujícího

V programu jsou implementovány heuristické algoritmy, a to metoda nejbližšího souseda a hladový algoritmus. Cesta získaná těmito heuristikami se následně ještě vylepšuje pomocí algoritmu 2-opt. Dále 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 místo, ID tohoto místa a díky tomu například vypsat uživateli názvy poboček v nalezené cestě. Dalším vstupním údajem je ID pobočky, která je výchozí, což je v tomto případě vždy In Investments s. r. o. Jako poslední údaj se předává ID konečné pobočky. Pokud je tato hodnota nevyplněná, předpokládá se, že konečná pobočka je výchozí pobočka.

Metody zlepšující řešení mají 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.

5.3.1 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 míst 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 12 - Implementace metody nejbližšího souseda, cyklus pro vytvoření cesty z každé pobočky, zdroj: autor

Metoda `FindNextCity` již nalézá trasu pomocí metody nejbližšího souseda. Nejprve je nalezena množina míst, které ještě nejsou součástí doposud nalezené cesty a zároveň kam je

cesta z posledního navštíveného místa nejkratší. Každé místo z množiny nejbližších míst je postupně zařazeno do řešení. Pro zjištění následujícího nejbližšího místa je metoda opět rekurzivně zavolána. Pokud trasa již obsahuje všechny pobočky, 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 13 - Rekurzivní metoda metody nejbližšího souseda, zdroj: autor

5.3.2 Implementace hladového algoritmu

Pro hladový algoritmus potřebujeme seřazený seznam vzdáleností mezi všemi místy, 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 míst. Kontrolou stupňů vrcholů se zabrání tomu, aby se z jednoho místa 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 míst tvořící výslednou kružnici. Níže je část kódu metody hladového algoritmu.

```
//Seřazení kolekce s cestami
List<RouteIndex> orderedIndexRoutes = indexRoutes.OrderBy(d =>
d.RouteLentght).ToList();
//Pole, kam se zaznamenávají skupiny již vybraných míst
int[] citiesGoups = new int[matrixRoute.placeIDs.Count()];
//Pole, kde se zaznamenávají stupně vrcholů grafu
int[] citiesDeg = new int[matrixRoute.placeIDs.Count()];
//List, kam se přiřadí cesty vybrané hladových algoritmem
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 14 - Implementace hladového algoritmu, zdroj: autor

5.3.3 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 15 - Implementace kolekce kombinací výběru dvou hran pro 2-opt, zdroj: autor

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 libovolnými dvěma hranami v grafu s tím, že vždy vybereme ten menší počet. V tomto for cyklu je vnořený for cyklus, kde se 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, pokolikáté probíhá 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 16 - Implementace zjišťování všech možností výběru dvou hran pro 2-opt, zdroj: autor

Zbývá už jen procházet 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 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.

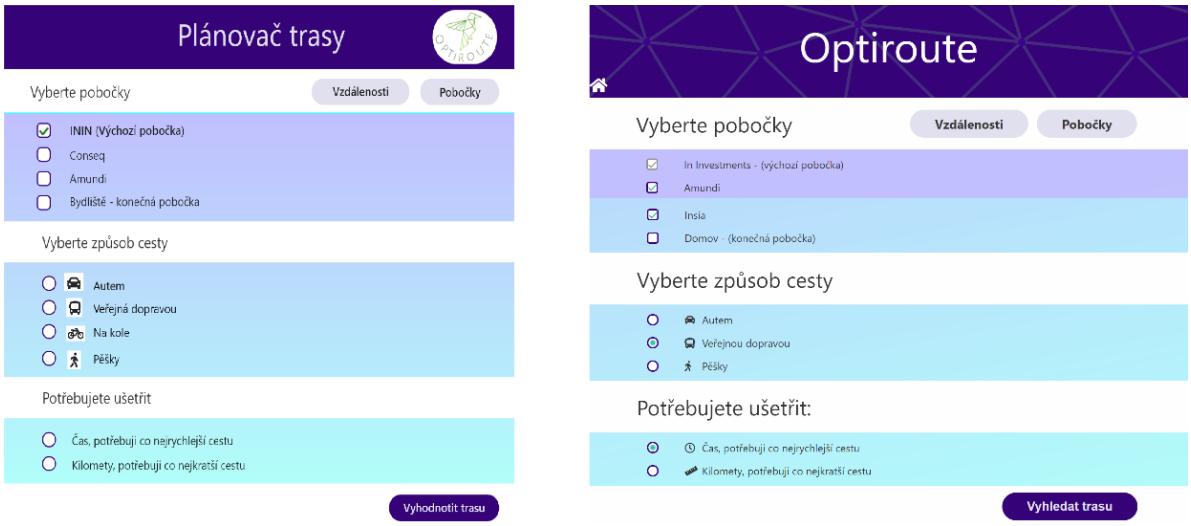
```
List<int> indexRoute = GetIndexRouteFromIdsRoute(inputRoute.AddressisRoute,
matrixRoute.placeIDs).ToList();

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 17 - Implementace prohazování dvou hran metodou 2-opt, zdroj: autor

5.4 Uživatelské rozhraní aplikace

Aplikaci byla pojmenována Optiroute a byla naprogramována podle návrhu, který byl vytvořen v programu Adobe XD. Uživatel si pomocí grafického uživatelského rozhraní webové aplikace volí jednotlivé zaškrtávací pole, kde každé z nich reprezentuje pobočku, kterou uživatel potřebuje zahrnou do své trasy. Defaultně jsou zde vybrané pobočky, na které firma rozváží smlouvy nejčastěji. Pomocí přepínačů si dále uživatel volí způsob dopravy a zdali potřebuje co nejrychlejší trasu anebo co nejkratší trasu. Na obrázku je vlevo návrh úvodní stránky a vpravo již screenshot skutečné webové aplikace, počet poboček je zde pro ilustraci menší než ve skutečnosti.



