České vysoké učení technické
Fakulta stavební, Obor geoinformatika
Katedra mapování a kartografie

DOKUMENTACE

Úvod do zpracování prostorových dat

Semestrální projekt

Skupina b12
Tereza Fiedlerová
Pavel Klimunda
Jan Špička

letní semestr 2011/2012

Obsah

1 Zadání

- Navrhněte a vytvořte tématické vrstvy (např. vodní toky, vodní plochy, lesy, silnice, železnice a pod.) na základě dat OSM (viz cvičná databáze pgis_student schéma osm). Pro tento účel byla na serveru 'geo102' založena databáze pgis_uzpd.
- Aplikujte testy datové integrity a odstraňte případné nekonzistence v datech.
- Vytvořte tutoriál pro výuku PostGIS - tj. sadu atributových a prostorových dotazů nad databází *pgis_uzpd*.

2 Použitý software

PostGIS v. 2.0.0

pgRouting v. 1.05

PgAdmin III v. 1.12.2

Quantum GIS v. 1.7.4-Wroclaw

3 Úvod

3.1 PostGIS a pgRouting

PostGIS je rozšíření objektově-relačního databázového systému PostgreSQL sloužící k ukládání a zpracování prostorových (geografických) dat. PgRouting je nadstavba PostGISu umožňující provádění síťových analýz. Umožňuje především vyhledání nejkratší cesty, ale při kompletní instalaci všech částí také řešení problému obchodního cestujícího, tvorbu izočar pro vzdálenost aj.

3.2 OSM

OSM - OpenStreetMap je projekt sloužící k tvrobě a vizualizaci geografických dat, kam může každý ukládat data naměřená nejčastěji ručním GPS přijímačem. Do PostGISu lze tato data importovat pomocí aplikace osm2pgsql. V databázi *pgis_uzpd* jsou tato data již importována ve schématu *osm*.

3.3 Náš cíl

Cílem tohoto projektu je vytvořit tématické vrstvy související především s dopravou a upravit data tak, aby bylo možné nad nimi provádět různé analýzy. To spočívá v základní validaci dat a vytvoření nezbytné topologie pro síťové analýzy. Dalším úkolem projektu je vytvoření několika ukázkových prostorových dotazů a především dotazů týkajících se síťových analýz s využitím nadstavby pgRouting.

4 Vytvoření vrstev

Po seznámení s daty OSM z nich byly vybrány následující tématické vrstvy:

1. Bodové vrstvy:

   benzinky

   parkoviste

   motorismus

2. Liniové vrstvy:

   silnice

3. Polygonové vrstvy:

trava

prum_zony

K vrstvám ze schématu *osm* byla přidána ještě vrstva obcí ze schématu *gis1*, a to bodová i polygonová:

4. Vrstvy z gis1:

   obce_bod

   obce

Obsah vrstev je zřejmý z názvů jednotlivých vrstev. Upřesnit je třeba snad jen vrstvu *motorismus*, která obsahuje opravny aut, automyčky a obchody s autodíly. Vzhledem k nízkému množství dat v těchto třech kategoriích byly sloučeny do společné tématické vrstvy. Dále je třeba dodat, že do liniové vrstvy silnice byly pro větší přehlednost zahrnuty pouze dálnice, rychlostní silnice a silnice 1. a 2. třídy.

4.1 Postup vytvoření

Vytvoření vrstev bylo provedeno následujícím postupem (uveden příklad pro vrstvu silnic). Kompletní zdrojové kódy pro vytvoření všech vrstev jsou uvedeny v přiloženém SQL souboru.

- Výběrem příslušných prvků z tabulky *czech_line* ze schématu *osm* byla vytvořena tabulka silnic.

```
CREATE TABLE b12.silnice AS
 SELECT osm_id, name AS nazev, highway AS popis, geom
 FROM osm.czech_line
 WHERE highway IN ('motorway', 'motorway_link', 'trunk', 'trunk_link', 'primary',
                   'primary_link', 'secondary', 'secondary_link');
```

- Poté byl do tabulky přidán primární klíč.

```
ALTER TABLE b12.silnice ADD PRIMARY KEY (osm_id);
```

- Nakonec byl vytvořen prostorový index.

```
CREATE INDEX silnice_geom ON silnice USING gist (geom);
```

Všechny vrstvy z dat OSM byly vytvořeny se sloupci osm_id, nazev, popis a geom. Sloupec osm_id je typ integer a označuje identifikátor, který byl nastaven i jako primární klíč. Ve sloupečku nazev je název (v OSM jako name) ukládaný jako textový řetězec, v mnoha případech je tato hodnota nevyplněná (hodnota NULL). Do sloupce popis jsou ukládány informace o druhu dat ve formátu textového řetězce, příkladem může být popis *„motorway“* přiřazený k dálnicím v tabulce silnic. Geometrie objektů je uložena ve sloupečku geom.

