VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

OKRUŽNÍ PROBLÉMY A JEJICH ŘEŠENÍ

ROUTING PROBLEMS AND THEIR SOLUTIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Václav Pospíšil
AUTHOR

VEDOUCÍ PRÁCE — prof. RNDr. Ing. Miloš Šeda, Ph.D.
SUPERVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Václav Pospíšil**
Studijní program:	Aplikovaná informatika a řízení
Studijní obor:	Bez specializace
Vedoucí práce:	**prof. RNDr. Ing. Miloš Šeda, Ph.D.**
Akademický rok:	2021/22

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Okružní problémy a jejich řešení

Stručná charakteristika problematiky úkolu:
Okružní problémy patří do třídy NP-těžkých úloh a mají několik specifických podob. Jednou z nich je úloha obchodního cestujícího (Travelling Salesman Problem), kdy cílem je najít nejkratší okružní cestu, na níž obchodník navštíví všechny zákazníky. V úloze plánování rozvozu vozidly (Vehicle Routing Problem) je nutné zohlednit jejich kapacitu.

Cíle diplomové práce:
Provést rozbor problémů a pro reálná data v závislosti na jejich počtu implementovat jejich řešení (deterministické pro menší rozsah a heuristické pro velké instance).

Seznam doporučené literatury:
CARIC, T., GOLD, H.: Vehicle Routing Problem. In Tech, 2008

LABADIE, N., PRINS, C., PRODHON, C.: Metaheuristics for Vehicle Routing Problems. New York: Wiley, 2016.

POTVIN, J.-Y.: Bio-inspired Algorithms for the Vehicle Routing Problem. Berlin, Springer-Verlag, 2009

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2021/22

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D.	doc. Ing. Jaroslav Katolický, Ph.D.
ředitel ústavu	děkan fakulty

ABSTRAKT

Práce je v první části věnována úvodu a ucelenému popisu všech důležitých pojmů teorie grafů, na kterou navazuje popis a modifikace dvou vybraných typů okružních problémů: problému obchodního cestujícího a problému plánování rozvozu. Další část práce se věnuje následné možnosti řešení problémů skrze deterministické a stochastické algoritmy. Součástí je taktéž část praktická, která se v závěru práce zabývá optimalizací nejkratší cesty dvou vytvořených modelů pomocí metody nejbližšího souseda, genetického algoritmu a řešiče v modelovacím jazyce GAMS.

ABSTRACT

The first part of the thesis is devoted to an Introduction and a comprehensive description of all important concepts of graph theory, which is followed by descriptions and modifications of two selected types of routing problems: the travelling salesman problem and the vehicle routing problem. The next part of the thesis deals with subsequent possibilities of solving problems through deterministic and stochastic algorithms. It also includes a practical part, which at the end of the thesis deals with the shortest path optimization of the two created models using Nearest neighbour algorithm, Genetic algorithm and solver in GAMS.

KLÍČOVÁ SLOVA

Okružní problémy, Problém obchodního cestujícího, Problém plánování rozvozu, Metoda nejbližšího souseda, Genetický algoritmus, optimalizace nejkratší cesty

KEYWORDS

Routing problems, Travelling Salesman Problem, Vehicle Routing Problem, Nearest Neighbour algorithm, Genetic algorithm, shortest path optimization

ÚSTAV AUTOMATIZACE A INFORMATIKY

2022

BIBLIOGRAFICKÁ CITACE

POSPÍŠIL, Václav. Okružní problémy a jejich řešení. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky, 2022, 83 s. Diplomová práce. Vedoucí práce: prof. RNDr. Ing. Miloš Šeda, Ph.D.

PODĚKOVÁNÍ

Chtěl bych poděkovat vedoucímu mé diplomové práce prof. RNDr. Ing. Miloši Šedovi, Ph.D. za jeho cenné rady, ochotu a trpělivost, které mi pomohly tuto práci dokončit. Dále bych rád poděkoval mé rodině za podporu po celou dobu studia.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že, že tato práce je mým původním dílem, vypracoval jsem ji samostatně pod vedením vedoucího práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury.

Jako autor uvedené práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následku porušení ustanovení § 11 a následujících autorského zákona c. 121/2000 Sb., včetně možných trestně právních důsledků.

V Brně dne 20. 5. 2022 ……

Václav Pospíšil

OBSAH

1 ÚVOD

S plánováním různých cest a tras se člověk setkává již od pradávna, ale vytvořit ručně jednoduchou a efektivní trasu se může zdát jako prakticky nemožný úkol, a proto se v dnešní době ve společnosti rozšiřují stále více okružní problémy, které se plánováním nejrůznějších cest a tras zabývají. Řešení těchto optimalizačních problémů pomáhá především s minimalizací nákladů na cestu. Své využití v praxi nalézají téměř v každém scénáři, kde je potřeba provést jakoukoliv dodávku, ať už se jedná o zboží nebo o pouhou přepravu osob.

Cílem diplomové práce je čtenáře seznámit s problematikou týkající se okružních problémů, jejich vývojem, základní charakteristikou a přiblížit dva významné problémy: problém obchodního cestujícího a problém plánování rozvozu. Dále rozebrat jejich možné řešení skrze deterministické a stochastické algoritmy a následné vytvoření testovacích modelů pro reálná data. Smyslem je tedy vytvoření uceleného přehledu daného tématu, provedení následné optimalizace a porovnání testovacích modelů pomocí deterministického a stochastického přístupu.

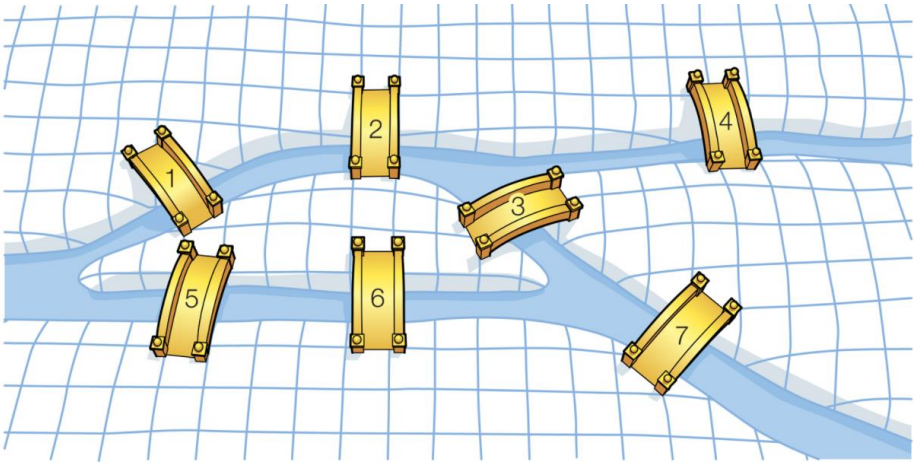
K základnímu pochopení daného tématu autor využil doporučené literatury, která mu dodala základní poznatky a pomohla s výběrem daných metod k řešení okružních problémů a následnou implementací.

2 OKRUŽNÍ PROBLÉMY

2.1 Základy teorie grafů

Teorie grafů je poměrně mladý obor matematiky, který se zabývá sítěmi bodů (uzlů), které jsou spojeny úsečkami (hranami). Tento obor měl své počátky spíše v rekreačních matematických úlohách, ale přerostl do důležité oblasti matematiky s využitím v mnoha oblastech. Své uplatnění nachází v informatice, chemii, biologii, ale také například ve společenských vědách [1].

Za počátek teorie grafů lze považovat rok 1736, kdy švýcarský matematik Leonhard Euler poprvé představil tzv. „problém Sedmi mostů města Königsbergu“ (dnešního Kaliningradu v Rusku). Tento problém spočíval ve vyřešení hádanky, která měla za úkol najít cestu přes každý ze sedmi mostů, které se tyčily nad rozvětvenou řekou tak, aby osoba prošla přes daný most právě jednou a vrátila se zpět do místa počátku. Euler převedl danou hádanku na graf, kde každý břeh reprezentoval jeden vrchol a mosty představovaly hrany grafu. Matematicky dokázal, že daná úloha je neřešitelná [1,2].

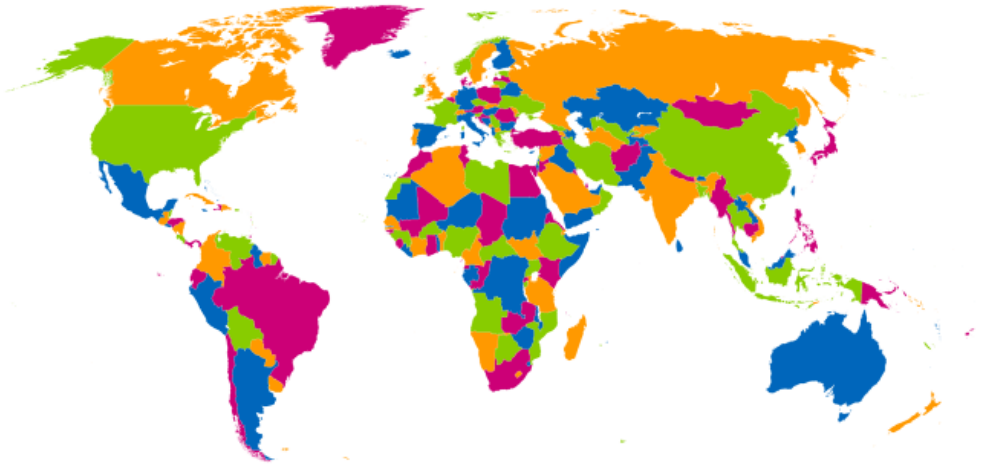


Obr. 1: Problém Sedmi mostů města Königsbergu. Zdroj: [1]

Další zmínky o teorii grafů se objevily v roce 1847, kdy se Gustav Kirchhoff zabýval výpočtem proudů v elektrických sítí pomocí teorie stromů. Problematiku stromů rozvíjel i matematik Arthur Cayley v oblasti chemie [3].

Roku 1859 byla vytvořena hra „The Icosian Game“, jejímž autorem je William Rowan Hamilton. Hra spočívá na principu nalezení hamiltonovské kružnice v grafu, který odpovídá pravidelnému dvanáctistěnu [3].

Nejrozšířenějším problémem teorie grafů v 19. století se stal bezpochyby „problém čtyř barev“. Jedná se o obarvení libovolné mapy pouze čtyřmi barvami. Problém se podařilo vyřešit až v roce 1976 s využitím počítače [3].



Obr. 2: Řešení problému čtyř barev na mapě světa. Zdroj: [4]

O rozvoj teorie grafů ve 20. století se zasloužil také brněnský matematik Otakar Borůvka, který se zabýval ekonomickou elektrifikací moravských vesnic. Zasloužil se o vytvoření vlastního algoritmu (Borůvkův algoritmus) pro nalezení minimální kostry ohodnoceného grafu. Stejný problém později vyřešil i Vojtěch Jarník pomocí vlastního algoritmu (Jarníkův algoritmus) [3].

2.1.1 Základní pojmy

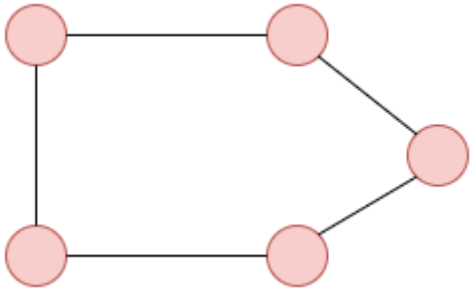
V reálném světě lze mnoho situací popsat **grafem**, který udává přehled o daných objektech a vztahy mezi nimi. Matematicky takový graf lze popsat uspořádanou dvojicí G = (V, E), kde označení V a E jsou odvozena od anglických slov vertices (vrcholy) a edges (hrany). Z tohoto zápisu vyplývá, že graf se skládá z vrcholů a hran [5].

Podle vlastností se grafy dělí na různé druhy. Grafy můžeme dělit podle počtu vrcholů, počtu hran, vzájemné propojenosti a jejich celkové struktuře [6].

K popisu byly vybrány základní typy grafů: nulový, neorientovaný, orientovaný, souvislý, úplný, biparitní a ohodnocený. Dále byly do popisu zařazeny další pojmy (stupeň vrcholu, Hamiltonova cesta a Eulerovský tah), které jsou využívány v následujících částech práce.

Nejzákladnějším typem grafu je **nulový**, někdy také označovaný jako **prázdný** graf. Jde o případ, kde mezi jednotlivými vrcholy neexistují žádné hrany.

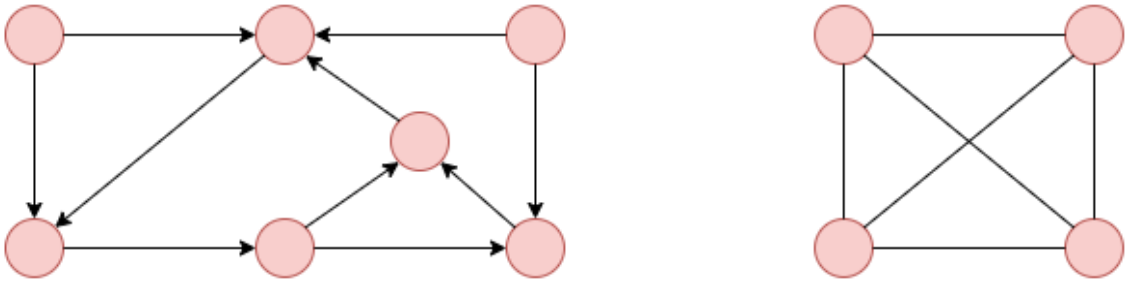
Neorientovaný graf je takový, jehož hrany nemají daný směr. Graf, ve kterém z každého vrcholu vede cesta do všech ostatních, se nazývá **souvislý** [6].



Obr. 3: Neorientovaný souvislý graf

Naopak, pokud budou mít hrany v grafu daný směr, bude se hovořit o **orientovaném** grafu.

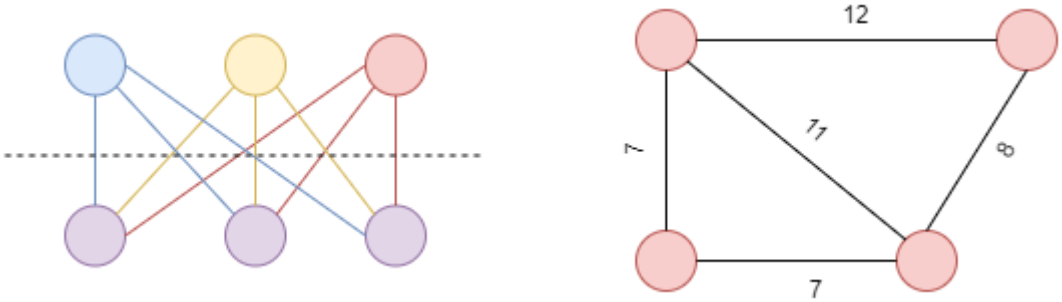
Úplný graf je takový, který obsahuje všechny možné hrany. Tudíž každá dvojice vrcholů je spojena hranou [6].



Obr. 4: Orientovaný graf (vlevo) a úplný graf (vpravo)

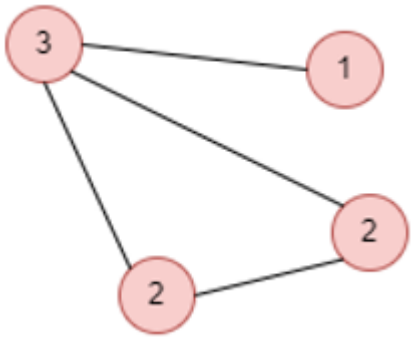
Pokud lze graf rozdělit na dvě množiny tak, že hrany spojují vrcholy pouze mezi množinami, nikoli uvnitř nich, jedná se o **biparitní** graf.

Hrany hrají v grafu velikou roli, jelikož mohou být orientované, neorientované, ale také mohou mít svou délku. Jestliže se takové hrany v grafu nacházejí, jde o graf **ohodnocený** [6].



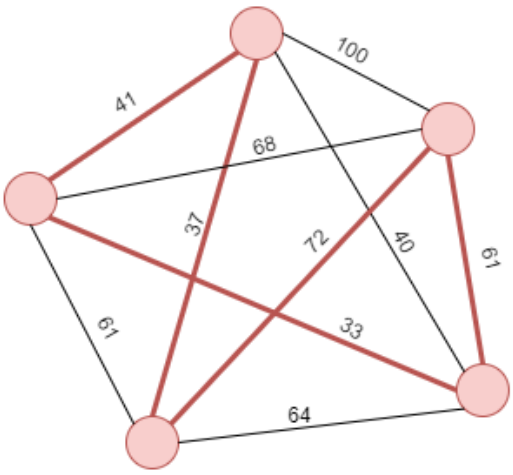
Obr. 5: Biparitní graf (vlevo) a ohodnocený graf (vpravo)

Dalším velice důležitým pojmem je **stupeň vrcholu**. U orientovaného grafu se určí počtem hran, se kterými je daný vrchol incidentní. Jedná se o takové hrany, které z vrcholu vystupují nebo naopak do něj vstupují. U neorientovaného grafu je stupeň vrcholu roven počtu hran, které jsou s daným vrcholem spojeny [5].



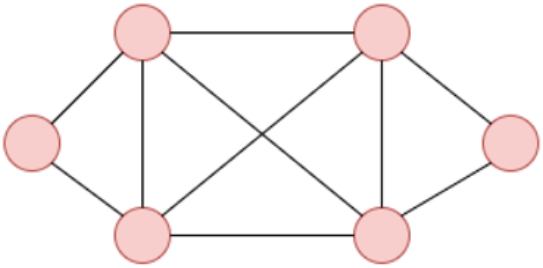
Obr. 6: Graf s označenými stupni vrcholů

Hamiltonova cesta (nebo také hamiltonovská) je taková, která prochází všemi vrcholy grafu právě jednou. Podmínkou nalezení této cesty je, že graf musí být souvislý a každý uzel musí mít stupeň roven alespoň 2 [5].



Obr. 7: Příklad nalezení nejkratší hamiltonovské cesty

Eulerovský tah je tah v neorientovaném souvislém grafu, který prochází všemi hranami právě jednou. Jestliže tah začíná i končí ve stejném vrcholu, jedná se o tah uzavřený, v jiném případě je tah otevřený. Podmínkou pro nalezení uzavřeného eulerovského tahu je, aby byl graf souvislý a měl všechny vrcholy sudého stupně [5].



