

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA STAVEBNÍ
KATEDRA GEOMATIKY

Název předmětu

Geoinformatika

Úloha

U3

Název úlohy:

Nejkratší cesta grafem

akademický rok
2024/2025

semestr
zimní

studijní skupina
C102

vypracoval
Matyáš Pokorný
Tereza Černohousová

datum
3.12.2024

klasifikace

Technická zpráva

1 Bonusové úlohy

Z bonusových úloh námi byly zpracovány:

- Řešení úlohy pro grafy se záporným ohodnocením.
- Nalezení cesty mezi všemi dvojicemi uzlů
- Nalezení minimální kostry Kruskal
- Nalezení minimální kostry Prime

2 Pracovní postup

2.1 Dijkstra algoritmus

Dijkstra algoritmus je metoda pro hledání nejkratší cesty v grafu. Jeho princip spočívá v postupném prozkoumávání grafu, přičemž se začíná od počátečního uzlu. Algoritmus udržuje množinu již navštívených uzlů a pro každý uzel počítá vzdálenost od počátečního uzlu. V každém kroku se vybere uzel s nejmenší vzdáleností, který ještě nebyl navštíven, a z jeho sousedních uzlů se aktualizují vzdálenosti. Tento proces pokračuje, dokud se všechny uzly nezpracují nebo dokud není nalezena cesta k cílovému uzlu.

Dijkstra algoritmus garantuje nalezení optimální cesty, přičemž jeho časová složitost závisí na použité struktuře pro uložení uzlů. V naší implementaci byla pro uložení uzlů použita prioritní fronta.

Náš algoritmus vytváří seznam předchůdců v optimální cestě pro daný počáteční uzel. Poté, je vytvořena cesta další zadaný bod - koncový.

Algoritmus pracuje s různým oceněním hran, v našem případě se jedná o:

- vzdálenost danou délkou polylinie,
- transportní čas daný délkou polylinie a rychlostí danou třídou silnice,
- transportní čas daný délkou polylinie a rychlostí podle třídy silnice, navíc s uvažováním křivosti silnice.

Kde délka polylinie je atribut této entity (d), transportní čas je vypočten z této délky a z rychlosti která je dána třídou silnice:

$$t = \frac{d}{v}, \quad (1)$$

kde v je podle třídy silnice z popisu dat databáze ArcČR500 v.3.3:

- 1 - dálnice = 130 km/h
- 2 - rychlostní silnice = 110 km/h

- 3 - silnice I. třídy = 90 km/h
- 4 - silnice II. třídy = 80 km/h
- 5 - silnice III. třídy = 60 km/h
- 6 - neevidovaná silnice = 50 km/h

a transportní čas s opravením o křivost silnice je spočten jako:

$$t = \frac{d}{v} \frac{d}{d_e}, \quad (2)$$

kde d_e je přímá vzdálenost mezi dvěma body.

Python skript *dijkstra_better.py* poté obsahuje další funkce, které provedou na vytvořeném grafu nalezení nejkratší cesty dijkstra algoritmem a cestu graficky zobrazí. Spolu s vrcholy, které cestu tvoří je zároveň vypočtena celková cena cesty.

2.2 Bellman-Fordův algoritmus

Pro ošetření grafů se záporným ohodnocením byl Dijkstrův algoritmus upraven do podoby Bellman-Fordova algoritmu, který dokáže detekovat záporné cykly a v případě jejich existence výpočet zastavit. Tento algoritmus implementuje skript *modif_dijkstra.py* a je testován na příkladovém grafu, a nikoliv na reálné síti (ta neobsahuje záporné hrany).

2.3 Nalezení cest mezi všemi vrcholy

Pro nalezení cest mezi všemi dvojicemi vrcholů byl výpočet pro nalezení cest zabalen do dvou *for* cyklů, které pro každý počáteční a koncový bod vytvoří cestu a uloží do určené proměnné. Tento postup je implementován ve skriptu *dijkstra_all2all.py*. Postup je sice velmi jednoduchý, ale v jazyku Python trvá velmi velmi dlouho v reálné silniční síti.

