UNIVERZITA KARLOVA
Přírodovědecká fakulta
Katedra aplikované geoinformatiky a kartografie

Studijní program:
Geoinformatika, kartografie a dálkový průzkum Země

Jakub Fuchsig, Šimon Kohout

Nejkratší cesta grafem

Úkol č. 3

Ústí nad Labem, Zákupy 2025

Úloha 3: Nejkratší cesta grafem

Implementujte Dijkstra algoritmus pro nalezení nejkratší cesty mezi dvěma zadanými 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 *w* hran grafu tak, aby nalezená cesta měla:

- nejkratší Eukleidovskou vzdálenost,
- 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 různých typech komunikací dle jejich návrhové rychlosti *v* a klikatosti κ, příkladem může být

$$t(u,v) = \kappa\left(\frac{l(u,v)}{v(u,v)}\right) \tag{1.1}$$

Pro výpočet κ využijte poměr délky polylinie *l* představující diskrétní křivku a vzdálenosti *s* koncových bodů polylinie

$$\kappa = \frac{l(u,v)}{s(u,v)} \tag{1.2}$$

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).

Krok	**Hodnocení**
Dijsktra algoritmus.	20b
Návrh jiného ohodnocení hran zohledňující křivolakost silniční sítě.	*+5b*
Zohlednění vlivu DMT.	*+15b*
Nalezení nejkratších cest mezi všemi dvojicemi uzlů.	*+15b*
Nalezení minimální kostry Borůvka/Kruskal.	*+15b*
Nalezení minimální kostry Jarník/Prime.	*+15b*
Využití heuristiky Weighted Union	*+5b*
Využití heuristiky Path Compression	*+5b*
Max celkem:	**95b**

Úvod

Cílem práce byla implementace Dijkstrova algoritmu pro vyhledání nejkratší cesty v silniční síti. Aplikace měla umožnit uživateli výběr počátečního a koncového bodu (města - obce) a volbu kritéria pro výpočet trasy. Bylo požadováno otestovat tři varianty ohodnocení hran: eukleidovskou vzdálenost, transportní čas a transportní čas se zohledněním klikatosti trasy.

Teoretická část

Pro hledání optimální cesty v grafu byl zvolen Dijkstrův algoritmus. Silniční síť je v tomto modelu reprezentována jako graf, kde křižovatky a konce silnic tvoří uzly a jednotlivé úseky silnic představují hrany s definovanou váhou.

Řešení topologie: Při přípravě dat byl identifikován problém s nekonzistencí souřadnic mezi datovou sadou měst (body) a uzly silniční sítě. Ačkoliv by teoreticky měly souřadnice odpovídat, při exportu dat docházelo k odchylkám, kvůli kterým nebylo možné města v grafu přesně lokalizovat.

Tento nesoulad jsme ošetřili zavedením prostorové tolerance. Algoritmus nevyžaduje absolutní shodu souřadnic, ale vyhledává nejbližší uzel v definovaném okruhu (1 metr). Pokud se v této vzdálenosti uzel nachází, je k němu město automaticky přiřazeno.

Výpočet vah a klikatosti:

Pro variantu zohledňující klikatost byl aplikován vzorec využívající poměr délky linie a přímé vzdálenosti koncových bodů (*1.2*), čímž se uměle navýšil průjezdní čas u klikatých cest.

1. Data

Jako podkladová data byly využity vektorové mapy silniční sítě a bodovou vrstvu sídel. Pro zmenšení objemu dat byla data oříznuta zhruba po celém území ústeckého kraje.

- Zdroj dat: https://download.geofabrik.de/europe/czech-republic.html
- Formát: ESRI Shapefile (`.shp`).

1.1 Zpracování v ArcGIS

Před použitím v Pythonu bylo nutné data v prostředí ArcGIS Pro vyčistit a doplnit o potřebné atributy, které náš skript vyžaduje.

