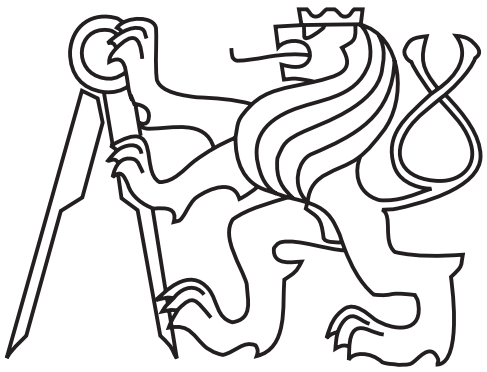
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ

FAKULTA STAVEBNÍ

OBOR GEOMATIKA



Semestrální práce

UZPD - IPR a síťové analýzy

Vedoucí práce: Ing. Martin Landa, Ph.D.

Leden 2016 **David Mráz, Pavel Kulmon**

Obsah

1 Úvod

Tato práce byla zpracována v rámci předmětu *UZPD* vyučovaného na katedře *geomatiky*. Zadáním bylo navrhnout a vytvořit tématické vrstvy na základě OSM (OpenStreet Map) a dalších otevřených zdrojů. Dále bylo zadáním aplikovat testy datové integrity a případně odstranit inkonzistence v datech. Nakonec bylo úkolem vytvořit tutoriál k navrženým a vytvořeným datovým vrstvám, který bude obsahovat atributové a prostorové dotazy nad databází *pgis_uzpd*.
Vzhledem k tomu, že datová sada OSM byla v rámci tohoto předmětu již mnohokrát zpracovávána v minulých letech, rozhodli jsme se použít data z jiných zdrojů. Konkrétně jsme se rozhodli použít data IPR Praha (Institut plánování a rozvoje - Pražská otevřená data).

2 Data IPR

Institut Plánování a Rozvoje Praha poskytuje Otevřená data (Open data) na základě licenčních podmínek, jejichž kompletní znění je dostupné z [2]. K datům je povoleno jejich kopírování a distribuce jakýmkoli způsobem (např. jejich nahrání na GIS server jako v případě našeho využití). Dále je povoleno data upravovat a začlenit do vlastního díla za předpokladu zachování licence k datům.
Velkou výhodou dat IPR Praha je, že ke všem datovým vrstvám jsou dodávána metadata se základními informacemi (tzv. data o datech).

3 Formát dat IPR

Data IPR jsou poskytována ve formě datových souborů v různých formátech jako jsou GeoJSON (Geo JavaScript Online Notification), GML, Esri Shapefile nebo DXF. Z důvodu pohodlného převodu Shapefile na SQL dávku pomocí programu `shp2pgsql` jsme zvolili právě tento formát. Kompletní příkaz pro převod souboru je

```
shp2pgsql -s 5514 -W latin2 file.shp table > file.sql
```

kde přepínač `-s` slouží k zadání *SRID* identifikačního čísla souřadného systému ve kterém jsou data v Shapefile uložena (S-JTSK = 5514) a přepínač `-W` slouží ke změně kódování vstupního souboru pomocí kterého se texty konvertují do UTF-8 (je viditelné v hlavičce vzniklé SQL dávky.
Vytvořenou dávku lze následně pohodlně importovat v prostředí postgres pomocí příkazu

```
\i file.sql;
```

kterým se vykoná celý obsah SQL souboru (v případě souboru který je výstupem z programu `shp2pgsql` příkaz `CREATE TABLE` a další, následované importy všech dat originálního Shapefile).

4 Vrstvy

Pro účely zpracování našeho projektu jsme zvolili následujících šest datových vrstev s definicemi o jaké vrstvy se jedná

- cyklotrasy - MultiLineString
- městské část - MultiPolygon
- objekty Městské Pořádkové policie - Point
- pěší trasy - MultiLineString
- veřejná WC - Point
- veřejné stavby - Point

Všechny specifikace těchto vrstev lze najít v poskytovaných metadatech.