Obrázek 18 - Rozhraní titulní stránky aplikace, zdroj: autor

Na další stránce se zobrazí trasa vybraných poboček, která byla pomocí použitých algoritmů nalezena jako nejkratší. Pro zobrazení porovnání efektivity algoritmů a tras jednotlivých algoritmů je možné rozkliknout detailní výstup tlačítkem Zobrazit více. Na obrázku je stránka s nejkratší nalezenou trasou, kdy nalevo je návrh a napravo skutečný výstup webové aplikace.

Obrázek 19 - Rozhraní stránky s nalezenou trasou, zdroj: autor

6 Výsledky a diskuse

6.1 Cesta po hlavních městech všech evropských států

Jednotlivé metody je vhodné porovnávat na vzorku s větším počtem míst, aby byly více patrné rozdíly v délce nalezených tras a v délce doby výpočtu těchto algoritmů.

Pro příklad s více než 20 místy byla zvolena úloha pro cestu přes hlavní města všech států Evropy s tím, že počátek a konec bude vždy v české republice. Uznaných států je nyní v Evropě přesně 50. Dále jsou v této kapitole představeny výsledky nalezených tras po Evropě a diskuze o efektivitě jednotlivých implementovaných algoritmů pro nalezení trasy.

6.1.1 Co nejkratší cesta autem po Evropě

Nejkratší nalezená cesta autem pomocí mé aplikace vede přes všechna města v tomto pořadí:

Praha, Česko → Berlín, Německo → Kodaň, Dánsko → Reykjavík, Island → Oslo, Norsko → Stockholm, Švédsko → Helsinki, Finsko → Tallinn, Estonsko → Riga, Lotyšsko → Vilnius, Litva → Minsk, Bělorusko → Varšava, Polsko → Kyjev, Ukrajina → Moskva, Rusko → Nur-Sultan, Kazachstán → Baku, Ázerbájdžán → Jerevan, Arménie → Tbilisi, Gruzie → 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 → Bělehrad, Srbsko → Sarajevo, Bosna a Hercegovina

→ Podgorica, Černá Hora → Tirana, Albánie → Skopje, Severní Makedonie → Sofie, Bulharsko → Bukurešť, Rumunsko → Kišiněv, Moldavsko → Budapešť, Maďarsko → Bratislava, Slovensko → Vídeň, Rakousko → Praha, Česko.

Celková délka této cesty autem je 35 579,8 km.

V tabulce níže je vidět, že si jednoznačně nejlépe vedla metoda nejbližšího souseda, která ale zároveň zabrala nejvíce výpočetního času. Stále se ale jedná o rychlost, která by se dala kategorizovat jako velmi rychlá. Zajímavé je, že optimalizace 2-opt náhodné trasy našla lepší cestu, než optimalizace 2-opt trasy hladového algoritmu i přes to, že cesta získaná hladovým algoritmem je kratší než náhodně vybraná cesta. Úplně nejlepšího výsledku dosáhla optimalizovaná cesta metody nejbližšího souseda. Výpočet trval nejdéle u metody nejbližšího souseda, naopak optimalizace 2-opt trvala v průměru nejkratší dobu.

Tabulka 2 - Přehled efektivity algoritmů pro nalezení nejkratší cesty autem po všech státech Evropy, zdroj: autor

	Délka (m)	Výpočet ms
Random	113 464 134	10
2-opt: Random	36 102 481	9
NS	42 717 561	153
2-opt: NS	**35 579 780**	**< 1**
HA	47 045 208	7
2-opt: HA	36 135 043	**< 1**

6.1.2 Co nejrychlejší cesta autem po Evropě

Pro nejrychlejší trasu přes všechny hlavní města v Evropě byla nalezena pomocí mé aplikace tato trasa:

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 → Baku, Ázerbájdžán → Tbilisi, Gruzie → Jerevan, Arménie → Nikósie, Kypr → Ankara, Turecko → Atény, Řecko → Tirana, Albánie → Podgorica, Černá Hora → Sarajevo, Bosna a Hercegovina → Záhřeb, Chorvatsko → Lublaň, Slovinsko → San Marino, San Marino → Valletta, Malta → Řím, Itálie → Vatikán → Monaco-Ville, Monako → Andorra la Vella, Andorra → Madrid, Španělsko → Lisabon, Portugalsko → Paříž, Francie → Dublin, Irsko → Londýn, Velká Británie → Amsterdam, Nizozemsko → Bruselas, Belgie → Lucemburk, Lucembursko → Bern, Švýcarsko → Vaduz, Lichtenštejnsko → Vídeň, Rakousko → Bratislava, Slovensko → Budapešť, Maďarsko → Bělehrad, Srbsko → Skopje, Severní Makedonie → Sofie, Bulharsko → Bukurešť, Rumunsko → Kišiněv, Moldavsko → Kyjev, Ukrajina → Varšava, Polsko → Praha, Česko.

Celkový čas autem činí 22 dní a 21 hodin a 32 minut.

Z tabulky níže si můžeme všimnout, že výsledky hladového algoritmu a metody nejbližšího souseda nejsou tolik odlišné, jako v předchozím příkladě. Avšak metoda nejbližšího souseda opět nalezla nejkratší cestu a po optimalizaci 2-opt této trasy byla získána ze všech nejrychlejší trasa. Rychlost výpočtu je opět nejdelší u metody nejbližšího souseda. 2-opt náhodné trasy má lepší výsledek než hladový algoritmus nebo metoda nejbližšího souseda.

6. Tabulka 3 - Přehled efektivity algoritmů pro nalezení nejrychlejší trasy autem po všech státech Evropy, zdroj: autor

	Délka (min)	Výpočet ms
Random	79 062	9
2-opt: Random	34 433	8
NS	35 923	154
2-opt: NS	**32 972**	5
HA	38 194	2
2-opt: HA	33 594	**< 1**

Co nejkratší cesta pěšky po Evropě

Pro nejkratší cestu evropskými hlavními městy pěšky byl nalezen následující okruh:

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

Celková vzdálenost cesty je 34 484,9 km.