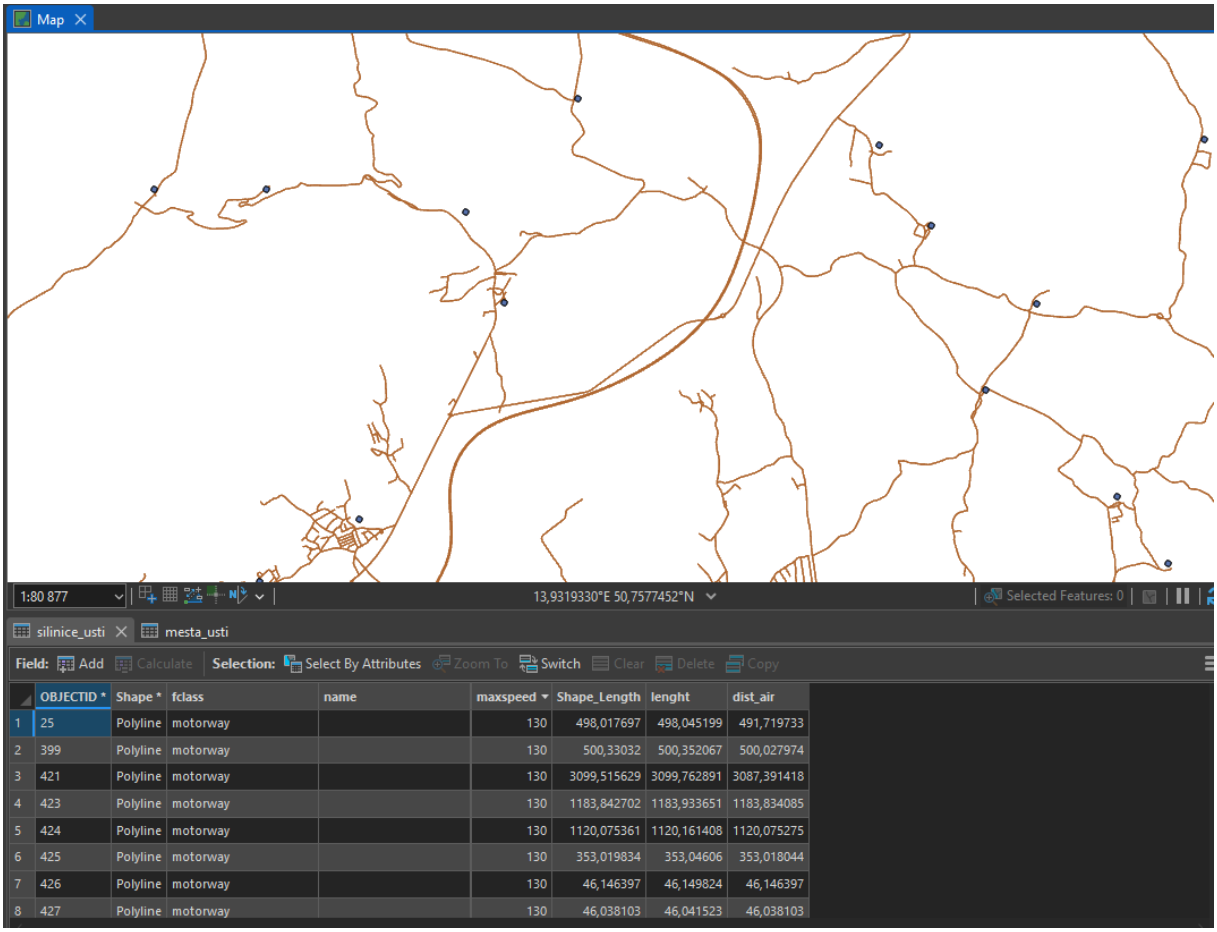
1.1.1 Úprava vrstev

V atributové tabulce silnic byly vytvořeny a vypočítány následující sloupce:

`lenght` (Délka hrany):
- Vypočítáno pomocí nástroje *Calculate Geometry*
- Jednotka: metry [m]

`dist_air` (Vzdušná vzdálenost):
- Vzdálenost mezi počátečním a koncovým bodem úseku vzdušnou čarou

`maxspeed` (Rychlost):
- Přidáno pole pro maximální povolenou rychlost
- Hodnoty byly doplněny na základě typu silnice (např. dálnice = 130, obec = 50…)



	OBJECTID *	Shape *	fclass	name	maxspeed	Shape_Length	lenght	dist_air
1	25	Polyline	motorway		130	498,017697	498,045199	491,719733
2	399	Polyline	motorway		130	500,33032	500,352067	500,027974
3	421	Polyline	motorway		130	3099,515629	3099,762891	3087,391418
4	423	Polyline	motorway		130	1183,842702	1183,933651	1183,834085
5	424	Polyline	motorway		130	1120,075361	1120,161408	1120,075275
6	425	Polyline	motorway		130	353,019834	353,04606	353,018044
7	426	Polyline	motorway		130	46,146397	46,149824	46,146397
8	427	Polyline	motorway		130	46,038103	46,041523	46,038103

Obrázek 1: Datová vrstva silnic s příslušnou atributovou tabulkou v prostředí ArcGIS Pro

1.2 Export pro Python

Klíčovým krokem před samotným programováním byl, v prostředí ArcGIS, export do formátu, který dokáže zpracovat Python skript `lines_to_graph2.py`.

Struktura exportovaných dat:

Soubor silnic: Každý řádek reprezentuje jednu hranu grafu ve tvaru `x1 y1 x2 y2 w`.

- `x1, y1`: Souřadnice počátečního bodu úseku.
- `x2, y2`: Souřadnice koncového bodu úseku.
- `w`: Váha hrany (vzdálenost, čas, nebo čas zohledňující klikatost).

Soubor měst: Každý řádek definuje jedno sídlo ve tvaru `název_města x y`.