5 Validita

Všechny tři typy *Point, MultiLineString, MultiLineString* jsou jedny z možných typů atributového sloupce ST_Geometry který je definován v ANSI SQL:1999 SQL Multimedia and Application Packages (SQLMM) a je dostupný za poplatek 60USD z [4]. Tento dokument dále zavádí, které geometrie jsou *simple* a které jsou dále *valid*. Dle specifikací to nemusí vždy být totéž což ilustrováno v [5]. Nebudeme zde opisovat jednotlivé případy kdy je jaká geometrie *simple* a *valid*. Pro zjištění jestli je dle výše uvedených definicí geometrie uložená ve sloupci geom (všechna data stažená z IPR Praha pro účely tohoto projektu již implicitně tento sloupce mají) validní slouží funkce `ST_IsValid` která zavádí specifikace [6]. Chcemeli tedy zjistit které záznamy v relaci nejsou valdní, použijeme příkaz

```
SELECT gid FROM table WHERE ST_IsValid(table.geom) = false;
```

který nám vypíše všechny primární klíče `gid` teré nemají validní geometrii. Dle definice *POSTGisu* v tomoto prostředí vrací funkce hodnotu `NULL` pokud i vstup ve sloupci geom je hondnoty `NULL`. Z tohoto důvodu se příkaz s pozměněnou hodnotou

```
SELECT gid FROM table WHERE ST_IsValid(table.geom) IN (NULL, false);.
```

V případě že bychom nalezli nějaké `gid` u nějž by topologie nebyla validní, je k dispozici další funkce a to `ST_MakeValid` která se, v případě že jí je jako argument předložena nevalidní geometrie, pokouší geometrii zvlaidovat bez ztráty vrcholů. Podporuje geometrie ST_Geometry *Point, MultiPoint, LineString, MultiLineString, Polygon, MultiPolygon, GeometryCollection*. Tuto funkci jsme ovšem nevyužili, neboť všecny záznamy ve všech našich relacích se ukázaly být validní. Podrobný popis funckí pro validitu lze nalézt na [7].

6 PgRouting

6.1 Nejkratší vzdálenost Dijkstra

Příklad 1: Vypočtěme délku trasy od veřejných záchodků v Petrovicích k veřejným záchodkům na Praze 17 (řekněme že jdeme na pěší výlet a dodržujeme pitný režim). Pro pohyb použijeme pěší trasy, pro výpočet nejkratší vzdálenosti Dijkstrův algoritmus [9]. Výsledek se pokusíme uvést co nejpřesněji vzhledem k lomovým bodům linií ve vrstvě `pesitrasy`.
Protože vrstva `pesitrasy` obsahuje po převodu pomocí programu `shp2pgsql` geometrii typu `MultiLineString` bylo nejprve potřeba ji převést pomocí funkce `ST_Dump` na jednodušší geometrii `LineString` [10]

```
CREATE TABLE pesi_single AS SELECT (st_dump(pesitrasy.geom)).geom FROM pesitrasy;
ALTER TABLE pesi_single ADD COLUMN gid serial;
ALTER TABLE pesi_single ADD PRIMARY KEY (gid);
CREATE INDEX ON pesi_single USING gist (geom);
```

abychom ji mohli použít v nadstavbě `pgrouting`. Pro použití přímo funkce `pgr_dijkstra` bylo dále nutné přidat do tabulky `pesi_single` další dva nové sloupce `source` a `target` jejich obsah dále vyplnit funkce `pgr_createTopology` která přiřadí indexy (čísla vrcholů grafu) a také byl přidán sloupec nákladů (úměrných délce geometrie)

```
ALTER TABLE pesi_single ADD COLUMN source INTEGER;
ALTER TABLE pesi_single ADD COLUMN target INTEGER;
SELECT pgr_createTopology('pesi_single','0.01','geom','gid');
ALTER TABLE pesi_single ADD COLUMN cost double precision;
UPDATE pesi_single SET cost = ST_length(geom);
```

