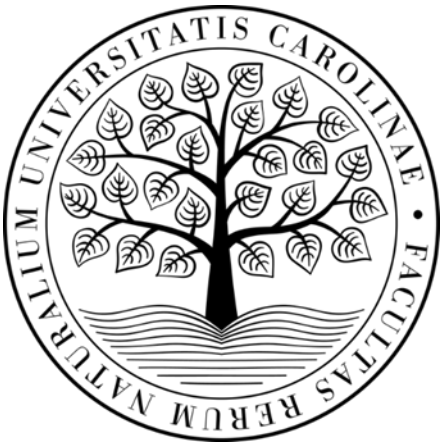
Univerzita Karlova

Přírodovědecká fakulta

Studijní program: Geoinformatika, kartografie a dálkový průzkum Země



Karolína Drápelová, Anna Wildová

1. ročník studijního programu Geoinformatika, kartografie a dálkový průzkum Země

Nejkratší cesta grafem

Geoinformatika

Akademický rok 2025/2026

Praha, 2026

1) Zadání

Implementujte Dijkstra algoritmus pro nalezení nejkratší cesty mezi dvěma uzly grafu. Vstupní data budou představována silniční sítí doplněnou vybranými sídly (alespoň 100 uzlů). S využitím přiloženého skriptu konvertujte podkladová data do grafové reprezentace. Otestujte různé varianty volby ohodnocení hran grafu tak, aby nalezená cesta měla:

a. nejkratší Euklidovskou vzdálenost
b. nejmenší transportní čas (2 varianty bez/se zohledněním klikatosti komunikací)

Pro druhou variantu optimální cesty navrhněte také vhodnou metriku, která zohledňuje rozdílnou dobu jízdy na rozdílných typech komunikací dle jejich návrhové rychlost v a klikatosti k.

Ve vybraném GIS konvertujte podkladová data do grafové reprezentace představované neorientovaným grafem. Pro druhou variantu optimální cesty navrhněte vhodnou metriku, která zohledňuje rozdílnou dobu jízdy na různých typech komunikací.

Každou z variant otestujte pro dvě různé cesty tvořené alespoň 20 uzly. Výsledky umístěte do tabulky, vlastní cesty vizualizujte. Dosažené výsledky porovnejte s vybraným navigačním SW (alespoň 3).

2) Bonusové úlohy

Nalezení nejmenší kostry – Jarníkův algoritmus:

Cílem algoritmu je najít minimální kostru grafu (MST – Minimum Spanning Tree). To znamená propojit všechny vrcholy (uzly) v grafu tak, aby:

- existovala cesta mezi každými dvěma body (graf byl souvislý),
- nevznikly žádné cykly,
- součet vah hran byl co nejmenší.

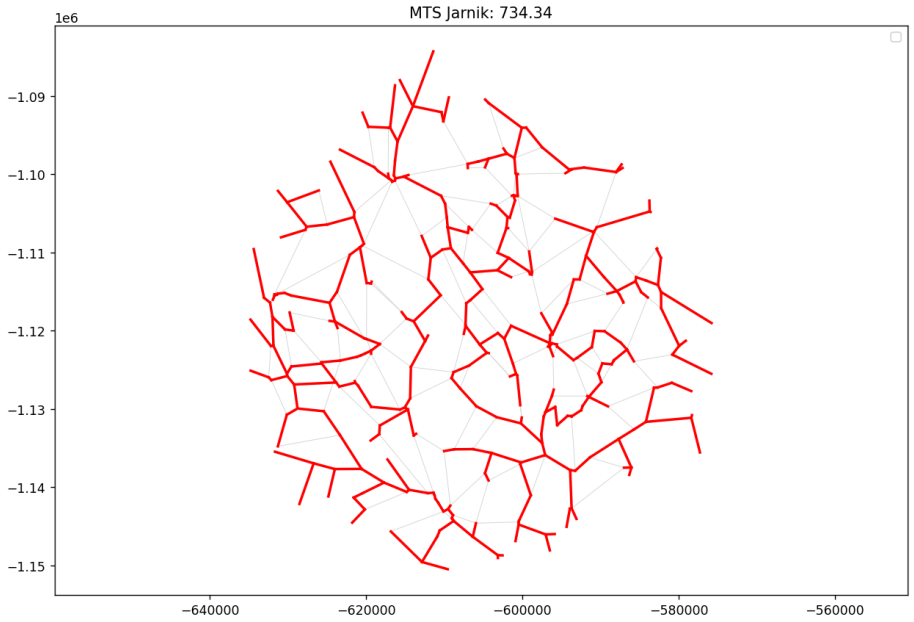
Implementace:

Vybereme startovní vrchol (start_node=0). Vytvoříme si dvě množiny:

- Visited: Uzly, které byly připojeny ke kostře,
- Priority Queue: Navštívené hrany, které vedou k nenavštíveným hranám seřazené podle váhy (ceny).