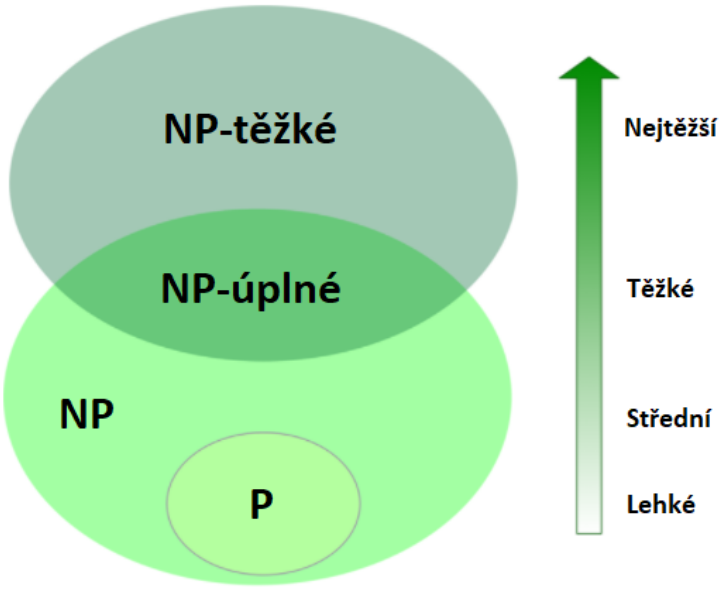
Obr. 8: Graf, který obsahuje uzavřený eulerovský tah

2.2 Klasifikace problémů

V informatice se používají dvě základní klasifikace úloh, P a NP úlohy [7]:

- P (Polynomial) – úlohy, které je možno řešit v polynomiálním čase (lineární, kvadratický nebo logaritmický čas). P úlohy jsou podmnožinou NP úloh. Mezi tyto úlohy lze zařadit například všechny základní matematické operace, algoritmy nejkratší cesty (Dijkstra, Bellman-Ford, Floyd-Warshall) a jiné.
- NP (Non-deterministic Polynomial) – úlohy, které není možno řešit v polynomiálním čase, ale dají se v tomto čase ověřit. Mezi základní úlohy lze považovat faktorizaci celých čísel nebo izomorfismus grafu.

NP úlohy lze dále rozdělit na NP-těžké a NP-úplné úlohy, které se používají pro řešení složitějších a sofistikovanějších problémů. Všechny problémy lze tedy podle složitosti rozdělit následujícím způsobem (Obr. 9) od nejlehčího po nejtěžší [7].



Obr. 9: Klasifikace úloh. Upraveno ze zdroje: [7]

Problém obchodního cestujícího (TSP – Travelling Salesman Problem) a problém plánování rozvozu (VRP – Vehicle Routing Problem) spadají do kategorie NP-těžkých a NP-úplných úloh.

Mezi **NP-těžké úlohy** patří ty nejsložitější problémy. Nejenže je těžké je vyřešit, ale také ověřit. Nemusí se jednat o problém s rozhodováním. Jako další ukázkové problémy lze uvést shlukování metodou nejbližších středů (K-means clustering) nebo obarvení grafu. Za **NP-úplné úlohy** se považují takové, které spadají zároveň do NP a NP-těžkých úloh a mohou být řešeny nedeterministickým algoritmem v polynomiálním čase. Další příklad této třídy je Knapsack problem [7].

3 PROBLÉM OBCHODNÍHO CESTUJÍCÍHO

Problém obchodního cestujícího (TSP) je kombinatorický problém v oblasti teoretické informatiky, který se zabývá hledáním nejkratší a nejúčinnější cesty, kterou lze použít pro dosažení definovaných oblastí. Jedná se tedy o optimalizační problém v teorii grafů, kde je úkolem projet všechna města (uzly) právě jednou a vrátit se zpět do výchozího uzlu za použití minimálních nákladů na cestu (cestovní vzdálenost) [8]. Úkolem je nalezení hamiltonovské cesty v ohodnoceném grafu.

Jestliže je zadaný počet měst n, které je potřeba navštívit, tak počet možných cest S je vyjádřen následující rovnicí (1) [9].

$$S = \frac{(n-1)!}{2} \tag{1}$$

Tab. 1: Celkový počet možných cest pro n měst

počet měst n	počet možných cest S
3	1
4	3
5	12
6	60
7	360
8	2 520
9	20 160
10	181 440
15	43 589 145 600

TSP patří mezi NP-úplné úlohy, jelikož je možno jeho řešení testovat v polynomiálním čase, ale nalézt nikoliv. Problém optimalizace TSP je však NP-těžká úloha, ale ne NP, z čehož vyplývá, že řešení a ověření nelze provést v polynomiálním čase [10]. Složitost řešení výpočtu nejkratší cesty se s každým přidaným uzlem zvyšuje (viz Tab. 1).

Matematický model TSP lze popsat následovně [30]:

Základem matematického modelu je účelová funkce (2), která minimalizuje celkové náklady na výsledné cestě. Do modelu jsou zapracovány dvě podmínky (3) a (4), které zaručují, že daný vrchol bude navštíven právě jednou. Poslední rovnice (5) zamezuje vzniku subcyklů.

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{2}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1, \dots, n \tag{3}$$

$$\sum_{\mathrm{j}=1}^{n} x_{ij} = 1, \qquad \mathrm{i} = 1, \dots, n \tag{4}$$

$$u_i - u_j + n x_{ij} \leq n - 1, \qquad i = 1,2, \dots, n, \qquad j = 2,3, \dots, n, \qquad i \neq j \tag{5}$$

Vysvětlení symbolů:

- n – počet měst
- c_{ij} – náklady na cestu z i do j
- x_{ij} – následuje-li j po i, tak rovno 1, jinak 0
- u_i – pořadové číslo místa i v rámci okruhu

3.1 Historie

Původ TSP není známý, ačkoliv se první zmínky objevily v roce 1800, kdy William Rowan Hamilton a Thomas Penyngton Kirkman řešili matematické problémy, které úzce souvisely s problémem obchodního cestujícího. Příkladem je velice známý problém The Icosian Game z roku 1859 [11].

O obecné formě problému bylo možno poprvé slyšet ve 20. letech 20. století díky propagaci Karla Mengera mezi svými kolegy ve Vídni. V psané podobě se problém vyskytuje poprvé v roce 1932, kdy Menger publikoval „Das botenproblem“ v Ergebnisse eines Mathematischen Kolloquiums. Označil problém jako „problém poslů“ a prohlásil, že hledání trasy, kde se z výchozího bodu jde do bodu, který je mu nejblíže a takto pokračovat, obecně nevede k nejkratšímu řešení [12].

Při průzkumu zemědělské půdy v Bengálsku v roce 1938, kde hlavním problémem byla přeprava pracovníků a vybavení z jednoho místa do druhého. Řešením TSP prostřednictvím náhodně vybraných míst v euklidovské rovině se zabývali ve 40. letech statistici Mahalanobis, Jessen, Gosh a Marks. V této době proslavil TSP také Merill Meeks Flood mezi svými kolegy v americké společnosti RAND Corporation jako

prototyp těžkého problému, jelikož bylo obtížné prozkoumávat všechny cesty kvůli velkému počtu bodů [12].

V následující tabulce (Tab. 2) lze nalézt postupně rostoucí velikost instancí v průběhu let.

Tab. 2: Vývoj velikosti řešení TSP v průběhu let. Zdroj: [11,13,14]

Rok vyřešení	Počet měst
1954	49
1971	57
1971	64
1975	67
1977	120
1980	318
1987	532
1987	666
1987	1 002
1987	2 392
1992	3 038
1993	4 461
1994	7 397
1998	13 509
2001	15 112
2004	24 978
2004	33 810
2006	85 900
2020	1 437 195

V 50. letech Merill Meeks Flood popsal některé heuristické metody k získání vhodných cest, nejznámějším popsaným algoritmem byl algoritmus nejbližšího souseda. V roce 1957 L. L. Barachet vydal dokument, ve kterém se zabýval grafickým řešením TSP, popsal a vyřešil ručně problém s instancí 10 měst. Konec 50. let znamenal pro TSP velký krok kupředu, jelikož se k jeho řešení začal využívat počítač, jedním takovým příkladem je aplikace na problému s 10 městy v roce 1958 na počítači IBM 650 [12].

V roce 1963 byl aplikován na TSP nový algoritmus, metoda větví a mezí (B&B – Branch and Bound), který byl implementován na počítači IBM 7090. O jeho publikování se postarali J. D. C. Malý, K. G. Murty, D. W. Sweeney a C. Karel. Roku 1965 byl publikován článek, který řešil problém až pro 105 měst, jeho autorem byl Shen Lin.

O aplikaci celočíselného programování a využití algoritmu Branch-and-Cut se v roce 1976 postaral P. Miliotis. O rok později byl použit pro instanci 120 měst další algoritmus, a to algoritmus řezné roviny. Autorem byl M. Grtschel a M. Padberg [12].

3.2 Využití v praxi

Aplikace a využití TSP sahá až daleko za hranici pouhého plánování trasy pro projití určitého počtu měst. Své využití nalézá při optimalizaci v oblasti matematiky, informatiky, elektrotechniky, strojírenství či genetiky.

3.2.1 Logistika

Nejčastější aplikací je samozřejmě pohyb osob, vozidel či vybavení. Běžné plánování výletů či pracovních cest je prováděno pomocí softwarů, které dokáží vyřešit TSP jen pro malé instance a jsou dostačující pro tuto aplikaci. Jako další příklad lze uvést výše zmíněná přeprava osob a vybavení v Bengálsku. Co se týče více sofistikovanějších problémů, tak sem lze zařadit například plánování rozvozů školních autobusů. Tímto problémem se zabývá mnoho velkých společností a nabízí školním institucím jejich služby při optimalizaci vyzvednutí dětí do škol [14].

Velké využití nachází TSP u poštovních služeb. Mohou nastat situace takové, že není nutno projíždět dům od domu nebo například domy se nacházejí daleko od sebe. Jsou vytvořeny poštovní aplikace pro doručování balíků, kde aplikace dodržují různá dopravní omezení. Nejde jen o doručování dopisů či balíků, ale také o pomoc potřebným při rozvážení jídla či léků [14].

Obr. 10: Použití TSP pro rozvoz školního autobusu. Zdroj: [14]

3.2.2 Strojírenství a elektrotechnika

Vrtání desek plošných spojů patří mezi častý problém obchodního cestujícího. Nutnost vyvrtat desítky otvorů různého průměru přináší optimalizační problém. Cílem je minimalizovat dobu přejezdu hlavy stroje tak, aby vyvrtal všechny díry jednoho průměru,

následně proběhla výměna hlavy a postup se opakoval [9]. Tato aplikace může být použita u jakéhokoliv problému s nutností vyvrtání velkého množství děr.

K využití TSP dochází i v integrovaných obvodech, kde je nutno optimalizovat skenovací řetězce, což jsou trasy zahrnuté na čipech. Dochází k minimalizaci jejich délky skrze časovou a energetickou úlevu [11].

3.2.3 Genetika

V genetickém inženýrství je TSP využito pro výpočet sekvencí DNA. Zde se TSP aplikovalo na kolekci řetězců DNA, které jsou vloženy následně do jednoho univerzálního, je potřeba minimalizovat délku tohoto řetězce [11].

Dalším úkazem důležitosti řešiče TSP je ke konstrukci radiačních hybridních map, které jsou využívány pro sekvenování genomů. Problém tkví v sestavení optimální hybridní mapy z více místních map [11].

3.2.4 Další možnosti využití

Při použití farmaceutických robotů pro automatický výdej potřebných léků klientem je nutno využít TSP. Úkolem je vyřešit problém minimalizace uražené vzdálenosti manipulátoru pro dosažení určitých léků [15].

Jedním z dalších příkladů lze uvést použití k analýze struktury krystalů. K získání informací o struktuře je použit rentgenový difraktometr, který měří intenzitu rentgenových odrazů z různých pozic. Zisk těchto informací je realizován ze stovek tisíc pozic, a tudíž zde nastává problém optimalizace celkového času pohybu při polohování [9].

Mezi další možné příklady využití TSP patří [9,14]:

- modelový problém při minimalizaci plýtvání tapetami
- vyskladňování předmětů ve velkých skladech
- ve sklářství při plánování výroby pro řezání
- shromažďování geofyzikálních seismických dat
- komprese velkých datových sad polí
- spojování součástek na desce počítače
- plánování vojenských misí kvůli optimalizaci uražené vzdálenosti a času
- návrh mapovacích sítí globálního navigačního satelitního systému

3.3 Varianty problému

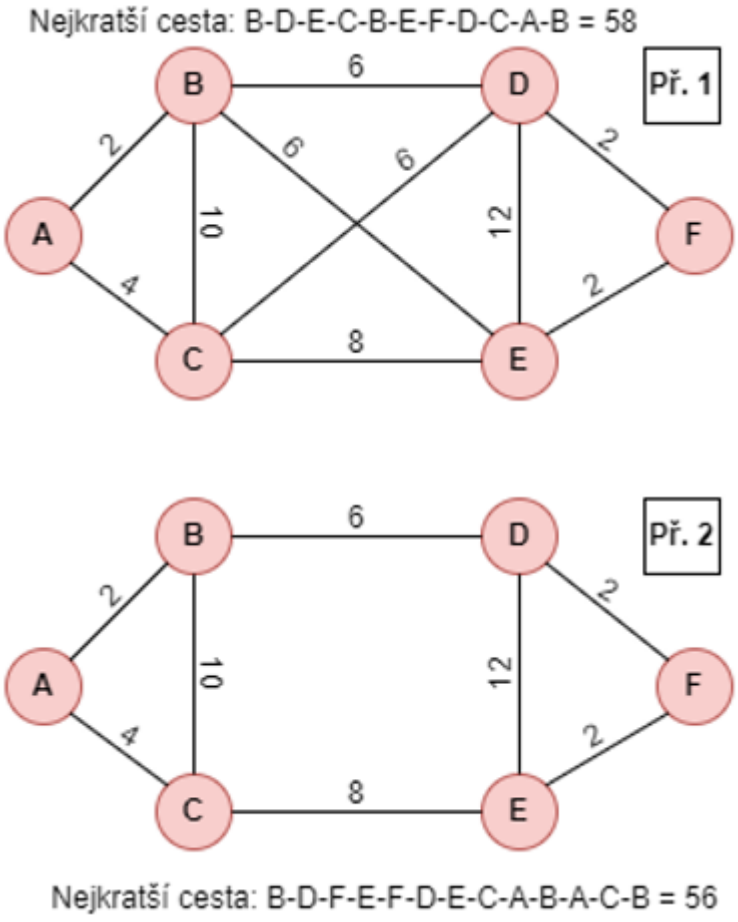
U problému obchodního cestujícího dochází k mnoha modifikacím a úpravám zadání úlohy a problematiky. Mezi nejznámější patří problém čínského listonoše a problém kanadského cestujícího.

3.3.1 Problém čínského listonoše

Poprvé se o tomto problému zmínil čínský matematik Kwan Mei-Ko počátkem 60. let. Jde o problém velmi podobný TSP, avšak s jedním velkým rozdílem, jde o nalezení nejkratší cesty v grafu, která prochází nikoliv všemi uzly, ale všemi hranami ohodnoceného grafu. Inspiraci získal u čínského pošťáka, který každý den doručoval dopisy napříč městem a musel projít všechny ulice tohoto města. Jestliže je daná úloha reprezentována souvislým grafem a má všechny stupně vrcholů sudé, tak je nejkratší cestou nalezený uzavřený eulerovský tah. V jiném případě je potřeba k nalezení nejkratší možné cesty navštívit některé hrany vícekrát. Pokud jsou všechny hrany grafu neorientované, lze problém řešit v polynomiálním čase [16,17].

Problém nalézá využití u plánování tras při doručovaní pošty, svozu odpadu, úklidu silnic od nečistot či odhrnování sněhu [17].

Řešení jednoduchých úloh lze vidět na obrázku (Obr. 11). V příkladu č. 1 se nachází o dvě hrany více, ale v délkách nejkratších cest není velký rozdíl. Pokud graf splňuje podmínky existence eulerovského tahu, je optimální řešení jím určeno, protože každou hranu obchodník projde pouze jednou.



Obr. 11: Aplikace problému čínského listonoše na triviálních příkladech

3.3.2 Problém kanadského cestujícího

Tento optimalizační problém byl představen informatiky Christosem Papadimitrouem a Mihalisem Yannakakisem roku 1991. Název je odvozen od kanadských řidičů, kterým cestování komplikoval sníh na silnicích. Problém spočívá v nalezení nejkratší cesty z vrcholu A do vrcholu B za nepředvídatelného omezení. Některé hrany mohou být zablokovány, ale pro zjištění, zda je daná hrana volná, musí cestující dorazit do vrcholu, ze kterého hrana vychází. Úkolem je vytvoření ideální strategie, která zajistí minimální cestovní náklady na základě dílčích informací [18,19].

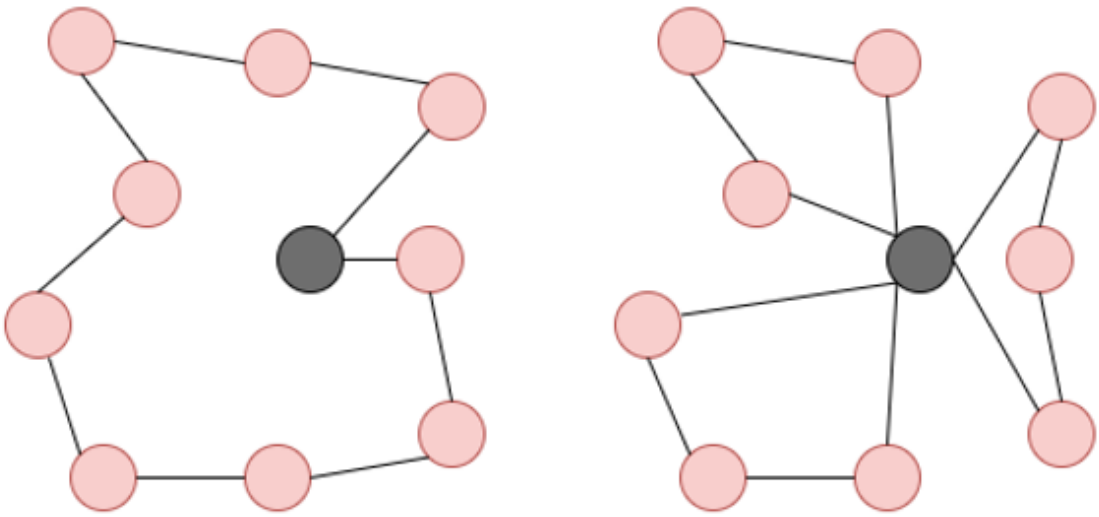
4 PROBLÉM PLÁNOVÁNÍ ROZVOZU

Problém plánování rozvozu (VRP) je kombinatorický problém v oblasti optimalizace. První zmínky o VRP vznikly roku 1959, kdy byl problém popsán jako zobecněná forma problému obchodního cestujícího. Autoři této studie, která byla zaměřena na dodávku benzínu, byli George Dantzig a John Ramser. Často je tento problém formulován jako rozvoz zboží zákazníkům do širokého okolí z centrálního depa. [20]. VRP spadá do NP-těžkých úloh.