Výběr bodů z bodové vrstvy `verejnawc` s použitím polygonové vrstvy městských částí `mestskecast` provedeme dotazem

```
CREATE TABLE wc_petr AS
SELECT A.* FROM verejnawc AS A
JOIN mestskecast AS B
ON st_within(A.geom,B.geom)
WHERE B.nazev LIKE 'Praha-Petrovice';

CREATE TABLE wc_17 AS
```

```
SELECT A.* FROM verejnawc AS A
JOIN mestskecast AS B ON
st_within(A.geom,B.geom)
WHERE B.nazev LIKE 'Praha 17' LIMIT 1;
```

kde jsme využili funkce pro práci s geometriemi `ST_Within` [12].
Protože chceme počítat trasu až k lomovému bodu poslední linie (a nikoli k poslendímu vrtexu) porovedeme identifikaci potřebných geometrií následným způsobem

1. identifikace nejbližšího LineStringu vzhledem k cílovému bodu
2. identifikace nejbližšího lomového bodu tohoto LineStringu vzhledem k cílovému bodu
3. identifikace nejbližšího vertexu grafu vzhledem k tomuto bodu.

Nejprve tedy identifikace příslušných LineStringů z vrstvy `pesi_single` [13, 14, 15, 16, 17, 18]

```
CREATE TABLE pesi_wc_petr_lin AS
SELECT netw.gid FROM
pesi_single AS netw, wc_petr AS pnt
WHERE ST_distance(netw.geom, pnt.geom) = (
SELECT min(st_distance(netw.geom, pnt.geom))
FROM pesi_single AS netw, wc_petr AS pnt);

CREATE TABLE pesi_wc_17_lin AS
SELECT netw.gid FROM
pesi_single AS netw, wc_17 AS pnt
WHERE ST_distance(netw.geom, pnt.geom) = (
SELECT min(st_distance(netw.geom, pnt.geom))
FROM pesi_single AS netw, wc_17 AS pnt);
```

Tabulka souřadnic lomových bodů identifikovaných LineStringů

```
CREATE TABLE pesi_wc_petr_xy AS
SELECT num AS id, ST_X(ST_PointN(geom,num)) AS x, ST_Y(ST_PointN(geom, num)) AS y
FROM pesi_single, (
SELECT generate_series(1, (
SELECT ST_NumPoints(geom)
FROM pesi_single, pesi_wc_petr_lin
WHERE pesi_single.gid = pesi_wc_petr_lin.gid)) AS num) AS series, pesi_wc_petr_lin
WHERE pesi_single.gid = pesi_wc_petr_lin.gid;

CREATE TABLE pesi_wc_17_xy AS
SELECT num AS id, ST_X(ST_PointN(geom,num)) AS x, ST_Y(ST_PointN(geom, num)) AS y
FROM pesi_single, (
SELECT generate_series(1, (
SELECT ST_NumPoints(geom)
FROM pesi_single, pesi_wc_17_lin
WHERE pesi_single.gid = pesi_wc_17_lin.gid)) AS num) AS series, pesi_wc_17_lin
WHERE pesi_single.gid = pesi_wc_17_lin.gid;
```

Nejbližší bod LineStringu k našemu bodu

```
CREATE TABLE pesi_start_pnt AS
SELECT ST_SetSRID(ST_Point(nodes.x,nodes.y),ST_SRID(pnt.geom)) AS geom
FROM pesi_wc_petr_xy AS nodes, wc_petr AS pnt
WHERE nodes.id = (
SELECT nodes.id
FROM pesi_wc_petr_xy AS nodes, wc_petr AS pnt
WHERE ST_distance(ST_SetSRID(ST_Point(nodes.x, nodes.y),ST_SRID(pnt.geom)),pnt.geom) = (
SELECT min(ST_distance(ST_SetSRID(ST_Point(nodes.x, nodes.y),ST_SRID(pnt.geom)),pnt.geom))
FROM pesi_wc_petr_xy AS nodes, wc_petr AS pnt));
```

```
CREATE TABLE pesi_end_pnt AS
SELECT ST_SetSRID(ST_Point(nodes.x,nodes.y),ST_SRID(pnt.geom)) AS geom
FROM pesi_wc_17_xy AS nodes, wc_17 AS pnt
WHERE nodes.id = (
SELECT nodes.id
FROM pesi_wc_17_xy AS nodes, wc_17 AS pnt
WHERE ST_distance(ST_SetSRID(ST_Point(nodes.x, nodes.y),ST_SRID(pnt.geom)),pnt.geom) = (
SELECT min(ST_distance(ST_SetSRID(ST_Point(nodes.x, nodes.y),ST_SRID(pnt.geom)),pnt.geom))
FROM pesi_wc_17_xy AS nodes, wc_17 AS pnt));
```

