1. **Zadání:**

Úloha: Řešení problému obchodního cestujícího

Vstup: množina uzlů U reprezentujících body.

Výstup: nalezení nejkratší Hamiltonovské kružnice mezi těmito uzly.

Nad množinou U nalezněte nejkratší cestu, která vychází z libovolného uzlu, každý z uzlů navštíví pouze jedenkrát, a vrací se do uzlu výchozího. Využijte níže uvedené metody konstrukčních heuristik:

- Nearest Neighbor,
- Best Insertion.

Výsledky porovnejte s výstupem poskytovaným nástrojem Network Analyst v SW ArcMap.

Otestování proveďte nad dvěma zvolenými datasety, které by měly obsahovat alespoň 100 uzlů. Jako vstup použijte existující geografická data (např. města v ČR s více než 10 000 obyvateli, evropská letiště, ...), ohodnocení hran bude představovat vzdálenost mezi uzly (popř. vzdálenost měřenou po silnici); pro tyto účely použijte vhodný GIS.

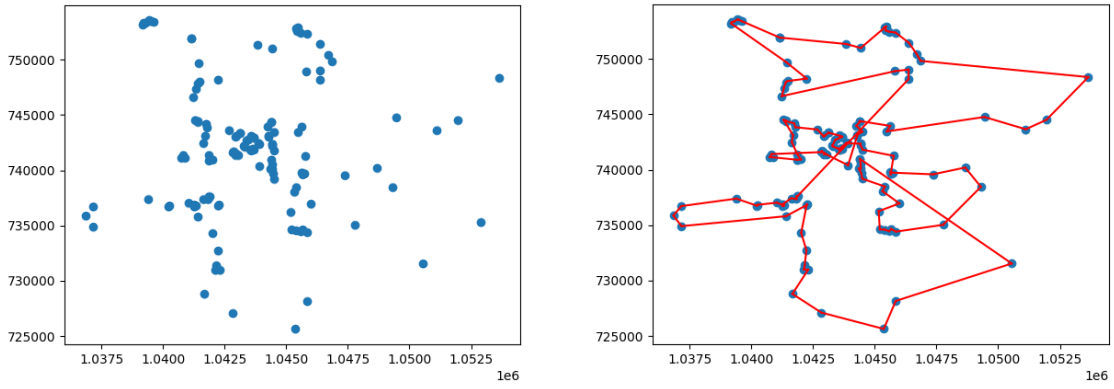
Výsledky s uvedením hodnot W, k, uspořádejte do přehledné tabulky (metodu Best Insertion nechte proběhnout alespoň 10x), a zhodnoťte je.

Pro implementaci obou konstrukčních heuristik použijte programovací jazyk Python, vizualizaci výstupů proveďte ve vhodné knihovně, např. `matplotlib`.

Čas zpracování: 3 týdny

2. Popis a rozbor problému

Problém obchodního cestujícího (anglicky Traveling Salesman Problem, zkracováno na TSP) je obtížný optimalizační problém v teorii grafů, který hledá nejkratší možnou cestu mezi zadanými body. Dá se si představit takto: Obchodní cestující chce podniknout cestu skrze zadaná města. Každé město navštíví právě jednou a vrátí se zpět do města, ze kterého vyrážel. V jakém pořadí má navštívit všechna města, aby náklady (to může být například vzdálenost či čas) byly co nejmenší?



Obrázek 1 - Možné řešení úlohy TSP

Zadání úlohy není složité pochopit, i přesto se ale jedná o jednu z nejtěžších úloh matematiky spadající do kategorie NP-úplných problémů. Za vyřešení těchto problémů je dokonce vypsaná odměna ve výši jednoho milionu dolarů od Clay Mathematics Institute. Úlohy v matematice jsou dělené na P a NP. Úlohy spadající do kategorie P jsou deterministické a lze je spočítat v polynomiálním čase. Úlohy spadající do kategorie NP (mohou být NP-těžké či NP-úplné) jsou nedeterministické, to znamená, že ze stejných vstupních algoritmů nevytvoří vždy stejný výsledek. Pro NP úlohy platí, že P nerovná se NP, tedy že pro ně neexistuje žádný polynomiální algoritmus, který by našel řešení, které je nejhůře k-násobkem optima. Pro NP-úplné problémy je jednoduché zkontrolovat, že se nejedná o optimální řešení, nicméně je nemožné optimální řešení najít. Této vlastnosti NP-úplných problémů se využívá například v šifrování.