Silniční síť, která je využívána při řešení VRP, bývá popsána grafem, kde hrany reprezentují silnice a vrcholy zákazníky, depa nebo pouhé křižovatky. Hrany jsou buď orientované nebo neorientované v závislosti na silničních omezeních. Každá hrana je taktéž ohodnocena svou cenou z hlediska vzdálenosti mezi vrcholy nebo čas potřebný k překonání dané hrany.

Vrcholy grafu nabývají různých vlastností, od informací o množství zboží, které je nutno dodat či vyzvednout, až po časové rozmezí, kdy je možno daný vrchol navštívit. Cesty, které jsou vytvořeny za účelem obsluhy zákazníka, začínají a končí v jednom či více centrálních dep, které jsou předem určeny [21].

Na obrázku níže (Obr. 12) lze vidět nalezení cesty u TSP (graf vlevo) a tras u VRP (graf vpravo), kde se mohou vyskytnout různá omezení, například kapacita vozidla při rozvozu osob. Černý vrchol se bere jako počáteční uzel, může být označován i jako centrální depo.



Obr. 12: Porovnání hledání cesty mezi TSP a VRP

V praxi je tento problém rozšířen o mnohé podmínky a omezení. Tato omezení se mohou týkat například kapacity vozidla, maximální délky jedné trasy, doby trasy, doby dodání a mnoho dalších. Mezi TSP a VRP je zásadní rozdíl v možnosti využití vozidel,

TSP vyhledá takovou trasu, kterou cestující projede jednou a zamíří do všech měst. VRP nachází mnoho různých tras pro více vozidel.

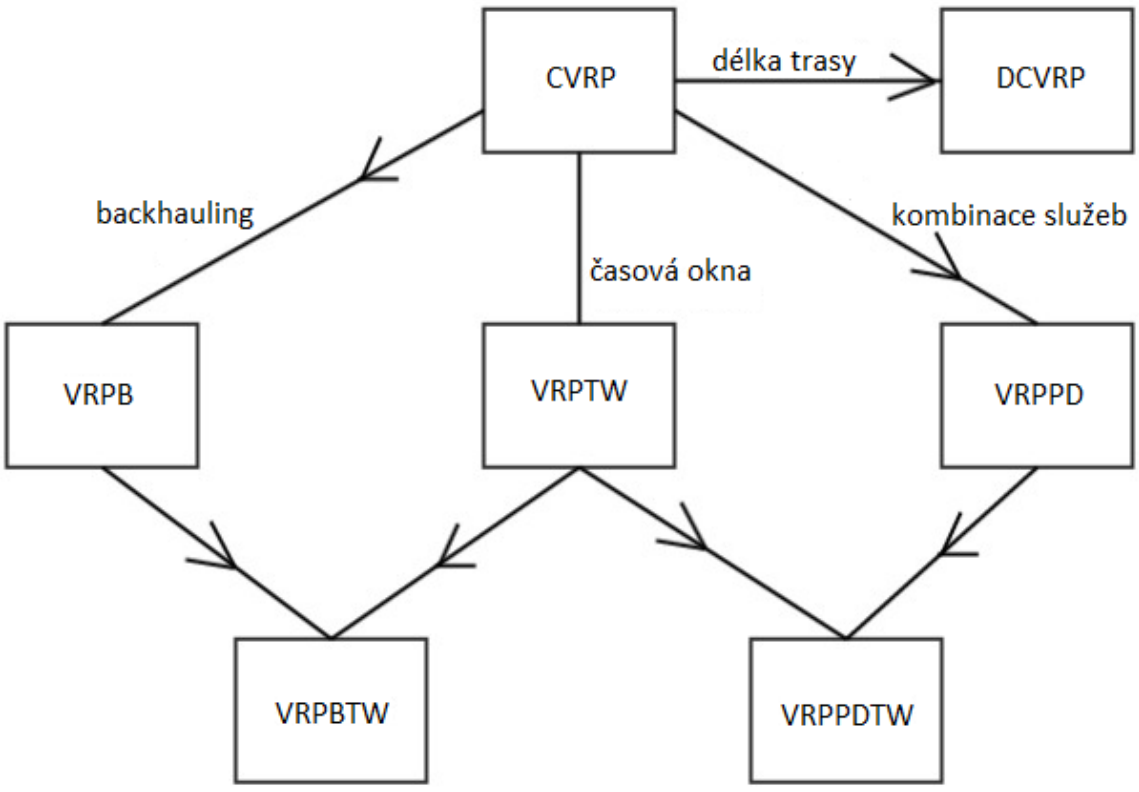
U problému plánování rozvozu lze brát v úvahu větší počet cílů k jeho ukončení, nejen pár nejdůležitějších níže uvedených [21]:

- minimalizace nákladů na dopravu
- minimalizace počtu vozidel potřebných k obsluze všech zákazníků
- vyvážení všech tras z důvodu zatížení vozidel a dobu jízdy

4.1 Typy dopravních okružních úloh

Na základě různých omezení a úprav základního problému vzniklo mnoho rozšíření. K detailnějšímu rozboru byly vybrány tyto typy VRP:

- s omezením kapacity
- s omezením vzdálenosti
- s časovými okny
- s vyzvednutím a doručením
- se zpětným chodem
- dynamický



Obr. 13: Závislosti mezi jednotlivými typy VRP. Upraveno ze zdroje: [21]

4.1.1 VRP s omezením kapacity nebo vzdálenosti

Problém plánování rozvozu s omezením kapacity (CVRP – Capacitated Vehicle Routing Problem) patří mezi nejjednodušší a nejstudovanější typy VRP. U tohoto problému jsou požadavky deterministické, tedy předem známé. Všechna vozidla jsou stejná a existuje pouze jedno centrální depo, ze kterého všechna vozidla vyrážejí a zároveň zde i jejich trasa končí. Jediným omezením je kapacita vozidel, která se označuje písmenem C. Hlavním cílem je minimalizace celkových nákladů na cestu, které jsou úměrné celkové ujeté vzdálenosti vozidel k obsloužení všech zákazníků [21].

Matematický model VRP s omezením kapacity lze popsat následovně [30]:

Základem celého modelu je účelová funkce (6), která minimalizuje náklady na rozvoz. Rovnice (7) a (8) v tomto pořadí znázorňují, že k zákazníkovi *j* přijede právě jedno vozidlo a od zákazníka *i* právě jedno vozidlo odjede. Další rovnice zaznamenávají, že právě K vozidel se od zákazníků vrátí zpět do depa (9) a právě K vozidel vyrazí z depa k zákazníkům (10). Rovnice (11) zaznamenává balanci převozu nákladu a zároveň eliminuje subcykly. Aby byla dodržena stanovená kapacita vozidel zajišťuje rovnice (12).

$$\min_{x_{ij},u_i} \sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij} x_{ij} \tag{6}$$

$$\sum_{i=0}^{n} x_{ij} = 1, \qquad j = 1, \dots, n \tag{7}$$

$$\sum_{j=0}^{n} x_{ij} = 1, \qquad i = 1, \dots, n \tag{8}$$

$$\sum_{i=1}^{n} x_{i0} = \mathrm{K} \tag{9}$$

$$\sum_{\mathrm{j}=1}^{n} x_{0j} = \mathrm{K} \tag{10}$$

$$u_i - u_j + d_j \leq \mathrm{C}(1 - x_{ij}), \qquad i, j = 1, \dots, n \tag{11}$$

$$d_i \leq u_i \leq C, \qquad i = 1, \dots, n \tag{12}$$

Vysvětlení symbolů:

- n – počet zákazníků
- 0 – centrální depo

- K – počet dostupných vozidel
- d_j – poptávka zákazníka ($d_j \geq 0$)
- C – kapacita vozidla ($C > 0$)
- c_{ij} – náklady na cestu z i do j
- x_{ij} – následuje-li j po i, tak rovno 1, jinak 0
- u_j – horní odhad doposud převezeného nákladu po navštívení zákazníka j

K řešení toho typu problému se využívají především heuristické algoritmy, nejčastěji používaný je algoritmus mravenčí kolonie (ACO – Ant Colony Optimization), který je podrobněji popsán v kapitole 5 [22].

Dalším typem je **problém plánování rozvozu s omezením vzdálenosti** (DVRP – Distance-Constrained Vehicle Routing Problem). Zde je omezení kapacity vozidel nahrazeno maximální délkou nebo časem, po kterém se dané vozidlo musí dostat zpět do centrálního depa. Cílem je tedy minimalizace celkové délky tras nebo jejich doba trvání [21].

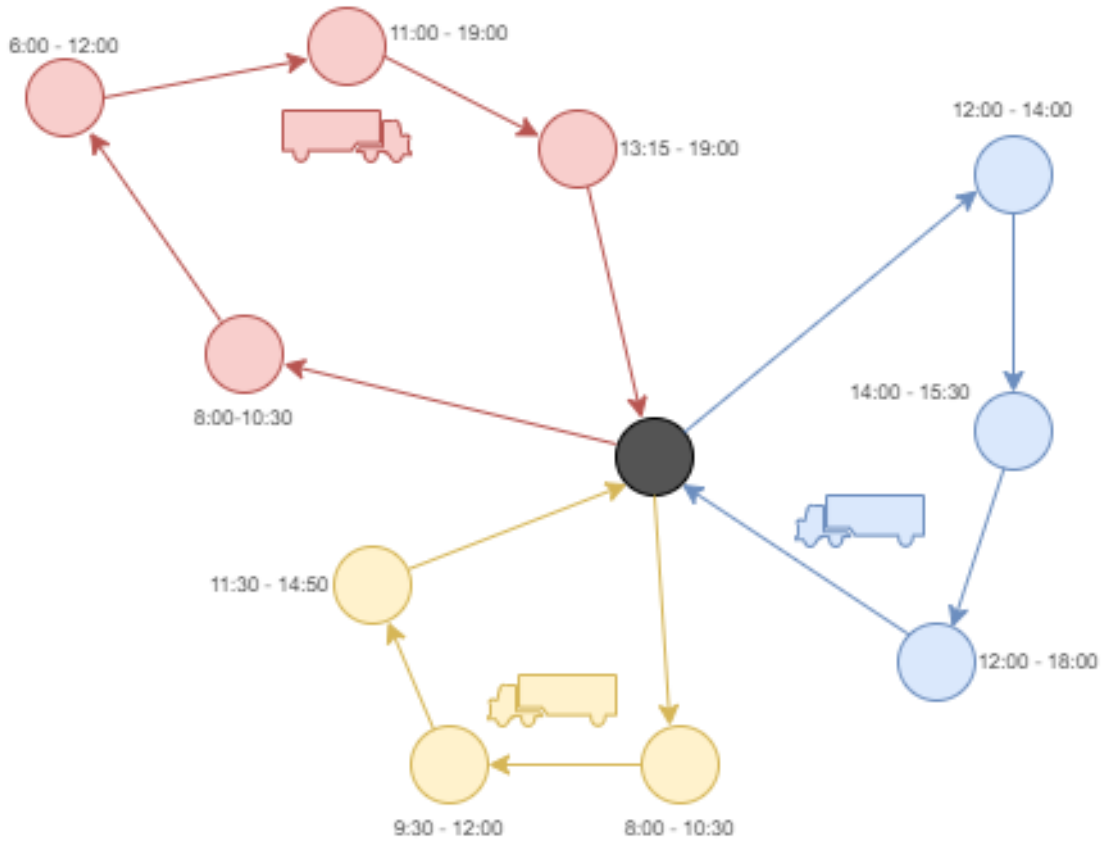
Může dojít i ke kombinaci dvou předešlých rozšíření základního problému (DCVRP – Distance-Constrained Capacitated Vehicle Routing Problem), kdy nastane omezení kapacitní i vzdálenostní či časové.

4.1.2 VRP s časovými okny

Velice podobný CVRP je **problém plánování rozvozu s časovými okny** (VRPTW – Vehicle Routing Problem with Time Windows), který taktéž pracuje s kapacitami vozidel. Navíc je tento problém spojen s časovými okny, což jsou časové intervaly, ve kterých je zákazník k dispozici pro přijetí dodávky. Intervaly mohou být nastaveny na libovolnou časovou šířku, od minut až po dny. Zákazník může být obsluhován pouze v tomto časovém intervalu. Je nutno zohlednit taktéž dobu, během které je zákazník obsluhován [20,21].

K řešení VRPTW jsou využívány metaheuristiky, mezi které patří: genetický algoritmus (GA – Genetic Algorithm), evoluční algoritmy (EA – Evolutionary Algorithms) a již dříve zmíněný ACO [23].

Cílem je obsloužení všech určených zákazníků v předem definovaných časových oknech s minimálními náklady na cestu. Velké využití tento problém nalézá při poštovním doručování.



Obr. 14: Příklad nalezení tras pro VRPTW

4.1.3 VRP s vyzvednutím a doručením a s blackhaulem

V praxi je potřeba řešit i takové problémy, ve kterých nejde pouze o doručení zboží zákazníkům, ale současně i o jeho vyzvednutí a dovoz zpět do centrálního depa. O takovém problému lze mluvit jako o problému **plánování rozvozu s vyzvednutím a doručením** (VRPPD – Vehicle Routing Problem with Pickup and Delivery). Vrcholy mají dvojí označení, vrcholy pro doručení a vrcholy pro vyzvednutí zboží. Mohou nastat situace takové, že jeden vrchol má dvojí označení. Předpokládá se, že nejprve dané vozidlo v daném vrcholu zboží doručí a následně jej vyzvedne. Je potřebné zohlednit jak kapacitu vozidel, tak v případě nutnosti i časová okna (VRPPDTW – Vehicle Routing Problem with Pickup and Delivery with Time Windows), ve kterých je možno zákazníky obsluhovat [21].

Velice podobný je **problém plánování rozvozu s blackhaulem** (VRPB – Vehicle Routing Problem with Backhauls), kde je velký rozdíl v označení vrcholů. U jednoho zákazníka může vozidlo zboží pouze doručit nebo vyzvednout. Kdykoliv trasa zahrnuje obě možnosti, musí nejprve obsloužit všechny zákazníky, u kterých má být zboží doručeno a následně může být provedeno vyzvednutí zboží a převoz do centrálního depa [21]. Stejně jako u VRPPD je nutno splnit kapacitní omezení vozidel a taktéž je v určitých případech nutno pracovat s časovými okny (VRPBTW – Vehicle Routing Problem with Backhauls with Time Windows).

Cílem obou verzí problému je vytvoření tras s minimálními náklady na cestu.

4.1.4 Dynamický VRP

Ve všech předešlých případech rozšíření VRP se předpokládá, že informace o poptávce zákazníků a době jízdy vozidla jsou známé a vytvořená trasa se již nebude měnit. U **dynamického problému plánování rozvozu** mohou nové požadavky zákazníků přicházet během provádění distribuce a s rozvojem inteligentního dopravního systému se dopravní situace na určitých úsecích tras mění.

Systém nejprve podle dostupných informací naplánuje trasy vozidel tak, aby náklady na cestu byly minimální. V průběhu distribuce mohou být obdrženy nové informace a systém musí zajistit rychlou odezvu. Následuje přeplánování tras vozidel do momentu, kdy se neobjeví novější informace.

Hlavními důvody přeplánování tras bývají [22]:

- dopravní zácpy nebo nehody
- obsluha nových zákazníků
- zrušení objednávky stávajících zákazníků
- změna místa dodání nebo vyzvednutí zboží
- změna časových oken zákazníků

4.1.5 Další možné typy VRP

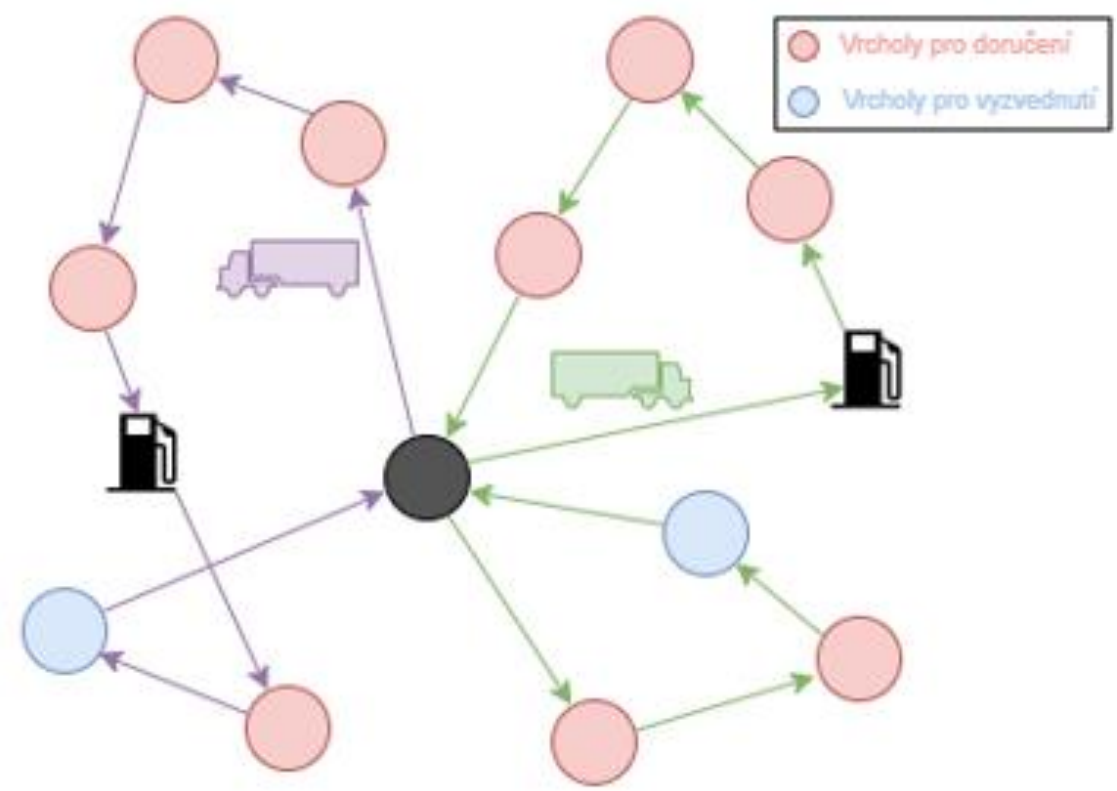
Existuje mnoho dalších rozšíření základního problému [20, 24]:

- **Problém plánování rozvozu se zisky** – cílem je maximalizace zisků z obsloužených zákazníků, není nutno navštívit všechny zákazníky, vozidla začínají i končí v centrálním depu.