Metoda nejbližšího souseda našla nepatrně kratší cestu než hladový algoritmus. Hladový algoritmus má nejlepší výpočetní složitost ze všech algoritmů pro nalezení řešení, je dokonce rychlejší než nalezení náhodné cesty. Pravděpodobně to bude tím, že matematický algoritmus pro generování pseudo-náhodných čísel je složitější. 2-opt je nejrychlejší ze všech vybraných algoritmů. Je ale vidět, že čas výpočtu je zde ovlivněný kvalitou řešení na vstupu. Kvalita nalezené trasy pomocí metody 2-opt zde ale není tak výrazně ovlivněná kvalitou neboli celkovou délkou trasy na vstupu metody. Všechny cesty po optimalizaci 2-opt mají totiž podobnou délku. 2-opt hladového algoritmu má dokonce lepší výsledek než 2-opt metody nejbližšího souseda i přes to, že metoda nejlepšího souseda našla kratší cestu.

Tabulka 4 - Přehled efektivity algoritmů pro nalezení nejkratší trasy pěšky po všech státech Evropy, zdroj: autor

	Délka (m)	Výpočet ms
Random	108 183 524	14
2-opt: Random	35 166 632	13
NS	40 051 177	153
2-opt: NS	34 898 107	**< 1**
HA	40 942 329	7
2-opt: HA	**34 484 864**	**< 1**

6.1.4 Co nejrychlejší cesta pěšky po Evropě

Pro nejrychlejší cestu evropskými hlavními městy pěšky byl pomocí mé aplikace nalezen následující okruh:

Praha, Česko → Berlín, Německo → Kodaň, Dánsko → Reykjavík, Island → Oslo, Norsko → Stockholm, Švédsko → Tallinn, Estonsko → Helsinki, Finsko → Riga, Lotyšsko → Vilnius, Litva → Minsk, Bělorusko → Varšava, Polsko → Budapešť, Maďarsko → Bělehrad, Srbsko → Sarajevo, Bosna a Hercegovina → Podgorica, Černá Hora → Tirana, Albánie → Skopje, Severní Makedonie → Sofie, Bulharsko → Bukurešť, Rumunsko → Kišiněv, Moldavsko → Kyjev, Ukrajina → Moskva, Rusko → Nur-Sultan, Kazachstán → Baku, Ázerbájdžán → Tbilisi, Gruzie → Jerevan, Arménie → Ankara, Turecko → Nikósie, Kypr → Atény, Řecko → 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 → Valletta, Malta → Řím, Itálie → Vatikán → San Marino, San Marino → Lublaň, Slovinsko → Záhřeb, Chorvatsko → Bratislava, Slovensko → Vídeň, Rakousko → Praha, Česko.

Cesta celkově trvá 244 dní a 14 hodin a 18 minut.

I v tomto případě byla nejlepší cesta nalezená pomocí optimalizace 2-opt cesty nalezené hladovým algoritmem, kdy tyto oba algoritmy v součtu mají nejrychlejší výpočet. Zatím poprvé dosáhl hladový algoritmus lepších výsledků než metoda nejbližšího souseda. 2-opt náhodné cesty jen nepatrně zaostává za zbylými 2-opt optimalizovanými trasami v délce nalezené cesty. Metoda nejbližšího souseda opět zabírá nejvíce výpočetního času.

Tabulka 5 - Přehled efektivity algoritmů pro nalezení nejrychlejší trasy pěšky po všech státech Evropy, zdroj: autor

	Délka (min)	Výpočet ms
Random	1 109 529	10
2-opt: Random	369 033	8
NS	429 266	149
2-opt: NS	352 471	**< 1**
HA	411 764	6
2-opt: HA	**352 218**	5

6.2 Cesta po partnerských společnostech

Jako další vzorek pro porovnání efektivity různých algoritmů byla zvolena cesta přes všechny pobočky partnerských společností firmy In Investments s. r. o. a přes všechny pobočky, které firma jezdí nejčastěji. Jedná se o trasy zvolené tak, aby celkový čas cesty byl co nejmenší.

6.2.1 Cesta přes všechny partnerské společnosti

Protože jsou pobočky rozmístěny v různých městech České republiky, je jako způsob přepravy zvolena jízda autem. Partnerských společností je dohromady dvacet. Pro nejrychlejší cestu po všech pobočkách partnerských společnostech se začátkem a koncem cesty v sídle firmy byl mou aplikací nalezen tento okruh:

In Investments → Amundi → Insia → INVESTIKA → Generali → Avant → Česká spořitelna → EIC → Efekta → Moventum → Uniqa → EKKA-Gold → Artesa → ČSOB → Hello bank! → NN → Tesla → Conseq → Allianz → J&T Banka → KB → In Investments.

Celkový čas trasy je 11 hodin a 27 minut.

Hladový algoritmus dokázal nalézt nepatrně rychlejší trasu než metoda nejbližšího souseda. Obě tyto trasy bylo možné vylepšit pomocí metody 2-opt. Trasa získaná metodou nejbližšího souseda byla pomocí 2-opt optimalizována na rychlejší trasu než trasa získaná hladovým algoritmem. Náhodná cesta byla metodou 2-opt zkrácena přibližně na třetinový čas, čímž se získala cesta, která je nejrychlejší ze všech. V rychlosti výpočtu jednotlivých algoritmů mezi přibližně dvaceti místy není tak výrazný rozdíl, jako tomu bylo u trasy s padesáti místy.