Dokud nemáme všechny vrcholy, podíváme se do prioritní fronty (pq) a vybereme hranu s nejmenší váhou. Pokud tato hrana vede do uzlu, který jsme už zpracovali (u in visited), zahodíme ji (protože by vytvořila cyklus). Pokud vede do nového uzlu, přijmeme ji. Jakmile připojíme nový uzel u, musíme se podívat na všechny jeho sousedy. Pokud soused ještě není v naší kostře, přidáme hranu k němu do prioritní fronty. Kroky 2 a 3 se opakují, dokud nejsou navštíveny všechny uzly (len(visited) < num_nodes) nebo dokud nedojdou dostupné hrany.

Na obrázku 1 je vyobrazena minimální kostra po spuštění skriptu. Celkově je v minimální kostře 343 hran a celková cena (váha) je 734,34 (km).



Obrázek 1: Vizualizace minimální kostry s využitím Jarníkova algoritmu.

Nalezení nejmenší kostry – Borůvkův algoritmus:

Borůvkův algoritmus pracuje ve fázích. Na začátku je každý bod v grafu sám za sebe (je to samostatná komponenta). V každé fázi se každá komponenta (skupina bodů) "rozhlédne" a najde nejlevnější hranu, která vede pryč k jiné komponentě.Všechny tyto nejlevnější hrany se najednou přidají do kostry. Komponenty se spojí. Opakuje se to, dokud nezbude jen jedna velká komponenta (celý graf je propojen).

Implementace:

Zavedení datové struktury Union-Find, která je klíčová pro implementaci Borůvkova algoritmu. Tato struktura řeší správu množin komponent, a zamezuje tedy tomu, aby se musel prohledávat celý graf

- parent: Slovník, který pro každý uzel říká, kdo je jeho rodič.
- Find (i): Funkce, která zjistí, do jaké skupiny uzel patří (najde hlavního reprezentanta skupiny).
- union (i, j): Funkce, která spojí dvě skupiny dohromady.

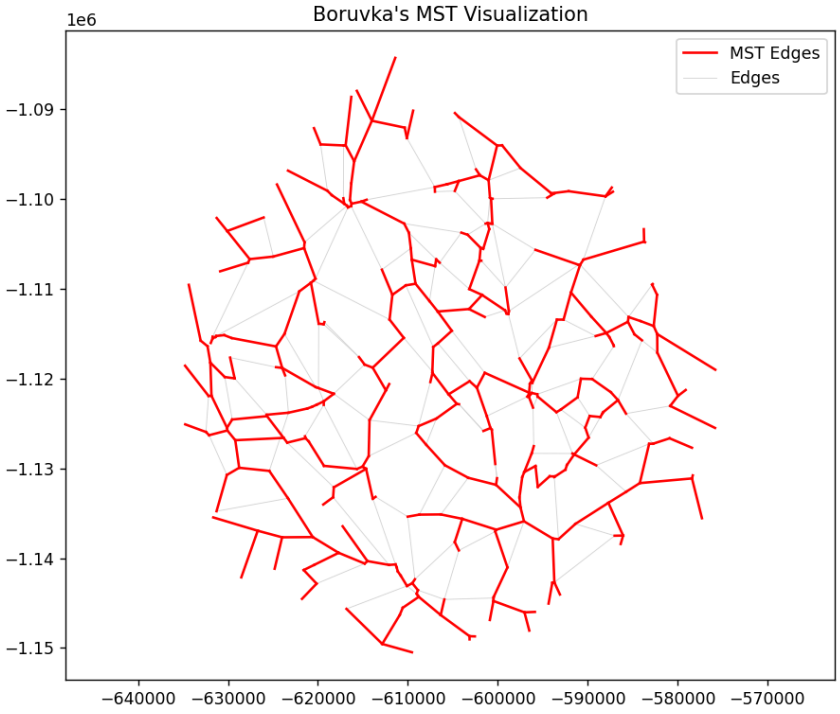
Algoritmus běží, dokud máme více než jednu skupinu (num_components > 1). V každém průchodu smyčkou se počet skupin obvykle sníží na polovinu. Procházíme úplně všechny hrany v grafu. Pro každou hranu (u, v) zjistíme, do jakých komponent uzly patří (root_u, root_v). Pokud patří do stejné komponenty (root_u == root_v), hranu ignorujeme (vytvořila by cyklus uvnitř už propojené části). Pokud jsou z různých komponent, zkontrolujeme, zda je tato hrana levnější než ta, kterou jsme pro danou komponentu (root_u) našli doposud (obrázek 2). Jakmile máme pro každou komponentu vybranou tu nejlepší hranu, projdeme seznam cheapest a spojíme je pomocí union. Tím se z mnoha malých částí stávají větší celky.

```
for u in G:
    for v, data in G[u].items():
        root_u = find(u)
        root_v = find(v)

        if root_u != root_v:
            w = data['weight']
            #check if this edge is the cheapest for component root_u
            if root_u not in cheapest or w < cheapest[root_u][0]:
                cheapest[root_u] = (w, u, v, data['id'])
```