Pro problém obchodního cestujícího to znamená, že jednoduché zkontrolovat, jestli jsme našli nejkratší cestu, ale je nemožné ji najít v polynomiálním čase. Cílem algoritmů vytvořených k řešení tohoto problému je nalézt optimální řešení rychle. Nicméně algoritmy, které naleznou řešení rychle, nenaleznou řešení optimální, a opačně algoritmy, které naleznou optimální řešení, ho nenaleznou dostatečně rychle. O algoritmech využívaných pro řešení problému TSP bude řeč dále.

První zmínka o problému obchodního cestujícího pochází z 19. stol., kdy byl problém formulovaný matematiky W. R. Hamiltonem a Thomasem Kirkmanem. Poprvé byl studovaný ve Vídni ve 30. letech 20. stol. Karlem Mengerem. Ten problém obchodního cestujícího definoval. V 50. a 60. letech se problém těšil větší a větší oblibě a byly vyhlášeny prémie pro toho, kdo problém vyřeší. Roku 1972 ukázal Richard M. Karp, že se jedná o NP-úplný problém.

TSP se využívá v mnoha oblastech. Například v dopravní obsluze, při plánování cesty dronu (při efektivním naplánování trasy je možné nalétnou více snímků kvůli šetření baterie). Nebo se dá využít například v orientačním běhu při závodě zvaném „scorelauf“, kdy má běžec na mapě několik kontrol, které musí proběhnout libovolně zvoleném pořadí, nicméně v co nejkratším čase. V zájmu běžce tedy je, aby vzdálenost, kterou bude muset překonat, byla minimální.

I přes to, že se jedná o jeden z nejvíce studovaných problémů současné matematiky, stále není známé řešení. Ale za posledních 50 let byly již nalezeny metody, které sice optimální řešení nenaleznou, ale alespoň jsou schopny se mu přiblížit.

2.1. Popis úlohy

Problém obchodního cestujícího je v teorii grafů definován takto: Nalezni nejkratší hamiltonovskou kružnici v ohodnoceném grafu.

Každý bod v grafu je nazvaný vrchol, vrcholy jsou propojené hranami, a každá hrana má hodnotu, tzv. *cost* (to může být například vzdálenost či čas). Hamiltonovská kružnice je taková kružnice, která obsahuje všechny vrcholy grafu (právě jednou). Délka hamiltonovské kružnice je součet hodnot hran mezi vrcholy hamiltonovské kružnice.

2.1.1. Definice

Graf G je uspořádaná dvojice G = (V,E), kde V je množina *vrcholů* a E je množina *hran* – množina (některých) dvouprvkových podmnožin množiny V.

Kružnice je graf, na *n* vrcholech (kde *n* > 2), které jsou spojeny po řadě *n* hranami do jediného cyklu tak, že každý vrchol je spojen s následujícím vrcholem a poslední vrchol také s prvním vrcholem.

Cesta je graf na *n* vrcholech, které jsou spojeny po řadě *n* – 1 hranami.

Hamiltonovská kružnice je taková kružnice, která prochází všemi vrcholy grafu právě jedenkrát.

Definice byly převzaty z učebnice Úvod do Teorie grafů od Petra Kováře (2011).

2.2. Řešení problému

Jak již bylo řečeno, problém TSP je optimalizační problém, kdy cílem je nalézt nejkratší hamiltonovskou kružnici grafu. Algoritmy využívané pro řešení tohoto problému by se dalo rozdělit do 3 kategorií – metoda brute-force, heuristické algoritmy a aproximační algoritmy.