- **Problém plánování rozvozu (LIFO – Last In First Out) –** zboží, které je jako poslední naloženo je jako první doručeno kvůli zkrácení doby setrvání v místech dodání.

- **Green VRP** – cílem problému je minimalizace spotřeby paliva.

- **Problém plánování rozvozu elektrickými vozidly** – problém se zabývá plánováním tras pro elektromobily a zahrnuje do nich i v případě nutnosti polohy a využití dobíjecích stanic.



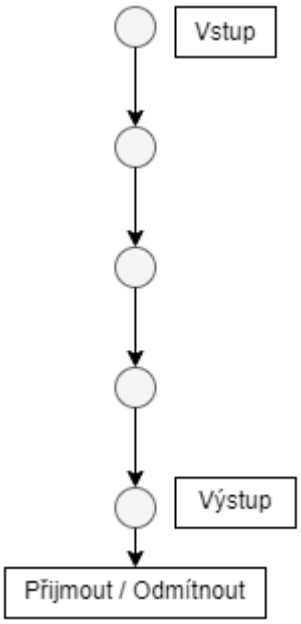
Obr. 15: Příklad kombinace VRP s blackhaulem a VRP s elektrickými vozidly

5 METODY ŘEŠENÍ PROBLÉMŮ

Okružní problémy se dají implementovat pomocí mnoha různých metod, pro menší instance jsou vhodnější deterministické algoritmy, a naopak pro velké instance měst se používají algoritmy stochastické. Níže jsou popsány různé metody přístupu řešení těchto problémů.

5.1 Deterministické algoritmy

Deterministické algoritmy patří mezi nejstudovanější a nejpraktičtější algoritmy vůbec, jejich chování je předvídatelné a vždy probíhá stejná sekvence kroků k jejich kompletaci. Pro daný konkrétní vstup bude mít daná úloha vždy stejný výstup (Obr. 16). Díky tomu se tyto algoritmy dají považovat za spolehlivé a dokáží nalézt optimální řešení [25,26].



Obr. 16: Postup řešení deterministických algoritmů. Upraveno ze zdroje: [29]

Úlohy se dají řešit v polynomiálním čase, vždy se dá určit další krok provedení a tento počet kroků je pevně daný. Na konci každého řešení nastane fáze přijetí či odmítnutí stejného výsledku [27].

Deterministických algoritmů pro řešení Okružních problémů je mnoho, k detailnějšímu popisu byly vybrány tyto metody: metoda hrubé síly, metoda větví a mezí a metoda nejbližšího souseda.

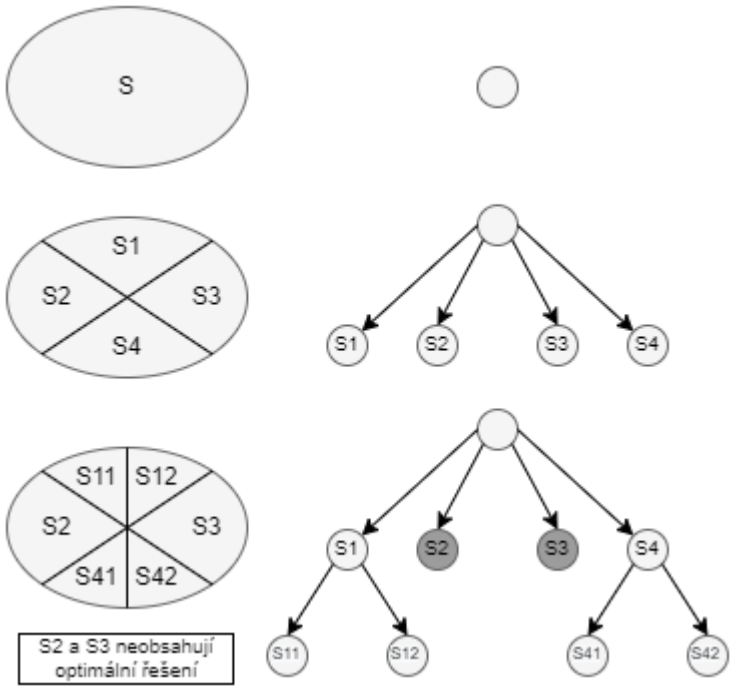
5.1.1 Metoda hrubé síly

Metoda hrubé síly neboli metoda pokusu a omylu, je nejzákladnější a nejjednodušší typ algoritmu. Metoda je snadno implementovatelná a vždy najde optimální řešení, pokud existuje. Zároveň poskytuje všechna možná řešení daného problému, avšak při rostoucích velikostech problému roste i náročnost výpočtu tohoto algoritmu. Je velice výhodné ji použít, pokud je problém malý a jednoduchý [31]. Jako ukázkový příklad lze uvést prolomení 4-místného číselného kódu, kde by tato metoda vyprodukovala 10^4 možností pro nalezení správného výsledku [32]. Z pohledu aplikace na okružních problémech je tato metoda tou nejvhodnější, jelikož umožní vždy cestovat mezi městy či zákazníky s minimálními náklady. Nevýhodou metody je výpočetní náročnost a její využití pro tento typ problémů se příliš nepraktikuje.

Časté využití metody hrubé síly nastává v situacích, kdy není k dispozici žádný jiný algoritmus, který by daný proces zrychlil a je třeba zkontrolovat všechny možné varianty. Algoritmus nepřináší žádné kreativní nebo konstruktivní výsledky. Velice často se používá pro porovnání s jinými algoritmy [31].

5.1.2 Metoda větví a mezí

Metoda větví a mezí (B&B) je paradigma návrhu algoritmu, který se používá pro řešení diskrétních optimalizačních problémů. K zisku optimálního výsledku algoritmus prohledává celý prostor řešení. První zmínky o metodě B&B přišly od A. Landa a A. Doiga, kdy se zabývali řešením problémů celočíselného programování [33].



Obr. 17: Ukázka prohledávaného prostoru při použití B&B. Upraveno ze zdroje: [34]

Základní kámen celého algoritmu tvoří dynamicky se generující prohledávací strom. Na počátku celého procesu vždy existuje množina všech řešení S, která je označována za kořen stromu (Obr. 17), zahrnuje všechna možná řešení a nejlepší řešení problému je nastaveno na hodnotu ∞. Nastává situace, kdy se množina S rozdělí na vícero podmnožin. V kterémkoliv okamžiku procesu je stav řešení popsán dosud neprozkoumanými podmnožinami. Neprozkoumané podmnožiny jsou reprezentovány jako vrcholy stromu. Každá iterace algoritmu zpracovává jeden takový vrchol a skládá se ze tří důležitých částí. Jedná se o výběr vrcholu ke zpracování, výpočet mezí a v neposlední řadě větvení daného stromu. U každé podmnožiny se ověřuje, zda se podmnožina skládá pouze z jednoho možného řešení. Jestliže ano, tak se řešení porovnává s doposud optimálním řešením. V opačném případě dochází k výpočtu tzv. mezní funkce, která ohraničuje danou podmnožinu a daný proces se znovu opakuje. Pokud je horní mez řešení z S1 nižší než dolní mez řešení v S2, pak zjevně nemá cenu zkoumat řešení v S2. Pokud tato situace nastane, tak je možno celou tuto podmnožinu vypustit a dále s ní nepracovat (podmnožiny S2 a S3). Proces vyhledávání je ukončen v případě, kdy ve vyhledávacím prostoru nezůstávají žádné neprozkoumané podmnožiny a optimálním řešením se stává doposud nejlepší nalezené [34].

Velkou výhodou metody B&B je zcela jistě to, že není potřeba prohledávat všechny vrcholy stromu, a to je důvodem, že časová náročnost je v porovnání s předchozí metodou mnohem menší [35]. Díky faktu, že algoritmus postupně poskytuje lepší a lepší řešení a meze se postupně zužují, může i při předčasném přerušení procesu dojít ke smysluplnému výsledku.

5.1.3 Metoda nejbližšího souseda

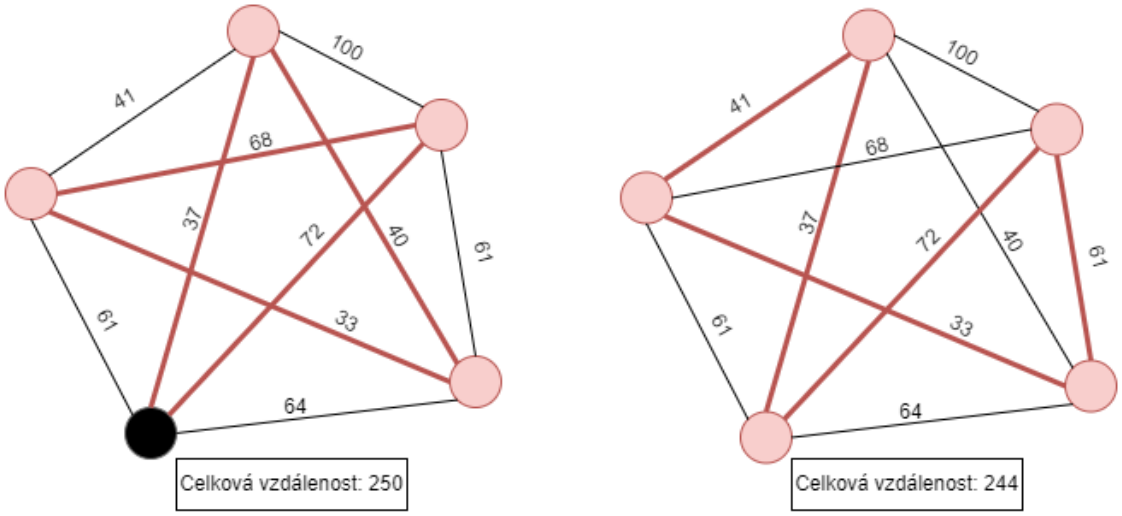
Metoda nejbližšího souseda (NN – Nearest Neighbour) patří mezi nejjednodušší algoritmy, které se využívají pro řešení Okružních problémů. Poprvé byl tento algoritmus představen J. G. Skellamem a jeho základní princip spočívá v přesunu do doposud nenavštíveného města, které je nejblíže naposledy navštívenému městu. Tento postup se opakuje, dokud nedojde k navštívení všech měst. Poté následuje návrat do města počátečního [36].

Kroky NN algoritmu jsou následující:

1. Výběr náhodného vrcholu, který se označí jako počáteční.
2. Výběr hrany s nejmenším ohodnocením, která spojuje aktuální vrchol a vrchol nenavštívený. Přesun po zvolené hraně do nenavštíveného vrcholu.
3. Jestliže stále existují nějaké nenavštívené vrcholy, opakuje se krok č. 2.
4. Pokud jsou navštíveny všechny vrcholy, následuje přesun zpět do počátečního vrcholu.

Výstupem algoritmu je sekvence všech navštívených vrcholů. Tato výsledná cesta nemusí být nutně optimální. Například na obrázku (Obr. 18) si lze povšimnout, že nalezená cesta, která měla počátek v černém vrcholu není optimální a lze nalézt cestu kratší.

Pro zlepšení nalezeného řešení lze NN algoritmus aplikovat více než jednou, přičemž se jako počáteční vrchol vybere jiný, než byl již použit. Algoritmus se opakuje do té doby, než se jako počáteční vrchol využijí všechny vrcholy grafu. Tento způsob lze nazvat jako opakující se metoda nejbližšího souseda [37].

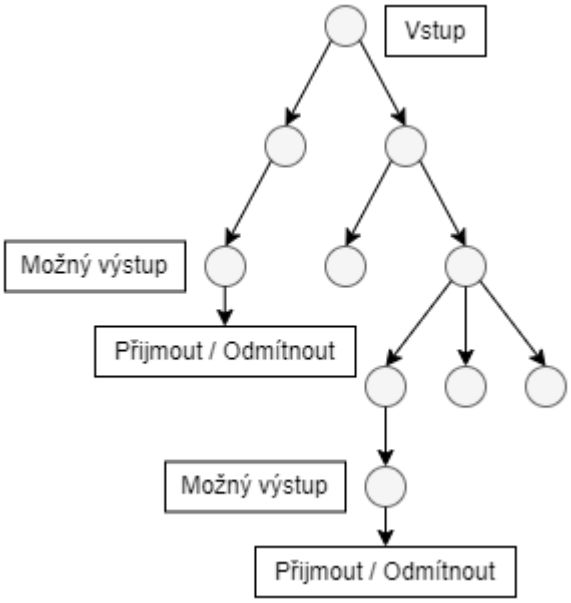


Obr. 18: Nalezení cesty pomocí NN algoritmu (vlevo) a optimální cesta (vpravo)

5.2 Stochastické algoritmy

Stochastické algoritmy, též známé jako nedeterministické, jsou takové, které pro daný vstup mohou vykazovat různé výsledky a jejich chování je nepředvídatelné. Používají se pro nalezení aproximace řešení u takových problémů, kde by bylo velice náročné a nákladné nalezení přesného řešení. Oproti deterministickým algoritmům se liší v tom, že cesta k výsledku může probíhat pomocí různých cest (Obr. 17) [28].

Tyto algoritmy bývají označovány jako nespolehlivé, jelikož vykazují různé výsledky pro různá provedení. Problémy nelze řešit v polynomiálním čase [27].



Obr. 19: Postup řešení stochastických algoritmů. Upraveno ze zdroje: [29]

Pro řešení Okružních problémů je vhodné využít genetický algoritmus nebo algoritmus mravenčí kolonie. Bližší charakteristiky obou metod jsou popsány v další části kapitoly.

5.2.1 Genetický algoritmus

Genetický algoritmus (GA), který patří mezi evoluční algoritmy, je vyhledávací heuristika, která našla inspiraci v procesu přirozené evoluce. První zmínky pochází od Johna Hollanda během 70. let 20. století. Algoritmus je reprezentován populací chromozomů a následnou simulací jejího vývoje. Řešení z jedné populace se využívají k vytváření nových populací. Motivace celého procesu spočívá v tom, že nová populace bude vykazovat jiné výsledky než ta předešlá. Každý chromozom je ohodnocen fitness funkcí *f(x)* a podle ní mohou být chromozomy vybrány k reprodukci. Do nových populací se mohou přijímat jak potomci s lepším ohodnocením, tak i s horším. Tento přístup přijímání i horších řešení je základem toho, aby algoritmus neuvázl v lokálním extrému [38].

Kvůli správnosti fungování GA, tak aby nové populace nebyly stejné, je proto proti tomu nutné zavést různá opatření. Běžně jsou využíváno třech různých operátorů: selekce, křížení a mutace.

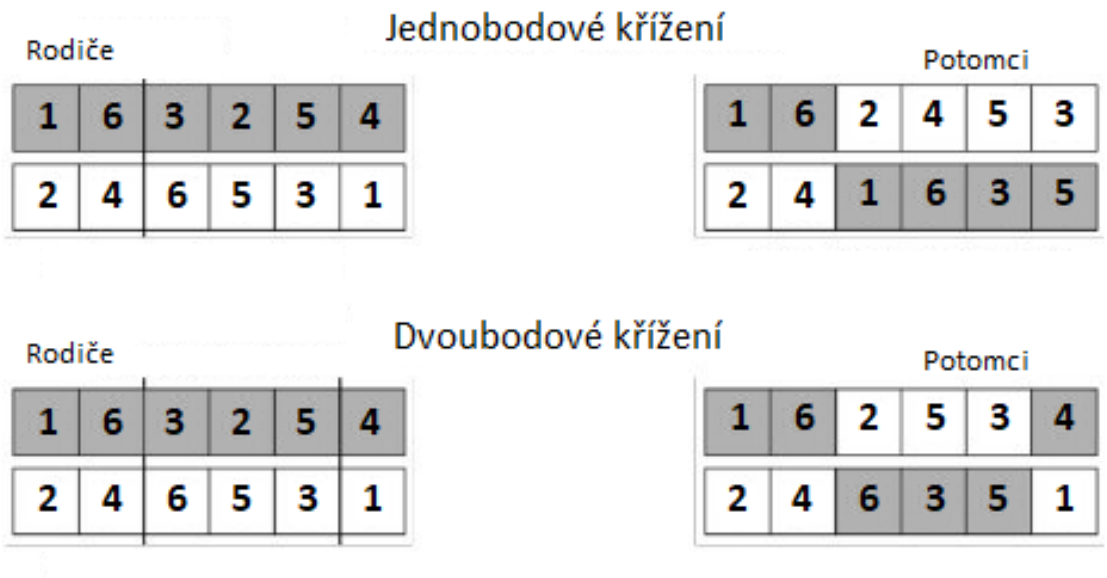
V případě **selekce** jde o výběr vhodných chromozomů pro reprodukci. Čím je chromozom vhodnější, tím větší šance je, že bude pro reprodukci vybrán. Existují různé druhy selekce [38,39]:

- **Turnajová selekce** – dojde k náhodnému výběru několika chromozomů a z něj vybere ten nejlepší.

- **Selekce ruletou** – k výběru jsou chromozomy vybírány s pravděpodobností, která je úměrná kvalitě fitness funkce.
- **Selekce ruletou založená na pořadí –** velice podobný systém jako u předchozího případu, rozdíl je v tom, že nejprve dojde k rozdělení pravděpodobností a to tak, že nejhorší chromozom bude mít novou hodnotu fitness funkci rovnu 1, druhý nejhorší rovnu 2, a to až se dojde k nejlepšímu, který bude mít hodnotu fitness funkce rovnu počtu chromozomů v populaci.

Operátor **křížení** představuje proces, při kterém standardně vzniknou ze dvou rodičů dva noví potomci. Možnosti křížení [45]:

- **Jednobodové křížení** – je vybrán jeden bod křížení, do bodu křížení je řetězec převzat od jednoho rodiče a za ním následuje řetězec, který obsahuje zbylá města v pořadí tak, jak jsou uspořádána v druhém rodiči.
- **Dvoubodové křížení** – jsou vybrány dva body křížení, do prvního bodu a od druhého bodu křížení je řetězec převzat od jednoho rodiče, středová část obsahuje města, která se nacházejí ve středové části prvního rodiče, ale v pořadí takovém, jak se vyskytují v rodiči druhém.



Obr. 20: Operátory křížení

U operátoru **mutace** dochází ke změnám v samotných jedincích. Může jít o bitovou inverzi nebo o změnu pořadí. U **bitové inverze** jsou vybrané bity invertovány. Při **změně pořadí** dojde k vybrání a vzájemné výměně dvou čísel [38].

Jako nevýhodu GA lze považovat nepřímočarost implementace, velké množství vyhodnocování výsledného řešení a problém s nalezením přesného optima [41].