Obrázek 2:Ukázka kódu Borůvkova algoritmu.

Na obrázku 3 je vyobrazena minimální kostra po spuštění skriptu. Stejně jako u Jarníkova algoritmu je v minimální kostře 343 hran a celková cena (váha) je 734,34 (km).



Obrázek 3: Vizualizace minimální kostry s využitím Borůvkova algoritmu.

3) Teoretická část

Grafový algoritmus se využívá při hledání řešení problému vyskytujících se v grafech, který je tvořen sítí uzlů a hran s různým ohodnocením. Nejčastější problémem s řešením grafových algoritmů je hledání nejkratší cesty mezi dvěma uzly. Graf rozlišujeme na orientovaný a neorientovaný.

V této práci má být graf zjednodušený model obcí a silničních komunikací. Graf bude nazván proměnnou G v naši implementaci. Uzly tohoto grafu budou představovat sídla a silniční komunikace mezi sídly budou znázorněny pomocí hran. Pro nalezení nejkratší cesty mezi dvěma sídly se tedy bude hledat komunikace, které mají nejnižší ohodnocení. Ohodnocení se dá buďto vnímat jako nejkratší euklidovská vzdálenost nebo nejmenší čas přesunu mezi dvěma sídly, jelikož i v reálném světě nejkratší vzdálenost mezi dvěma určitými místy se ne nutně shoduje s cestou s nejkratším transportním časem. Čas transportu může být ovlivněn několika faktory, např. omezenou rychlostí na trase, sklonem a klikatostí terénu.

V této práci pro nalezení nejkratší cesty s nejkratším časem transportu je doba trasy počítána pomocí vzorce:

$$t(u,v) = \frac{s\,(u,v)}{v}$$

kde t(u,v) je čas překonání trasy z počátečního (u) do koncového uzlu (v) v hodinách, s(u,v) je vzdálenost po hranách [km] a v je návrhová rychlost dle třídy komunikace. Návrhová rychlost byla určena dle kategorie ČSN 73 6110, pro místní komunikace se jedná o projektování místních komunikací a Zákona o PK 13/1997. 4., 5. a 6. třída silnic se řadí obvykle do místních komunikací. Na těchto komunikacích nebude rychlost vyšší než 80 km/h, touto rychlostí se jezdí pouze v okrajových částech nebo na přechodech do vyšších kategorií větších měst. Jelikož na vybraném území se velké město nevyskytuje, tak pro 4. a 5. třídu byla definovaná maximální povolená rychlost 60 km/h a 40 km/h (4. třída má technické parametry odpovídající vyšším rychlostem, 5. třída jsou užší komunikace v obcích, v obydlených oblastech apod.). Pro 6. třídu silnic byla uvažovaná maximální povolená rychlost 30 km/h, jedná se o velmi úzké, nezřídka o nepřehledné a neznačené komunikace.