A nejbližší vertex k našemu bodů na LineStringu

```
CREATE TABLE pesi_start_vert AS
SELECT vertex.id
FROM pesi_single_vertices_pgr AS vertex, pesi_start_pnt AS pnt
WHERE ST_distance(vertex.the_geom,pnt.geom) = (
SELECT min(ST_distance(vertex.the_geom,pnt.geom))
FROM pesi_single_vertices_pgr AS vertex, pesi_start_pnt AS pnt);

CREATE TABLE pesi_end_vert AS
SELECT vertex.id
FROM pesi_single_vertices_pgr AS vertex, pesi_end_pnt AS pnt
WHERE ST_distance(vertex.the_geom,pnt.geom) = (
SELECT min(ST_distance(vertex.the_geom,pnt.geom))
FROM pesi_single_vertices_pgr AS vertex, pesi_end_pnt AS pnt);
```

Body `pesi_start_vert` a `pesi_end_vert` jsou body identifikující vrcholy grafu vytvořeného funkcí `pgr_createTopology` a proto je lze použít ve funkci `pgr_Dijkstra` k identifikaci výchozího a cílového bodu (pomocí subselectu)

```
CREATE TABLE pesi_shortest AS
SELECT seq, id1 AS node, id2 AS edge, cost
FROM pgr_dijkstra(
'SELECT gid AS id, source, target, cost FROM pesi_single',(
 SELECT pesi_start_vert.id FROM pesi_start_vert),(
 SELECT pesi_end_vert.id FROM pesi_end_vert),false,false);
```

Abychom dostáli zadání, dopočítali bychom vzdálenost přes lomové body linie od koncového vertexu nalezené cesty až k námi identifikovanému nejbližšímu bodu.
Pro `cyklotrasy` by byl postup obdobný (s využitím stejných funkcí), ovšem při testování této možnosti bylo zjištěno že funkce vrací prázdnou tabulku. To může být způsobeno například přítomností odděleného podgrafu který funkci brání provést vyhodnocení. Délku trasy nalezené pomocí Dijkstrova algoritmu získáme pomocí

```
SELECT sum(cost) FROM pesi_dijkstra;
```

dále tabulku s geometrií této cesty (např. pro vizualizaci) získáme

```
CREATE TABLE pesi_dijkstra_geom AS
SELECT A.* FROM pesi_single AS A
JOIN pesi_dijkstra AS B
ON A.gid = B.edge;
```

a například městské části (polygony) přes které po této trase pojedeme

```
CREATE TABLE pesi_dijkstra_casti AS
SELECT DISTINCT(A.gid),A.nazev,A.geom
FROM mestskecast AS A JOIN pesi_dijkstra_geom AS B
ON ( ST_Intersects(A.geom,B.geom) OR ST_Within(B.geom,A.geom));
```

6.2 Nejkratší vzdálenost A*

Příklad 2: Nalezněte nejkratší cestu mezi stejnými body pomocí algoritmu A*.
Pro využití funkce `pgr_astar` [8] je nutné k tabulce (grafu) přidat další sloupce se souřadnicemi počátečních a koncových vertexů

```
ALTER TABLE pesi_single ADD COLUMN x1 double precision;
ALTER TABLE pesi_single ADD COlUMN y1 double precision;
ALTER TABLE pesi_single ADD COlUMN x2 double precision;
ALTER TABLE pesi_single ADD COLUMN y2 double precision;

UPDATE pesi_single SET x1 = ST_X(ST_PointN(geom,1));
UPDATE pesi_single SET y1 = ST_Y(ST_PointN(geom,1));
UPDATE pesi_single SET x2 = ST_X(ST_PointN(geom, ST_NumPoints(geom)));
UPDATE pesi_single SET y2 = ST_Y(ST_PointN(geom, ST_NumPoints(geom)));.
```