Import do Pythonu a tvorba grafu: Tato textová data byla následně vstupem pro zadaný algoritmus `lines_to_graph2.py`. Cílem skriptu bylo vytvořit grafovou strukturu silniční sítě a přiřadit jednotlivá města do příslušných uzlů, aby bylo možné mezi nimi vyhledávat cesty. Řešení problému s napojením měst (viz. teoretická část).

2 Implementace algoritmu

Samotné řešení úlohy bylo realizováno v jazyce Python. Projekt je strukturován do dvou hlavních skriptů, které zajišťují načtení dat a následný výpočet optimální trasy.

2.1 Konstrukce grafu a řešení tolerance

Prvním krokem je načtení textových souborů pomocí skriptu `lines_to_graph2.py`. Zde dochází k vytvoření topologie sítě. Jak bylo zmíněno v předchozí kapitole, při párování měst na uzly sítě byl řešen problém nekonzistentních souřadnic.

Tento nesoulad byl ošetřen implementací prostorové tolerance (metoda nejbližšího souseda). Algoritmus nehledá bod se shodnými souřadnicemi, ale identifikuje nejbližší uzel sítě v okruhu 1 metru. Pro tuto část byla použita AI ke konzultaci řešení problému ohledně neshodných souřadnic.

Ukázka kódu – řešení tolerance: Následující fragment kódu ukazuje, jak je pro každé město nalezen nejbližší uzel sítě na základě minimální eukleidovské vzdálenosti.

```
    # Calculating difference between all nodes, saving the lowest
     for ux, uy, uid in node_list:
        dist = math.sqrt((x - ux)**2 + (y - uy)**2)
        if dist < min_dist:
            min_dist = dist
            nearest_id = uid
     # Applying the tolerance (1 m) for the coordinates, then assigns city to the node
     if nearest_id is not None and min_dist < 1.0:
        city_to_id[name] = nearest_id
     else:
        print(f"City '{name}' does not exist in a node .")
```

2.2 Dijkstrův algoritmus

Princip algoritmu: Dijkstrův algoritmus je jedním z nejznámějších grafových algoritmů pro hledání nejkratší cesty v grafu s nezáporným ohodnocením hran. Algoritmus pracuje na principu "prohledávání do šířky" s prioritou – v každém kroku vybírá z množiny dosud nenavštívených uzlů ten, který má nejmenší vzdálenost od startu. Postupně tak "relaxuje" hrany a aktualizuje vzdálenosti k sousedním uzlům, dokud nenajde cíl. Pro efektivní výběr uzlu s minimální vahou naše implementace využívá prioritní frontu (heap queue). V průběhu tvorby algoritmu byla využita AI pro nastínění struktury algoritmu a dále pro opravu syntaxí a errorů.

Uživatel definuje startovní a cílové město. Zde je nutné dodržet syntaxi názvů – pokud se název města skládá z více slov, musí být mezery nahrazeny podtržítkem (např. `Ústí_nad_Labem`), aby odpovídaly formátu v datovém souboru.

Ukázka konfigurace v kódu: Níže uvedená část kódu ukazuje, jak uživatel volí typ výpočtu (odkomentováním příslušného řádku) a zadává trasu:

```
# Load roads
# Choose what calculation you want to use
# file = 'silnice_usti_vzdal.txt'          # For calculating Euclid distance
# file = 'silnice_usti_cas.txt‘            # For calculating fastest time without any consideration of curvature
of the roads
# file = 'silnice_usti_cas_klikatost.txt'  # For calculating fastest time considering curvature of the roads
                                             -> slowing down
```

A taky start a cíl:

```
# Choose start and end
start_city = "Ústí_nad_Labem"   # If you want a city with 2 or more words, use "_" between each word
end_city = "Bečov"     # For example "Ústí_nad_Labem"
file_cities = 'mesta_usti.txt
```

Vlastní výpočet:

1. **Inicializace:** Všechny vzdálenosti jsou nastaveny na nekonečno, pouze startovní uzel má vzdálenost 0.
2. **Relaxace:** Algoritmus iterativně prochází sousední uzly a pokud nalezne výhodnější cestu, aktualizuje hodnotu vzdálenosti a uloží předchůdce.
3. **Rekonstrukce:** Po dosažení cílového uzlu funkce zpětně zrekonstruuje trasu pomocí seznamu předchůdců.