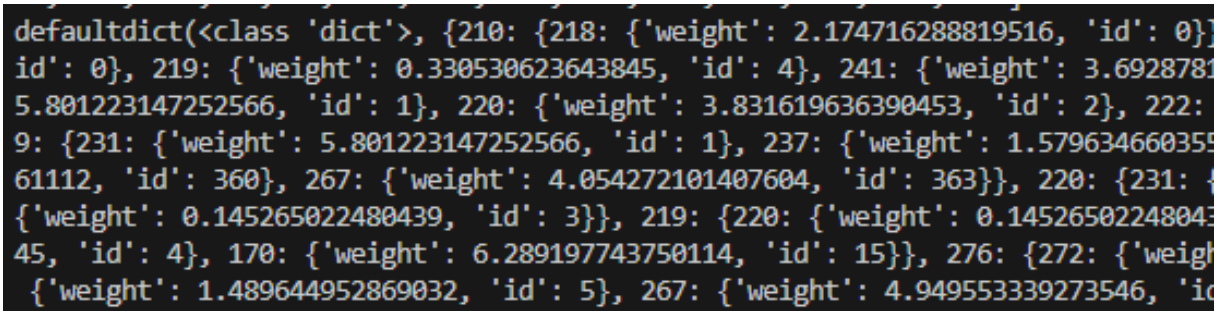
Dijkstra (podle autora Edsger W. Dijkstra, 1956) algoritmus je jedním z nejčastěji používaných algoritmů pro řešení problému hledání nejkratší cesty mezi dvěma uzly. Tento algoritmus patří do algoritmů hladových, pro nalezení nejkratší cesty algoritmus najde pro každý uzel jeho nejkratší cestu k sousednímu uzlu, tj, lokální minimum, poté sestaví z nejkratších cest tu nejkratší trasu mezi počátečním a koncovým uzlem, tím najde minimum globální (nejkratší cestu). Jelikož pro každý bod hledá všechny možné varianty, tak vedlejším výstupem je nalezení nejkratší vzdálenosti od všech bodů. Realizuje se opakovanou relaxací hrany (u, v). Heuristická optimalizace je zde úspěšná.

4) Implementace

Neorientovaný graf s Euklidovskou vzdáleností pro Dijkstra algoritmus:

Pro zpracování v pythonu byly využity knihovny *numpy* (sloučení seznamů a odstranění identických hodnot), *collections* (funkce defaultdict pro vytvoření složeného slovníku a vnořeného slovníku), *matplotlib* (pro vizualizaci grafových výstupů), *heapq* (vytvoření prioritní fronty). Pomocí funkcí loadEdges(), pointsToIDs(), edgesToGraph() naskriptované v python byl textový soubor převeden do formátu slovníku, kde jsou přítomny incidující hrany, klíč slovníku je ID uzlu a hodnota daného klíče je slovník sousedních uzlů. Tento vnořený slovník má dva klíče *weight* a *id* a k nim jsou jejich příslušné hodnoty. V tomto případě weight byla euklidovská vzdálenost (v km), význam tohoto slovníku je zobrazení trasy jako neorientovaný graf s kladným ohodnocením.

Pro hledání nejkratší euklidovské vzdálenosti pomocí dijkstra algoritmu se nejprve vzdálenosti všech uzlů stanoví na nekonečnu. Použitím knihovny heapq se vytvořila prioritní fronta, do které je na začátku vložen počáteční uzel s vahou 0. Poté se pomocí cyklu se zpracovávají uzly z prioritní fronty, dokud se fronta nevyprázdní. Z fronty je vyjmut uzel ‘node’, hrany sousedící s tímto uzlem jsou postupně procházeny za podmínky, že vzdálenost incidujícího uzlu je od začátku. Algoritmus takto opakovaně hledá lokální minimum, po vyprázdnění všech možných možností tímto způsobem nalezne minimum globální. Pro vizualizaci byla použita knihovna *matplotlib*, kde pomocí cyklu for byl nejdříve zkonstruován síť uzlů a hran, dále byl pomocí for cyklu vizualizovaná nejkratší cesta mezi konkrétními 2 body. V obrázcích níže je vidět výsledek vizualizace pomocí knihovny *matplotlib.* Pro nalezení ukázek skriptů s použití python knihovny heapq byla použita AI Google Gemini Pro.



Obrázek 4: ukázka složeného slovníku, klíč je ID uzlu a poté ve vnořeném slovníku je klíč ID incidujících uzlů a hodnota je vzájemná euklidovská vzdálenost (=’weight‘)

Neorientovaný graf s nejmenším transportním časem pro Dijkstra algoritmus:

V případě hledání nejkratšího transportního času se využila podmínka a třída silnice. Pro danou třídu silnice jsme volili maximální povolenou rychlost (vel) následovně, klikatost tedy nebyla použita:

1) třída 3 měla vel rovno 90 km/h
2) třída 4 měla vel rovno 60 km/h
3) třída 5 měla vel rovno 40 km/h
4) třída 6 měla vmax rovno 30 km/h

```python
def edgesTravelTimeToGraph(L, PS, PE, C, IDs):
    GT = defaultdict(dict)
    for i in range(len(PS)):
        u = D[PS[i]]
        v = D[PE[i]]
        l = L[i]
        c = C[i]
        if c == 6:
            vel = 30
        if c == 5:
            vel = 40
        if c == 4:
            vel = 60
        if c == 3:
            vel = 90
        travel_time_mins = (l/1000)/vel*60
        edge_id = IDs[i]
        length_data = {'travel time [mins]': travel_time_mins,'id': edge_id}
        GT[u][v] = length_data
        GT[v][u] = length_data

    return GT
```

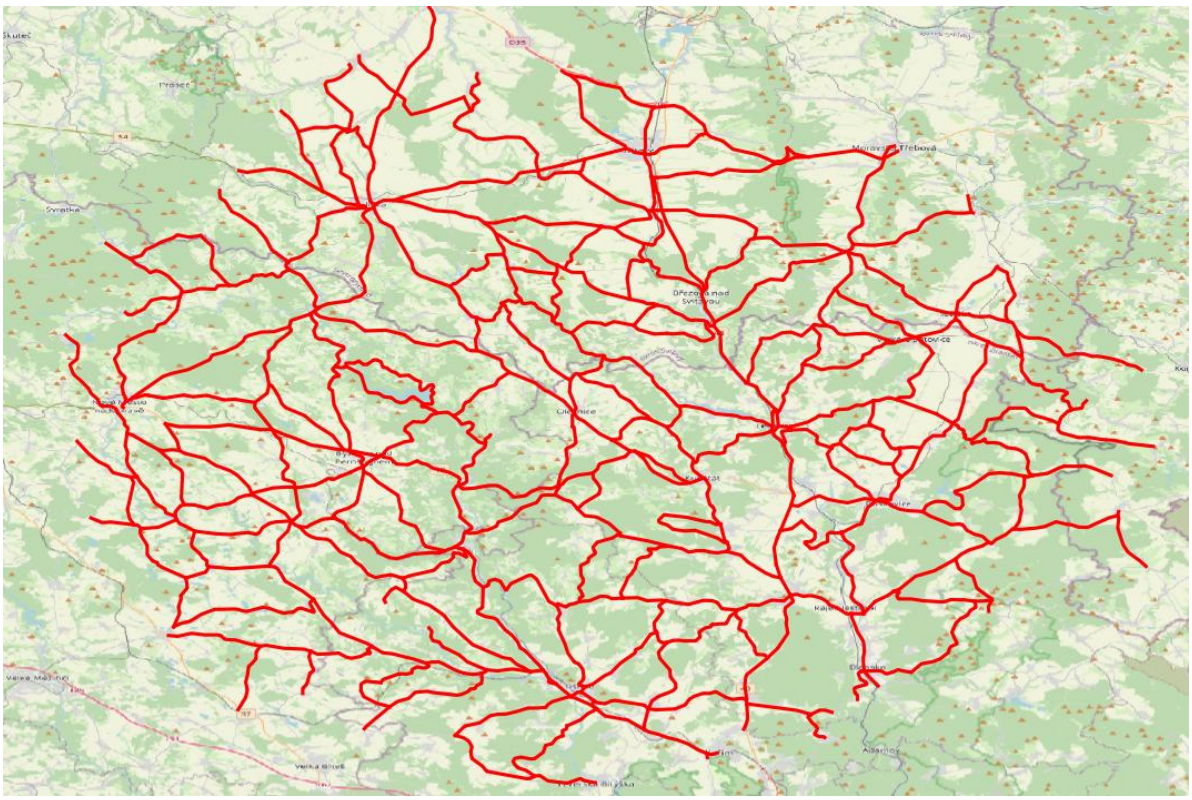
Obrázek 5: Ukázka kódu pro vytvoření přibližného transportního času pomocí návrhových rychlostí dle známých tříd komunikací.

Pro odhad transportního času byla vytvořena funkce edgesTravelTimeToGraph, jeho definice je vidět v obrázku 5. Nejprve byla pozměněna funkce loadEdges a pointsToIDs, jejich vstup byl textový soubor, kde byla obsažena informace o třídě komunikace v atributu.

5) Data

Pro vybrání silnic byly využita data ArcData. Po importu liniového shapefilu se dále určila požadovaná oblast, která se nacházela mezi Novým Městem na Moravě a Boskovicemi. V obrázku níže je printscreen vybraných bodů a linií otevřený v projektu SW QGIS.