2.2.1. Brute-force

Brute-force metoda byla navržena jako jedno z prvních řešení, kdy se začal o problém TSP zajímat Karl Menger v první polovině 20. století. Jedná se o metodu, která nejdříve nalezne všechna možná řešení, ta si postupně ukládá, vypočte délku cesty, ze kterých poté vybere tu nejkratší. Výsledkem této metody by tedy mělo být řešení optimální. Dá se ale využít pouze pro velmi malá data, jelikož čas potřebný na jeho výpočet se zvyšuje exponenciálně s rostoucím počtem vrcholů (pro obrázek nahlédněte do tabulky č. 1). Mějme n vrcholů, pak počet možných cest grafem je roven (n-1)!/2. Pro graf se třemi vrcholy existuje pouze jedno řešení, pro graf s šesti vrcholy již je to 60, a pro graf s 16 vrcholy už je počet možných řešení 643 837 184 000. Proto je využití této metody jako řešení problém TSP zcela vyloučeno.

počet vrcholů	počet kombinací (možných cest grafem)
3	1
4	3
5	12
6	60
10	181 440
50	3.0414093e+62
100	4.666311e+155

Tabulka 1 - počet možných cest grafem pro určitý počet vrcholů

2.2.2. Heuristické algoritmy

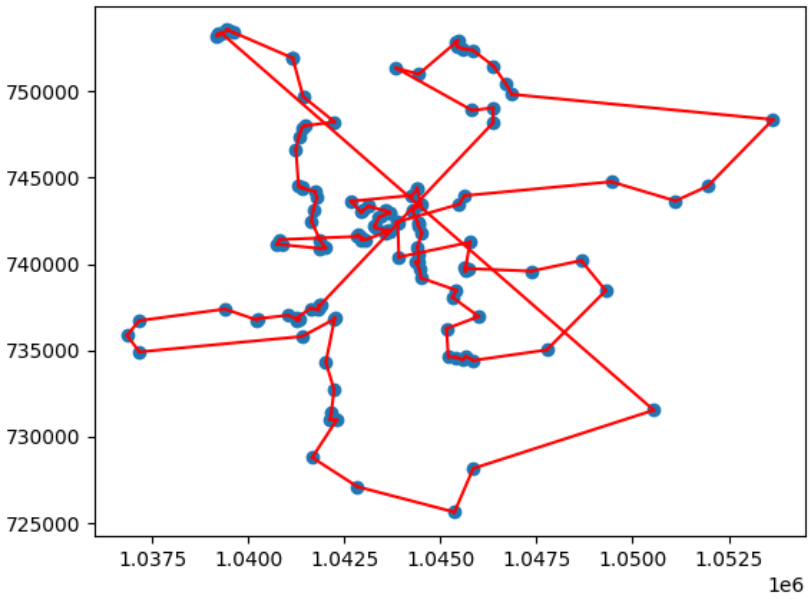
Další algoritmy, které mohou být využité pro řešení problému TSP, jsou heuristické algoritmy. Tyto algoritmy naleznou řešení v polynomiálním čase, nicméně negarantují nalezení optimálního řešení. Většinou je nalezeno nějaké lokální minimum, nikoli globální. Většinou tyto algoritmy dokáží najít řešení ve blízké optimu, nicméně je také možné, že tato metoda vyhodnotí jako řešení tu nejhorší možnou cestu. Příklady těchto heuristik může být Greedy strategie, Nearest Neighbor, Best Insertion (Nearest Insertion, Cheapest Insertion, Random Insertion, Farthest Insertion)

Greedy strategie spočte délky všech hran, a od hrany s nejnižší délkou postupně zařazuje vrcholy, mezi kterými se tato hrana nachází, do hamiltonovské kružnice, dokud nejsou všechny vrcholy propojené.