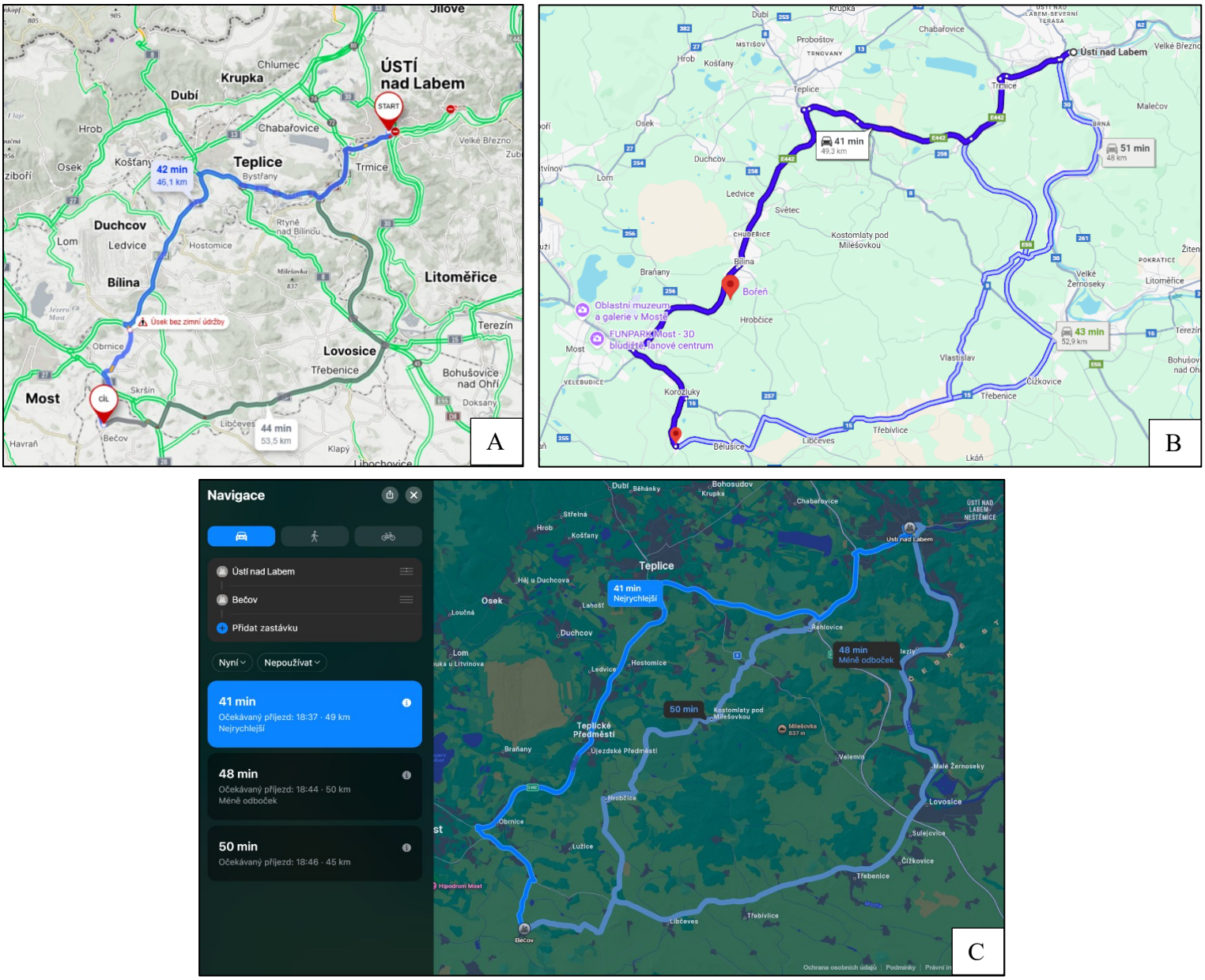
Poslední částí byla vizualizace nejlepší trasy, která byla vykreslena společně s vybranými městy v grafu pomocí knihovny *matplotlib.pyplot*.

3 Přehled výsledků

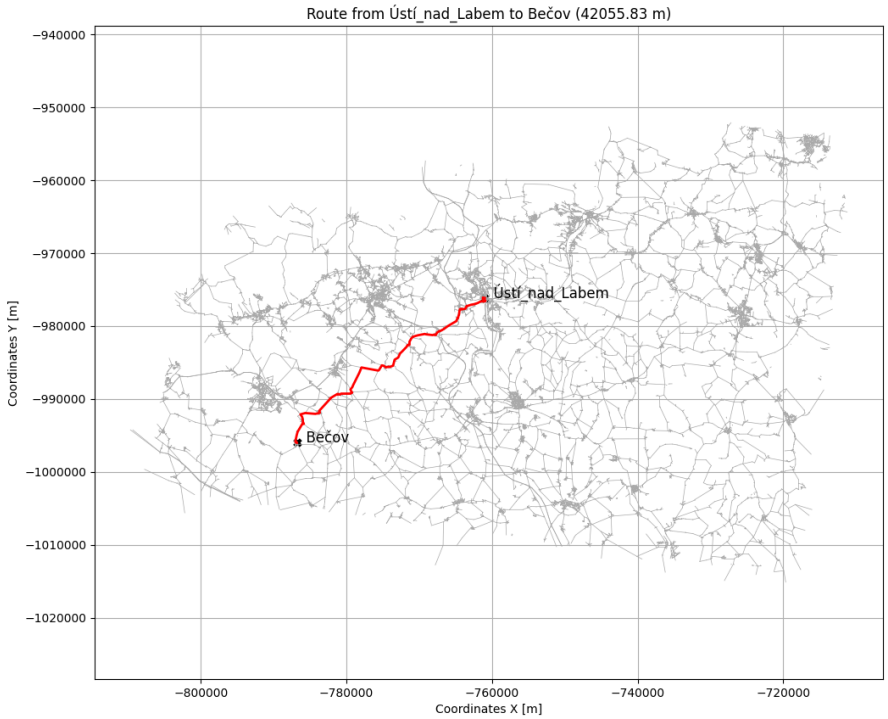
Funkčnost algoritmu byla ověřena na dvou testovacích trasách: **Zákupy – Most** a **Ústí nad Labem – Bečov**. Pro každou trasu byly vypočítány tři varianty (vzdálenost, čas, čas s klikatostí) a výsledky byly porovnány s komerčními plánovači (Mapy.cz, Google Maps, Apple Maps). Výsledky jsou shrnuty v následující tabulce.