Kroky základního genetického algoritmu mohou probíhat následovně [38]:

1. Vytvoření náhodné populace chromozomů o velikosti N
2. Vyhodnocení fitness funkce každého z chromozomu v populaci
3. Provedení selekce
4. Provedení křížení
5. Provedení mutace
6. Umístění nových potomků vytvořených pomocí operátorů do nové populace
7. Použít novou generaci pro další běh algoritmu
8. Pokud je splněna ukončovací podmínka, tak najít optimální řešení, jinak celý proces opakovat od kroku č. 2

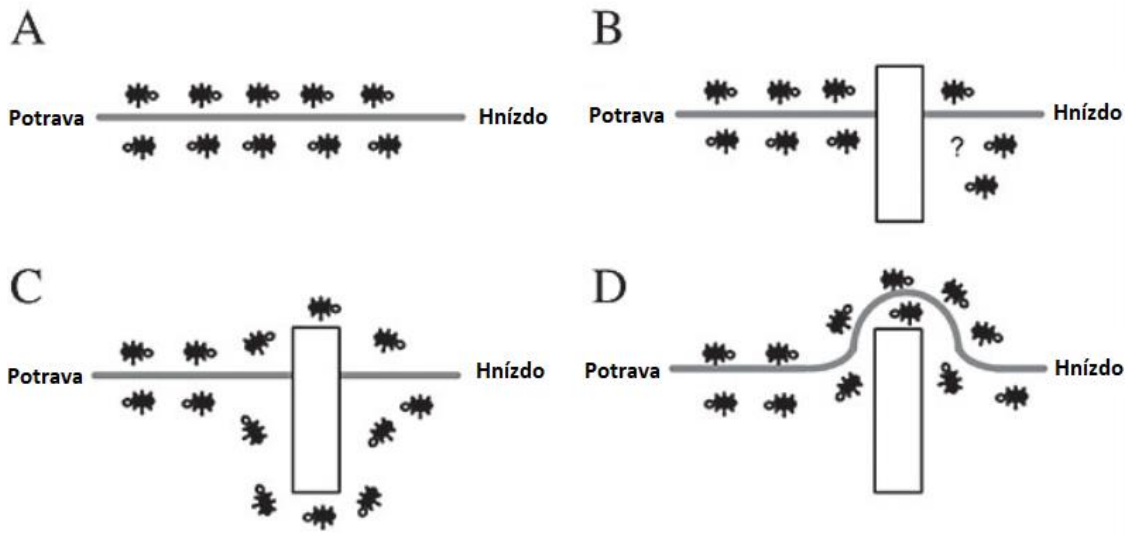
5.2.2 Algoritmus mravenčí kolonie

Algoritmus mravenčí kolonie (ACO) patří mezi další zástupce evolučních algoritmů. Algoritmus byl poprvé představen Marcem Dorigem na konci 20. století. Inspiraci čerpá z chování mravenčích kolonií, které se snaží hledat potravu [42].

Hlavní myšlenkou fungování algoritmu je pozorování pohybu mravenců při hledání potravy z jejich hnízd. Nejprve se mravenci pohybují zcela náhodně, tento náhodný pohyb nabízí mnoho možných cest. Na základě množství potravy, kterou mravenci nosí zpět do hnízda, vypouštějí na zpáteční cestě potřebnou koncentraci feromonů. Na základě této koncentrace ostatní mravenci vybírají novou adekvátní cestu k potravě, která by měla být co nejkratší [42].

Jednoduchý princip ACO lze popsat na následujícím obrázku (Obr. 21):

- **A** – počáteční situace je taková, že mravenci mají jedinou a volnou cestu z hnízda k potravě.
- **B** – na cestě vznikla překážka, kterou je nutno překonat, čímž vznikají dvě možné přístupové cesty.
- **C** – mravenci se s určitou pravděpodobností (většinou 0,5) vydávají jednou či druhou cestou.
- **D** – mravenci, kteří nosí potravy více a absolvují kratší vzdálenost, tak vypouštějí více feromonové stopy, které se nadále přizpůsobí i ostatní mravenci.



Obr. 21: Postupný postup ACO. Upraveno ze zdroje: [43]

6 MODELY PRO ŘEŠENÍ ÚLOH

Pro řešení úloh byly vytvořeny dvě sady dat, které podstoupily otestování pomocí tří různých přístupů. Modely se liší primárně v počtu obsažených měst. K testování bylo využito deterministického přístupu pomocí metody nejbližšího souseda, stochastického přístupu pomocí genetického algoritmu a jako poslední byly modely otestovány pomocí GAMS.

6.1 Vizuální zobrazení pomocí knihovny Pygame

Veškeré vizuální zobrazení výsledných cest je provedeno v jazyce Python s využitím knihovny Pygame. Knihovna Pygame je primárně určena pro tvorbu počítačových her, ale slouží také pro práci se zvukem či grafikou. Výsledná zobrazení cest vznikají v samostatných oknech a mají rozlišení 1080 x 720.

Vytvoření okna je prováděno pomocí následujících příkazů.

Výpis 1: Kód pro vytvoření okna.

```
import pygame
WIDTH, HEIGHT = 1080, 720 # Šířka a výška okna
WINDOW = pygame.display.set_mode((WIDTH, HEIGHT)) # Zobrazení okna
pygame.display.set_caption("VOLITELNÝ POPISEK") # Popisek okna
FPS = 60
```

V další části kódu je vytvořena paleta barev, se kterou se v dalších částech pracuje.

Výpis 2: Kód pro vytvoření palety barev.

```
WHITE = (255, 255, 255)     # bílá
BLACK = (0, 0, 0)           # černá
RED = (255, 0, 0)           # červená
GRAY = (127, 127, 127)      # šedá
GREEN = (0, 255, 0)         # zelená
BLUE = (0, 0, 255)          # modrá
```

Poslední nedílnou složkou pro vytvoření základního okna v Pygame je hlavní smyčka, kde dojde na konci k vyvolání funkce draw_window(), která zajistí zakreslení měst, jejich vzájemné propojení a výslednou optimální cestu.

Výpis 3: Kód hlavní smyčky Pygame.

```
clock = pygame.time.Clock()
run = True
```

```
while run:
    clock.tick(FPS)
    for event in pygame.event.get():
        if event.type == pygame.QUIT: # Při kliknutí na tlačítko X dojde k zavření okna
            run = False
        if event.type == pygame.KEYDOWN
            if event.key == pygame.K_ESCAPE: # Při stisknutí klávesy ESC dojde k zavření
okna
                run = False

    draw_window()
```

Výpis 4: Zobrazení měst, vzájemného propojení a výsledné cesta v okně.

```
def draw_window():

    WINDOW.fill(BLACK) # Vyplnění okna černou barvou

    # Vypsání hodnoty nejkratší vzdálenosti v okně
    pygame.font.init()
    font = pygame.font.Font(pygame.font.get_default_font(), 25)
    text = font.render("The shortest path has the value: %d" %(shortest_distance), True,
WHITE)
    WINDOW.blit(text, (20,HEIGHT-50))

    # Vzájemné propojení všech měst
    for i in range(CITIES):
        for j in range(CITIES):
            if i != j:
                pygame.draw.line(WINDOW, GRAY, (CITY_INFO[i][1], CITY_INFO[i][2]),
(CITY_INFO[j][1], CITY_INFO[j][2]), 1)

    # Zvýraznění nejkratší cesty
    for i in range((len(Path))-1):
        pygame.draw.line(WINDOW, GREEN, (CITY_INFO[final_Path[i]][1],
CITY_INFO[final_Path[i]][2]), (CITY_INFO[final_Path[i+1]][1], CITY_INFO[final_
Path[i+1]][2]), 3)
    pygame.draw.line(WINDOW, GREEN, (CITY_INFO[final_Path[(len(Path))-1]][1],
CITY_INFO[final_Path[(len(Path))-1]][2]), (CITY_INFO[final_Path[0]][1],
CITY_INFO[final_Path[0]][2]), 3)
```

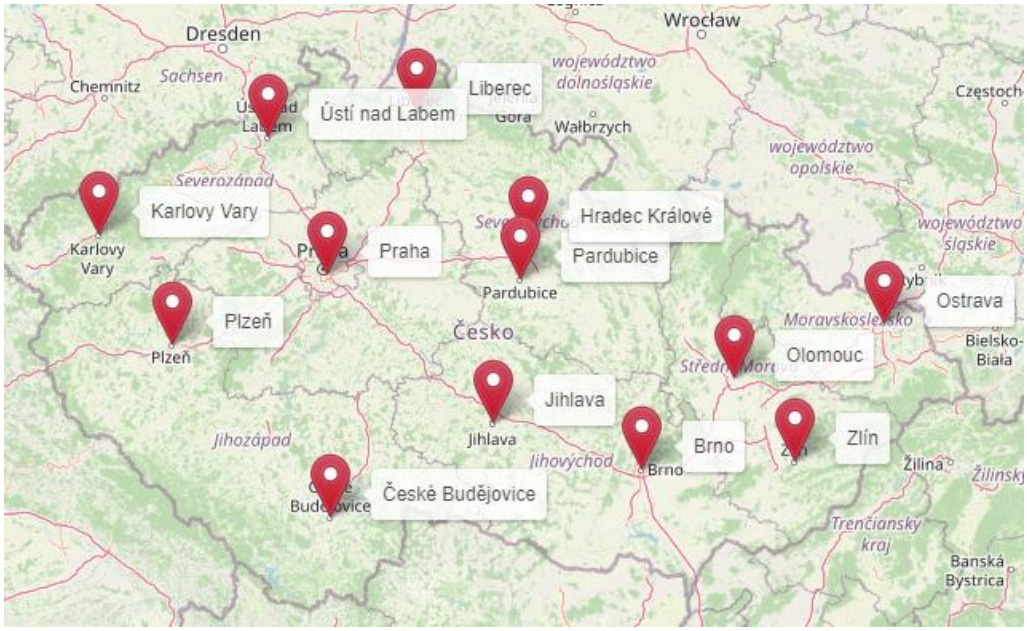
```
# Zobrazení všech měst
for i in range(CITIES):
    pygame.draw.circle(WINDOW, RED, (CITY_INFO[i][1], CITY_INFO[i][2]), 7)
pygame.draw.circle(WINDOW, BLUE, (CITY_INFO[final_Path[0]][1],
CITY_INFO[final_Path[0]][2]), 7) # Počáteční město

pygame.display.update() # Aktualizace okna
```

6.2 Krajská města České republiky

Pro méně náročná řešení byla vybrána všechna krajská města České republiky, a tedy nalezení nejkratší trasy, která má za úkol navštívit každé město právě jednou a vrátit se opět do města, ve kterém byla trasa započata.

V testovacím modelu jsou tedy zahrnuta tato města: Praha, Brno, Ostrava, Olomouc, Zlín, Jihlava, Pardubice, Hradec Králové, Liberec, Ústí nad Labem, Karlovy Vary, Plzeň a České Budějovice.

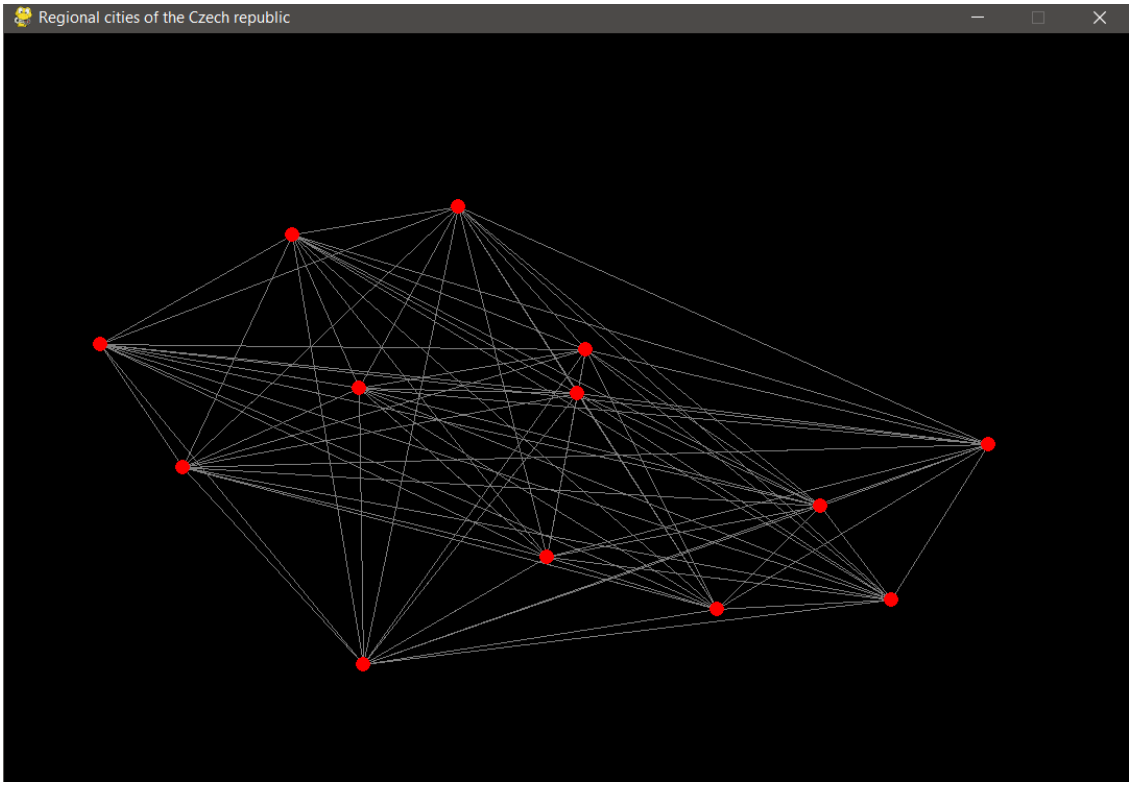
Na obrázku (Obr. 22) lze nalézt polohy všech měst na mapě České republiky pomocí OpenStreetMap. Dále byla vytvořena matice sousednosti jednotlivých měst, která je zobrazena v tabulce (Tab. 3), kde jsou jednotlivé vzdálenosti uváděny v kilometrech. Pro lepší přehled byly jednotlivé vzdálenosti zaokrouhleny na jednotky kilometrů. Ve výpočtech se pracuje s nezaokrouhlenými daty. Poslední obrázek (Obr. 24) zobrazuje rozložení měst a jejich vzájemné propojení, které bylo vytvořeno pomocí knihovny Pygame v jazyce Python. Tato knihovna je použita i pro následné vykreslení nejkratších cest tohoto modelu.



Obr. 22: Rozložení krajských měst ČR

Tab. 3: Matice sousednosti vybraných měst

	Pr	Br	Os	Ol	Zl	Ji	Pa	HK	Li	ÚnL	KV	Pl	ČB
Pr		183	276	208	250	110	95	100	90	73	114	84	121
Br	183		140	64	77	77	112	126	208	246	293	242	158
Os	276	140		79	81	200	181	180	251	315	389	352	292
Ol	208	64	79		51	122	117	122	204	258	321	279	213
Zl	250	77	81	51		152	163	171	253	305	362	315	234
Ji	110	77	200	122	152		73	92	157	179	216	164	94
Pa	95	112	181	117	163	73		19	96	142	208	175	151
HK	100	126	180	122	171	92	19		83	137	211	183	169
Li	90	208	251	204	253	157	96	83		73	166	165	204
ÚnL	73	246	315	258	305	179	142	137	73		95	112	190
KV	114	293	389	321	362	216	208	211	166	95		65	181
Pl	84	242	352	279	315	164	175	183	165	112	65		117
ČB	121	158	292	213	234	94	151	167	204	190	181	117	



Obr. 23: Zobrazení a vzájemné propojení krajských měst pomocí knihovny Pygame

Vytvoření matice v jazyce Python, která obsahuje informace o městech (pozice města, souřadnice x a souřadnice y), je k dispozici v souboru dataset_cz.py (Výpis 5).

Celý kód využívá knihovnu NumPy a je napsán v jazyce Python. Nejprve je vytvořena prázdná matice o rozměrech 13x3, která je následně postupně zaplněna

hodnotami, které nesou informace o číslování měst a jejich poloze. Veškeré hodnoty o poloze byly stejně posunuty tak, aby s nimi bylo možné dále pracovat s využitím knihovny Pygame.