V případě vrstvy *motorismus*, kam byla ukládána data s různými klíčovými slovy ('amenity', 'shop'), bylo třeba nejdříve vložit prvky se zadanou hodnotou jednoho klíče ('amenity') a poté data s příslušnou hodnotou druhého klíče ('shop'). Přitom ovšem musel být ošetřen duplicitní zápis dat (prvky s hledanými hodnotami obou klíčů). Pro jednoduchost byl typ objektu ve sloupci *amenity* zvolen jako prioritní, tzn. že datům s příslušnou hodnotou v obou sloupcích byla ponechána hodnota ze sloupce *'amenity'*.

```
CREATE TABLE motorismus AS
 SELECT osm_id, name AS nazev, amenity AS popis, geom FROM czech_point
 WHERE amenity ='car_wash';

INSERT INTO motorismus (osm_id, nazev, popis, geom)
 SELECT osm_id, name AS nazev, shop AS popis, geom FROM czech_point
 WHERE shop IN ('car_parts','car_repair') AND amenity != 'car_wash';
```

4.2 Převedení obcí na multiprvky

Pro správné výsledky analýz prováděných nad vrstvou *obce* ze schématu *gis1* bylo nutné převést geometrii prvků z typu 'Polygon' na 'MultiPolygon'. Důvodem je, že některé obce jsou vyjádřeny více polygony. Tento krok byl proveden následujícím postupem:

- Vytvoření dočasné tabulky s multipolygony:

```
CREATE TABLE obce_tmp1 AS SELECT kodob,ST_union(geom) AS geom FROM obce GROUP BY kodob;
```

- Zkopírování tabulky *obce* a příprava sloupců s typem geometrie 'MultiPolygon':

```
CREATE TABLE obce_tmp2 AS SELECT * FROM obce;
DELETE FROM obce_tmp2 WHERE ctid NOT IN (SELECT MAX(ctid) FROM obce GROUP BY kodob);
SELECT DropGeometryColumn('obce_tmp2', 'geom');
SELECT AddGeometryColumn('obce_tmp2', 'geom', 2065, 'MultiPolygon', 2);
```

- Překopírování geometrie z dočasné tabulky:

```
UPDATE obce_tmp2 AS a SET geom = ST_multi(b.geom) FROM obce_tmp1 AS b
 WHERE a.kodob = b.kodob;
```

- Smazání dočasné tabulky:

```
DROP TABLE obce_tmp1;
```

- Přejmenování nové tabulky a nastavení primárního klíče:

```
ALTER TABLE obce_tmp2 RENAME TO obce;
ALTER TABLE obce ADD PRIMARY KEY(ogc_fid);
```

4.3 Transformace

Data ze schématu *osm* jsou uložena v souřadnicovém systému Google Mercator (kód srid 900913). To však neplatí o vrstvách ze schématu *gis1*, které mají souřadnicový systém S-JTSK (kód srid 2065). Proto bylo třeba provést transformaci těchto vrstev do systému Google Mercator, a to následujícím postupem (uveden pouze případ polygonové vrstvy):

```
ALTER TABLE b12.obce RENAME COLUMN geom TO geom1;
SELECT AddGeometryColumn('obce', 'geom', 900913, 'multipolygon', 2);
UPDATE obce SET geom = ST_Transform(geom1, 900913);
SELECT DropGeometryColumn('obce','geom1');
```

Poznámka:

Po vizualizaci dat z vrstev *obce* a *obce_bod* byl zjištěn posun oproti datům ze schématu *osm*. Tato chyba je pravděpodobně dána nepřesnou definicí souřadných systémů v tabulce *spatial_ref_sys*. Dočasně byl problém byl vyřešen tak, že data byla transformována v programu *QGIS* a následně vyexportována jejich geometrie ve formátu WKT do textového souboru. Poté byla nová geometrie vložena do tabulky v naší databázi do sloupečku geom.

Pro bodovou vrstvu *obce_bod* byl použit příkaz:

```
UPDATE obce_bod
 SET geom = ST_GeomFromText('POINT(1605694.96282113622874 6461514.72681927587837)', 900913)
 WHERE ogc_fid = 1;
```

Pro polygonovou vrstvu *obce* pak:

```
UPDATE obce
 SET geom = ST_GeomFromText('MULTIPOLYGON (((1920326.893898334819824 6404670.577448925934732,
 1918900.472809662576765 6404808.423255076631904,1917909.86095187580213 6406810.109475350938737,
 1918853.132074539316818 6407155.18160492926836,1919031.419374293182045 6406884.865542201325297,
 1919407.649014645488933 6407024.65515595767647,1920145.921946217073128 6406333.285755157470703,
 1920326.893898334819824 6404670.577448925934732)))', 900913 )
 WHERE ogc_fid = 3075;
```

Takto byla provedena aktualizace položky geom ve všech řádcích tabulky.

Problém s definicí souřadných systémů je nyní už vyřešen. Proto výše uvedený postup s transformací v programu *QGIS* již není nutný.

4.4 Převedení polygonových vrstev na bodové

Některé druhy dat se nacházely ve schématu *osm* jak v tabulce *czech_points* tak v tabulce *czech_polygons*, tedy ve formě bodů i polygonů. Jednalo se o data pro vrtsvy *benzinky, parkoviste* a *motorismus*. Bylo proto nutné tato data převést na bodová (vzhledem k zamýšleným analýzám a velikosti území by tato data ve formě polygonů nebyla přínosná). K tomu byla využita funkce *ST_PointOnSurface()*.