		Testovaná trasa	
		Zákupy - Most	**Ústí nad Labem - Bečov**
vlastní implementace	**vzdálenost** [km]	88,0	42,1
	čas [min]	66,3	32,2
	čas s křivostí [min]	68,1	33,1
Mapy.com	**nejkratší trasa** [km]	91,7	46,1
	nejrychlejší trasa [min]	86,0	42,0
Google maps	**nejkratší trasa** [km]	91,9	48,0
	nejrychlejší trasa [min]	89,0	41
Apple Maps	**nejkratší trasa** [km]	90	45,0
	nejrychlejší trasa [min]	86,0	41,0

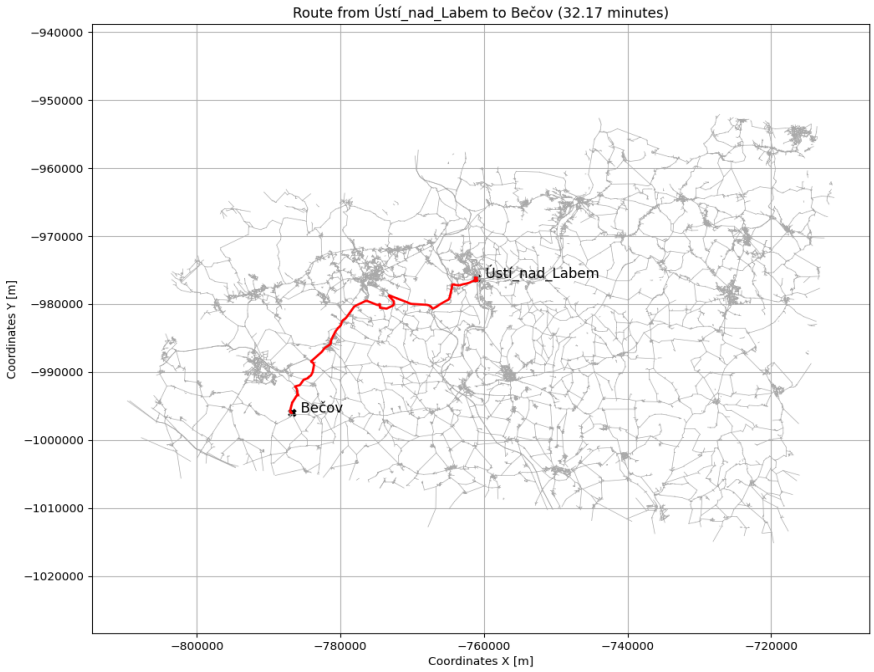
Tabulka 1: Porovnání výsledků vlastní implementace s vybranými komerčními plánovači