Další metodou je Nearest Neighbor, která je pravděpodobně jedna z nejjednodušších heuristik v řešení problému obchodního cestujícího. Klíčem je v každém kroku iterace přidat do hamiltonovské kružnice nejbližší město, aniž by nás zajímal celkový výsledek. Postup probíhá takto:

1. výběr náhodného vrcholu
2. dokud se v grafu nachází ještě nějaké nenavštívené vrcholy, nalézt nejbližšího souseda a „přemístit“ se tam
3. vrátit se zpět do výchozího vrcholu

Právě kvůli tomu, že se nalézá lokální minimum pro každý bod, je možné, že poté na závěr, kdy je již většina bodů přiřazena do hamiltonovské kružnice, se délka hamiltonovské kružnice může velmi prodloužit. Může docházet i ke křížení hran, což vede k neoptimálním výsledkům, a může dojít i k nalezení nejhorší možné cesty (tedy cesty, která je dvojnásobkem optima). Nicméně většinou dokáže tento algoritmus najít cestu v 25 % optima. Metoda Nearest Neighbor, pokud se začne pokaždé ze stejného vrcholu, je deterministická.



Obrázek 2 - Možné řešení problému TSP pomocí metody Nearest Neighbor

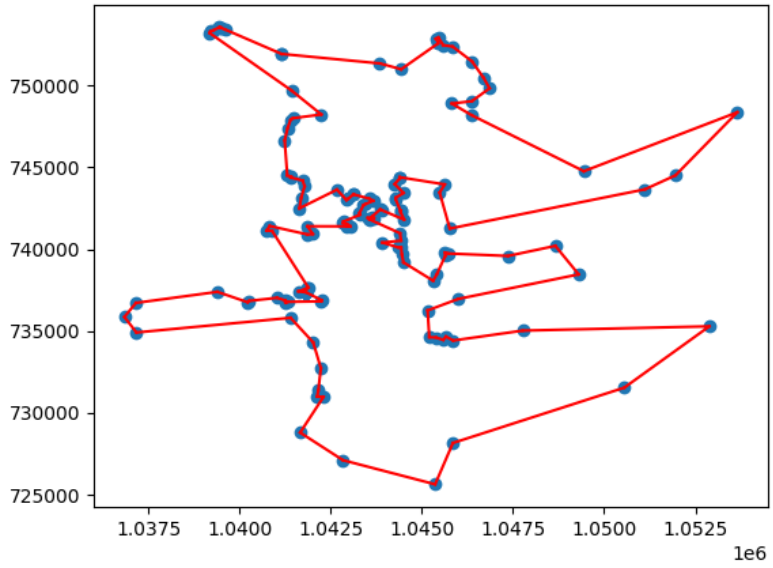
Další heuristickou metodou je strategie Best Insertion, která je účinnější než Nearest Neighbor, protože nepřidává vrchol na konec cesty (když zbývá přidat posledních pár bodů, nevznikne tam naráz mnohem

větší nárust délky hamiltonovské kružnice), nýbrž vkládá body do hamiltonovské kružnice tak, aby v každém kroku minimalizovala nárust vzdálenosti.

Postup probíhá takto:

1. Je vybrána počáteční hamiltonovská kružnice o třech bodech.
2. Dokud se v grafu nachází nezpracované vrcholy, vybere jeden, a vloží ho na to místo do hamiltonovské kružnice, kde se minimalizuje její vzdálenost.

Existuje více druhů tohoto algoritmu: Random Insertion (vybírá body, které vkládá do hamiltonovské kružnice náhodně), Nearest Insertion (vybere nejbližší bod ke kterémukoliv vrcholu hamiltonovské kružnice) a další.



Obrázek 3 - Řešení problému TSP pomocí metody Best Insertion (Random Insertion)

2.2.3. Aproximační algoritmy

Třetí kategorií algoritmů jsou aproximační algoritmy, které jsou schopné nalézt kvalitní řešení (v 1% optima) v polynomickém čase.

Jedním z příkladů je Christofidesův algoritmus, který garantuje, že řešení bude nejhůře 3/2 optima. Postup je takový, že je nejdříve vytvořen nejkratší možný strom, který obsahuje všechny body, a z toho je poté vytvořena kružnice vytvářením „zkratek" v bodech, ve kterých by obchodní cestující byl dvakrát.