2.4 Borůvkův algoritmus

Borůvkův (nebo Kruskalův) algoritmus slouží pro nalezení minimální kostry v souvislém a váženém grafu. Algoritmus pracuje iterativně postupným spojováním komponent grafu. V každé iteraci identifikuje pro každou komponentu hranu s nejmenší vahou, která ji spojuje s jinou komponentou. Tyto hrany jsou poté přidány do kostry, čímž se komponenty propojí. Proces pokračuje, dokud všechny komponenty nesplynou v jednu souvislou kostru. Algoritmus implementuje skript *boruvka.py*.

2.5 Jarníkův algoritmus

Jarníkův (Primův) algoritmus je algoritmus pro nalezení minimální kostry v souvislém a váženém grafu. Algoritmus začíná u libovolného vrcholu a postupně přidává hrany s nejmenší vahou, které spojují již navštívené vrcholy s dosud nenavštívenými. Tento proces pokračuje,

dokud nejsou zahrnuty všechny vrcholy grafu. Klíčovou vlastností algoritmu je jeho hladový (chamtivý) přístup, kdy v každém kroku volí pouze nejlevnější hranu přidávající nový vrchol. Algoritmus implementuje skript *jarnik.py*, přičemž běžný Jarníkův algoritmus si neporadí s reálnou sítí, která obecně nemusí být souvislá, proto je prezentován skript *jarnik-nesouvisly.py*, který je ošetřen tak, aby přeskakoval malé nesouvislé komponenty grafu.

2.6 Implementace v reálné síti

Síť komunikací, ze které byl tvořen graf a počítána cesta, byl vzat z databáze ArcČR500 verze 3.3. Z těchto dat byla použita datová vrstva Silnice_2015, a to z okresů Cheb a Tachov.

Vrstva silnic byla exportovaná jako Geopackage, jakožto otevřený a standardizovaný moderní formát pro práci s prostorovými daty. K výběru dat na konkrétním území a k uložení do požadovaného formátu byl využit program QGIS.

Z tohoto formátu je síť načítána funkcí *lines2graph_gpkg* a převádí ho na dva slovníky, jeden obsahující graf, který obsahuje ke každému vrcholu, který má celočíselný index (ten začíná od 1 a jde až do $N = \text{počet vrcholů}$) a k němu odpovídající indexy sousedních vrcholů a cena cesty (vzdálenost, čas,...) k nim.

Názvy obcí a souřadnice jejich definičních bodů byly získány obdobnou cestou jako silniční síť pomocí skriptu s funkcemi *obce_load.py*.

Pseudokód pro funkci `lines2graph_gpkg`

Algorithm 1 `lines2graph_gpkg`

Require: Typ váhy `type_of_weight`: 'Euclidean distance', 'Transport time', nebo 'Transport time with deviality'

Require: Soubor `file`: název geopackage souboru (implicitně 'silnice.gpkg')

Ensure: Graf `G.m` a souřadnice uzlů `C.m`

```
1: Načti geopackage soubor pomocí knihovny fiona
2: Inicializuj prázdné slovníky G, C, a W
3: Definuj očekávané rychlosti podle tříd silnic (road_classes)
4: for každou čáru line v síti do
5:   Získej souřadnice začátku a konce čáry z geometrie (line.geometry)
6:   Urči startovní a koncový bod: start a end
7:   if start není v C then
8:     Přidej start do C
9:   end if
10:  if end není v C then
11:    Přidej end do C
12:  end if
13:  if start není v G then
14:    Inicializuj G[start] jako prázdný seznam
15:  end if
16:  if end není v G then
17:    Inicializuj G[end] jako prázdný seznam
18:  end if
19:  Přidej end do seznamu sousedů G[start]
20:  Přidej start do seznamu sousedů G[end]
21:  if type_of_weight je 'Euclidean distance' then
22:    Vypočítej váhu w jako délku čáry (line.properties['SHAPE.Length'])
23:  else if type_of_weight je 'Transport time' then
24:    Vypočítej vzdálenost d a rychlost v podle třídy silnice
25:    Vypočítej čas t = d / v
26:    Nastav váhu w = t
27:  else if type_of_weight je 'Transport time with deviality' then
28:    Vypočítej čas t a poměr curvature_ratio = d / Euclidean_distance
29:    Nastav váhu w = t * curvature_ratio
30:  end if
31:  if start není v W then
32:    Přidej w do W[start]
33:  end if
34:  if end není v W then
35:    Přidej w do W[end]
36:  end if
37: end for
```

Pseudokód pro funkce `obcefromgpkg`, `obec2node_of_graph`, a `node2obec_of_graph`