Tabulka 6 - Přehled efektivity algoritmů pro nalezení nejrychlejší trasy autem po všech pobočkách, zdroj: autor

🕒🚗	Délka (min)	Výpočet ms
Random	1 995	10
2-opt: Random	**687**	9
NS	714	18
2-opt: NS	688	**< 1**
HA	701	8
2-opt: HA	695	**< 1**

6.2.2 Cesta po nejčastějších partnerských společnostech

Všechny tyto pobočky se nacházejí v Praze. Nejčastěji se při rozvozu smluv začíná v sídle firmy a končí v místě bydliště rozvozce. Většinou jsou smlouvy rozváženy za použití veřejné hromadné dopravy. Poboček, do kterých se nejčastěji rozváží smlouvy, je

dohromady 13 s tím, že není započítáno sídlo firmy ani domov brigádníka. Nejkratší trasa byla nalezena v tomto pořadí:

In Investments → Amundi → ČSOB → NN → Hello bank! → Tesla → Conseq → J&T Banka → Allianz → Insia → INVESTIKA → Česká spořitelna → Avant → Generali → domov.

Celkový čas trvání této trasy je 3 hodiny a 55 minut. Na obrázku níže jsou uvedeny časy mezi jednotlivými pobočkami.

Délka trasy:
3 h 55 min

Z	Délka	Do
In Investments	14 min	Amundi
Amundi	25 min	ČSOB
ČSOB	9 min	NN
NN	4 min	Hello bank!
Hello bank!	17 min	Tesla
Tesla	13 min	Conseq
Conseq	11 min	J&T Banka
J&T Banka	5 min	Allianz
Allianz	17 min	Insia
Insia	18 min	INVESTIKA
INVESTIKA	28 min	Česká spořitelna
Česká spořitelna	14 min	Avant
Avant	5 min	Generali
Generali	50 min	Domov

Obrázek 20 - Trasa autem po všech nejčastějších partnerských společnostech, zdroj: autor

Cesta nalezená hladovým algoritmem je i v tomto případě rychlejší než cesta nalezená metodou nejbližšího souseda. Optimalizací 2-opt těchto algoritmů byla nalezena totožná trasa. Nejlepší cesta byla i v tomto případě nalezená optimalizací 2-opt náhodně zvolené cesty.

Tabulka 7 - Přehled efektivity algoritmů pro nalezení nejrychlejší trasy veřejnou dopravou po všech nejčastějších pobočkách, zdroj: autor

	Délka (min)	Výpočet ms
Random	362	< 1
2-opt: Random	**235**	< 1
NS	252	< 1
2-opt: NS	236	< 1
HA	248	< 1
2-opt: HA	236	< 1

6.3 Optimalizace tras firmy In Investments s. r. o.

Firma jezdí pokaždé jiné okruhy podle toho, jaké smlouvy je zrovna v ten den potřeba rozvést. Právě proto, že se trasa vždy mění, byla naprogramovaná webová aplikace, kde si zaměstnanci mohou zvolit požadované pobočky. Jak bylo zmíněno, firma si nyní intuitivně volí, na jaké pobočky pojede brigádník a na jaké pobočky se smlouvy pošlou poštou. Z tohoto důvodu si firma volí pobočky, o kterých ví, že jsou blízko sebe a naplánovat na ně trasu není složité. S použitím aplikace je možné, že firma bude moci s podobným úsilím rozvážet větší množství smluv. Pro konkrétní ilustraci zde uvedu nějaké příklady tras, které firma už jela a uvedu i její optimalizovanou trasu pomocí aplikace. Jedná se o příklady tras dvou brigádníků. Nyní rozváží smlouvy už jen jeden z nich s tím, že první dva příklady jsou trasy, které jel nynější brigádník a zbylé příklady jsou trasy bývalého brigádníka.

Délky mezi pobočkami jsou uvedeny v minutách bez vteřin. Součet všech vzdáleností mezi pobočkami tedy bývá menší než skutečný celkový součet.

6.3.1 První trasa

Například firma již jela trasu v tomto pořadí: In Investments → 14 min. → Amundi → 14 min. → JT Banka → 5 min. → Allianz → 13 min. → Conseq → 55 min. → domov.

Konečná zastávka byla doma u brigádníka, který rozváží smlouvy. Brigádník využil MHD. Celkový čas této trasy je cca 103 minut.

Má webová aplikace našla trasu v pořadí: In Investments → 14 min. → Amundi → 16 min. → Allianz → 5 min. → JT Banka → 11 min. → Conseq → 55 min. → domov. Nalezená trasa je sice o něco jiná, ale její délka je stejná jako je délka trasy původní. Na obrázku níže je vidět výstup z aplikace, kde je uvedena délka celkové trasy 103 minut, nicméně po sečtení jednotlivých délek v tabulce dostáváme číslo 101. Zkreslený výsledek je způsoben zaokrouhlováním čísel získané z Google API.

Trasa:
In Investments -> Amundi -> Allianz -> J&T Banka -> Conseq -> Domov

Délka trasy:
1 h 43 min

Z	Délka	Do
In Investments	14 min	Amundi
Amundi	16 min	Allianz
Allianz	5 min	J&T Banka
J&T Banka	11 min	Conseq
Conseq	55 min	Domov

Obrázek 21 - Optimalizovaná první trasa firmy ve webové aplikaci, zdroj: autor

Stejná trasa byla nalezena jak metodou nejbližšího souseda, tak hladovým algoritmem. Optimalizační metoda 2-opt už v této trase nenalezla žádné vylepšení. Je tedy pravděpodobné, že trasa nalezená programem je zanedbatelně lepší než trasa původní. Přehled o tom, jak dlouhé trasy nalezly jednotlivé algoritmy, je na obrázku níže.