2.2.4. Vylepšení cest

Existují 2-opt a 3-opt algoritmy, které výsledek optimalizují tak, že z cesty odstraní dvě hrany (2-opt), a cestu opět spojí (existuje pouze jeden způsob). Cestu ponechá takto spojenou, pokud je délka hamiltonovské kružnice po této úpravě kratší. 3-opt algoritmy fungují podobně, ale v jednom kroku odstraní 3 hrany.

2.2.5. Řešení problému TSP v programu ArcGIS Pro

Program ArcGIS Pro obsahuje rozšíření Network Analyst, s jehož pomocí je možné nalézt optimální cestu mezi body. Funkce Route Solver nejdříve vygeneruje matici hodnot všech hran a využívá

metaheuristickou strategii *Tabu-search* k nalezení nejlepšího možného řešení. Tato metoda vylepšuje cestu pomocí 2-opt metody. Nicméně, protože ArcGIS Pro je proprietární, přesnou implementaci těchto metod neposkytuje.

3. Řešení problému, popis algoritmu

Import využitých knihoven

Nejdříve byly importované všechny knihovny, které byly při řešení problému využité:

- Pandas (čtení .csv souboru souřadnic)
- Matplotlib.pyplot (vizualizace výsledků)
- random (pro náhodný výběr startovního bodu)
- math (využití odmocniny při výpočtu vzdálenosti)
- numpy (hledání minima v seznamu)

Čtení dat (funkce read_input(name))

Nejdříve byla vytvořena funkce na čtení dat pomocí knihovny Pandas. Funkce read_input má parametr name, což je název .csv souboru obsahující data. Výstupem funkce je seznam tuplů souřadnic x a y.

Výpočet vzdálenosti (funkce calc_distance(node, current_node)

Funkce calc_distance má dva parametry: node a current_node, tedy dva vrcholy. Výstupem je vzdálenost mezi těmito dvěma vrcholy.

Hledání nejbližšího souseda (funkce find_nearest_neighbor(current_node, unvisited_nodes))

Tato funkce nachází nejbližšího souseda k současnému bodu. Parametry jsou *current_node*, tedy současný vrchol, pro kterého hledáme nejbližšího souseda, a seznam *unvisited_nodes*, což je seznam zatím nenavštívených vrcholů. Pro každý bod v tomto seznamu je spočtena vzdálenost od současného vrcholu pomocí funkce *calc_distance()*. Tyto vzdálenosti jsou postupně ukládány do seznamu *distances*. Nejbližší soused je nalezen pomocí funkce *argmin()* knihovny *numpy*, která vrací index nejnižší hodnoty. Výstupem je vrchol (tedy x a y souřadnice vrcholu) s nejmenší vzdáleností od současného vrcholu (aby mohl být uložen do seznamu *path*), a jeho vzdálenost (aby mohla být přičtena k délce hamiltonovské kružnice).

Řešení TSP pomocí NN (funkce TSP_by_NN(nodes, plot))

Vstupem je seznam vrcholů (x a y souřadnic uložených v tuplech). Toto je hlavní funkce, která řeší problém TSP pomocí metody Nearest Neighbor. Nejdříve je vytvořen seznam s názvem „status“, do kterého jsou pro začátek uloženy hodnoty „N“ (jako new) pro všechny vrcholy, proto je vytvořen stejně dlouhý jako seznam vrcholů. Také je zde vytvořena proměnná path_length, což je délka hamiltonovské kružnice, která je inicializovaná hodnotou nula. Dále je také vytvořen seznam „path“, do kterého se postupně ukládají vrcholy v hamiltonovské kružnici, aby byla možná vizualizace výsledku.

Dalším krokem je inicializace hodnoty „current_node“, tedy současný vrchol, jako počátek hamiltonovské kružnice. Startovní vrchol je vybrán náhodně.

Dále je vytvořen while-cyklus, kde jako podmínka je uvedeno „dokud počet vrcholů v seznamu „status“ s hodnotou „N“ je větší než jedna, vykonej cyklus“. Cyklus je tedy vykonáván, dokud je počet nenavštívených vrcholů větší než 1. Na začátku cyklu je hodnota současného vrcholu (proměnná current_node) nastavena na „Z“, tedy označení pro již navštívený vrchol (změní hodnotu v seznamu „status“ na stejné pozici jako je „current_node“ v seznamu „nodes“ pomocí indexu). Poté je vytvořen

seznam nenavštívených vrcholů zvaný „unvisited_nodes“, kam jsou uloženy všechny zatím nenavštívené vrcholy (také pomocí indexů).