Algorithm 2 `obcefromgpkg`

Require: Název souboru `file`, ve kterém je geopackage (implicitně `'obce.gpkg'`)

Ensure: Slovník `O`, který mapuje názvy obcí na jejich souřadnice

- 1: Načti geopackage soubor pomocí knihovny `fiona` a ulož jako `obce`
 - 2: Inicializuj prázdný slovník `O`
 - 3: **for** každá obec `obec` v `obce` **do**
 - 4: Získat souřadnice `coords` z `obec.geometry['coordinates']`
 - 5: Získat název obce `name` z `obec.properties['NAZ_OBEC']`
 - 6: Přidat dvojici klíč-hodnota (`name`, `coords`) do `O`
 - 7: **end for**
 - 8: **return** `O`
-

Algorithm 3 `obec2node_of_graph`

Require: Slovník `O` (obce a jejich souřadnice) a slovník `C` (uzly a jejich souřadnice)

Ensure: Slovník `towns_closest_to_points_from_C`, který mapuje uzly na názvy nejbližších obcí

- 1: Inicializuj prázdný slovník `towns_closest_to_points_from_C`
 - 2: **for** každá obec `town`, její souřadnice `def_coord` v `O` **do**
 - 3: Inicializuj `closest_coord` na `None` a `min_distance` na nekonečno
 - 4: **for** každý uzel `id`, jeho souřadnice `coord` v `C` **do**
 - 5: Spočítej vzdálenost `distance` mezi `def_coord` a `coord`
 - 6: **if** `distance < min_distance` **then**
 - 7: Aktualizuj `min_distance` na `distance`
 - 8: Aktualizuj `closest_coord` na `id`
 - 9: **end if**
 - 10: **end for**
 - 11: Přidej dvojici (`closest_coord`, `town`) do `towns_closest_to_points_from_C`
 - 12: **end for**
 - 13: **return** `towns_closest_to_points_from_C`
-

Algorithm 4 node2obec_of_graph

Require: Slovník O (obce a jejich souřadnice) a slovník C (uzly a jejich souřadnice)

Ensure: Slovník towns_closest_to_points_from_C, který mapuje názvy obcí na nejbližší uzly

```
1: Inicializuj prázdný slovník towns_closest_to_points_from_C
2: for každá obec town, její souřadnice def_coord v  $O$  do
3:   Inicializuj closest_coord na None a min_distance na nekonečno
4:   for každý uzel id, jeho souřadnice coord v  $C$  do
5:     Spočítej vzdálenost distance mezi def_coord a coord
6:     if distance < min_distance then
7:       Aktualizuj min_distance na distance
8:       Aktualizuj closest_coord na id
9:     end if
10:  end for
11:  Přidej dvojici (town, closest_coord) do towns_closest_to_points_from_C
12: end for
13: return towns_closest_to_points_from_C
```

Pseudokód pro funkce dijkstra, rPath, a další

Algorithm 5 rPath

Require: Začátek cesty u , konec cesty v , seznam předchůdců P

Ensure: Seznam path, obsahující body na cestě mezi u a v

```
1: Inicializuj prázdný seznam path
2: while  $v \neq u$  and  $v \neq -1$  do
3:   Přidej  $v$  do path
4:   Aktualizuj  $v$  na  $P[v]$ 
5: end while
6: Přidej  $v$  do path
7: if  $v \neq u$  then
8:   Vypiš 'Incorrect path'
9: end if
10: return path
```

Algorithm 6 `sum_of_weights`

Require: Cesta `path`, graf `G`**Ensure:** Celková váha `w` cesty

```
1: Inicializuj váhu w na 0
2: for i od 0 do len(path) - 2 do
3:   Přičti váhu G[path[i]][path[i+1]] k w
4: end for
5: return w
```

Algorithm 7 `dijkstra`

Require: Graf `G`, startovní bod `start`**Ensure:** Seznam předchůdců `P`, vzdálenosti `d` od `start`

```
1: Inicializuj délku grafu n na len(G) + 1
2: Inicializuj vzdálenosti d na nekonečno (inf) a předchůdce P na -1
3: Inicializuj prioritní frontu PQ
4: Přidej do PQ dvojici (0, start)
5: Nastav d[start] na 0
6: while PQ není prázdná do
7:   Získej du, u z PQ
8:   for každého souseda v a jeho váhu wuv v G[u] do
9:     if d[v] > d[u] + wuv then
10:      Aktualizuj d[v] na d[u] + wuv
11:      Nastav P[v] na u
12:      Přidej (d[v], v) do PQ
13:     end if
14:   end for
15: end while
16: return P, d
```