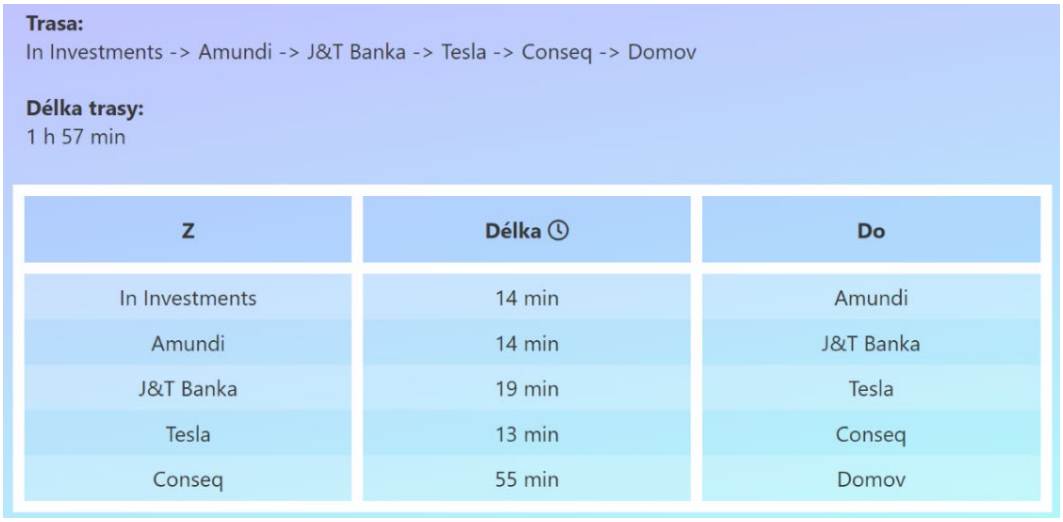
Tabulka 8 - Přehled efektivity algoritmů pro optimalizace první trasy firmy, zdroj: autor

	Délka (min)	Výpočet ms
Random	114	< 1
2-opt: Random	103	< 1
NS	103	< 1
2-opt: NS	103	< 1
HA	103	< 1
2-opt: HA	103	< 1

6.3.2 Druhá trasa

Dalším příkladem cesty, kterou firma jela za využití MHD, je: In Investments → 14 min. → Amundi → 14 min. → J&T Banka → 11 min. → Conseq → 13 min. → Tesla → 68 min. → domov. Její celková délka činí 123 minut.

Trasa nalezená mou aplikací je v pořadí: In Investments → 14 min → Amundi → 14 min. → J&T Banka → 19 min. → Tesla → 13 min. → Conseq → 55 min. → domov. Celková délka činí 117 minut, je tedy o 6 minut kratší než trasa původní.



Obrázek 22 - Optimalizovaná druhá trasa firmy ve webové aplikaci, zdroj: autor

Metoda nejbližšího souseda i hladový algoritmus nalezly opět stejnou trasu, tato trasa ale není optimální, protože ji bylo dále možné vylepšit metodou 2-opt, která našla stejnou trasu i po optimalizaci náhodné cesty.

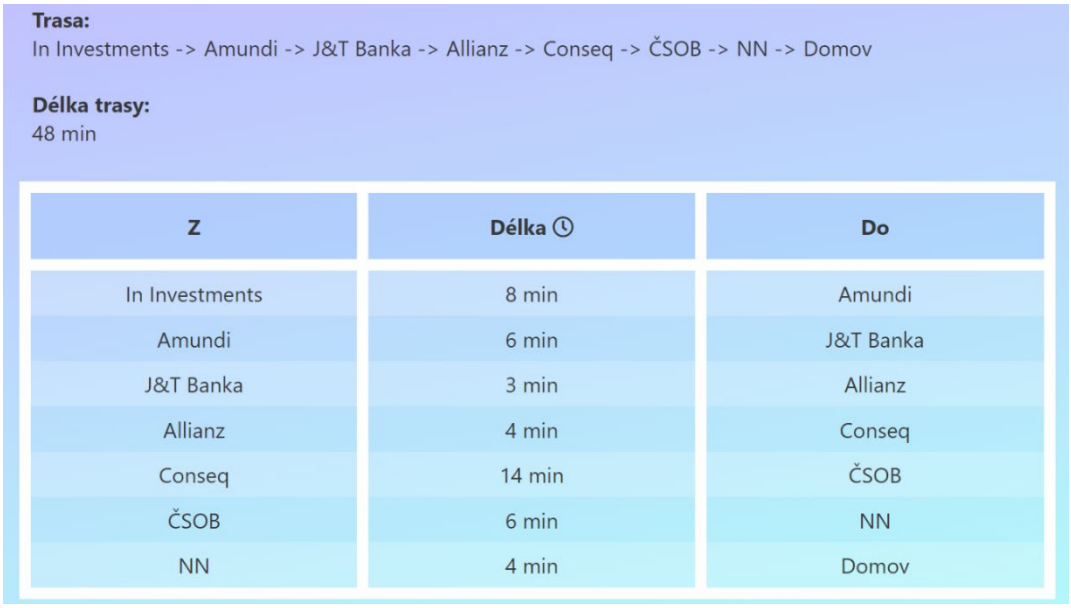
Tabulka 9 - Přehled efektivity algoritmů pro optimalizaci druhé trasy firmy, zdroj: autor

	Délka (min)	Výpočet ms
Random	131	< 1
2-opt: Random	**117**	< 1
NS	123	< 1
2-opt: NS	**117**	< 1
HA	123	< 1
2-opt: HA	**117**	< 1