Pro vytvoření jednotlivých uzlů se použil nástroj Field Calculator v atributové tabulce. Byly vytvořeny nové pole X1, Y1 (souřadnice počátečního uzlu silnice) X2, Y2 (souřadnice koncového uzlu silnice). Pro výpočet X1 a Y1 se pro výpočet pole použil výraz start_point($geometry), pro X2 a Y2 byl výraz end_point($geometry). Tyto výrazy se použily pro výpočet všech prvků v atributové tabulce, tudíž se tak určily počáteční a koncové body všech komunikací. Pro získání nejkratšího transportního času byl vytvořen další soubor, která obsahovala informaci o třídě komunikací v atributu s názvem ‘TRIDA’.



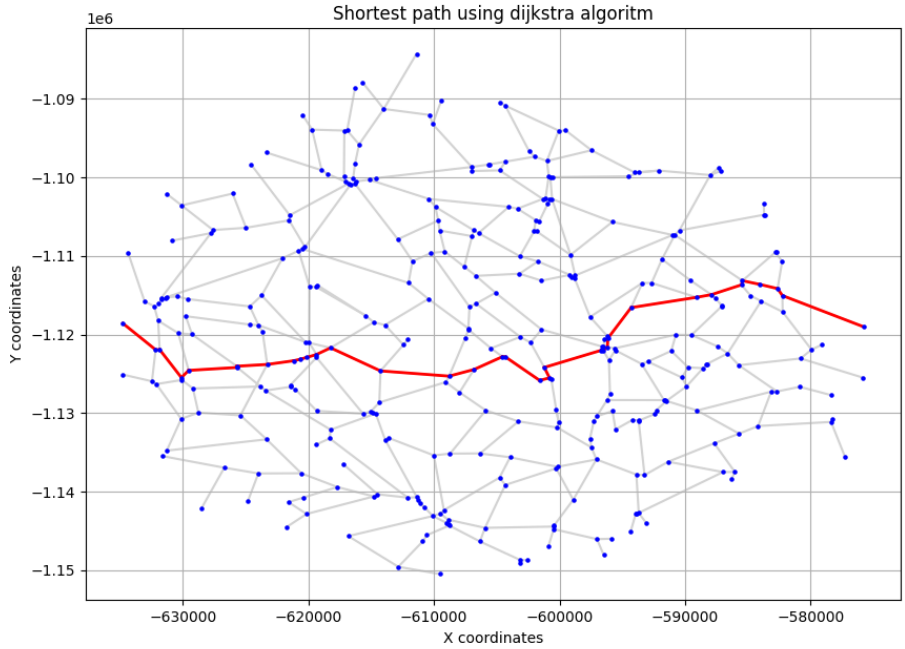
Obrázek 6: Vybraná silniční síť vyobrazená v SW QGIS.

6) Přehled výsledků

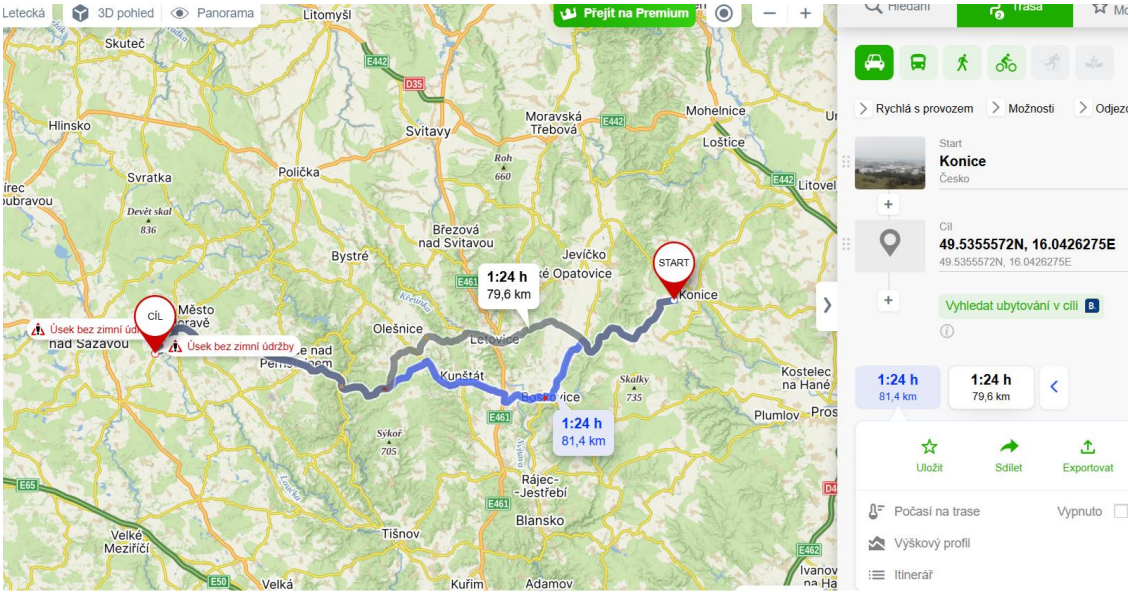
Pro srovnání výsledků mezi používaným SW a naší implementací se použil navigační SW plánování tras v Mapy.cz, kde pro stejné dva uzly vyšly dvě varianty tras s délkami 79,6 km a 81,4 km, ve srovnání v naší implementaci dijkstry trasa měřila 82,546 km. V druhém případě trasy z Městečko Trnávka do Katov náš algoritmus spočítal nejkratší trasu 71,9 km, Mapy.cz pak nabízí dvě trasy - 79,1 km a 78,1 km. Transportní časy jsou uvedeny v následující tabulce.