Algorithm 8 Hlavní skript

```
1: Načti graf G a souřadnice C pomocí funkce lines2graph_gpkg
2: Načti obce O a jejich uzly OC pomocí funkcí obcefromgpkg a obec2node_of_graph
3: Spuště algoritmus Dijkstry na grafu G se startem v bodě 83, ulož P, D
4: Získej cestu path pomocí rPath(83, 268, P)
5: Spočítej náklady cost cesty pomocí sum_of_weights(path, G)
6: Graficky zobraz graf a vypočtenou cestu
```

Pseudokód pro Borůvkův algoritmus a pomocné funkce

Algorithm 9 dict2lists

Require: Graf G v podobě slovníku

Ensure: Vrcholy V a hrany E ve formě seznamů

- 1: Inicializuj seznam vrcholů V jako klíče slovníku G
 - 2: Inicializuj prázdný seznam hran E
 - 3: **for** každý vrchol v v V **do**
 - 4: **for** každého souseda k vrcholu v **do**
 - 5: Přidej hranu $[v, k, G[v][k]]$ do E
 - 6: **end for**
 - 7: **end for**
 - 8: **return** V, E
-

Algorithm 10 find

Require: Vrchol u , pole předchůdců P

Ensure: Kořen komponenty obsahující u

- 1: **while** $P[u] \neq u$ **do**
 - 2: Nastav $u = P[u]$
 - 3: **end while**
 - 4: **return** u
-

Algorithm 11 union

Require: Vrcholy u, v , pole předchůdců P

Ensure: Spojení komponent obsahujících u a v

- 1: Najdi kořeny komponent u a v pomocí **find**
 - 2: **if** kořeny nejsou shodné **then**
 - 3: Spoj komponenty: nastav $P[\text{root}_v] = \text{root}_u$
 - 4: **end if**
-

Algorithm 12 make_set

Require: Vrchol u , pole předchůdců P

Ensure: Inicializace komponenty obsahující pouze u

- 1: Nastav $P[u] = u$
-

Algorithm 13 boruvka

Require: Graf G ve formě slovníku

Ensure: Minimální kostra T a její váha wt

```
1: Transformuj graf pomocí dict2lists( $G$ ) na vrcholy  $V$  a hrany  $E$ 
2: Inicializuj prázdný seznam  $T$  a váhu  $wt = 0$ 
3: Inicializuj pole předchůdců  $P$  velikosti  $|V| + 1$  s hodnotami inf
4: for každý vrchol  $v$  v  $V$  do
5:   Inicializuj množinu pro  $v$  pomocí make_set(v, P)
6: end for
7: Seřaď hrany  $E$  podle jejich vah
8: for každou hranu  $[u, v, w]$  v  $E$  do
9:   if find(u, P)  $\neq$  find(v, P) then
10:    Spoj komponenty pomocí union(u, v, P)
11:    Přidej hranu  $[u, v, w]$  do  $T$ 
12:    Aktualizuj váhu:  $wt = wt + w$ 
13:   end if
14: end for
15: return  $T, wt$ 
```

Algorithm 14 Hlavní skript

```
1: Načti graf  $G$  a souřadnice  $C$  pomocí lines2graph_gpkg
2: Načti obce  $O$  a jejich uzly  $OC$  pomocí funkcí obcefromgpkg a obec2node_of_graph
3: Spuště Borůvkův algoritmus na grafu  $G$ , výsledky ulož do  $T, weight$ 
4: Vypiš minimální kostru  $T$  a celkovou váhu weight
5: Graficky zobraz graf a minimální kostru (zelená pro hrany v kostře, modrá pro ostatní)
```