Nejdříve byla vytvořena dočasná tabulka s příslušnými daty ve formě polygonů (příklad pro vrstvu benzinek):

```
CREATE TABLE b12.benzinky_pol AS
 SELECT osm_id, name AS nazev, amenity AS popis, geom FROM czech_polygon
 WHERE amenity = 'fuel';

 ALTER TABLE b12.benzinky_pol ADD PRIMARY KEY (osm_id);
```

Přitom byla provedena kontrola, jestli se tu neobjevují duplicitní data. Bylo testováno, zda nějaký prvek z bodové vrstvy neleží uvnitř elementu z vrstvy polygonové, a případně byly tyto řádky odstraněny. Tento krok bylo nutné provést ještě před přidáním centroidů polygonů do bodové vrstvy. Aby správně fungovala funkce *ST_Contains()* a následně funkce *ST_PointOnSurface()*, musela být také zkontrolována validita dat dočasných polygonových vrstev. Nevalidní polygon se objevil pouze ve vrstvě *motorismus*. Jeho oprava byla provedena funkcí *ST_MakeValid()* (detailnější informace o opravě nevalidních dat jsou uvedeny v následující podkapitole).

- Kontrola validity polygonové vrstvy:

  ```
  SELECT ST_IsValidReason(geom) FROM benzinky_pol
   WHERE ST_IsValid(geom) = false;
  ```

- Odstranění duplicitních dat:

  ```
  DELETE FROM benzinky_pol WHERE osm_id IN
   (SELECT b.osm_id FROM benzinky_bod AS a
   JOIN benzinky_pol AS b
   ON ST_Contains(b.geom, a.geom));
  ```

- Vložení centroidů polygonové vrstvy do vrstvy bodové:

  ```
  INSERT INTO benzinky_bod
   (SELECT osm_id, nazev, popis, ST_PointOnSurface(geom) AS geom FROM benzinky_pol);
  ```

- Smazání polygonové vrstvy:

  ```
  DROP TABLE IF EXISTS benzinky_pol;
  ```

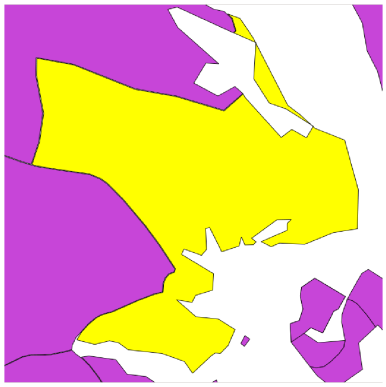
5 Validace dat

Pro provádění různých dotazů nad daty v PostGISu je vhodné, aby cílová data měla validní geometrii. To lze zajistit pomocí implementovaných funkcí *ST_IsValid()*, *ST_IsValidReason()* a *ST_MakeValid()*. Pomocí první uvedené funkce byly vyhledány v každé tématické vrstvě všechny nevalidní prvky. Nevalidní data se nacházela pouze v polygonových vrstvách. Chyba v geometrii byla zjištěna funkcí *ST_IsValidReason()*. Na opravu tohoto problému slouží v PostGISu nově od verze 2.0 funkce *ST_MakeValid()*.

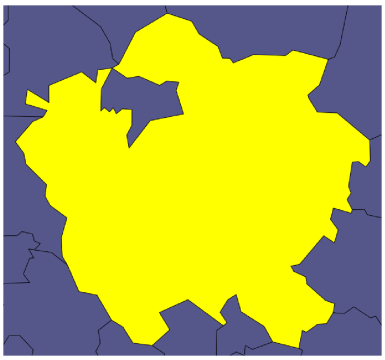
V našich datech se vyskytovaly následující chyby validity:

```
Self-intersection
Ring Self-intersection
```

obrázek 5.1 - self-intersection



obrázek 5.2 - ring self-intersection



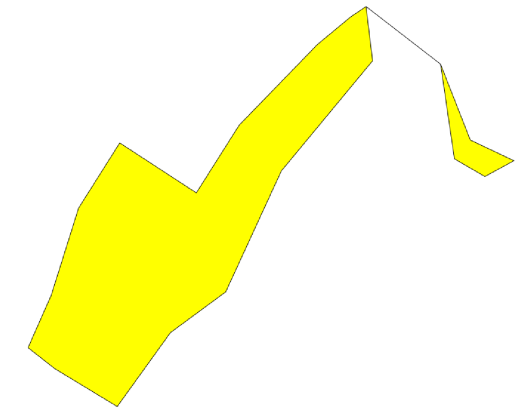
Funkce *ST_MakeValid()* řeší tyto problémy převedením jednoduchých prvků na multiprvky. Toto řešení šlo bez problémů použít u nevalidních dat ve vrstvě *prum_zony* a ve vrstvě *obce*, v nichž se v každé objevil jeden nevalidní prvek.