Dijkstra implementace	SW Mapy.cz
trasa: 82,546 km transportní čas: 1 h 25 min (145 minut)	trasa: 79,6 km nebo 81,4 km doba trasy: 1 h 24 minut pro obě varianty
trasa: 71,9 km transportní čas: 1 h 13 min (88 min)	trasa: 79,1 km nebo 78,1 km transportní čas: 1h 21 min nebo 1h 27 min

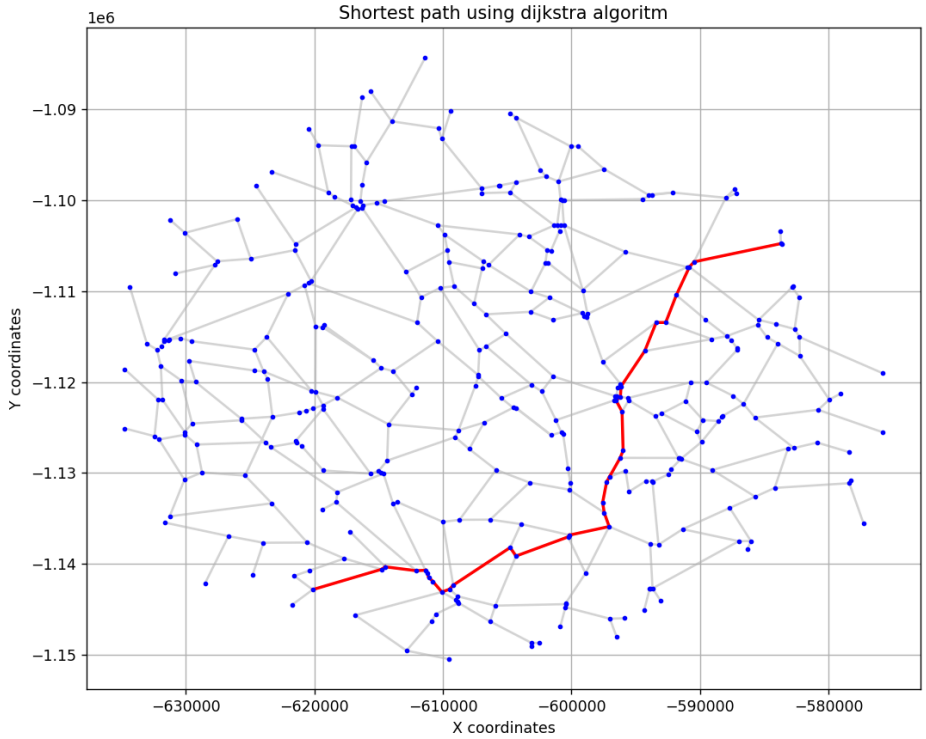
Tabulka 1. Srovnání výsledků délky tras a transportních časů implementaci Dijkstry a Map.cz.