6.3.3 Třetí trasa

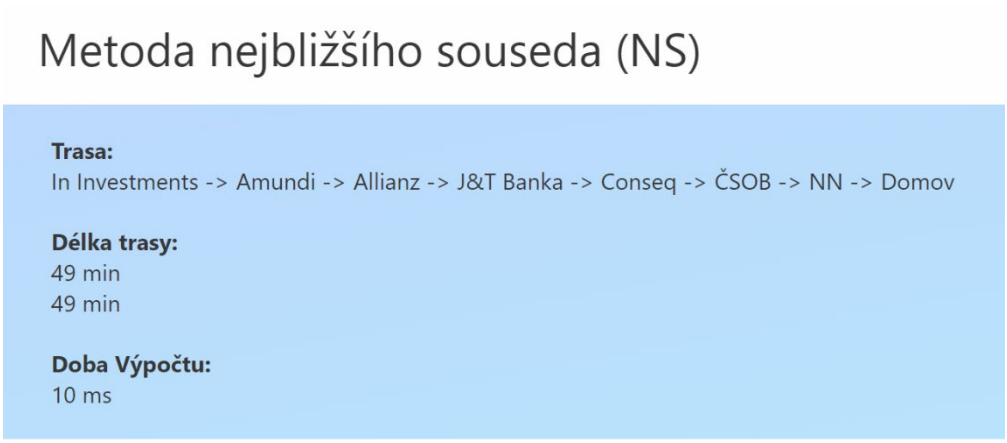
Tato trasa se jela autem a byla zde jiná adresa domova než v předchozích příkladech, tato adresa domova bude i v následujících příkladech. Jedná se o trasu v pořadí: In Investments → 8 min. → Amundi → 5 min. → Allianz → 3 min. → J&T Banka → 5 min. → Conseq → 14 min. → ČSOB → 6 min. → NN → 4 min. → domov. Celková délka činí 49 minut.

Trasa nalezená aplikací je v tomto pořadí: In Investments → 8 min. → Amundi → 6 min. → J&T Banka → 3 min. → Allianz → 4 min. → Conseq → 14 min. → ČSOB → 6 min. → NN → 4 min. → domov. Celková délka trasy činí 48 minut. Zde došlo k vylepšení trasy o jednu minutu.



Obrázek 23 - Optimalizovaná třetí trasa firmy ve webové aplikaci, zdroj: autor

Metoda nejbližšího souseda našla trasu ve stejném pořadí jako jela firma.



Obrázek 24 - Trasa nalezená metodou nejbližšího souseda, zdroj: autor

Nejlépe si vedl hladový algoritmus, který našel trasu, která se už dále nedala optimalizovat pomocí metody 2-opt. Metoda nejbližšího souseda našla trasu jen o minutu delší. Pomocí metody 2-opt došlo k vylepšení všech tras na stejnou celkovou vzdálenost.

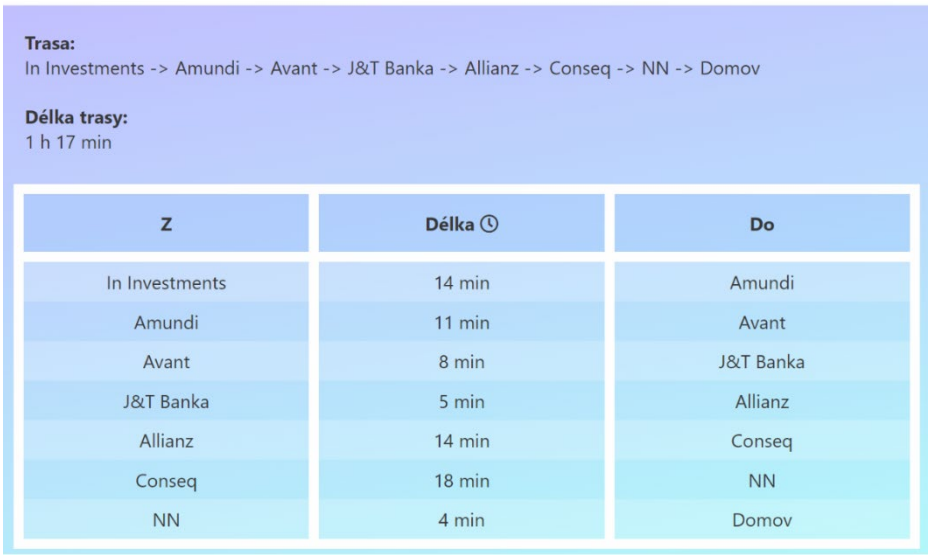
Tabulka 10 - Přehled efektivity algoritmů pro optimalizaci třetí trasy firmy, zdroj: autor

	Délka (min)	Výpočet ms
Random	77	10
2-opt: Random	**48**	1
NS	49	10
2-opt: NS	**48**	**< 1**
HA	**48**	8
2-opt: HA	**48**	**< 1**

6.3.4 Čtvrtá trasa

Tato trasa se jela veřejnou hromadnou dopravou v pořadí: In Investments → 14 min. → Amundi → 11 min. → Avant → 12 min. → Allianz → 5 min. → J&T Banka → 12 min. → Conseq → 18 min. → NN → 4 min. → domov. Celková délka této cesty je 78 minut.

Toto je trasa nalezená aplikací: In Investments → 14 min. → Amundi → 11 min. → Avant → 8 min. → J&T Banka → 5 min. → Allianz → 14 min. → Conseq → 18 min. → NN → 4 min. → domov. Celková délka cesty je 77 minut, tato trasa je tedy o minutu kratší než trasa původní.



Obrázek 25 - Optimalizovaná čtvrtá trasa firmy ve webové aplikaci, zdroj: autor

Metoda nejbližšího souseda i hladový algoritmus nalezly trasu, kterou již dále nebylo možné optimalizovat pomocí metody 2-opt. Doba výpočtu všech algoritmů je velmi podobná, kromě času výpočtu metody 2-opt se vstupní trasou získanou metodou nejbližšího souseda nebo hladovým algoritmem, kde byl vypočet velmi rychlý.