Je zavolaná funkce *find_nearest_neighbor*, která vrátí x a y souřadnice nejbližšího souseda a jeho vzdálenost od současného bodu. Souřadnice nejbližšího souseda jsou uloženy do seznamu *path* a jeho vzdálenost od současného bodu je přičtena k délce hamiltonovské kružnice. Tento vrchol (tedy nejbližší soused), se poté stane současným uzlem, pro který je dále vykonáván while-cyklus.

V komentáři je poté kód, který vizualizuje průběh řešení TSP pomocí metody NN bod po bodu.

Po skončení while cyklu je do seznamu *path* na konec uložený startovní bod. Rovněž jeho vzdálenost od předchozího bodu je přičtena do délky hamiltonovské kružnice.

Funkce vrací seznam bodů *path* a délku hamiltonovské kružnice *path_length*.

Vizualizační funkce plot_result(path)

Pomocí knihovny matplotlib je řešení problému TSP vizualizováno. Jako vstup tato funkce bere seznam *path*. Nejdříve je vytvořený scatterplot se všemi vstupními body, které jsou poté jeden za druhým spojeny čárou.

Řešení problému TSP pomocí metody Best Insertion (funkce TSP_by_best_insertion(nodes))

Podobně jako funkce *TSP_by_NN()* si tato funkce bere za vstup seznam vrcholů *nodes*. Na začátku jsou také vytvořeny seznamy *status* a *path*.

Nejprve je potřeba vytvořit počáteční hamiltonovskou kružnici o třech bodech, která je vytvořena pomocí while-cyklu. Tento cyklus probíhá dokud v seznamu *path* uložených méně než 4 body. Dokud tedy v hamiltonovské kružnici nejsou 3 body, vybírá je náhodně ze seznamu *nodes*. V cyklu je rovněž podmínka, že tento náhodně vybraný vrchol nesmí být v seznamu *status* označený jako již vybraný („Z“). Toto je udělané pomocí indexu. Poté je spočítaná délka této počáteční hamiltonovské kružnice pomocí již vytvořené funkce *calc_distance()*.

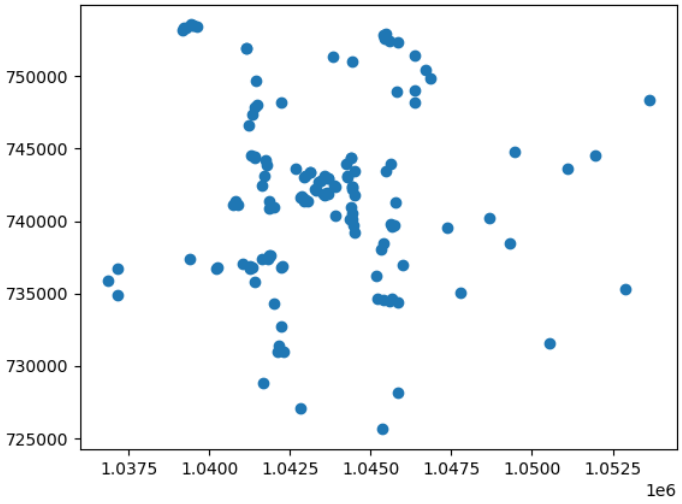
Další while-cyklus probíhá, stejně jako u NN, dokud je počet nenavštívených vrcholů větší než 1. Na začátku cyklu je vytvořen seznam s dosud nenavštívenými vrcholy, a vybrán náhodný vrchol, který je uložen do proměnné *random_node*. Rovněž je vytvořen prázdný seznam *distance changes*, který bude využit později v kódu. Dále je v while-cyklu for-cyklus, který prochází indexy v seznamu již navštívených vrcholů *path*. V tomto cyklu je vytvořena proměnná i2, do které je uložená hodnota o 1 vyšší, než v indexu i (protože toto bude umístění v seznamu, kam bude uložen vrchol, se kterým právě pracujeme, tedy *random_node*) i2 se rovná i + 1 pouze tehdy, pokud se index i nerovná poslední hodnotě v seznamu. V tomto případě je hodnota i2 rovna 0 a hodnota je uložena na začátek seznamu (tedy mezi první a poslední vrchol). Dále je spočtena hodnota *distance_change*, což je hodnota, o kterou by se prodloužila vzdálenost mezi body i a i2, pokud by byl vrchol *random_node* vložen právě mezi tyto dva vrcholy. Tato hodnota je poté uložena do seznamu *distance_changes.* For-cyklus tedy *random_node* zařazuje postupně mezi všechny dvojice po sobě jdoucích vrcholů a zaznamenává, o kolik by se prodloužila hodnota proměnné *path_length*.