Pokud jsou sloupce přidány, můžeme rovnou přejít k volání funkce `pgr_astar`

```
CREATE TABLE pesi_astar AS
SELECT seq, id1 AS node, id2 AS edge, cost
FROM pgr_astar('SELECT gid AS id, source, target, cost, x1, y1, x2, y2 FROM pesi_single',(
SELECT pesi_start_vert.id FROM pesi_start_vert)::integer,(
SELECT pesi_end_vert.id FROM pesi_end_vert)::integer,false,false);
```

Délku cesty již umíme vypočítat a tak můžeme například chtít zjistit, jestli jsou cesty nelezené pomocí různých algoritmů totožné [20]

```
SELECT A.gid FROM pesi_dijkstra_geom AS A
JOIN pesi_astar_geom AS B
ON ST_Equals(A.geom,B.geom)
WHERE ST_Equals(A.geom,B.geom) = false;
```

6.3 kDijkstra many to one

Verze Dijkstrova alortitmu *many to one* znamená, že hledáme pro každý ze skupiny výchozích bodů nejkratší cestu k bodu cílovému (nebo obráceně).
Příklad 3: Studenti pořádají závody. Každý student si vybere jeden z pražských záchodků jako své startovní místo. Studentů je o jednoho méně než záchodků a tak ten který zůstane se stane cílovým místem. Který ze studentů má největší šanci na výhru v závodě, vzhledem ke vzdálenosti startovního a cílového místa a podmínce pohybovat se pouze po pěších trasách? (určete pomocí Dijkstrova algoritmu ve verzi Many to One - pgr_kDijkstra).
Nejprve tedy jednoduše nasimulujeme výběr studentů (resp. nám stačí náhodně vybrat jeden záchodek který reprezentuje ten který po výběru zbyl) a k němu rovnou přiřadit číslo nejbližšího vertexu z topologie kterou vytvořila funkce `pgr_createTopology`

```
CREATE TABLE target_toal_vert AS
SELECT DISTINCT ON(A.gid) A.gid AS wcgid, B.id AS vertgid
FROM (
SELECT * FROM verejnawc
ORDER BY random() LIMIT 1
) AS A, pesi_single_vertices_pgr AS B
ORDER BY A.gid, ST_Distance(A.geom,B.the_geom);
```

dále pak pro všecny zbylé záchodky provedeme totéž

```
CREATE TABLE wc_nearest AS
SELECT DISTINCT ON(A.gid) A.gid AS wcgid ,B.id AS vertgid
FROM verejnawc AS A, pesi_single_vertices_pgr AS B, target_toal_vert AS C
WHERE A.gid <> C.wcgid
ORDER BY A.gid, ST_Distance(A.geom,B.the_geom);
```

Protože funkce `pgr_kDijkstraCost` a `pgr_kDijkstraPath` [21] vyžadují na vstupu pole `integerů` musíme náš sloupec s čísly vertexů převést na pole, tedy

```
CREATE TABLE array_wc_nearest AS
SELECT ARRAY(
SELECT wc_nearest.vertgid FROM wc_nearest) AS vertArray;
```

díky čemuž můžeme volat vyhodnocení nejkratších cest

```
CREATE TABLE pesi_dijkstra_many_cost AS
SELECT *
FROM pgr_kDijkstraCost('SELECT gid AS id, source, target, cost FROM pesi_single',(
SELECT target_toal_vert.vertgid FROM target_toal_vert)::integer,(
SELECT array_wc_nearest.vertArray FROM array_wc_nearest)::integer[],false,false);
```

Funkce `pgr_kDijkstraCost` vrací sekvenci řádků vždy s počátečním vrcholem, koncovým a cenou cesty. Pro vyhledání velikosti nejkratší cesty toto stačí. Neexistuje-li pro danou dvojici zdroj a cíl cesta (mezery v grafu) je cena rovna -1.0. Dále se díky charakteru mohlo stát, že dva veřejné záchodky sdílely stejný nástupní vrchol na pěší trase, a proto budeme eliminovat také ty výsledky kde je cena rovna 0. Pro získání cesty tedy zadáme

```
CREATE TABLE nearest_start AS
SELECT seq FROM pesi_dijkstra_many_cost
WHERE cost <> -1 AND cost <> 0
ORDER BY cost ASC LIMIT 1;
```