Do příslušných tabulek byl přidán sloupeček s geometrií typu multipolygon a do něho uložen výsledek této funkce, popřípadě původní geometrie objektu převedená na multiprvek pomocí funkce *ST_Multi()*.

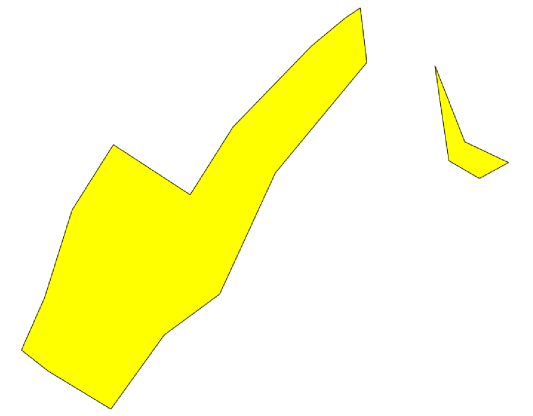
5.1 Validace vrstvy *trava*

Ve vrstvě *trava* však funkce *ST_MakeValid()* některé polygony převedla na typ 'GeometryCollection'. Po prohlédnutí těchto dat v QGISu bylo zjištěno, že některé polygony jsou tvořeny i liniovými částmi. Jelikož takovéto části nemají v dané vrstvě příliš smysl, byly tyto linie z řešení odstraněny a ponechány pouze polygony. K tomu byly využity funkce *ST_GeometryType()* a *ST_CollectionExtract()*. Prostřednictvím první uvedené funkce byly nalezeny všechny prvky typu 'GeometryCollection' získané validací polygonů. Následně byly z těchto prvků zachovány pouze polygony použitím funkce *ST_CollectionExtract()*.

obrázek 5.1.1 před validací



obrázek 5.1.2 po validaci



Příklad celého postupu validity je uveden u vrstvy *trava*, přičemž první dotaz slouží pouze ke zjištění chyb. Validace ostatních vrstev je v přiloženém SQL souboru.

- Výpis nevalidních polygonů včetně chyby:

```
SELECT osm_id, ST_IsValidReason(geom) AS reason FROM trava
 WHERE ST_IsValid(geom) = FALSE;
```

- Výsledek dotazu:

```
osm_id     | reason
-------------------------------------------------------------------
40981382   | "Self-intersection[1356407.77441725 6492899.36355119]"
109991951  | "Self-intersection[1359126.30622222 6491837.73755809]"
127256036  | "Ring Self-intersection[1844527.95 6246921.4]"
84067907   | "Self-intersection[1893399.00137311 6261627.80224262]"
...
```

- Přidání sloupce s typem geometrie 'MultiPolygon':

```
ALTER TABLE trava RENAME COLUMN geom TO geom1;
SELECT AddGeometryColumn('trava','geom',900913,'multipolygon',2,true);
```

- Převedení prvků na multipolygony a odstranění liniových či bodových prvků z řešení:

```
UPDATE trava SET geom = ST_Multi(geom1);
UPDATE trava SET geom = ST_CollectionExtract(ST_MakeValid(geom),3)
 WHERE ST_IsValid(geom) = false
 AND ST_GeometryType(ST_MakeValid(geom)) = 'ST_GeometryCollection';
```

- Validace ostatních prvků a odstranění sloupce s původní geometrií:

```
UPDATE trava SET geom = ST_MakeValid(geom)
 WHERE ST_IsValid(geom) = FALSE;
SELECT DropGeometryColumn('trava', 'geom1');
```

Poznámka:

Zajímavostí je, že se chyba objevila i ve vrstvě *obce* ze schématu *gis1*. Po hledání důvodu bylo zjištěno, že typ polygonu s dírou dotýkající se v jednom bodě vnější hranice polygonu (viz obrázek 5.2) je podle ESRI topologicky správný, pokud je uložen jako polygon bez díry, zatímco podle OGC je třeba u něho uložit zvlášť vnější a vnitřní hranici.

6 pgRouting

6.1 Příprava dat pro síťové analýzy

Nejříve bylo třeba liniovou vrstvu, nad níž měly být analýzy prováděny, upravit tak, aby s ní pgRouting uměl pracovat. Tento krok spočívá v přiřazení počátečního (source) a koncového (target) uzlu každé linii. To bylo uskutečněno přidáním sloupců *source* a *target* do tabulky *silnice* a následným spuštěním funkce *assign_vertex_id()*, která vytvoří tabulku s uzlovými body a jejich id uloží do sloupců *source* a *target*. Pro používání různých algoritmů (*A-Star, Shooting Star*) pro výpočet nejkratší cesty bylo nutné přidat ještě další sloupce se souřadnicemi uzlových bodů a náklady.