Výpis 5: Ukázka kódu pro vytvoření matice, která nese informace o všech krajských městech

```
import numpy as np

CITY_INFO_COMPUTE = np.zeros((13, 3))  # vytvoření matice 13x3
# Praha
CITY_INFO_COMPUTE[0][0] = 0      # pozice města
CITY_INFO_COMPUTE[0][1] = 300   # souřadnice x
CITY_INFO_COMPUTE[0][2] = 300   # souřadnice y
# Brno
CITY_INFO_COMPUTE[1][0] = 1
CITY_INFO_COMPUTE[1][1] = 455.9652231
CITY_INFO_COMPUTE[1][2] = 396.47657264
# Ostrava
CITY_INFO_COMPUTE[2][0] = 2
CITY_INFO_COMPUTE[2][1] = 574.492928
CITY_INFO_COMPUTE[2][2] = 324.9808134
# Olomouc
CITY_INFO_COMPUTE[3][0] = 3
CITY_INFO_COMPUTE[3][1] = 501.228419
CITY_INFO_COMPUTE[3][2] = 351.779
# Zlín
CITY_INFO_COMPUTE[4][0] = 4
CITY_INFO_COMPUTE[4][1] = 531.916507
CITY_INFO_COMPUTE[4][2] = 392.619157
# Jihlava
CITY_INFO_COMPUTE[5][0] = 5
CITY_INFO_COMPUTE[5][1] = 382.234066
CITY_INFO_COMPUTE[5][2] = 373.810319
# Pardubice
CITY_INFO_COMPUTE[6][0] = 6
CITY_INFO_COMPUTE[6][1] = 395.18087149
CITY_INFO_COMPUTE[6][2] = 302.342792584
# Hradec Králové
CITY_INFO_COMPUTE[7][0] = 7
CITY_INFO_COMPUTE[7][1] = 398.83250592
CITY_INFO_COMPUTE[7][2] = 283.3546965
```

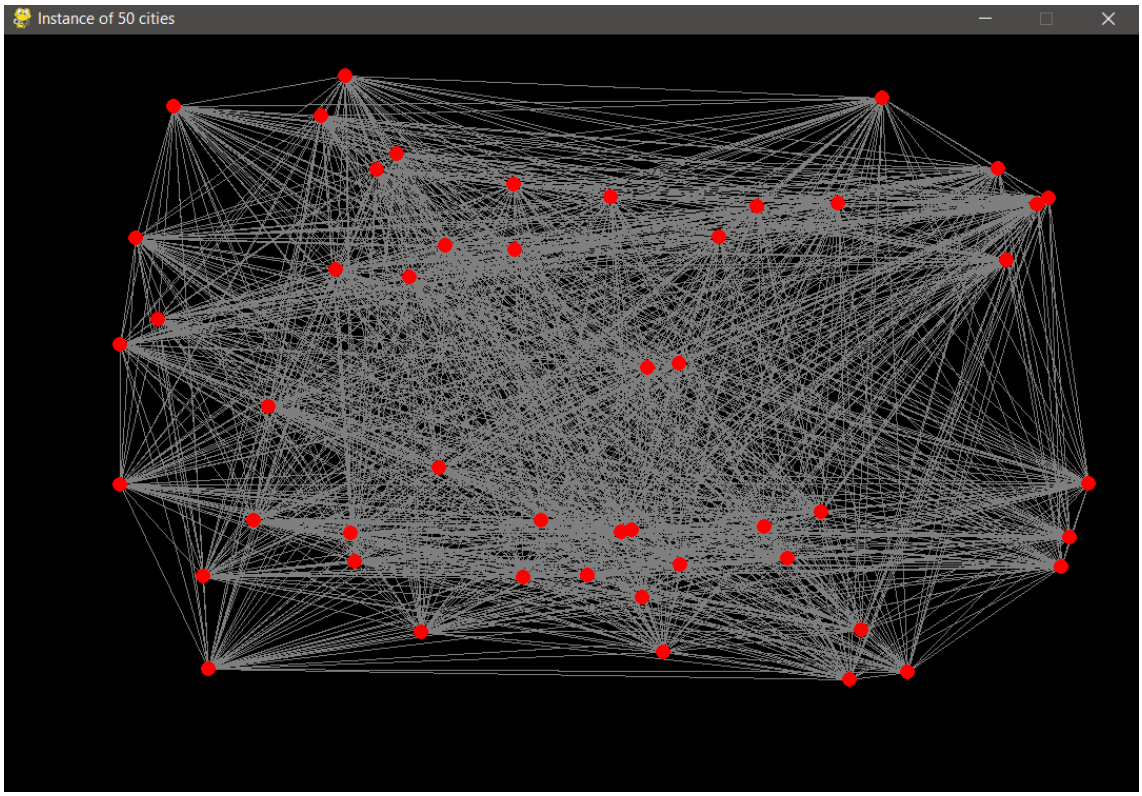
 # Jihlava
CITY_INFO_COMPUTE[8][0] = 8
CITY_INFO_COMPUTE[8][1] = 343.3650335
CITY_INFO_COMPUTE[8][2] = 220.9559815
 # Ústí nad Labem
CITY_INFO_COMPUTE[9][0] = 9
CITY_INFO_COMPUTE[9][1] = 271.0308531
CITY_INFO_COMPUTE[9][2] = 233.1842685
 # Karlovy Vary
CITY_INFO_COMPUTE[10][0] = 10
CITY_INFO_COMPUTE[10][1] = 187.7091511
CITY_INFO_COMPUTE[10][2] = 281.0074245
 # Plzeň
CITY_INFO_COMPUTE[11][0] = 11
CITY_INFO_COMPUTE[11][1] = 223.42393
CITY_INFO_COMPUTE[11][2] = 334.68811186
 # České Budějovice
CITY_INFO_COMPUTE[12][0] = 12
CITY_INFO_COMPUTE[12][1] = 302.000946366
CITY_INFO_COMPUTE[12][2] = 420.6687111

6.3 Model s velkým počtem instancí měst

Pro druhý testovací model bylo vybráno 50 měst. Tato varianta modelu byla vybrána z důvodu větší náročnosti na výpočet a s tím i spojená možnost rozdílnosti výsledků k porovnání.

Na obrázku (Obr. 24) lze nalézt vykreslení všech měst s využitím knihovny Pygame.

Tak jako u prvního modelu byl k vytvoření matice nesoucí informace o městech použit jazyk Python s využitím knihovny NumPy. V níže uvedené části kódu pro vytvoření tohoto modelu je vytvořeno pouze první město. Celý výčet informací je plně k dispozici v souboru dataset_50_cities.py.



Obr. 24: Zobrazení a vzájemné propojení 50 měst.

Výpis 6: Kód pro vytvoření matice, která nese informace o všech 50 městech.

```
import numpy as np

CITY_INFO_COMPUTE = np.zeros((50, 3)) # vytvoření matice 50x3

CITY_INFO_COMPUTE[0][0], CITY_INFO_COMPUTE[0][1], CITY_INFO_COMPUTE[0][2] = 0, 485, 142 # pozice města, souřadnice x, souřadnice y
```

7 OPTIMALIZACE POMOCÍ METODY NEJBLIŽŠÍHO SOUSEDA

První přístup k nalezení nejkratší cesty ve vytvořených modelech je použití deterministického algoritmu. Pro získání co nejlepších výsledků byla aplikována opakující se metoda nejbližšího souseda. I větší počet spuštění algoritmu nemusí vést k optimálnímu řešení.

Výpis 7: Ukázka kódu metody nejbližšího souseda.

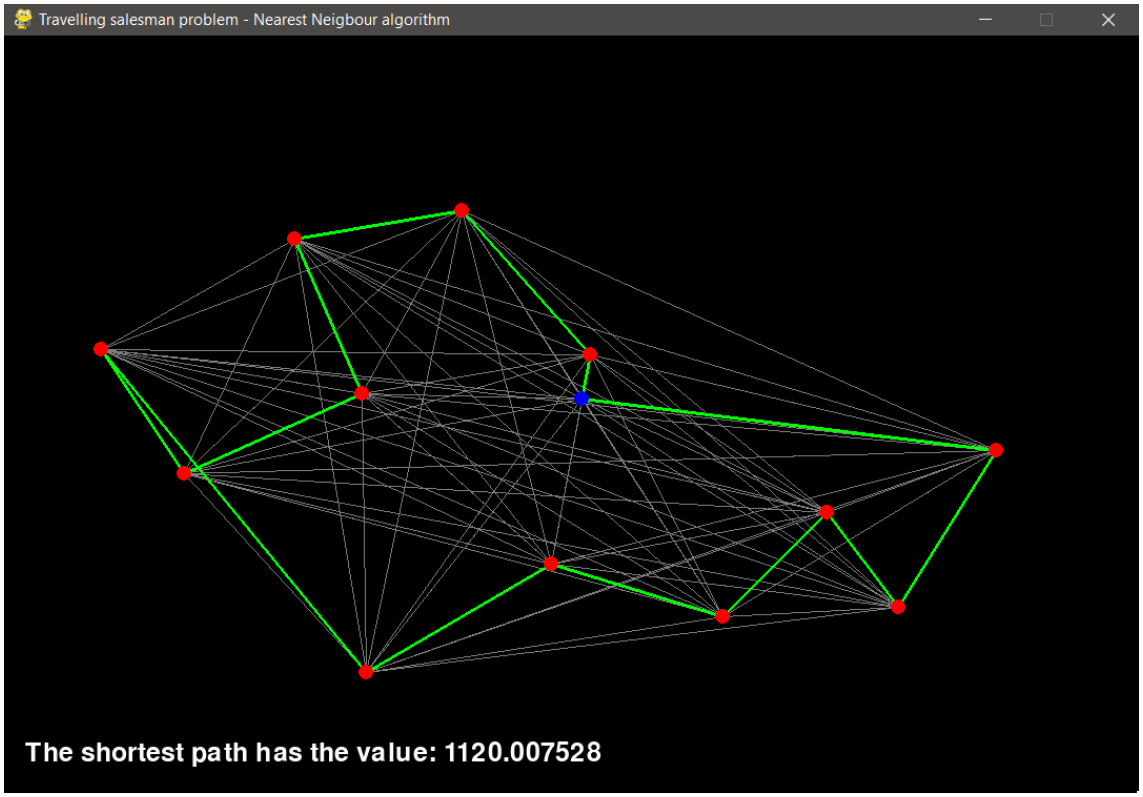
```
final_Path = []
shortest_distance = 9999999
for j in range(CITIES):
    CITY_INFO_COMPUTE = CITY_INFO.copy()
    starting_city = j
    actual_city = starting_city
    Path = []
    Path.append(actual_city)
    Total_distance = 0

    n = 0
    while n < CITIES-1:
        Distances = []

        for i in range(CITIES):

            D = math.sqrt((int(CITY_INFO_COMPUTE[actual_city][1]) - int(CITY_INFO_
            COMPUTE[i][1]))**2 + int((CITY_INFO_COMPUTE[actual_city][2]) - int(CI
            TY_INFO_COMPUTE[i][2]))**2)

            if D == 0:
                D = np.inf
            Distances.append(D)

        min = np.min(Distances)
        Total_distance += min
        min_index = Distances.index(min)
        CITY_INFO_COMPUTE[actual_city][1] = 9999999
        CITY_INFO_COMPUTE[actual_city][2] = 9999999
        actual_city = min_index
        Path.append(actual_city)
```

```
    n += 1
Path.append(starting_city)
LAST_PATH = math.sqrt((int(CITY_INFO[actual_city][1]) - int(CITY_INFO[starting_
city][1]))**2 + int((CITY_INFO[actual_city][2]) - int(CITY_INFO[starting_city][2]))*
*2)
Total_distance += LAST_PATH

if Total_distance < shortest_distance:
    shortest_distance = Total_distance
    final_Path = Path
```

Kompletní kód je k náhledu v souboru tsp_nn.py

7.1 Aplikace prvního modelu

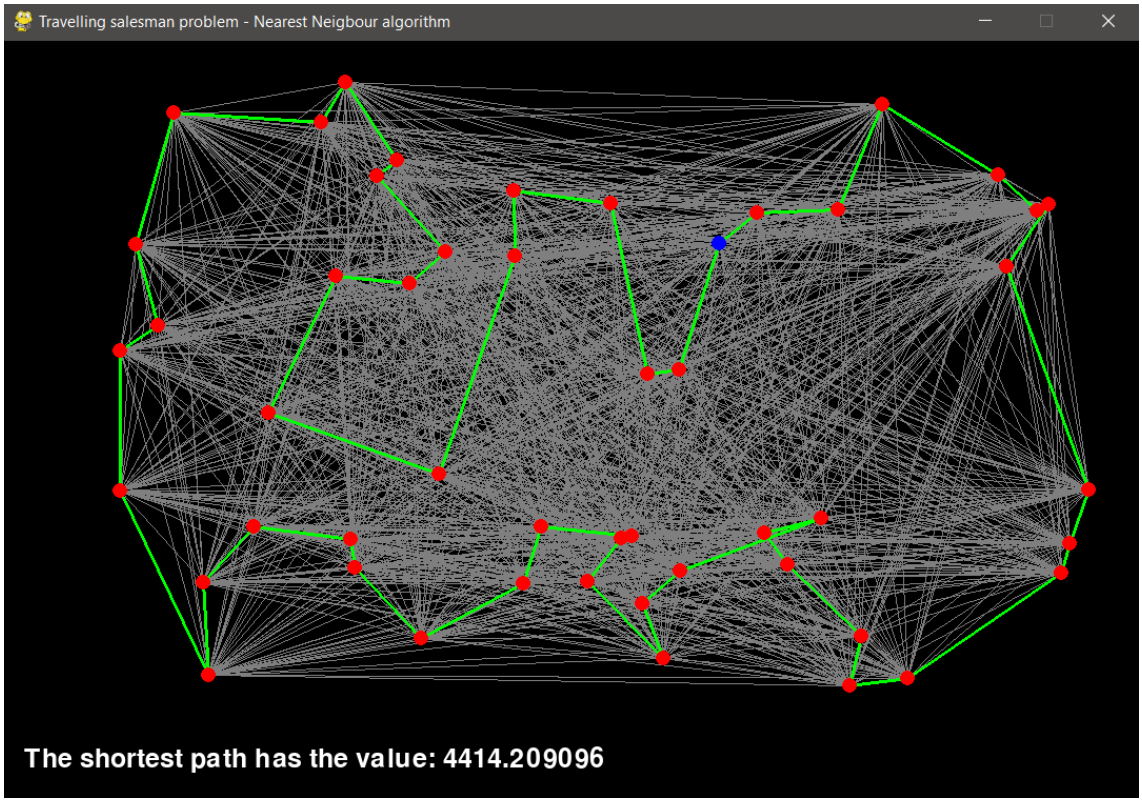
Nejkratší nalezená cesta je dlouhá 1120 km. Jako počáteční i konečné město bylo zvoleno město Pardubice. Trasa tedy vypadá následovně: Pardubice – Hradec Králové – Liberec – Ústí nad Labem – Praha – Plzeň – Karlovy Vary – České Budějovice – Jihlava – Brno – Olomouc – Zlín – Ostrava – Pardubice. Z obrázku (Obr. 25) lze posoudit, že nalezená cesta není nejkratší možná, jelikož dochází ke zbytečnému prodloužení trasy, které je způsobeno tím, že algoritmus vyhledává nejbližší nenavštívené město z města naposledy navštíveného.



Obr. 25: Nalezené řešení pro první model pomocí NN.

7.2 Aplikace druhého modelu

Nejkratší nalezená cesta má délku 4414,2 km.



Obr. 26: Nalezené řešení pro druhý model pomocí NN.

8 OPTIMALIZACE POMOCÍ GENETICKÉHO ALGORITMU

Druhý přístup testování vytvořených modelů byl pomocí stochastického algoritmu, přesněji pomocí genetického algoritmu. Algoritmus pro oba modely obsahoval následující vstupní parametry: počet měst, velikost populace, pravděpodobnost mutace, pravděpodobnost křížení, velikost turnaje a počet generací.

Jako první krok celého algoritmu je vytvoření počáteční populace. Počáteční populace nese název POP_SET a skládá se z určitého počtu náhodných tras.

Výpis 8: Funkce pro vytvoření počáteční populace.

```
def initial_population(CITY_INFO_COMPUTE,N_POP):
    POP_SET = []

    for i in range(N_POP):
        np.random.shuffle(CITY_INFO_COMPUTE)
        SET = []

        for j in range(CITIES):
            SET.append(int(CITY_INFO_COMPUTE[j][0]))

        POP_SET.append(SET)

    return np.array(POP_SET)
```

V dalších kroku algoritmu je pro tyto náhodné trasy vypočtena hodnota fitness funkce, která se rovná vzdálenosti mezi všemi městy.

Výpis 9: Funkce pro výpočet hodnoty fitness funkce a pro výpočet vzdálenosti mezi městy i a j.

```
def compute_distance(first, second):

 return math.sqrt((int(CITY_INFO[int(first)][1]) - int(CITY_INFO[int(second)][1]))**2
+ int((CITY_INFO[int(first)][2]) - int(CITY_INFO[int(second)][2]))**2)

def fitness(POP_SET):
    total = 0
    for j in range(CITIES-1):

        first = int(POP_SET[j])
```

```
        second = int(POP_SET[j+1])
        total += compute_distance(first, second)
    total += compute_distance(POP_SET[0], second)

    return total
```

Po výpočtu hodnot fitness funkcí pro všechny možné cesty obsažené v populaci je provedena následná selekce nejlepších jedinců, na které je dále aplikován operátor křížení a mutace. Tyto cesty jsou následně přidány do nové populace, která bude znovu testována v nové generaci. Pro operátor selekce byla vybrána turnajová selekce, jako operátor křížení bylo testováno jednobodové i dvoubodové křížení a pro operátor mutace byla vybrána inverze dvou měst.

Výpis 10: Funkce pro selekci, křížení a mutaci.

```
def selection(POP_SET):

    selection_index = random.randint(0, N_POP-1)

    for i in range(TS_VEL):
        index = random.randint(0, N_POP-1)

        if fitness(POP_SET[index]) < fitness(POP_SET[selection_index]):
            selection_index = index

    return POP_SET[selection_index]

def crossover(parent_1, parent_2):

    if np.random.rand() < R_CROSS:
        crossover_point = random.randint(0, CITIES)
        cross_1 = parent_1[:crossover_point]
        cross_2 = parent_2[:crossover_point]

        cross_3 = [city for city in parent_2 if city not in cross_1]
        cross_4 = [city for city in parent_1 if city not in cross_2]

        child_1 = np.concatenate((cross_1, cross_3))
        child_2 = np.concatenate((cross_2, cross_4))

    else:
        child_1, child_2 = parent_1.copy(), parent_2.copy()
    return child_1, child_2
```

```
def two_point_crossover(parent_1, parent_2):
    if np.random.rand() < R_CROSS:
        crossover_point1 = random.randint(0, CITIES)
        crossover_point2 = random.randint(crossover_point1, CITIES)

        cross_1_start = parent_1[:crossover_point1]
        cross_2_start = parent_2[:crossover_point1]

        cross_1_end = parent_1[crossover_point2:]
        cross_2_end = parent_2[crossover_point2:]

        cross_1_mid = [item for item in parent_2 if item not in cross_1_start and item not
in cross_1_end]
        cross_2_mid = [item for item in parent_1 if item not in cross_2_start and item not
in cross_2_end]

        child_1 = np.concatenate((cross_1_start,cross_1_mid,cross_1_end))
        child_2 = np.concatenate((cross_2_start,cross_2_mid,cross_2_end))
    else:
        child_1, child_2 = parent_1.copy(), parent_2.copy()

    return child_1, child_2

def mutation(child):
    if np.random.rand() < R_MUT:
        mut_1 = random.randint(0, CITIES-1)
        mut_2 = random.randint(0, CITIES-1)
        mut_city_1 = child[mut_1]
        mut_city_2 = child[mut_2]
        child[mut_1] = mut_city_2
        child[mut_2] = mut_city_1

    return child
```

Běh celého algoritmu probíhá a spouští se pomocí funkce genetického algoritmu.