Pseudokód pro Primův algoritmus (Jarník) a pomocné funkce

Algorithm 15 jarník

Require: Graf G ve formě slovníku, počáteční vrchol $start$

Ensure: Minimální kostra T a její váha wt

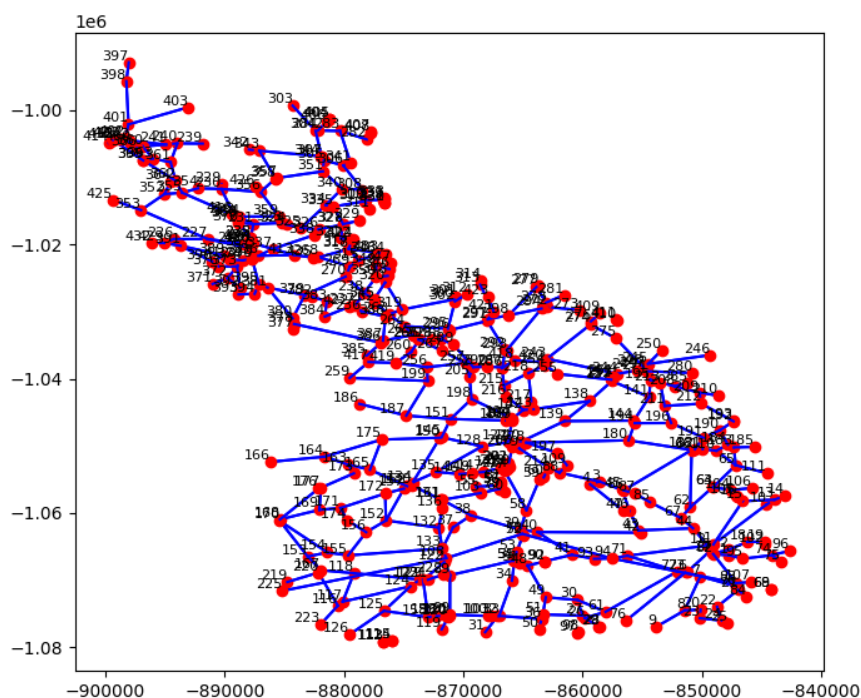
```
1: Inicializuj prázdný seznam hran  $T$ , váhu  $wt = 0$ 
2: Inicializuj množinu navštívených vrcholů  $visited = \{\}$ , nenavštívených vrcholů
    $unvisited = \{v \in G\}$  Nastav počáteční vrchol  $current\_node = start$ 
3: Inicializuj počítadlo komponent  $count\_of\_components = 0$ 
5: while  $|visited| < |G|$  do
6:   if  $current\_node$  není v  $visited$  then
7:     Přidej  $current\_node$  do  $visited$ , odeber z  $unvisited$ 
8:   end if
9:   Inicializuj  $min\_edge = None$ 
10:  for každý vrchol  $u$  v  $visited$  do
11:    for každého souseda  $v$  vrcholu  $u$  s váhou hrany  $w$  do
12:      if  $v$  není v  $visited$  a ( $min\_edge$  je prázdná nebo  $w < min\_edge[2]$ ) then
13:        Nastav  $min\_edge = (u, v, w)$ 
14:      end if
15:    end for
16:  end for
17:  if  $min\_edge$  then
18:    Přidej hranu  $min\_edge$  do  $T$ , přičti její váhu k  $wt$ 
19:    Přidej cílový vrchol  $v$  z  $min\_edge$  do  $visited$ , odeber z  $unvisited$ 
20:    Nastav  $current\_node = v$ 
21:  else
22:    Vypiš varování o nespojitosti grafu
23:    Zvýši  $count\_of\_components$  o 1
24:    if  $unvisited$  není prázdné then
25:      Vyber náhodný vrchol z  $unvisited$  jako  $current\_node$ 
26:    else
27:      Přerušeni cyklu
28:    end if
29:  end if
30: end while
31: Vypiš počet nespojených komponent  $count\_of\_components + 1$ 
32: return  $T, wt$ 
```

Algorithm 16 Hlavní skript

```
1: Načti graf  $G$  a souřadnice  $C$  pomocí lines2graph_gpkg
2: Načti obce  $O$  a jejich uzly  $OC$  pomocí funkcí obcefromgpkg a obec2node_of_graph
3: Spuště Primův algoritmus na grafu  $G$ , výsledky ulož do  $T$ , weight
4: Vypiš minimální kostru  $T$  a celkovou váhu weight
5: Graficky zobraz graf a minimální kostru:
6: for každý vrchol node a sousedy neighbors v  $G$  do
7:   Získej souřadnice  $(x, y)$  a  $(nx, ny)$  pro každou hranu
8:   if hrana patří do  $T$  then
9:     Zobraz ji zeleně
10:  else
11:    Zobraz ji modře
12:  end if
13:  Označ vrchol jako červený bod
14:  Přidej textový popis (pokud existuje) na základě  $OC[node]$ 
15: end for
16: Zobraz graf pomocí plt.show()
```