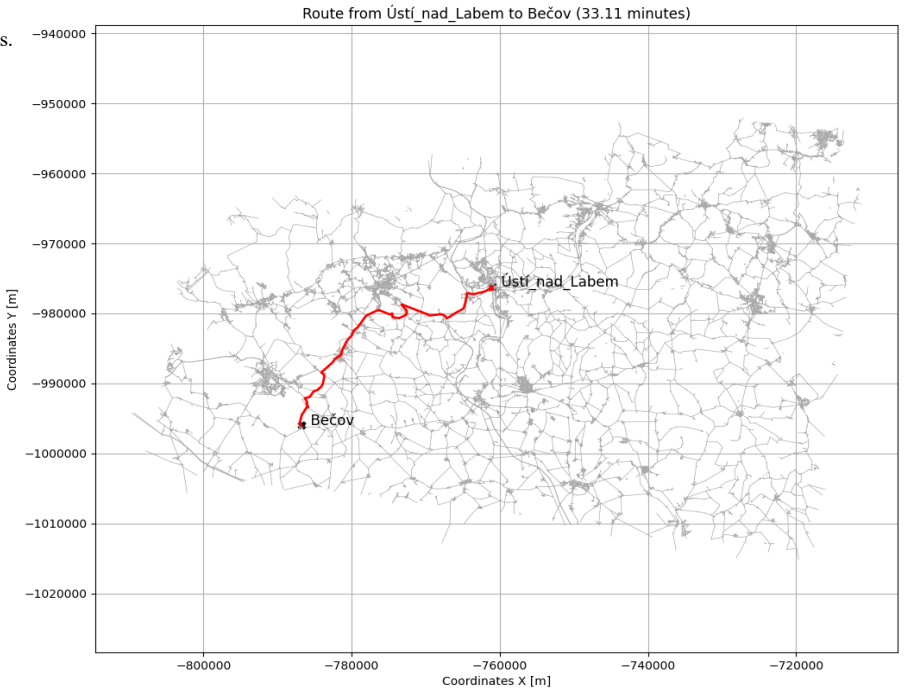
Obrázek 2: Srovnání testovacích tras v prostředí komerčních plánovačů (A - Mapy.cz, B - Google Maps, C - Apple Maps)
Trasa: Ústí nad Labem – Bečov



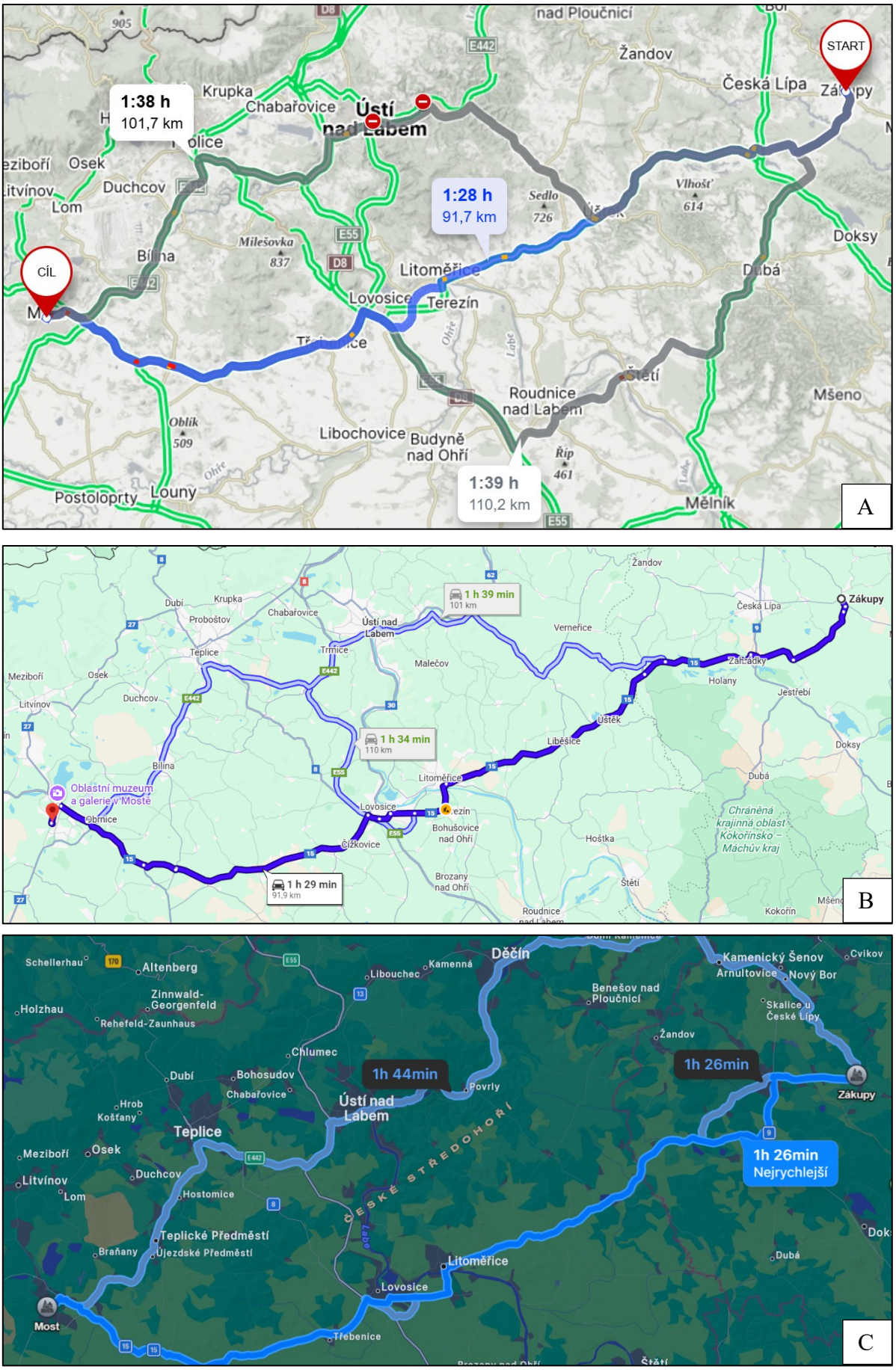
Obrázek. 3: Nejkratší cesta optimalizovaná na vzdálenost. Trasa Ústí nad Labem – Bečov s délkou 42,1 km