Výpis 11: Funkce pro běh genetického algoritmu.

```
def genetic_algorithm(SP = 99999):

    POP_SET = initial_population(CITY_INFO_COMPUTE, N_POP)
    pop_fitness = np.zeros(N_POP)
    generation = 0
```

```
while generation <= 50:
    for i in range(N_POP):
        distance = fitness(POP_SET[i])
        pop_fitness[i] = distance
    min = np.min(pop_fitness)
    min_idx = np.argmin(pop_fitness)

    if min < SP:
        SP = min
        shortest_path = []

        for i in range(CITIES):
            shortest_path.append(int(POP_SET[min_idx][i]))

        shortest_path.append(int(POP_SET[min_idx][0]))

    selected = [selection(POP_SET) for _ in range(N_POP)]
    new_population = []

    for i in range(0, N_POP, 2):
        parent_1, parent_2 = selected[i], selected[i+1]
        child_1, child_2 = crossover(parent_1, parent_2)
        child_1 = mutation(child_1)
        child_2 = mutation(child_2)
        new_population.append(child_1)
        new_population.append(child_2)

    POP_SET = new_population
    print("Generation: %d, The shortest path has the distance: %f" % (generation, SP))
    generations.append(generation)
    dist.append(SP)
    generation += 1

return shortest_path, SP
```

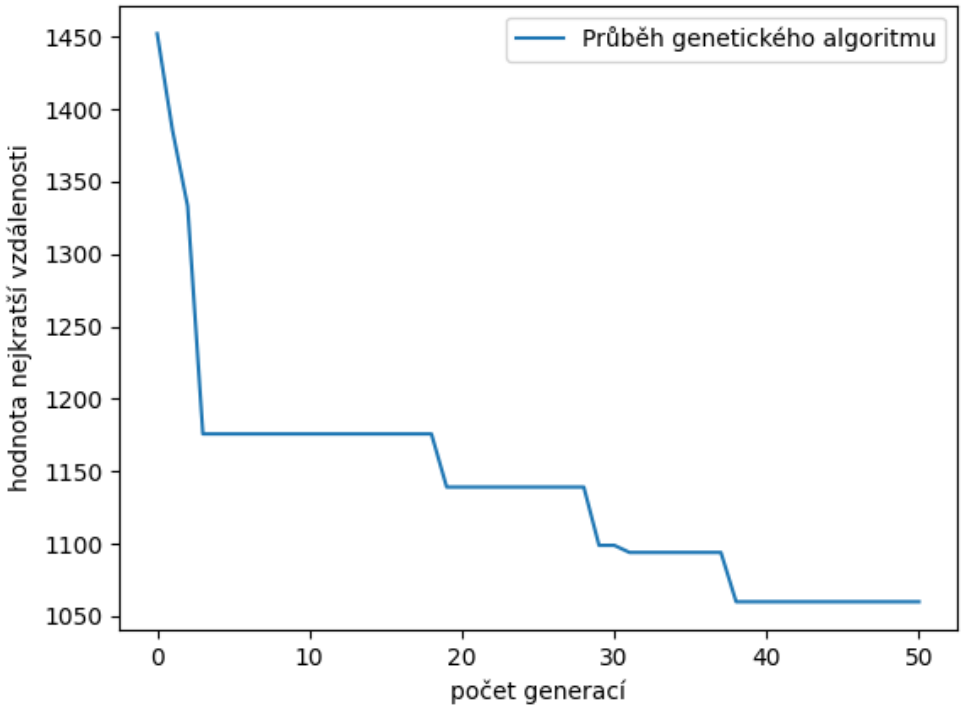
8.1 Aplikace prvního modelu

Pro aplikaci na prvním modelu byly zvoleny následující vstupní parametry:

- CITIES = 13 – počet měst
- N_POP = 100 – velikost populace

- R_MUT = 0.03 – pravděpodobnost mutace
- R_CROSS = 0.9 – pravděpodobnost křížení
- TS_VEL = 20 – velikost turnaje
- N_GEN = 50 – počet generací

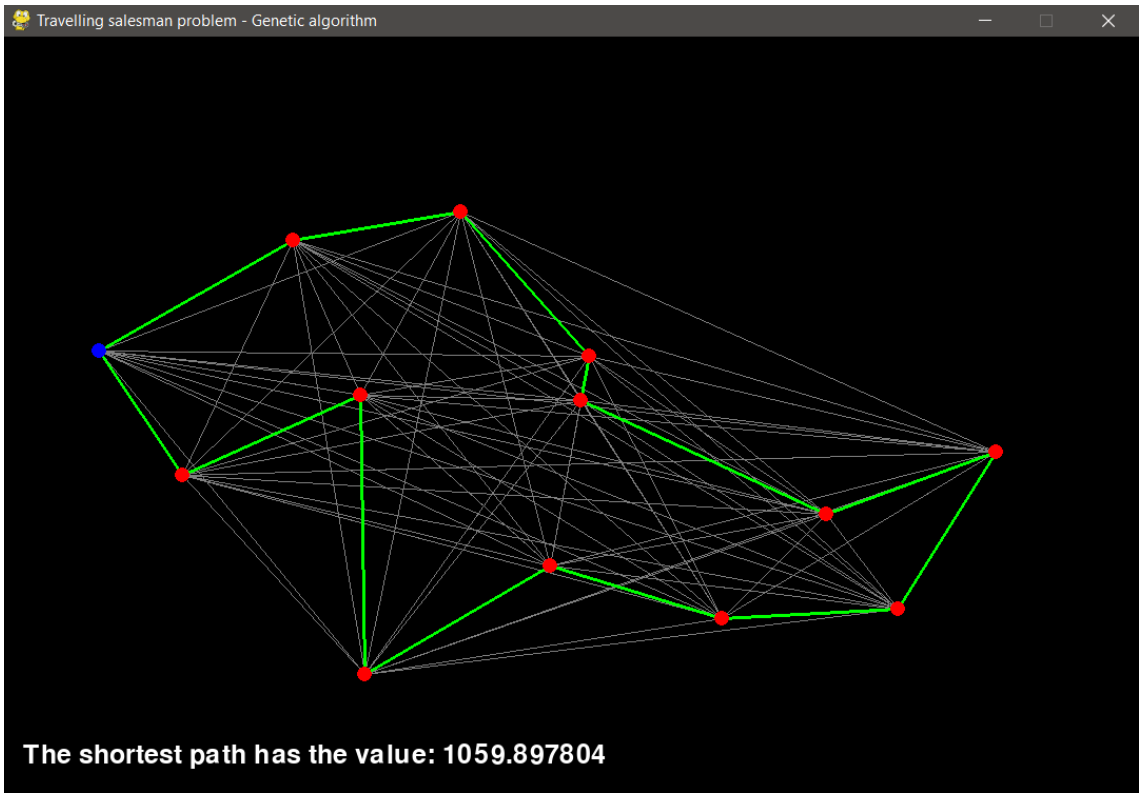
Ukončovací podmínkou algoritmu je splnění předem stanoveného počtu generací.



Obr. 27: Graf závislosti nejkratší uražené vzdálenosti na počtu generací.

Nalezená nejkratší cesta je dlouhá 1059,9 km. Počáteční a koncové město cesty jsou Karlovy Vary. Trasa tedy vypadá následovně: Karlovy Vary – Plzeň – Praha – České Budějovice – Jihlava – Brno – Zlín – Ostrava – Olomouc – Pardubice – Hradec Králové – Liberec – Ústí nad Labem – Karlovy Vary.

Kompletní kód pro řešení prvního modelu pomocí genetického algoritmu je k dispozici v souboru tsp_ga.py.



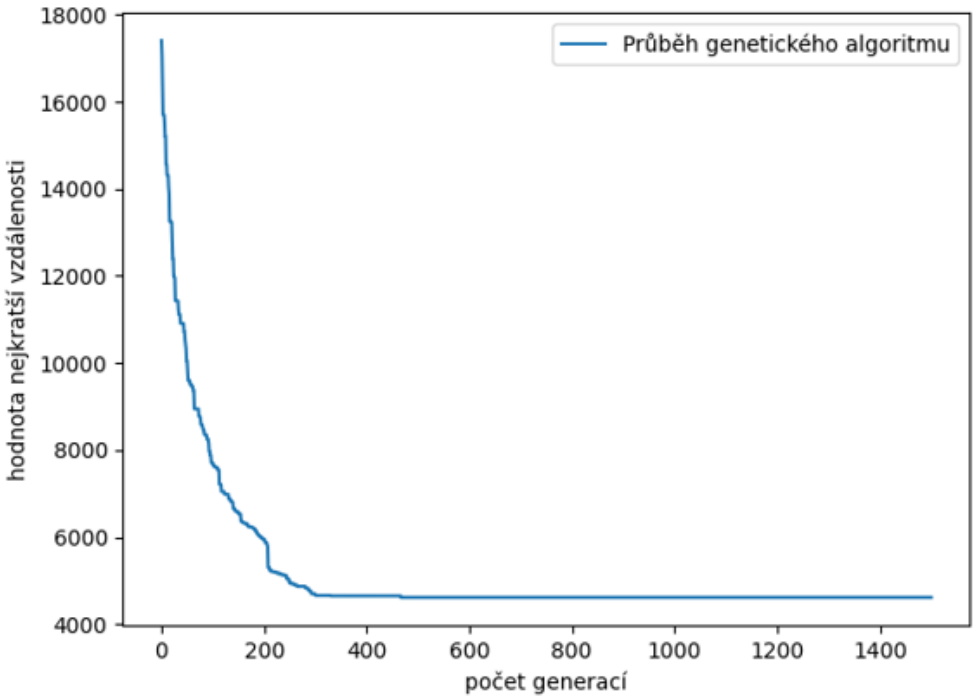
Obr. 28: Nalezené řešení pomocí GA pro první model.

8.2 Aplikace druhého modelu

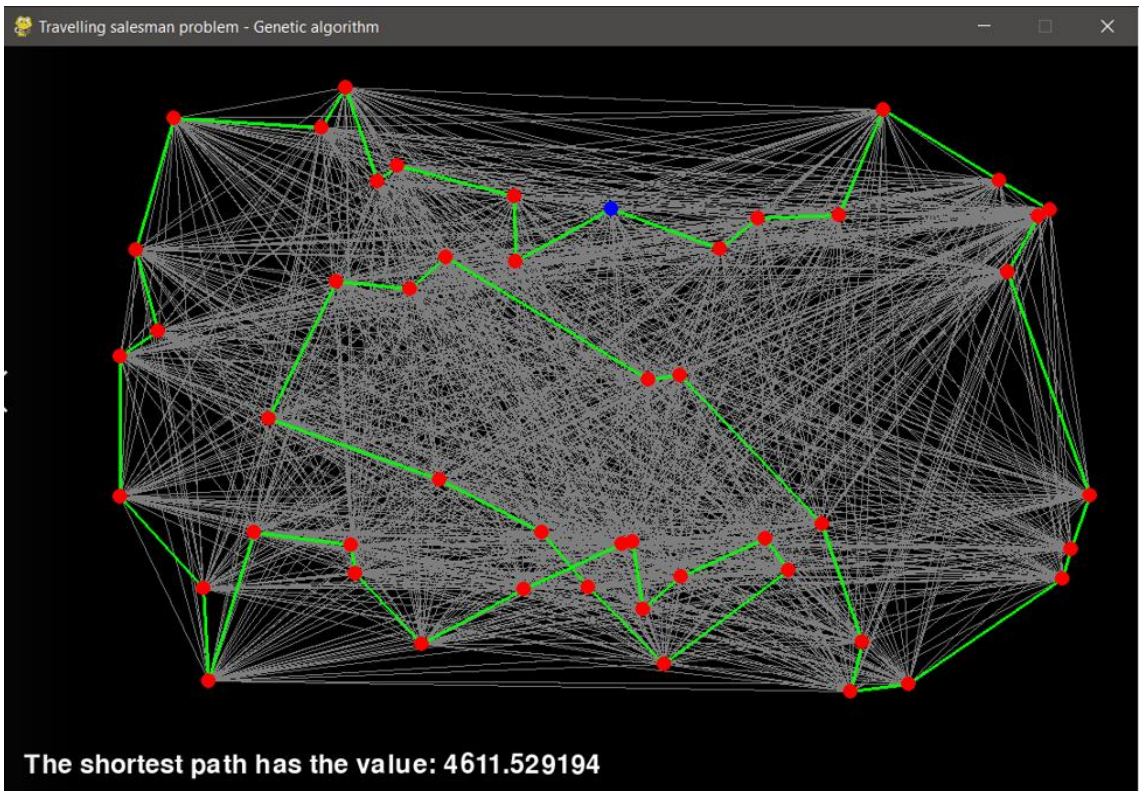
Pro získání výsledků pro druhý model byly použity následující vstupní parametry:

- CITIES = 50 – počet měst
- N_POP = 150 – velikost populace
- R_MUT = 0.2 – pravděpodobnost mutace
- R_CROSS = 0.5 – pravděpodobnost křížení
- TS_VEL = 4 – velikost turnaje
- N_GEN = 1500 – počet generací

Stejně jako u předchozího modelu je ukončovací podmínka splnění předem stanoveného počtu generací. Nejkratší nalezená cesta je dlouhá 4611,53 km.



Obr. 29: Graf závislosti hodnoty nejkratší vzdálenosti na počtu generací.



Obr. 30: Nalezená cesta pro 50 měst pomocí GA.

9 OPTIMALIZACE POMOCÍ GAMS

Jako poslední přístup k otestování modelů byl zvolen modelovací systém GAMS. GAMS (General Algebraic Modeling System) je modelovací jazyk, který umožnuje rychlé převedení optimalizačních problémů do počítačového kódu. Jazyk je velmi flexibilní, a to hlavně díky tomu, že umožňuje měnit použité řešiče, zatímco model zůstává beze změny [44].

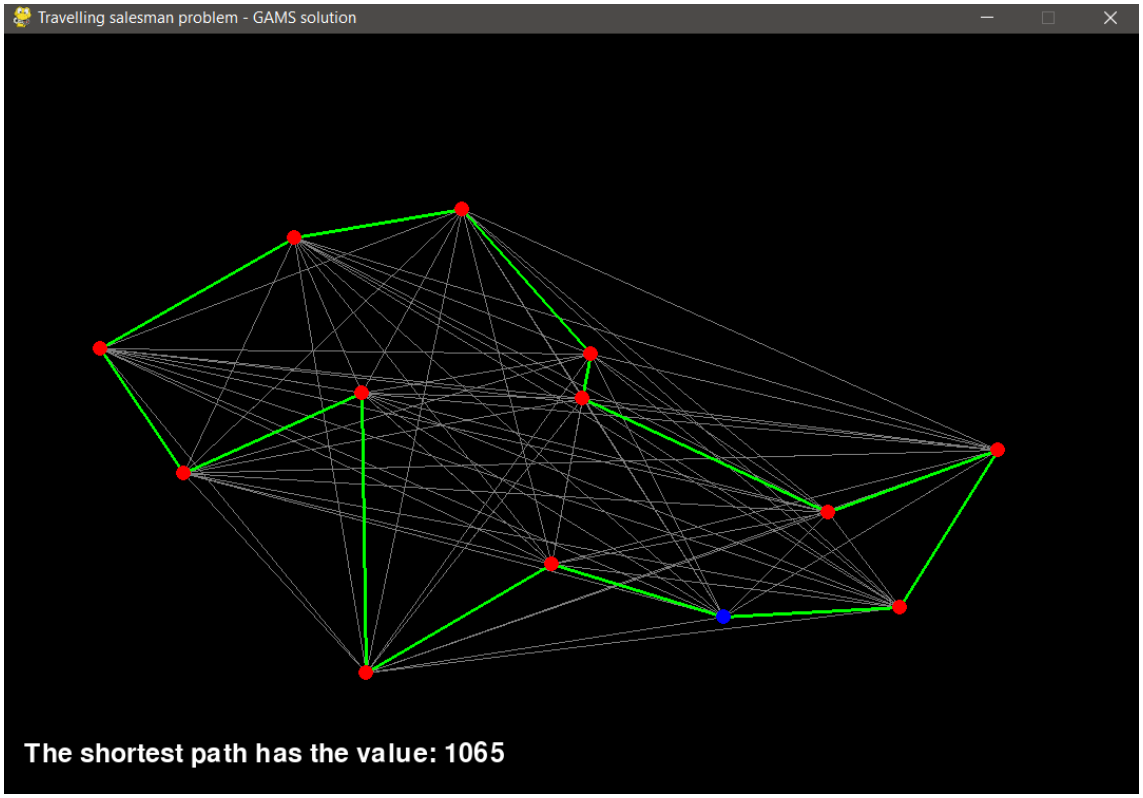
Výpis 12: Ukázka kódu, který slouží k nalezení nejkratší cesty v GAMS.

```
sets
   i /i1*i13/
;
alias (i,j);

table c(i,j)

set arcs(i,j);
arcs(i,j)$(not sameas(i,j)) = yes;

c(arcs(i,j)) = max(c(i,j),c(j,i));

scalar n 'počet uzlů';
n = card(i);

binary variables x(i,j);
positive variables y(i,j);
variable z;

scalar f /0.01/;
equations
  delkatrasy
  FlowY(i)
  TotalX
  ArcGain(i,j)
  FlowX(i)
;

set i2(i);
i2(i)$(ord(i)>=2) = yes;

delkatrasy..  z =e= sum(arcs, c(arcs)*x(arcs));
FlowY(i2(i)).. sum(arcs(i,j), y(i,j)) - sum(arcs(i,j), y(j,i)) =e= f;
```

```
TotalX.. sum(arcs, x(arcs)) =L= n;
ArcGain(arcs).. y(arcs) =l= (1+n*f)*x(arcs);
FlowX(i2(i)).. sum(arcs(i,j),x(i,j)) =e= sum(arcs(i,j),x(j,i));

option optcr=0;
option iterlim=10000000;

model tsp/all/;
solve tsp minimizing z using mip;

display x.l;
```

9.1 Aplikace prvního modelu

Nejkratší nalezená cesta je dlouhá 1065 km. Za počáteční i konečné město bylo zvoleno město Brno. Optimální trasa tedy vypadá následovně: Brno – Zlín – Ostrava – Olomouc – Pardubice – Hradec Králové – Liberec – Ústí nad Labem – Karlovy Vary – Plzeň – Praha – České Budějovice – Jihlava – Brno.



Obr. 31: Vizuální zobrazení trasy prvního modelu pomocí GAMS.

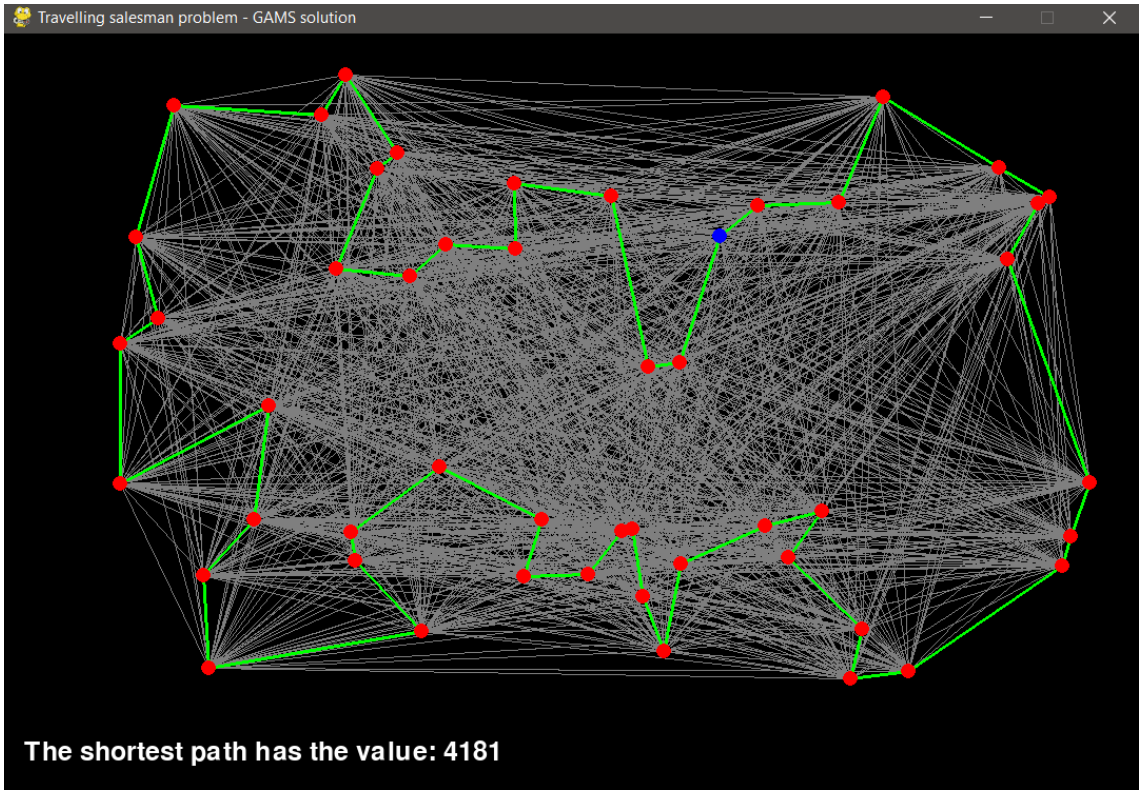
V uvedené ukázce kódu (Výpis 13) je na místě table c(i,j) matice sousednosti všech měst. Kompletní kód je k náhledu v souboru TSP_13_cities.gms.