Úprava vrstvy *silnice* pro síťové analýzy:

- Doplnění nezbytných sloupců:

```
ALTER TABLE silnice ADD COLUMN source INTEGER;
ALTER TABLE silnice ADD COLUMN target INTEGER;
ALTER TABLE silnice ADD COLUMN length FLOAT;
```

- Vytvoření indexů:

```
CREATE INDEX source_idx ON silnice(source);
CREATE INDEX target_idx ON silnice(target);
```

- Vytvoření topologie (uzlových bodů) a spočtení délky jednotlivých úseků:

```
SELECT assign_vertex_id('b12', 'silnice', 1, 'geom', 'osm_id');
UPDATE silnice SET length = ST_Length(geom);
```

- Přidání sloupců pro použití algoritmů *A-Star* a *Shooting Star*:

```
-- pro a-star
ALTER TABLE silnice ADD COLUMN x1 DOUBLE PRECISION;
ALTER TABLE silnice ADD COLUMN y1 DOUBLE PRECISION;
ALTER TABLE silnice ADD COLUMN x2 DOUBLE PRECISION;
ALTER TABLE silnice ADD COLUMN y2 DOUBLE PRECISION;

UPDATE silnice SET x1 = ST_x(ST_startpoint(geom));
UPDATE silnice SET y1 = ST_y(ST_startpoint(geom));
UPDATE silnice SET x2 = ST_x(ST_endpoint(geom));
UPDATE silnice SET y2 = ST_y(ST_endpoint(geom));

-- pro shooting star
ALTER TABLE silnice ADD COLUMN reverse_cost DOUBLE PRECISION;
UPDATE silnice SET reverse_cost = LENGTH;

ALTER TABLE silnice ADD COLUMN to_cost DOUBLE PRECISION;
ALTER TABLE silnice ADD COLUMN rule text;
```

Poznámka:

Při prvním pokusu o vytvoření topologie síťě funkce *assign_vertex_id()* neproběhla a skončila chybovou hláškou:

```
ERROR:  find_srid() - couldnt find the corresponding SRID - is the geometry
registered in the GEOMETRY_COLUMNS table?  Is there an uppercase/lowercase
missmatch?
```

Po konzultaci s vedoucím projektu bylo zjištěno, že tato chyba je způsobena tím, že daná funkce předpokládá práci s tabulkou ve schématu *public*. Proto byla funkce Ing. Martinem Landou upravena tak, aby bylo možné zadat schéma, v němž je umístěna tabulka pro síťovou analýzu, jako jeden z argumentů funkce.

6.2 Vyhledání 'správného' uzlu

Funkce pro vyhledávání nejkratší cesty rozšíření pgRouting vyžadují na vstupu jako argument počáteční (*source*) a koncový (*target*) uzel. Při hledání nejkratších cest však běžně není požadavkem vyhledání cesty mezi danými uzlovými body či úseky silnic, ale vyhledání cesty mezi dvěma místy, která mohou být charakterizována body vzdálenými i několik desítek metrů od silnice. Proto bylo třeba najít vhodný způsob, jak zjistit výchozí a cílový uzlový bod sítě silnic pro daná místa.

Metodou s nejpřesnějšími výsledky by bylo pravděpodobně nalézt nejbližší úsek silnic a na něm nejbližší bod zadanému místu (to lze provést pomocí funkcí *ST_Line_Interpolate_Point()* a *ST_Line_Locate_Point()*. Následně by pak bylo třeba tento bod vložit do tabulky silnic jako dočasný uzlový bod, popřípadě nejdříve najít všechny takové body pro zájmová místa (např. všechny benzinky) a vytvořit topologii sítě silnic znovu s uvážením těchto uzlových bodů. Otázkou však zůstává, zda je nutné snažit se o takovou přesnost vzhledem k uvážení přesnosti použitých dat a časové náročnosti postupu.

V projektu je proto zvolen jiný postup. Pro pgRouting existuje funkce *find_node_by_nearest_link_within_distance()*, jejíž zdrojový kód lze nalézt na stránkách [5]. Tato funkce slouží k nalezení nejbližšího existujícího uzlového bodu síťě cest ke zvolenému bodu. Navíc lze zadat také toleranci, v jaké maximální vzdálenosti se tento uzlový bod může nacházet. Použití funkce je zřejmé z dotazů, viz kapitola 7.3.

Pro náš případ bylo nutné zdrojový kód této funkce upravit tak, aby funkce vracela správné hodnoty. Pro funkčnost *find_node_by_nearest_link_within_distance()* bylo také třeba dodefinovat další funkci *find_nearest_link_within_distance()* z [5] a rovněž u ní provést několik úprav. Použité zdrojové kódy obou funkcí jsou uvedeny v příloze.

Popis změn provedených ve zdrojových kódech výše uvedených funkcí:

- Přejmenování proměnných the_geom na geom, gid na osm_id, aby odpovídaly názvům používaným v našem projektu.
- Nahrazení funkce *GetSrid()* funkcí *St_Srid()*.
- Připojení prefixu 'St' k funkcím *GeometryFromText(), X(), Y(), Distance(), SetSrid(), StartPoint(), EndPoint()*.