Ovšem můžeme získat také přímo nejkratší cestu jako geometrii [22, 23] (a z ní po sléze získat délku), a to pomocí funkce `pgr_kDijkstraPath` [21]

```
CREATE TABLE nearest_start_path AS
SELECT id1 AS path, ST_AsText(ST_LineMerge(ST_Union(b.geom))) AS geom
FROM pgr_kDIjkstraPath('SELECT gid AS id, source, target, cost FROM pesi_single',(
SELECT target_toal_vert.vertgid FROM target_toal_vert)::integer,(
SELECT array_wc_nearest.vertArray FROM array_wc_nearest)::integer[],false,false) AS a, pesi_single AS b, nearest
WHERE a.id3 = b.gid AND a.seq = c.seq
GROUP BY id1
ORDER BY id1;
```

což nám vrátí tabulku s jedním řádkem obsahujícím multigeometrii pro námi zvolený startovní záchodek s nejkratší existující nenulovou cestou k cílovému záchodku.

6.4 Ohodnocení s rychlostí

Vrstva `cyklo_single` (původně `cyklotrasy` obsahuje atribut `dopr_stav` kterým je určen typ omezení nebo charakter cyklotrasy. Následujcí hodnoty atributu (převzato z metadat vrstvy [24]) slouží pro pozměněnou rychlost na 2 (obecně je 3, bez fyzikálního rozměru)

- 6 - stezka pro chodce a cyklisty společná
- 9 - přejezd přes křižovatku nebo silnici
- 16 - cyklotrasa na komunikaci s dopravním stresem

neboť zde cyklisté musejí dbát zvýšené pozornosti a omezit rychlost jízdy (nebezpečí nehody nebo ohrození chodce). Pro následující hodnoty atributu omezíme rychlost na 1, neboť cyklisté zde buď musí kolo vést, nebo jsou omezeni rychlostí přívozu

- 15 - cyklisto veď kolo
- 19 - přívoz
- 20 - chodník.

Přiřadíme tedy rychlosti a vypočteme nové náklady

```
ALTER TABLE cyklo_single ADD COLUMN rychlost integer;
UPDATE cyklo_single SET rychlost = 3 WHERE dopr_stav NOT IN (6,9,16,15,19,20);
UPDATE cyklo_single SET rychlost = 2 WHERE dopr_stav IN (6,9,16);
UPDATE cyklo_single SET rychlost = 1 WHERE dopr_stav IN (15,19,20);

ALTER TABLE cyklo_single ADD COLUMN cost2 double precision;
UPDATE cyklo_single SET cost2 = (ST_length(geom)/rychlost)::float;
```

Příklad 4: Na Praze 15 jsou právě dve policejní stanice. Policisté potřebují mezi stanicemi převážet důležité dokumenty a v zájmu zdraví policistů se pohybují po cyklostezkách. Nalezněte nejkratší cestu mezi obema stanicemi za předpokladu, že policisté ctí omezení a doporučení plynoucí z dopravního stavu cyklotrasy.
Vybereme policejní stanice na Praze 15

```
CREATE TABLE mpp_p15 AS
SELECT * FROM obj_mpp
WHERE ST_Contains((
SELECT geom FROM mestskecast
WHERE nazev LIKE 'Praha 15'),obj_mpp.geom) = true;
```

a přiřadíme k nim čísla nejbližších vertexů sítě cyklostezek (pgr_createTopology)

```
CREATE TABLE mpp_nearest AS
SELECT DISTINCT ON(A.gid) A.gid AS mppgid, B.id AS vertgid
FROM mpp_15 AS A, cyklo_single_vertices_pgr AS B
ORDER BY A.gid, ST_Distance(A.geom,B.the_geom);
```

Nejkratší cestu nalezneme opět pomocí Dijkstrova algoritmu

```
CREATE TABLE mpp_dijkstra2 AS
SELECT seq, id1 AS node, id2 AS edge, cost
FROM pgr_dijkstra('SELECT gid AS id, source, target, cost2 AS cost FROM cyklo_single',(
SELECT vertgid FROM mpp_nearest LIMIT 1),(
SELECT vertgid FROM mpp_nearest LIMIT 1 OFFSET 1),false,false);
```

pro který jsme možnosti získání geometrie ukázali již v předchozích příkladech.