Výpis 13: Matice sousednosti měst.

```
table c(i,j) 'matice sousednosti (km)'
     i1    i2    i3    i4    i5    i6    i7    i8    i9   i10   i11   i12   i13
i1        183   276   208   250   110    95   100    90    73   114    84   121
i2              140    64    77    77   112   126   208   246   293   242   158
i3                     79    81   200   181   180   251   315   389   352   292
i4                           51   122   117   122   204   258   321   279   213
i5                                152   163   171   253   305   362   315   234
i6                                      73    92   157   179   216   164    94
i7                                            19    96   142   208   175   151
i8                                                  83   137   211   183   169
i9                                                       73   166   165   204
i10                                                            95   112   190
i11                                                                  65   181
i12                                                                       117
i13
;
```

9.2 Aplikace druhého modelu

Nejkratší nalezená cesta má délku 4181 km.



Obr. 32: Vizuální zobrazení trasy druhého modelu pomocí GAMS.

Při řešení druhého modelu byl v jazyce GAMS použit rozdílný přístup, kdy namísto matice sousednosti byla definována poloha každého z měst a z těchto informací následně vypočtena vzdálenost mezi jednotlivými městy.

Kompletní kód je k dispozici v souboru TSP_50_cities.gms.

10 ZÁVĚR

Práce je rozdělena do dvou základních částí, jedná se o část rešeršní, která dodává přehled o daném tématu a část praktická, která se zabývá následnou aplikací problému obchodního cestujícího. Úvod práce je věnován zasvěcení do problematiky teorie grafů, její historii a jsou zde podrobně popsány základní pojmy teorie grafů, které je potřeba znát k pochopení problematiky okružních problémů. V další kapitole je podrobně rozepsán problém obchodního cestujícího, jeho historie, charakteristika a možnosti využití v praxi a s ním spojené i různé varianty problémy. Na tuto kapitolu navazuje část o dalším okružním problému, a to problému plánování rozvozu. Kapitola je především zaměřena na základní charakteristiku problému a jeho různé modifikace způsobené různými podmínkami a omezeními. V poslední kapitole rešeršní části práce jsou více přiblíženy vybrané metody řešení problémů, k detailnějšímu popisu bylo vybráno pár deterministických i stochastických algoritmů.

Cílem práce bylo provést rozbor problémů a pro reálná data v závislosti na jejich počtu implementovat jejich řešení. První cíl práce byl splněn v části rešeršní a pro splnění druhého ze zadaných cílů byly vytvořeny dva testovací modely, kde první model představoval nalezení nejkratší cesty mezi všemi krajskými městy České republiky. Jako druhý testovací model byl použit náhodný výběr 50 měst, které představovaly složitější variantu problému. Pro testování těchto dvou modelů byly implementovány dvě metody, deterministická (opakující se metoda nejbližšího souseda) a stochastická (genetický algoritmus). Na závěr celého testování byl vytvořen i model v GAMSu, který sloužil především pro vyhodnocení a určení správnosti získaných výsledků pomocí algoritmů předešlých.

Pro oba modely proběhlo nejprve nalezení výsledků pomocí metody nejbližšího souseda, model stačilo spustit pro každý model pouze jednou, jelikož se jedná o deterministickou metodu, tak výsledek je při každém běhu stejný. Naopak pomocí genetického algoritmu bylo testování obou modelů spuštěno hned několikrát, a to z důvodu rozdílnosti výsledků každého běhu a hledání nejoptimálnějšího řešení. Taktéž se při různých bězích algoritmů měnily vstupní parametry, které mohly ovlivnit nalezené výsledky.

Pro první model krajských měst byla nalezena nejkratší cesta (1059.9 km) pomocí genetického algoritmu, jehož řešení v každém spuštěném běhu bylo velice přesné a pohybovalo se s rozdílem pár jednotek kilometrů od nalezené nejoptimálnější trasy. Co se týče metody nejbližšího souseda, tak zde řešení bylo horší (1120 km), což bylo způsobeno stavbou algoritmu, který jako další navštívené město vybírá to nejbližší, což nemusí být vždy nejoptimálnější volbou. Vytvořený model v GAMSu vykazoval velice podobnou hodnotu (1065 km) i cestu jako genetický algoritmus, vznikla zde malá odchylka výsledku kvůli zaokrouhlení hodnot ve vytvořené matici sousednosti, která zde byla k výpočtu použita.

Nejkratší nalezená cesta pro druhý model byla pomocí modelu v GAMSu, která měla délku 4181 km, z autorových implementovaných algoritmů dopadla lépe metoda nejbližšího souseda, což bylo u většího počtu instancí měst velice překvapivé. Pro tuto metodu měla nalezená cesta délku 4414,2 km. Při řešení pomocí genetického algoritmu bylo po velkém počtu běhů a úpravám vstupních parametrů nalezena nejkratší cesta 4611,53 km, což může být způsobeno nepřímočarostí implementace genetického algoritmu, která měla za následek, že populace v nových generacích mnohdy vykazovala horší výsledky. Obecně je známo, že řešení pomocí genetického algoritmu může mít problém s nalezením přesného optima a v tomto případě pro větší počet měst se toto tvrzení potvrdilo i praxi.

SEZNAM POUŽITÉ LITERATURY

[1] C. CARLSON, Stephan. Graph theory. Britannica [online]. 24.11.2020 [cit. 2022-02-23]. Dostupné z: https://www.britannica.com/topic/graph-theory

[2] PADMANAVA, Samanta. Introduction to Graph Theory. 2011. DOI: 10.13140/RG.2.2.25721.88166.

[3] ŠIŠMA, Pavel. Vznik a vývoj teorie grafů. Člověk-umění-matematika. Sborník přednášek z letních škol Historie matematiky. Praha: Prometheus, 1996, s. 155-165.

[4] 4 Colour Theorem: All The World's Countries Can Be Coloured Using Only 4 Colours, So That No Two Adjacent Countries Share The Same Colour. Brilliant Maps [online]. 11.04.2018 [cit. 2022-02-23]. Dostupné z: https://brilliantmaps.com/4-colour-theorem/

[5] KOVÁŘ, Petr. Úvod do teorie grafů. 2021. Dostupné také z: https://homel.vsb.cz/~kov16/files/uvod_do_teorie_grafu.pdf

[6] Types of Graphs [online]. [cit. 2022-02-27]. Dostupné z: https://www.javatpoint.com/graph-theory-types-of-graphs

[7] P, NP, NP-Complete and NP-Hard Problems in Computer Science [online]. 25.08.2021 [cit. 2022-02-28]. Dostupné z: https://www.baeldung.com/cs/p-np-np-complete-np-hard

[8] PATEL, Rakesh. Travelling Salesman Problem: Challenges & Solutions To Find the Minimum Cost Path [online]. 23.03.2022 [cit. 2022-03-03]. Dostupné z: https://www.upperinc.com/guides/travelling-salesman-problem/

[9] DAVENDRA, Donald. TRAVELING SALESMAN PROBLEM, THEORY AND APPLICATIONS. India: InTech, 2010, 336 s. ISBN 978-953-307-426-9.

[10] CHASE, Connor, Harrison CHEN, Alex NEOH a Maximum WILDER-SMITH. An Evaluation of the Traveling Salesman Problem [online]. [cit. 2022-03-05]. Dostupné z: https://scholarworks.calstate.edu/downloads/xg94hr81q

[11] Traveling Salesman Problem [online]. [cit. 2022-03-08]. Dostupné z: https://www.math.uwaterloo.ca/tsp/index.html

[12] CUMMINGS, Nigel. A brief History of the Travelling Salesman Problem [online]. [cit. 2022-04-03]. Dostupné z: https://www.theorsociety.com/about-or/or-methods/heuristics/a-brief-history-of-the-travelling-salesman-problem/

[13] MARIESCU-ISTODOR, Radu a Fränti PASI. Solving the Large-Scale TSP Problem in 1 h: Santa Claus Challenge 2020. Frontiers in Robotics and AI [online]. 04.10.2021, 2011(8) [cit. 2022-03-08]. Dostupné z: https://www.frontiersin.org/articles/10.3389/frobt.2021.689908/full#h2

[14] APPLEGATE, David L. The Travelling Salesman Problem: A Computational Study. Princeton: Princeton University Press, 2006, 593 s.

[15] OLIINYK, Andrii, et al. "Evolutionary method for solving the traveling salesman problem." 2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T). IEEE, 2018.

[16] RADA, Václav. CW05 – TEORIE ROZHODOVACÍCH PROCESŮ CW13 – LOGISTIKA - 26. a 24. přednáška – Čínský listonoš + TSP [online]. Ústav technologie, mechanizace a řízení staveb: Fakulta Stavební VUT v Brně, 2016 [cit. 2022-03-16]. Dostupné z: https://slideplayer.cz/slide/11381209/

[17] Chinese Postman Problem [online]. [cit. 2022-03-16]. Dostupné z: https://personal.utdallas.edu/~dzdu/cs6363/Chinese.htm

[18] LIAO, Chung-Shou a Yamming HUANG. The Covering Canadian Traveller Problem. Theoretical Computer Science [online]. 2014(530) [cit. 2022-03-20]. ISSN 0304-3975. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0304397514001327#se0010

[19] Canadian traveller problem. Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2022-03-20]. Dostupné z: https://en.wikipedia.org/wiki/Canadian_traveller_problem

[20] SAJAYKUMAR, J. VEHICLE ROUTING PROBLEM AND ITS VARIANTS [online]. 24.04.2020 [cit. 2022-04-04]. Dostupné z: https://www.linkedin.com/pulse/vehicle-routing-problem-its-variants-sajaykumar-j

[21] TOTH, Paolo a Daniele VIGO, ed. The Vehicle Routing Problem. Philadelphia: Society for Industrial and Applied Mathematics, 2002, 367 s. 47126854.

[22] ZHANG, Haifei, Hongwei GE, Jinlong YANG a Yubing TONG. Review of Vehicle Routing Problems: Models, Classification and Solving Algorithms. Archives of Computational Methods in Engineering. 2022, 29(1), 195-221. ISSN 1134-3060. Dostupné z: doi:10.1007/s11831-021-09574-x

[23] KUMAR, Suresh Nanda a Ramasamy PANNEERSELVAM. A Survey on the Vehicle Routing Problem and Its Variants. Intelligent Information Management. 2012, 04(03), 66-74. ISSN 2160-5912. Dostupné z: doi:10.4236/iim.2012.43010

[24] DERBEL, Houda, Bassem JARBOUI a Patrick SIARRY, ed. Green Transportation and New Advances in Vehicle Routing Problems. 1. Springer Nature Switzerland, 2020, 228 s. ISBN 978-3-030-45311-4.

[25] Difference between Deterministic and Non-deterministic Algorithms. GeeksforGeeks [online]. 24.11.2021 [cit. 2022-04-20]. Dostupné z: https://www.geeksforgeeks.org/difference-between-deterministic-and-non-deterministic-algorithms/

[26] Deterministic algorithm [online]. [cit. 2022-04-20]. Dostupné z: https://academickids.com/encyclopedia/index.php/Deterministic_algorithm

[27] SHARMA, Nitin. Difference between Deterministic and Non-deterministic Algorithms [online]. 09.06.2020 [cit. 2022-04-20]. Dostupné z: https://www.tutorialspoint.com/difference-between-deterministic-and-non-deterministic-algorithms

[28] 5 Difference Between Deterministic And Non-deterministic Algorithm [online]. [cit. 2022-04-20]. Dostupné z: https://vivadifferences.com/difference-between-deterministic-and-non-deterministic-algorithms/

[29] Difference between deterministic and Nondeterministic [online]. [cit. 2022-04-20]. Dostupné z: https://en.wikipedia.org/wiki/File:Difference_between_deterministic_and_Nondeterministic.png#filelinks

[30] BRANDA, Martin. Úvod do úloh plánování rozvozu (Vehicle Routing Problems) [online]. Univerzita Karlova v Praze Matematicko-fyzikální fakulta Katedra pravděpodobnosti a matematické statistiky, 2015 [cit. 2022-04-20]. Dostupné z: http://artax.karlin.mff.cuni.cz/~branm1am/download/Branda_VRP_2015.pdf

[31] BONELLO, Simon. WHAT IS A BRUTE FORCE ALGORITHM? [online]. 02.02.2021 [cit. 2022-04-22]. Dostupné z: https://www.chubbydeveloper.com/brute-force-algorithm/

[32] Most important type of Algorithms [online]. 25.03.2022 [cit. 2022-04-22]. Dostupné z: https://www.geeksforgeeks.org/most-important-type-of-algorithms/

[33] COCHRAN, James J., Louis A. COX, Pinar KESKINOCAK, Jeffrey P. KHAROUFEH a J. Cole SMITH. Wiley Encyclopedia of Operations Research and Management Science. 2010-06-15. Dostupné z: doi:10.1002/9780470400531.eorms0116

[34] CLAUSEN, Jens. Branch and Bound Algorithms – Principles and Examples. Department of Computer Science, University of Copenhagen, Universitetsparken 1, DK2100 Copenhagen, Denmark., 1999.

[35] DATTA, Subham. Branch and Bound Algorithm [online]. 10.10.2020 [cit. 2022-04-23]. Dostupné z: https://www.baeldung.com/cs/branch-and-bound

[36] ALSALIBI, Bisan, Marzieh BABAEIANJELODAR a Ibrahim VENKAT. A Comparative Study between the Nearest Neighbor and Genetic Algorithms: A revisit to the Traveling Salesman Problem [online]. 2012 [cit. 2022-04-23]. Dostupné z: https://www.researchgate.net/publication/258031702_A_Comparative_Study_between_the_Nearest_Neighbor_and_Genetic_Algorithms_A_revisit_to_the_Traveling_Salesman_Problem

[37] KOETHER, Robb T. The Traveling Salesman Problem Nearest-Neighbor Algorithm [online]. Hampden-Sydney College, 14.11.2016, s. 30 [cit. 2022-04-23]. Dostupné z: http://people.hsc.edu/faculty-staff/robbk/Math111/Lectures/Fall%202016/Lecture%2033%20-%20The%20Nearest-Neighbor%20Algorithm.pdf

[38] OBITKO, Marek. GENETIC ALGORITHMS [online]. 1998 [cit. 2022-04-23]. Dostupné z: https://www.obitko.com/tutorials/genetic-algorithms/index.php

[39] MITCHELL, Melanie. An Introduction to Genetic Algorithms. Cambridge, Massachusetts • London, England: A Bradford Book The MIT Press, 1999, 158 s.

[40] MEHBOOB, Usama, Junaid QADIR, Salman ALI a Athanasios VASILAKOS. Genetic algorithms in wireless networking: techniques, applications, and issues. Soft Computing [online]. 2016, 2016, 20(6), 2467-2501 [cit. 2022-04-23]. ISSN 1432-7643. Dostupné z: doi:10.1007/s00500-016-2070-9

[41] LUNER, Petr. Jemný úvod do genetických algoritmů [online]. [cit. 2022-04-23]. Dostupné z: https://cgg.mff.cuni.cz/~pepca/prg022/luner.html

[42] Introduction to Ant Colony Optimization [online]. 17.05.2020 [cit. 2022-04-23]. Dostupné z: https://www.geeksforgeeks.org/introduction-to-ant-colony-optimization/

[43] Team Ant Colony Optimization (TACO) [online]. [cit. 2022-04-23]. Dostupné z: https://github.com/itsmealves/taco

[44] GAMS: System Overview [online]. [cit. 2022-05-01]. Dostupné z: https://www.gams.com/products/gams/gams-language/

[45] SONI, Nitasha a Tapas KUMAR. Study of Various Crossover Operators in Genetic Algorithms. International Journal of Computer Science and Information Technologies [online]. Vol. 5 (6). 2014 [cit. 2022-05-19]. ISSN 0975-9646. Dostupné z: http://ijcsit.com/docs/Volume%205/vol5issue06/ijcsit2014050673.pdf

SEZNAM ZKRATEK

ACO	Anty Colony Algorithm
B&B	Branch-And-Bound
CVRP	Capacitated Vehicle Routing Problem
DCVRP	Distance-Constrained Capacitated Vehicle Routing Problem
DVRP	Distance-Constrained Vehicle Routing Problem EA
GA	Genetic Algorithm
GAMS	General Algebraic Modeling System
LIFO	Last In First Out
NN	Nearest Neighbour
NP	Non Polynomial
P	Polynomial
TSP	Travelling Salesman Problem
VRP	Vehicle Routing Problem
VRPB	Vehicle Routing Problem with Backhauls
VRPBTW	Vehicle Routing Problem with Backhauls with Time Windows
VRPPD	Vehicle Routing Problem with Pickup and Delivery
VRPPDTW	Vehicle Routing Problem with Pickup and Delivery with Time Windows
VRPTW	Vehicle Routing Problem with Time Windows

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

Název souboru
TSP_13_cities.gms
TSP_50_cities.gms
dataset_50_cities.py
dataset_cz.py
gams.py
tsp_ga.py
tsp_ga_50_cities.py
tsp_nn.py
tsp_nn_50_cities.py

PŘÍLOHY

dataset_cz.py
dataset_50_citiest.py
tsp_nn.py
tsp_nn_50_cities.py
gams.py
TSP_13_cities.gms
TSP_50_cities.gms
tsp_ga.py
tsp_ga_50_cities.py

Všechny přílohy jsou k dispozici na uvedeném odkazu:
https://github.com/vaclavpospisil/DP_2022_Okruzni_problemy