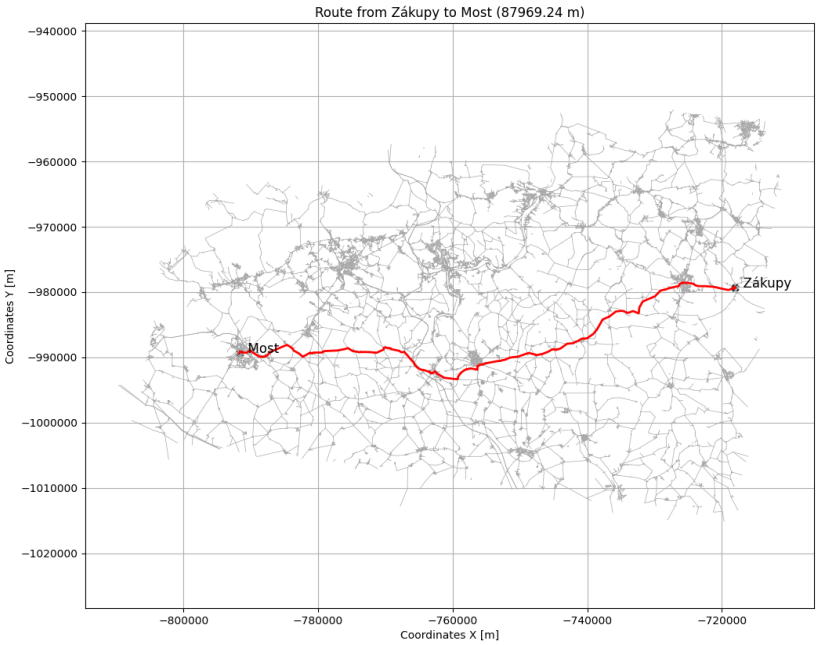
Obrázek. 4: Nejrychlejší cesta optimalizovaná na transportní čas. Trasa Ústí nad Labem – Bečov s časem 32,2 min klikatosti).



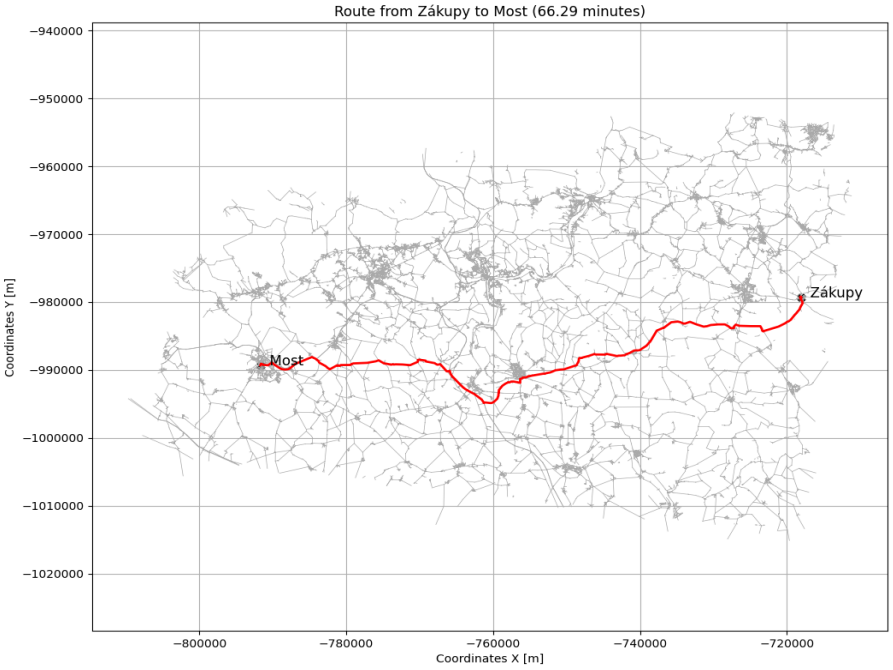
Obrázek. 5: Nejrychlejší cesta se zohledněním klikatosti.