Poznámka:

Problém při použití uvedeného postupu by mohl nastat v případě, kdy zájmový bod leží v blízkosti střední části dlouhého úseku cesty. Poté tento postup najde uzel na vzdálenějším úseku cesty (který může dojezdovou vzdálenost značně prodloužit) nebo nenajde pro zadanou toleranci žádný výchozí bod. V případě dat s příliš dlouhými úseky by proto bylo vhodné nejdříve použít funkci *ST_Segmentize()*, která rozdělí dlouhé linie na více kratších úseků. Důsledkem je však samozřejmě práce s větším objemem dat, proto tento postup v projektu není využit.

Poznámka 2:

Při použití funkce *shortest_path_shooting_star()* se nezadávají do dotazu id uzlových bodů, ale id počáteční a koncové hrany. Tady je proto třeba vyhledat nejbližší hranu, což lze např. pomocí výše zmíněné funkce *find_nearest_link_within_distance()*, nebo najít nejbližší úsek pomocí funkce *ST_Distance()*.

Poznámka 3:

Pro jednoduchost jsou ve většině následujících dotazů použity přímo id počátečních a koncových uzlů. Ty však mohou být různá pokaždé, když se spustí funkce pro vytvoření topologie sítě. Proto výsledky síťových dotazů při novém vytvoření tabulek mohou být odlišné od těch, které jsou uvedeny níže v kapitole 7.3.

7 Dotazy

7.1 Atributové dotazy

1. Jaký je nejdelší úsek silnic? Vypište název, popis a délku.

```
SELECT nazev,popis, length FROM silnice
ORDER BY length DESC LIMIT 1;

 nazev |  popis  |      length
-------+---------+------------------
       | primary | 47676.7835671838
(1 row)
```

2. Kolik automyček má v databázi uložen název?

```
SELECT COUNT(*) FROM motorismus
WHERE popis = 'car_wash' AND nazev != '';

 count
-------
    16
(1 row)
```

3. Které obce v Olomouckém kraji nezačínají na 'D' a končí na 'ov'? Vypište id a název.

```
SELECT ogc_fid, nazev_eng FROM obce
WHERE nazev_eng LIKE '%ov' AND nk='OL' AND nazev_eng NOT LIKE 'D%';

 ogc_fid |   nazev_eng
---------+----------------
    4087 | Grymov
     794 | Bezuchov
    3075 | Komarov
     333 | Cisarov
    1740 | Jindrichov
    2017 | Kolsov
    3707 | Vincencov
    3763 | Mutkov
    4252 | Radikov
    4677 | Skripov
    6142 | Zborov
    5180 | Starnov
    5318 | Tovacov
    93   | Bludov
 ...
(50 rows)
```

4. Kolik km dálnic je v ČR?

```
SELECT SUM(length)/1000 AS delka FROM silnice
WHERE popis IN ('motorway', 'motorway_link');

      delka
------------------
 2680.01107784901
(1 row)
```

7.2 Prostorové dotazy

1. Vypište jméno a geometrii každé benzínky, která má nejdále 50 m vzdálené parkoviště.

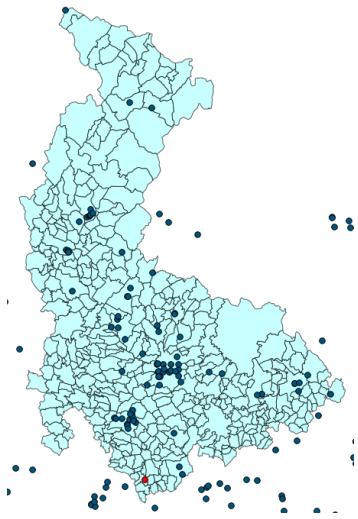
```
SELECT benzinky.nazev,ST_AsText(benzinky.geom) FROM benzinky
JOIN parkoviste ON ST_DWithin(benzinky.geom, parkoviste.geom, 50);

      nazev       |                st_astext
------------------+-----------------------------------------
                  | POINT(1365952.21245944 6474412.90923814)
                  | POINT(1375706.34906791 6468378.42461948)
                  | POINT(1380600.52161191 6460978.2525629)
 Nová Houžná      | POINT(1532395.43414606 6263843.15241299)
                  | POINT(1571936.75179143 6461425.37954551)
                  | POINT(1581002.26602995 6475167.06758177)
                  | POINT(1588605.06442355 6433713.04013696)
 bma              | POINT(1595571.60513566 6619717.6027474)
 ...
(38 rows)
```

2. Kolik benzínek je na území Prahy?

```
SELECT COUNT(*) FROM benzinky
JOIN (SELECT geom FROM obce
WHERE nazev_eng = 'Praha') AS Praha
ON ST_Intersects(benzinky.geom, Praha.geom);

 count
-------
   170
(1 row)
```

3. Jaká je nejjižnější benzínka v Olomouckém kraji a jaká je její Y souřadnice?

```
SELECT benzinky.nazev,ST_Y(benzinky.geom) AS y FROM benzinky
JOIN obce ON ST_Intersects(benzinky.geom, obce.geom)
WHERE obce.nk = 'OL' ORDER BY y ASC LIMIT 1;

  nazev  |        y
---------+------------------
 BENZHOR | 6330060.51406368
(1 row)
```