6.5 Driving distance

Příklad 5: Šerif Wiggum z Prahy 21 se rozhodl že pojede navštívit své kolegy v ostatních městských částech. Ví ovšem, že nikdy na kole neujel více než 12 km, a tak ho zajímá ke kterým policejním stanicím má šanci dojet. Úlohu řešte použitím funkce driving_distance.
Nejprve tedy vyhledáme Clancyho policejní stanici

```
CREATE TABLE mpp_p21 AS
SELECT * FROM obj_mpp
WHERE ST_Contains((
SELECT geom FROM mestskecast
WHERE nazev LIKE 'Praha 21'),obj_mpp.geom) = true;
```

dále k ostatním stanicím přiřadíme vertexy sítě cyklotras (vzdálenost od nástupního vertexu k cílové stanici již nepočítáme)

```
CREATE TABLE mpp_nearest_all AS
SELECT DISTINCT ON(A.gid) A.gid AS mppgid ,B.id AS vertgid
FROM obj_mpp AS A, cyklo_single_vertices_pgr AS B
ORDER BY A.gid, ST_Distance(A.geom,B.the_geom);
```

a pomocí funkce `pgr_drivingDistance` [25] nalezneme vrcholy které jsou v dané dojezdové vzdálenosti

```
CREATE TABLE mpp_driving_res AS
SELECT * FROM pgr_drivingDistance('
SELECT gid AS id, source, target, cost FROM cyklo_single',(
SELECT vertgid FROM mpp_nearest_all AS A, mpp_p21 AS B
WHERE A.mppgid = B.gid), 12000, false, false);
```

ze kterých můžeme udělat vrstvu s geometrií vertexů

```
CREATE TABLE mpp_driving_geom AS
SELECT A.* FROM cyklo_single_vertices_pgr AS A
JOIN mpp_driving_res AS B
ON B.id1 = A.id;
```

a také vrstvu s geometrií těch stanic, které mají jako nejbližší vrchol sítě nějaký z nalezených

```
CREATE TABLE mpp_driving_stat AS
SELECT A.* FROM obj_mpp AS A
JOIN mpp_nearest_all AS B
ON A.gid = B.mppgid
JOIN mpp_driving_res AS C
ON B.vertgid = C.id1;
```

Můžeme se například také dozvědět, do kterých městských částí by se šerif mohl podívat

```
CREATE TABLE mpp_driving_part AS
SELECT DISTINCT(A.nazev), A.geom FROM mestskecast AS A, mpp_driving_stat AS B
WHERE ST_Contains(A.geom,B.geom);
```

7 Závěr

V rámci semestrální práce jsme si jednak vyzkoušeli práci s Postgres databází a dále jsme se naučili používat některé funkce pro práci s geometriemi které poskytuje PostGIS. Jako naše osobní zaměření jsme si vybrali problematiku síťových analýz, i přes to že tato probelmatika byla probírána již v pracech z roku 2012 a 2013. Motivovala nás k tomu především změna a rozšíření funkcí PgRoutingu které přinesla verze 2.1 a oproti předchozím pracem jsme použili větší množství algoritmů. Druhotným produktem naší práce je pak velké množství nejrůznějších prostorových a atributových dotazů které nám nestálo za to formulovat do příkladů. Příkladů samotných je v naší práci celkem 5, přičemž jsme se snažili aby každý z příkladů ilustroval jinou funkci kterou PgRouting nabízí. Jsme si vědomi že práce není málo ale ani nikterak velké množství, což nás mrzí neboť nás práce na projektu převážně bavila, je to ale způsobeno především nejrůznějšími komunikačními problémy mezi členy týmu i jinými.
Závěrem děkujeme Ing. Landovi, Ph.D. za ochotnou spolupráci a posyktnuté rady.