3 Výsledky

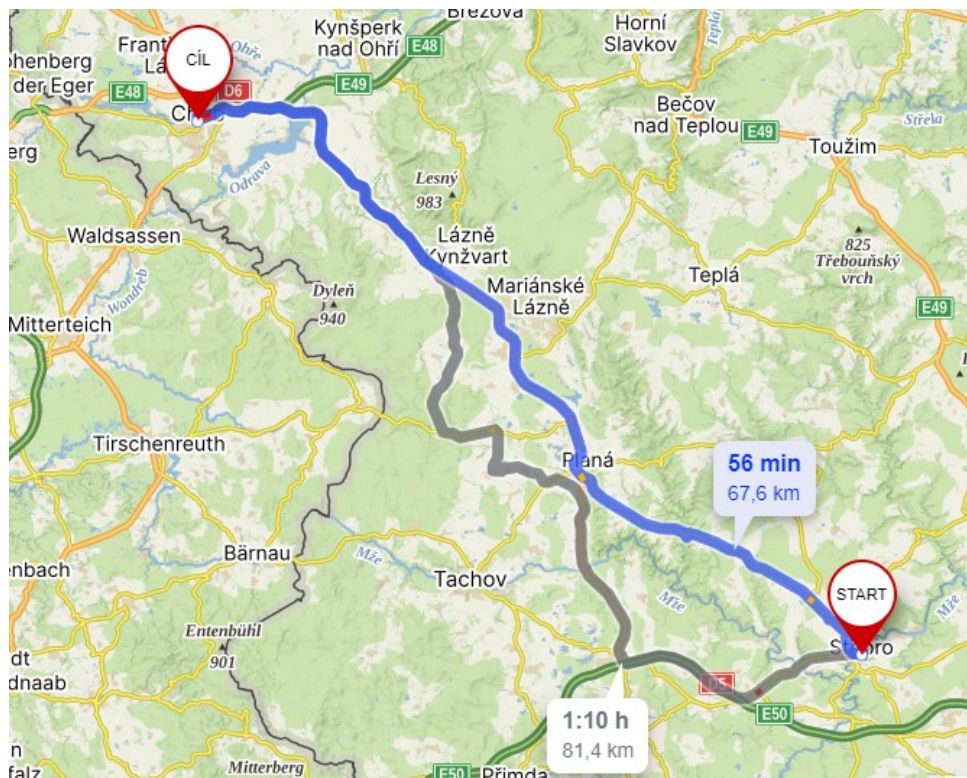
3.1 Vizualizace sítě



Obrázek 1: Vizualizace silniční sítě okresů Cheb a Tachov

3.2 Trasa zvolená pro testování

Stříbro ⇒ Cheb



Obrázek 2: Navrhované trasy ze služby Mapy.cz

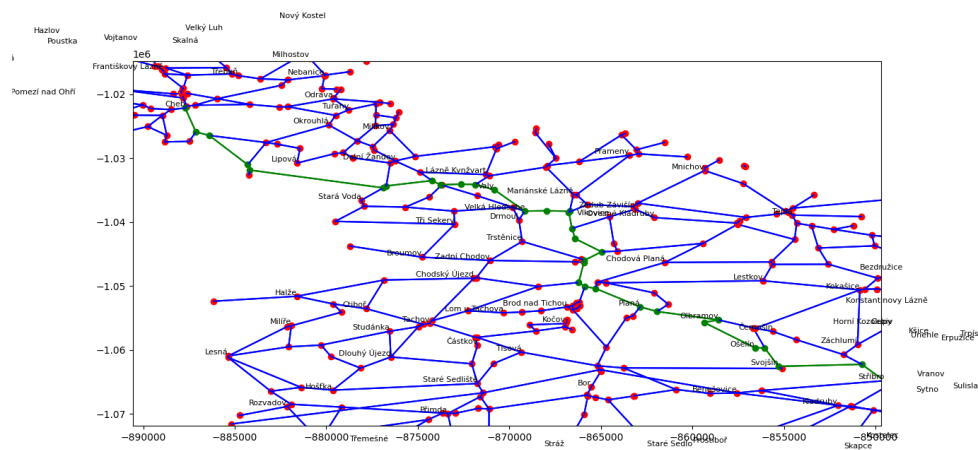
3.3 Cesta pomocí nejkratší vzdálenosti

Vrcholy přes které vede cesta:

396, 395, 381, 380, 378, 386, 387, 265, 266, 299, 288, 289, 258, 290, 287, 286, 215, 216, 142, 207, 189, 127, 206, 179, 214, 88, 3, 4, 47, 46, 43, 44, 12

Vzdálenost ujetá po nejkratší cestě:

cca. 55.4 km



Obrázek 3: Nejkratší cesta

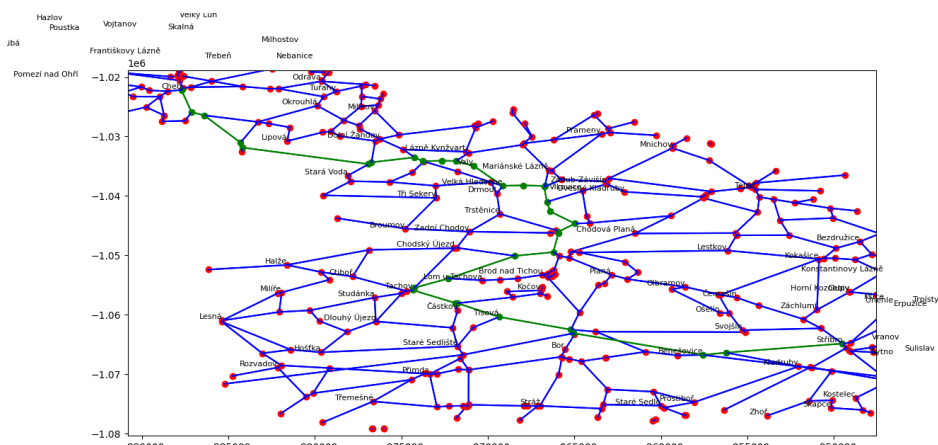
3.3.1 Cesta pomocí nejmenšího času

Vrcholy přes které vede cesta:

396, 395, 381, 380, 378, 386, 387, 265, 266, 299, 288, 289, 258, 290, 287, 286, 215, 216, 142, 207, 189, 127, 128, 135, 134, 130, 131, 161, 38, 39, 78, 94, 71, 12

Doba jízdy po nejrychlejší cestě:

cca. 49 minut



Obrázek 4: Nejrychlejší cesta

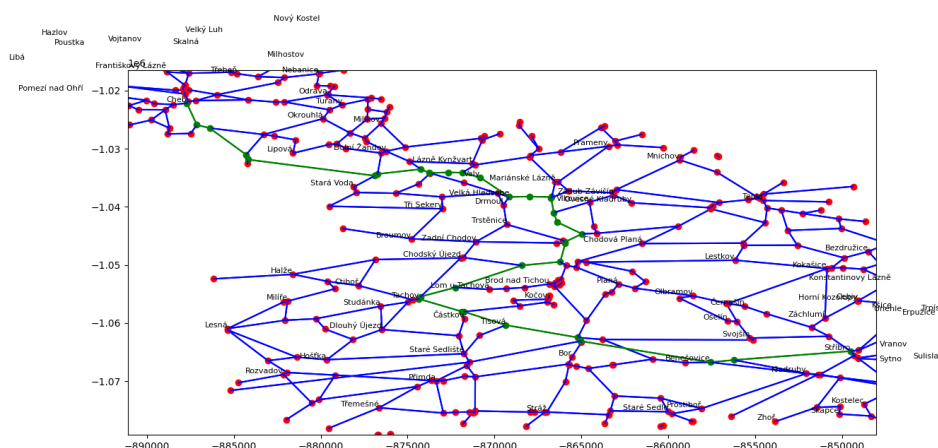
3.3.2 Cesta pomocí nejmenšího času s uvážením klikatosti

Vrcholy přes které vede cesta:

396, 395, 381, 380, 378, 386, 387, 265, 266, 299, 288, 289, 258, 290, 287, 286, 215, 216, 142, 207, 189, 127, 128, 135, 134, 130, 131, 161, 38, 39, 78, 94, 71, 12