4. Vypište název a obvod všech průmyslových zón které jsou vzdálené do 5 km od benzínové pumpy.

```
SELECT DISTINCT prum_zony.nazev, ST_Perimeter(prum_zony.geom) AS obvod
FROM prum_zony
JOIN benzinky ON ST_DWithin(prum_zony.geom, benzinky.geom, 5000);
```

```
            nazev                |      obvod
--------------------------------+-----------------
 Exmont-Energo a.s.              | 1529.80687669602
                                 | 1520.94687456981
 cihelna                         | 1885.56161043678
                                 | 768.789651447271
                                 | 2921.05567823668
ZD Podlesí                       | 1551.74113951414
čistička odpadních vod           | 814.901163457725
Jaderná elektrárna Temelín       | 8283.75151838138
...
(2798 rows)
```

5. Vypište průmyslové zóny, které mají v okolí 500 m více než 2 ha travnatých ploch a výměru těchto ploch v ha.

```
SELECT DISTINCT prum_zony.nazev, SUM(ST_Area(trava.geom))/1e4 AS area
FROM prum_zony
JOIN trava ON ST_DWithin(prum_zony.geom,trava.geom,500)
WHERE trava.popis = 'grass' group by prum_zony.geom, prum_zony.nazev
HAVING SUM(ST_Area(trava.geom))/1e4 > 2 ORDER BY area DESC;
```

```
         nazev          |       area
------------------------+------------------
                        |     101.83448677
                        | 51.3805287900574
 Letov                  | 44.3834900250581
 Skanska Transbeton     | 44.3834900250581
                        | 44.3834900250581
 Čivice                 | 38.9504373699698
                        | 34.5903949849904
 Elektrárna Opatovice   | 28.2517931700006
...
(153 rows)
```

6. Jaká je průměrná rozloha (ha) a počet průmyslových zón, které jsou jižněji než Praha a severněji než České Budějovice?

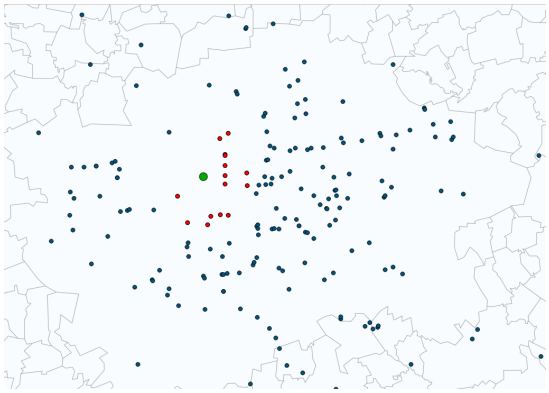
```
SELECT DISTINCT (SUM(ST_Area(prum_zony.geom))/1e4)/COUNT(*) AS prumer, COUNT(*) AS pocet
FROM prum_zony
JOIN obce_bod AS ob1 ON (ST_y(ST_centroid(prum_zony.geom)) < ST_y(ob1.geom))
JOIN obce_bod AS ob2 ON ((ST_y(ST_centroid(prum_zony.geom)) > ST_y(ob2.geom)))
WHERE ob1.nazob_eng='Praha' and ob2.nazob_eng='Ceske Budejovice';
```

```
      prumer      | pocet
------------------+-------
 13.4854556402118 |  2533
(1 row)
```

7. Vypište název, popis a geometrii benzínek, které jsou v okruhu 5 km od centra Prahy (bodová vrstva).

```
SELECT nazev, popis, ST_Astext(geom) FROM benzinky
WHERE ST_Intersects(geom,ST_Buffer((SELECT geom FROM obce_bod WHERE nazob_eng = 'Praha'),5000));
```

```
      nazez        | popis |                st_astext
------------------+-------+-----------------------------------------
                   | fuel  | POINT(1603198.86042138 6459632.03752206)
                   | fuel  | POINT(1604175.31046669 6457073.08983793)
                   | fuel  | POINT(1606087.66799876 6456871.95724026)
                   | fuel  | POINT(1606383.98935126 6457706.18899193)
                   | fuel  | POINT(1607313.91898137 6457847.42377609)
                   | fuel  | POINT(1607747.95367591 6463616.46194955)
                   | fuel  | POINT(1607756.5252767 6463530.38471496)
                   | fuel  | POINT(1607760.20995185 6461600.85725784)
                   | fuel  | POINT(1607785.90249032 6460789.62048764)
                   | fuel  | POINT(1608080.6097102 6457812.82588802)
                   | fuel  | POINT(1608094.63596604 6465658.59332991)
                   | fuel  | POINT(1609858.72706835 6461848.84685715)
                   | fuel  | POINT(1609912.71702137 6460626.23418358)
                   | fuel  | POINT(1607275.45551884 6465159.795)
 Rohanské nábřeží  | fuel  | POINT(1607750.55121804 6462543.105)
(15 rows)
```