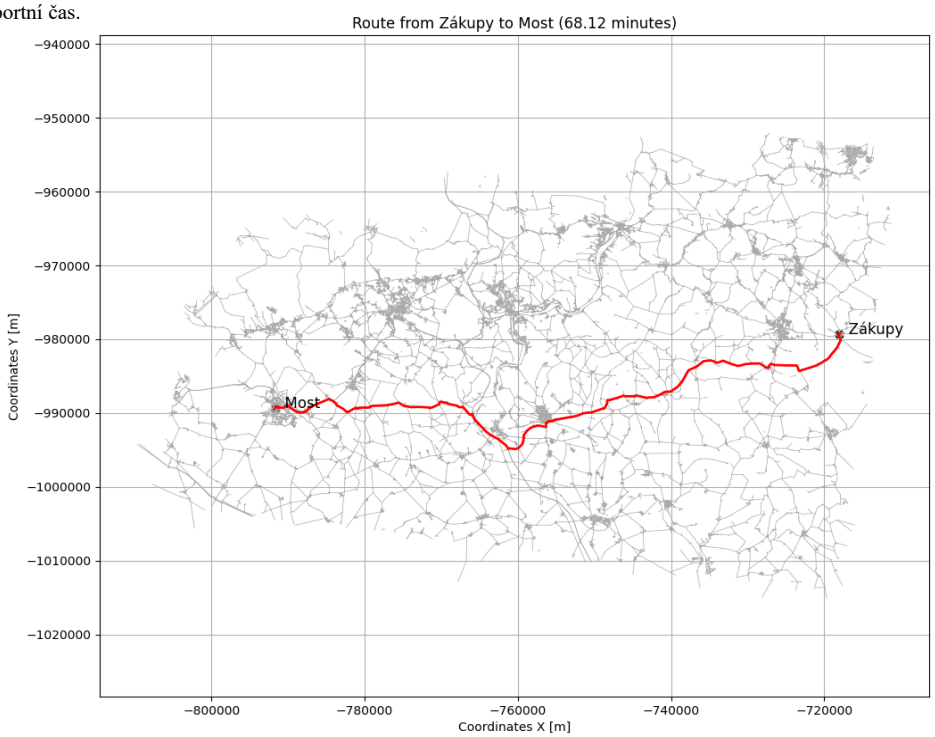
Obrázek 6: Srovnání testovacích tras v prostředí komerčních plánovačů (A - Mapy.cz, B - Google Maps, C - Apple Maps)
Trasa: Zákupy –Most



Obrázek 7: Nejkratší cesta optimalizovaná na vzdálenost.
Trasa Zákupy – Most s délkou 88,0 km.



Obrázek 8: Nejrychlejší cesta optimalizovaná na transportní čas.
Trasa Zákupy – Most s časem 66,3 min.



Obrázek 9: Nejrychlejší cesta se zohledněním klikatosti.
Trasa Zákupy – Most s časem 68,1 min.

Závěr a diskuse

Cílem práce byla implementace Dijkstrova algoritmu nad reálnou silniční sítí. Kritickým bodem realizace se ukázala být příprava grafové topologie, jelikož vstupní data měst (bodová vrstva) a silnic (liniová vrstva) nevykazovala absolutní polohovou shodu. Tento problém byl efektivně vyřešen implementací prostorové tolerance (metoda nejbližšího souseda s prahem 1 metr). Toto řešení se ukázalo jako robustní, umožnilo korektní napojení všech testovaných sídel na silniční graf a zprovoznění vyhledávání cest dle tří kritérií.

Z výsledků vyplývá, že geometrie trasy (vzdálenost) v naší implementaci velmi přesně odpovídá komerčním řešením (odchylka v řádu stovek metrů). V případě časových údajů však dosahuje vlastní model výrazně nižších hodnot (např. 66 min oproti 86 min u Mapy.cz). Tento rozdíl je způsoben tím, že model počítá s teoretickou maximální rychlostí a nezohledňuje reálné zpomalení v obcích, na křižovatkách či vlivem dopravy. Zavedení penalizace za klikatost (tortuosity) vedlo k mírnému navýšení času a výběru rovnějších úseků, čímž se model částečně přiblížil realitě.

Použitá literatura a zdroje:

GEOFABRIK GMBH. *OpenStreetMap Data Extracts: Czech Republic* [online datová sada]. Karlsruhe: Geofabrik GmbH, [cit. 2026-01-10]. Dostupné z: https://download.geofabrik.de/europe/czech-republic.html

PYTHON SOFTWARE FOUNDATION. *math — Mathematical functions* [online]. Python 3.13 Documentation, [cit. 2026-01-10]. Dostupné z: https://docs.python.org/3/library/math.html

THE MATPLOTLIB DEVELOPMENT TEAM. *Matplotlib documentation* [online]. [cit. 2026-01-10]. Dostupné z: https://matplotlib.org/

W3SCHOOLS (2026): DSA Graphs Dijkstra's Algorithm [online]. [cit. 2026-01-10] Dostupné z:

https://www.w3schools.com/dsa/dsa_algo_graphs_dijkstra.php