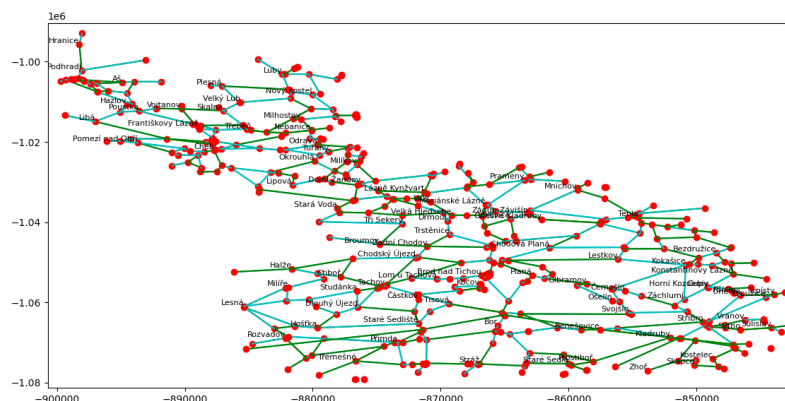
Doba jízdy po nejrychlejší cestě s uvážením křivosti silnic:

cca. 52 minut



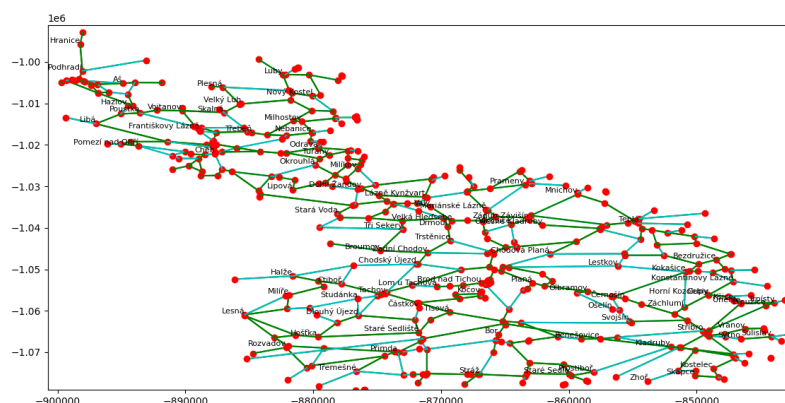
Obrázek 5: Nejrychlejší cesta s uváženou křivostí

3.4 Minimální kostra Borůvka



Obrázek 6: Minimální kostra (zeleně)

3.5 Minimální kostra Jarník



Obrázek 7: Minimální kostra (zeleně)

3.6 Nejkratší cesta v grafu se záporným ohodnocením hran

Prezentuje skript *modif_dijkstra.py*.

3.7 Nejkratší cesta v grafu mezi všemi vrcholy

Prezentuje skript *dijkstra_all2all.py*. Z časových důvodů skript prezentuje testovací graf o 9 vrcholech.

4 Závěr

Výsledkem zpracování této úlohy jsou skripty a funkce, které z veřejných dat ve formátu GeoPackage vytvoří graf, na kterém jsou prováděny různé grafové úlohy.

První část úloh se věnovala nalezení nejkratších a nejrychlejších cest. K tomu byl použit Dijkstrův algoritmus a Bellman-Fordův algoritmus pro grafy se záporně ohodnocenými hranami. Funkce cesty vykresluje v rámci silniční sítě, s vrcholy, kterými jsou různá pojmenovaná sídla. Algoritmus byl testován na trase Stříbro-Cheb, a vzdálenost i časová náročnost vypočtená z grafu se blíží hodnotám, které nám předkládá obdobná služba serveru Mapy.cz.

Druhá část byla zaměřena na hledání minimálních koster. Ty nacházeli Borůvkův a Jarníkův algoritmus. Při použití na reálných datech však bylo potřeba občas udělat lehké úpravy, jelikož reálná data často produkují nesouvislé grafy. Minimální kostry byly vypočteny s ohodnoceným grafem podle euklidovské vzdálenosti. Minimální kostry byly obdobně jako v první části, vyobrazeny do silniční sítě s pojmenovanými sídly na stejném území.

Algoritmus/SW	Vzdálenost [km]	Čas [min]
Mapy.cz	67,6	56
Nejkratší vzd.	55,4	-
Nejmenší čas	-	49
Nejmenší čas s klikatostí	-	52

Tabulka 1: Přehled výsledků z různých metod

V Praze dne: 21.1. 2025

**T. Černohousová
M. Pokorný**