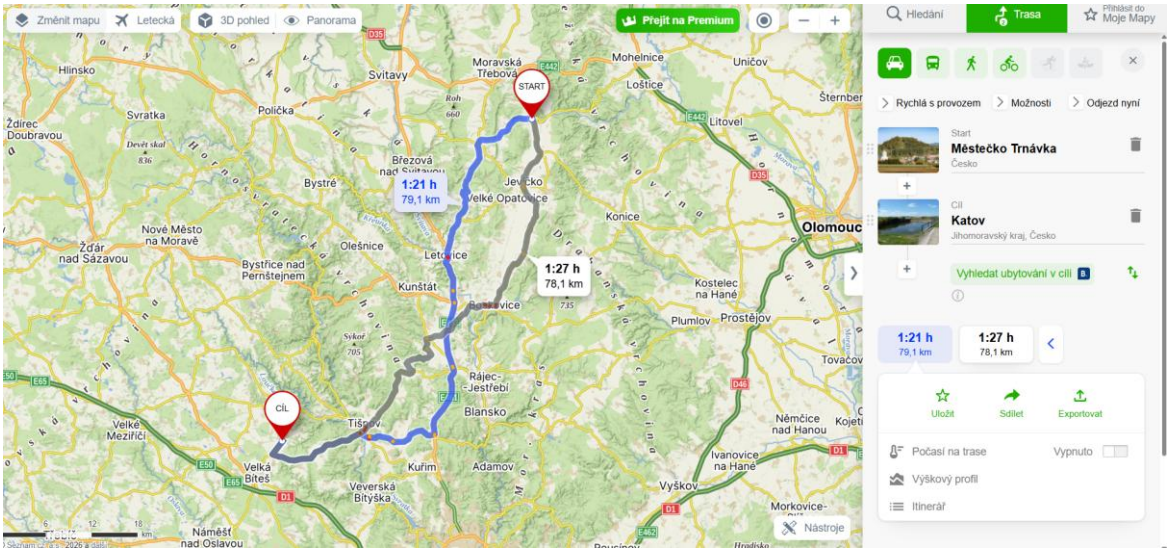
Obrázek 7: Vizualizace grafu a nejkratší cesty mezi dvěma uzly s použitím Dijsktra algoritmu (Konice – Hlinné).



Obrázek 8: Printscreen z Map.cz a návrh trasy mezi dvěma místy (Konice – Hlinné).



Obrázek 9: Vizualizace grafu a nejkratší cesty mezi dvěma uzly s použitím Dijkstra algoritmu (Městečko Trnávka – Katov).



Obrázek 10: Printscreen z [Map.cz](Map.cz) a návrh trasy mezi dvěma místy (Městečko Trnávka – Katov).

7) Závěr

S porovnáním navigačním SW Mapy.cz se podařilo pomocí Dijkstry algoritmu odhadnout nejkratší vzdálenosti mezi 2 uzly ve srovnání v naší implementaci dijkstry trasa v jednom případě rozdíl činil 1 km což mohlo být způsobeno vlivem měřítka a nižší podrobnosti dat stažené z ArcDat. V druhém případě trasy z Městečko Trnávka do Katov je rozdíl větší, mohlo to být způsobeno, že daná oblast se nachází na vlnitém terénu. V neposlední řadě v navigačním SW jsou informace o aktuálních uzavírkách, které v naší implementaci nebyly zahrnuty.

Bohužel se nepovedla implementace klikatosti, která byla nahrazena implementací tříd komunikací ve vstupních datech. Sice z těchto tříd vycházejí technické parametry a poté i návrhové rychlosti (viz v Kategorie pozemních komunikací dle ČSN), ale má to velké nedostatky, doporučení dle ČSN uvádějí i velké rozptyly pro danou třídu (např. třída 4 návrhová rychlost 50 až 80 km/h), takže je zde nutná znalost daného území (nachází se na daném území velké město, kde je komunikace pro motorová vozidla s maximální rychlostí 80 km/h?). Tento postup se nedá aplikovat v případě neznámého místa, kudy trasa vede.

8) Seznam literatury

ARCDATA PRAHA (2022): Arc ČR 500, 4.1. www.arcdata.cz

Bayer, T. (2025): Grafové algoritmy, úvod.
https://web.natur.cuni.cz/~bayertom/images/courses/Geoinf/geoinf_grafy1.pdf

Bayer, T. (2025): Nejkratší cesty v grafu.
https://web.natur.cuni.cz/~bayertom/images/courses/Geoinf/geoinf_grafy2.pdf

Mareš M., Valla, T. (2017): Průvodce labyrintem algoritmům, CZ NIC.

Observatoř bezpečnosti silničního provozu (2007): Kategorie pozemních komunikací dle České technické normy (ČSN) www.czrso.cz

W3schools (2025): DSA Dijkstra's Algorithm
https://www.w3schools.com/dsa/dsa_algo_graphs_dijkstra.php