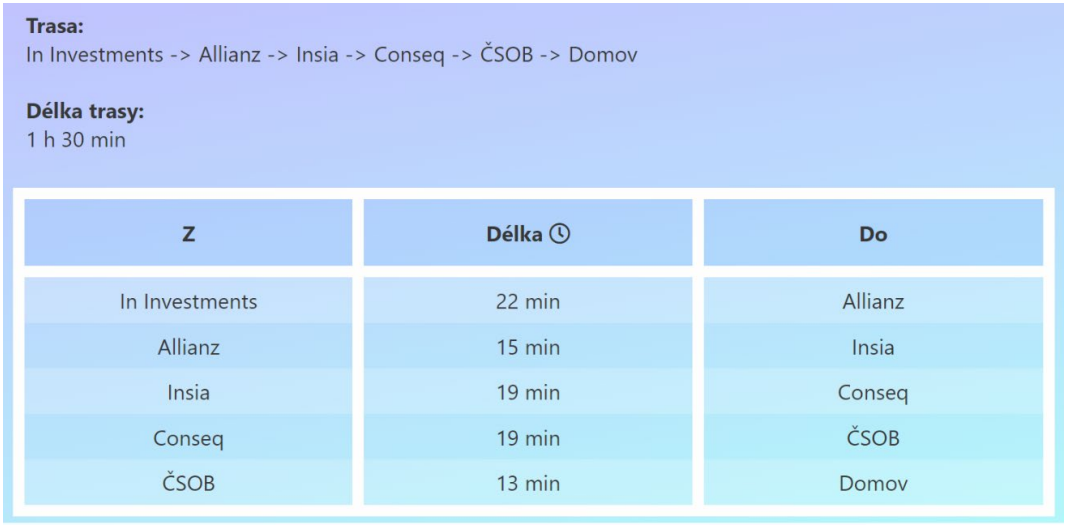
Tabulka 11 - Přehled efektivity algoritmů pro optimalizaci čtvrté trasy firmy, zdroj: autor

	Délka (min)	Výpočet ms
Random	108	10
2-opt: Random	**77**	9
NS	**77**	9
2-opt: NS	**77**	**< 1**
HA	**77**	8
2-opt: HA	**77**	**< 1**

6.3.5 Pátá trasa

Tuto trasu jela firma s využitím hromadné veřejní dopravy v tomto pořadí poboček: In Investments → 29 min. → Insia →15 min. → Allianz → 14 min. → Conseq → 19 min. → ČSOB → 13 min. → domov. Celkový čas trasy je 93 minut.

Pomocí aplikace byla nalezena tato trasa: In Investments → 22 min. → Allianz→15 min. → Insia → 19 min. → Conseq → 19 min. → ČSOB → 13 min. → domov. Celkový čas této trasy je 90 minut. Tato trasa je o tři minuty kratší než trasa, kterou jela firma.



Obrázek 26 - Optimalizovaná pátá trasa firmy ve webové aplikaci, zdroj: autor

Metoda nejbližšího souseda i hladový algoritmus našel stejnou trasu, avšak tuto trasu bylo možné dále vylepšit pomocí 2-opt. Doba výpočtu metody nejbližšího souseda i hladové algoritmu je stejná.

Tabulka 12 - Přehled efektivity algoritmů pro optimalizaci páté trasy firmy, zdroj: autor

	Délka (min)	Výpočet ms
Random	116	10
2-opt: Random	**90**	8
NS	94	9
2-opt: NS	**90**	**< 1**
HA	94	8
2-opt: HA	**90**	**< 1**

7 Závěr

Předmětem práce bylo porovnat některé heuristické metody pro řešení problému obchodního cestujícího. Zaměřila se na metodu nejbližšího souseda, hladový algoritmus a vylepšující metodu 2-opt. Tyto algoritmy byly implementovány do webové aplikace, kde na jejím výstupu je mimo jiné i porovnání jednotlivých algoritmů z hlediska délky nalezené cesty a rychlosti výpočtu. Testovacími příklady bylo nalezení co nejlepší cesty přes hlavní města všech států Evropy a pro rozvoz smluv firmy In Investments s. r. o.

Rychlost výpočtu byla velmi rychlá u všech algoritmů, nicméně metoda nejbližšího souseda byla výrazněji pomalejší než ostatní algoritmy. Rozdíl byl hlavně pozorovatelný na modelu o padesáti městech, kde výpočet metody nejbližšího souseda byl přibližně desetkrát delší než výpočet ostatních algoritmů. Časová výpočetní složitost metody zlepšující řešení 2-opt byla obvykle ze všech nejmenší, nicméně výpočet trval mnohem déle v případě, že na vstupu byla náhodná trasa, která měla obvykle velký prostor pro vylepšování. Metoda nejbližšího souseda s hladovým algoritmem dosahovaly velmi podobných výsledků v délce nalezené trasy. Vylepšující metoda 2-opt nejednou našla nejlepší řešení v případě, že na vstupu měla náhodnou trasu, která byla výrazně delší než trasy získané ostatními heuristikami. Z toho plyne, že u metody 2-opt nehraje významnou roli kvalita řešení na vstupu. Nejlepších výsledků a zároveň v nejrychlejším čase dosáhla kombinace využití hladového algoritmu a metody 2-opt.

Dalším cílem práce bylo navrhnout a vytvořit aplikaci pro rozvoz smluv, kterou bude moci firma In Investments s. r. o. využívat. Pro tyto účely byla vytvořena již zmíněná webová aplikace, která byla firmě představena a firma ji využívá. Zpětná vazba na aplikaci byla pozitivní. Jako námět na další vylepšení firma uvedla, že by aplikace mohla pracovat s aktuální dopravní situací a v případě jízdy MHD by mohla zobrazit i konkrétní dopravní spoje. Aplikace je navržena tak, že rozšíření o tyto funkce by bylo možné, vznikly by ale související poplatky za vyšší využívání Google API.

Vzhledem k tomu, že aplikace slouží k optimalizaci tras firmy, jsou v práci uvedeny některé příklady cest, které firma jela a tyto cesty jsou porovnány s cestami nalezenými pomocí mé aplikace. Pro čtyři z pěti uvedených tras se mi povedlo najít lepší řešení, ale jen o pár minut. Pro tu jednu zbylou trasu byla nalezena stejně dlouhá alternativa. Firma tedy měla své trasy na základě intuice promyšlené dobře.