Podobně jako ve funkci *TSP_by_NN()* je nalezen index minimální hodnoty pomocí funkce *argmin()* v knihovně numpy. *Random_node* je zařazen na místo v seznamu, kde bylo vyhodnoceno, že nejméně zvýší hodnotu *path_length*, ke které je poté přičtena vzdálenost, o kterou se prodloužila, a status vrcholu je změněn na „Z“.

Funkce vrací seznam bodů *path* a délku hamiltonovské kružnice *path_length*.

4. Experiments and results: vstupní datasety, dosažené výsledky, tabulky

Prvním vstupním datasetem byla bodová vrstva veřejných WC v Praze, která je volně dostupná na pražském geoportálu (https://www.geoportalpraha.cz/). Tato vrstva obsahuje 135 veřejných WC na území Prahy. Rozmístění bodů je zaneseno v obrázku č. 4.



Obrázek 4 - Veřejná WC v Praze

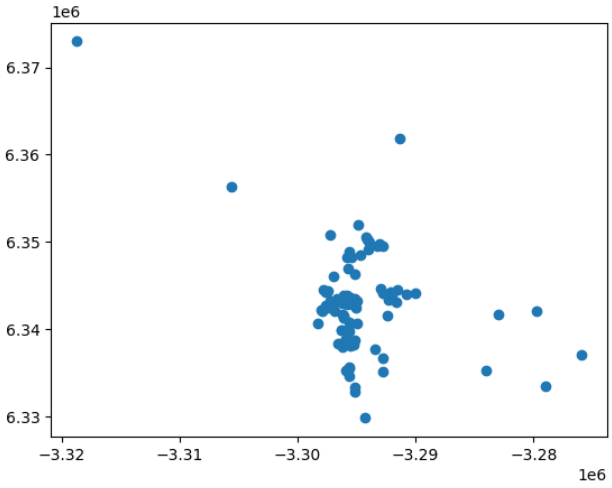
pokus	výsledek
1	151749.40345026835
2	145748.31314547386
3	134636.20773223924
4	131806.15484685588
5	132864.73317656966
6	132982.3880270161
7	131694.76801361862
8	133689.57250020572
9	127265.15876497068
10	131802.35114331284
průměr	**135423.9050800531**

Tabulka 2 - Výsledky metody Best Insertion na datech Veřejná WC v Praze

	ArcGIS Network Analyst	Nearest Neighbor	Best Insertion
délka H. kružnice	130665.302196	148729.78977757337	135423.9050800531

Tabulka 3 - porovnání metod na datech Veřejná WC v Praze

Druhým datasetem byly místa, kde se dá pořídit pivo z rukodělného pivovaru v New York City (restaurace, pivnice, obchody, bary). Dataset obsahuje 99 bodů, proto musela být do dat přidána dvě falešná místa. Data jsou volně dostupná zde: http://www.poi-factory.com/node/2312. Body jsou vizualizovány na obrázku č. 5.