Reference

[1] Pražská otevřená data, dostupné z:
http://www.geoportalpraha.cz/cs/clanek/271/prazska-otevrena-data#.Vp_-wF4oDtS
citováno dne: 20.1.2016

[2] Licenční podmínky pro Pražská otevřená data, dostupné z:
http://www.geoportalpraha.cz/cs/clanek/276/licencni-podminky-pro-otevrena-data#.VqKWjl4oDtQ
citováno dne: 22.1.2016

[3] American National Standarts Institute, dostupné z:
www.ansi.org
citováno dne: 23.1.2016

[4] Information technology - Database languages - SQL multimedia and application packages - Part 3: Spatial, dostupné z:
http://webstore.ansi.org/RecordDetail.aspx?sku=INCITS%2fISO%2fIEC+13249-3%3a2011[2012]
citováno dne: 23.1.2016

[5] Ensuring OpenGIS compliancy of validity, dostupné z:
http://postgis.net/docs/using_postgis_dbmanagement.html#OGC_Validity
citováno dne: 23.1.2016

[6] OGC - Simple Feature Access - Part 2: SQL Option, dostupné z:
http://www.opengeospatial.org/standards/sfs
citováno dne: 23.1.2016

[7] POSTGis Manual, dostupné z:
http://postgis.net/docs/
citováno dne: 23.1.2016

[8] Shortest Path A^*, dostupné z:
http://docs.pgrouting.org/2.1/src/astar/doc/index.html#pgr-astar
citováno dne: 23.1.2016

[9] Dijkstrův algoritmus, dostupné z:
http://docs.pgrouting.org/2.1/src/dijkstra/doc/index.html#pgr-dijkstra
citováno dne: 7.2.2016

[10] PostGIS funkce ST_Dump, dostupné z:
http://postgis.net/docs/ST_Dump.html
citováno dne: 7.2.2016

[11] PGRouting funkce pgr_createTopology, dostupné z:
http://docs.pgrouting.org/2.1/src/common/doc/functions/create_topology.html#pgr-create-topology
citováno dne: 7.2.2016

[12] PostGIS funkce ST_Within, dostupné z:
http://postgis.net/docs/manual-2.0/ST_Within.html
citováno dne: 7.2.2016

[13] PostGIS funkec ST_Distance, dostupné z:
http://postgis.net/docs/ST_Distance.html
citováno dne: 7.2.2016

[14] PostGIS funkce ST_SetSRID, distupné z:
http://postgis.net/docs/manual-2.0/ST_SetSRID.html
citováno dne: 7.2.2016

[15] PostGIS funkce ST_SRID, dostupné z:
http://postgis.net/docs/ST_SRID.html
citováno dne: 7.2.2016

[16] PostGIS funkce ST_Point, dostupné z:
http://postgis.net/docs/ST_Point.html
citováno dne: 7.2.2016

[17] Postgres funkce generate_series, dostupné z:
http://www.postgresql.org/docs/9.1/static/functions-srf.html
citováno dne: 7.2.2016

[18] PostGIS funkce ST_PointN, dostupné z:
http://postgis.net/docs/ST_Point.html
citováno dne: 7.2.2016

[19] PostGIS funkce ST_NumPoints, dostupné z:
postgis.net/docs/manual-2.0/ST_NumPoints.html
citováno dne: 7.2.2016

[20] PostGIS funkce ST_Equals, dostupné z:
http://postgis.net/docs/ST_Equals.html
citováno dne: 7.2.2016

[21] PgRouting funkce pgr_kDijkstra, dostupné z:
http://docs.pgrouting.org/2.1/src/kdijkstra/doc/index.html#pgr-kdijkstra
citováno dne: 9.2.2016

[22] PostGIS funkce ST_LineMerge, dostupné z:
http://postgis.net/docs/manual-2.0/ST_LineMerge.html
citování dne: 9.2.2016

[23] PostGIS funkce ST_Union, dostupné z:
http://postgis.net/docs/manual-2.1/ST_Union.html
citováno dne: 9.2.2016

[24] Metadata liniové vrstvy cyklotrasy, dostupné z:
http://www.geoportalpraha.cz/cs/fulltext_geoportal/id/%7B48409BAA-9EE2-4099-86B1-1C177C3EFD50%7D#.VrrpXV4oDtQ
citováno dne: 9.2.2016

[25] PgRouting funkce pgr_drivingDistance, dostupné z:
http://docs.pgrouting.org/2.1/src/driving_distance/doc/dd_driving_distance_v3.html#pgr-driving-distance-v3
citováno dne: 9.2.2016