8 Seznam použitých zdrojů

[1] Applegate D, Bixby R, Chvátal V, Cook W. Implementing the Dantzig-Fulkerson-Johnson algorithm for large traveling salesman problems. Mathematical Programming. 2003. Dostupné z: doi:10.1007/s10107-003-0440-4

[2] COOK, W. Po stopách obchodního cestujícího: matematika na hranicích možností. Praha: Dokořán, 2012. ISBN 978-80-7363-412-4.

[3] Du, P., Liu, N., Zhang, H., Lu. J. An Improved Ant Colony Optimization Based on an Adaptive Heuristic Factor for the Traveling Salesman Problem. Journal of Advanced Transportation. October 2021: Dostupné z: doi:10.1155/2021/6642009

[4] FORTNOW, Lance. The status of the P versus NP problem. Communications of the ACM. 2009, 52(9). ISSN 0001-0782. Dostupné z: doi:10.1145/1562164.1562186

[5] HELSGAUN, Keld. An Effective Implementation of K-opt Moves for the Lin-Kernighan TSP Heuristic. Computer Science, Roskilde University, 2006. Denmark. ISSN 0109-9779

[6] HLADÍK, M. Celočíselné Programování: text k přednášce [online]. 2017. Dostupné také z: https://kam.mff.cuni.cz/~hladik/CP/text_cp.pdf

[7] HLINĚNÝ, P. Základy teorie grafů pro (nejen) informatiky. Brno: Masarykova univerzita. Fakulta Informatiky, 2011

[8] In Investments s. r. o. [online]. Praha. Dostupné z: https://www.ininvest.cz/

[9] KUČERA, L. Kombinatorické algoritmy. Praha: SNTL - Státní nakladatelství technické literatury, 1989. ISBN 04-009-89

[10] LAPORTE, G. A Concise Guide to the Traveling Salesman Problem. The Journal of the Operational Research Society. 2016. Dostupné z: doi:10.1057/jors.2009.76

[11] LIN, Shen. Computer solutions of the traveling salesman problem. The Bell System Technical Journal. 1965, 2245 - 2269. Dostupné z: doi:10.1002/j.1538-7305.1965.tb04146.x

[12] MATOUŠEK, J. Kapitoly z diskrétní matematiky. Vyd. 2. Praha: Karolinum, 2000. ISBN 80-246-0084-6

[13] Misevicius, Alfonsas. Combining 2-OPT, 3-OPT and 4-OPT with K-SWAP-KICK perturbations for the traveling salesman problem. 2011.

[14] Ronald V. Reducibility Among Combinatorial Problems. Richard M. Karp Raymond E. Miller James W. Thatcher. The Journal of Symbolic Logic. 1975;40(4). Dostupné z: doi:10.2307/2271828

[15] SAVICKÝ, P. Výpočetní složitost I: pro obor logika na FF UK [online]. 2016. Dostupné také z: http://www.cs.cas.cz/~savicky/vyuka/vypsl/vypsl2016zs1.pdf

[16] TESAŘ, K. a M. MAREŠ. Složitost [online]. Dostupné také z: https://ksp.mff.cuni.cz/kucharky/programatorske-kucharky/01-slozitost.pdf

9 Přílohy

Partnerské společnosti firmy In Investments s. r. o.

In Investments - (výchozí pobočka)
In Investments s. r. o.
K Moravině 1871, 190 00 Praha 9-Libeň

Amundi
Amundi Czech Republic, investiční společnost, a.s.
Rohanské nábř. 693, 186 00 Praha 8-Karlín

Avant
AVANT investiční společnost, a.s.
City Tower, Hvězdova, 140 00 Praha 4-Nusle

Conseq
Conseq Investment Management, a.s. & Conseq penzijní společnost, a.s.
Rybná 682, 110 00 Praha-Staré Město

Efekta
EFEKTA obchodník s cennými papíry a.s.
nám. Svobody 91/20, 602 00 Brno-střed-Brno-město

EIC
European Investment Centre o.c.p., a.s., (EIC)
Veveří 2581/102, 616 00 Brno-Žabovřesky

Generali
Generali penzijní / investiční společnost, a.s.
123, Na Pankráci 1720, 140 00 Praha 4-Nusle

Hello bank!
Karla Engliše 3208/5, Smíchov, 150 00 Praha-Praha 5

INVESTIKA
INVESTIKA, investiční společnost, a.s.
U Zvonařky 291, 120 00 Vinohrady

J&T Banka
J&T Banka, a.s.
Pobřežní 14, 186 00 Karlín

Moventum
Moventum a.s.
Bašty 413/2, 602 00 Brno-střed-Brno-město

Tesla
Tesla investiční společnost, a.s.
Konviktská 291, 110 00 Praha-Staré Město

Artesa
Artesa, spořitelní družstvo
28. října 212/37, 702 00 Moravská Ostrava a Přívoz

EKKA-Gold
EKKA-Gold, s.r.o.
Kostelní 89/13, 702 00 Moravská Ostrava a Přívoz

Allianz
Allianz penzijní společnost, a.s.
Ke Štvanici 656/3, 186 00 Karlín

ČSOB
ČSOB Penzijní společnost, a. s.
Radlická 333/150, 150 00 Praha 5

Česká spořitelna
Česká spořitelna — penzijní společnost, a.s.
Poláčkova 1976, 140 00 Praha 4-Krč

KB
KB Penzijní společnost, a.s.
Bílinská 175/2, 400 01 Ústí nad Labem-město-Ústí nad Labem-centrum

NN
NN Penzijní společnost, a.s.
Nádražní 344, Smíchov, 150 00 Praha-Praha 5

Uniqa
Uniqa penzijní společnost a.s.
Úzká 488, 602 00 Brno-střed-Trnitá

Insia
INSIA a.s.
Vinohradská 2828/151, 130 00 Praha 3-Žižkov