Obrázek 5- NYC Craft Beers

pokus	výsledek
1	178821.4706277796
2	168241.5636369043
3	186520.41267814394
4	180863.08216474822
5	148493.3582688192
6	166205.33004289295
7	178550.47769934565
8	197155.52760640142
9	177672.85764725957
10	184473.35086993608
průměr	**176699.74312422308**

Tabulka 4 - Porovnání metody Best Insertion na datech NYC Craft Beers

	ArcGIS Network Analyst	Nearest Neighbor	Best Insertion
délka H. kružnice	200220.714444	220514.6185562381	176699.74312422308

Tabulka 5 - Porovnání metod na datech NYC Craft Beers

5. Závěr, možné či neřešené problémy, náměty na vylepšení

V této úloze byly testovány dva heuristické algoritmy na řešení úlohy obchodního cestujícího – Nearest Neighbor a Best Insertion, které byly následně porovnány s výsledky v ArcGIS Network Analyst. Metody byly testovány na dvou vstupních datasetech – jeden o 138 bodech, druhý o 101 bodech.

Podle očekávání vyšla v obou případech nejhůř metoda Nearest Neighbor. V případě datasetu Veřejná WC v Praze (138 bodů) dosáhl lepšího výsledky ArcGIS Pro Network Analyst (porovnáváno s průměrem algoritmu Best Insertion, nejlepší pokus Best Insertion měl lepší výsledek, než ArcGIS Pro Network Analyst). V případě druhého datasetu (NYC Craft Beer, 101 bodů), byl průměr výsledků Best Insertion lepší než ArcGIS Pro Network Analyst. Nicméně nejlepší z 10 pokusů Best Insertion byl vždy lepší než ArcGIS.

Výsledek metody Best Insertion je závislý na zadání počáteční kružnice. V tomto programu byla počáteční kružnice vybírána náhodně. Námětem na vylepšení je derandomizovat výběr počáteční hamiltonovské kružnice.

Problémem, na který jsem narazila, byly duplicitní hodnoty v datech, na kterých se program zacyklil. Předtím, než byl program spuštěn, byly v datech duplicitní hodnoty nalezeny pomocí následující funkce, a manuálně odstraněny. Dalším námětem na vylepšení kódu je mít toto přímo ošetřené v programu. Nicméně pro malé množství dat, které bylo v této úloze zpracováváno, bylo rychlejší hodnoty odstranit manuálně.

```
# kod pro nalezeni duplicitnich hodnot v datech

def find_duplicate(data):

    seen = set()
    duplicates = []

    for x in data:
        if x in seen:
            duplicates.append(x)
        else:
            seen.add(x)

    return duplicates
```

6. Seznam literatury

11 Animated Algorithms for the Traveling Salesman Problem (2019): STEM Lounge, https://stemlounge.com/animated-algorithms-for-the-traveling-salesman-problem/ (6. 1. 2022).

Algorithms used by the ArcGIS Network Analyst extension—ArcMap | Documentation (no date): https://desktop.arcgis.com/en/arcmap/latest/extensions/network-analyst/algorithms-used-by-network-analyst.htm#GUID-2A732430-E67E-4269-AB69-027A79EF9F75 (6. 1. 2022).

APPLEGATE, D., BIXBY, R., CHVATAL, V., COOK, W. (1998): On the Solution of Traveling Salesman Problems.

DOC. RNDR. ING. ŠEDA, M., Ph. D. (2001): Teorie Grafů. Vysoké učení technické v Brně, Brno.

Greedy algorithm - Encyclopedia of Mathematics (no date): https://encyclopediaofmath.org/index.php?title=Greedy_algorithm (6. 1. 2022).

KOVÁŘ, P. (2011): Úvod do Teorie grafů. Technická univerzita Ostrava, Západočeská univerzita v Plzni.

NILSSON, C. (no date): Heuristics for the Traveling Salesman Problem. 6.

Problém obchodního cestujícího (no date): https://www.algoritmy.net/article/5407/Obchodni-cestujici (6. 1. 2022).

TSP Poster (no date): http://comopt.ifi.uni-heidelberg.de/projects/projekttspsir/tspposter.html (6. 1. 2022).

WEISSTEIN, E. W. (no date): Traveling Salesman Problem, https://mathworld.wolfram.com/TravelingSalesmanProblem.html (6. 1. 2022).