8. Která průmyslová plocha s výměrou vetší než 10 ha leží nejblíže centru Ostravy (bod), jaká je její rozloha a vzdálenost od centra Ostravy?

```
SELECT nazev, ST_Area(prum_zony.geom)/1e4 AS vymera,
 ST_Distance(prum_zony.geom, obce_bod.geom) AS vzdalenost
FROM prum_zony
JOIN obce_bod ON (obce_bod.nazob_eng='Ostrava')
AND ST_Area(prum_zony.geom)/1e4 > 10 ORDER BY vzdalenost ASC LIMIT 1;
```

```
      nazev        |       vymera       |     vzdalenost
------------------+--------------------+-------------------
 pivovar Ostravar | 14.5537212549932 | 1037.32804977856
(1 row)
```

9. Jaká je rozloha všech travnatých ploch v ha ve Znojmě?

```
SELECT SUM(St_Area(St_Intersection(t.geom, o.geom)))/10000 AS plocha_ha FROM trava AS t
JOIN obce AS o ON St_Intersects(t.geom, o.geom)
WHERE nazev_eng = 'Znojmo';

   plocha_ha
------------------
 104.307467119909
(1 row)
```

10. Jaká je přímá vzdálenost v km z Karlových Varů do Prahy?

```
SELECT ST_Distance(a.geom, b.geom)/1000 AS delka_km FROM obce_bod AS a, obce_bod AS b
WHERE a.nazob_eng = 'Karlovy Vary' AND b.nazob_eng = 'Praha';

    delka_km
------------------
 172.779482614455
(1 row)
```

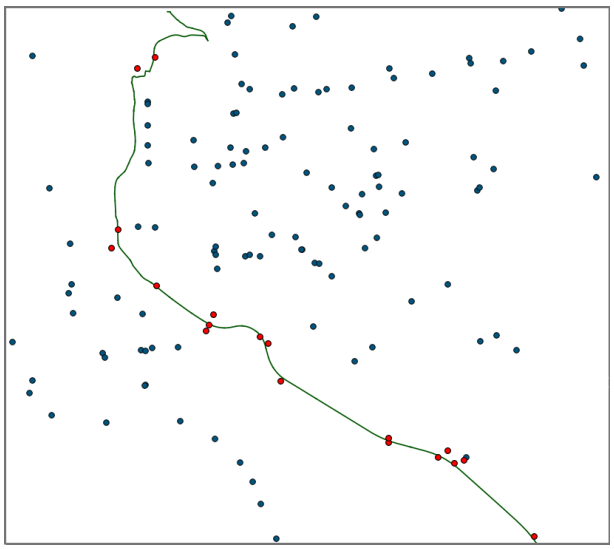
11. Vypište kraj s nejmenším poměrem plochy trávy ku ploše průmyslových zón.

```
SELECT ST_Area(trava.geom)/ST_Area(prum_zony.geom) AS pomer, obce.nk FROM obce
JOIN trava ON ST_Intersects(obce.geom,trava.geom)
JOIN prum_zony ON ST_Intersects(obce.geom,prum_zony.geom)
GROUP BY obce.nk, trava.geom, prum_zony.geom ORDER BY pomer ASC LIMIT 1;

        pomer         | nk
----------------------+----
 5.28771082241152e-06 | JM
(1 row)
```

7.3 Síťové analýzy

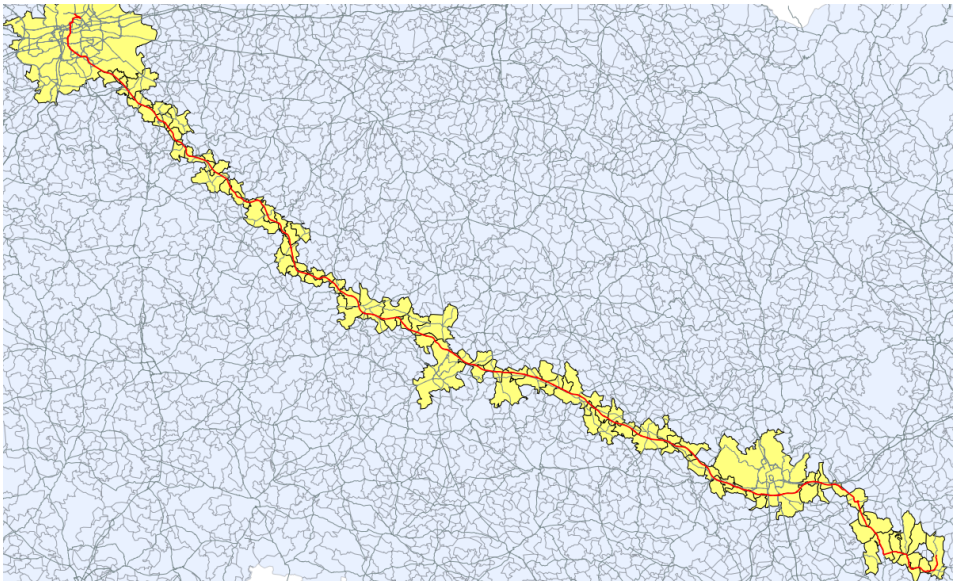
1. Které benzínky jsou vzdálené do 500 metrů od nejkratší trasy mezi dvěma uzlovými body 1099,8316?

```
SELECT DISTINCT benzinky.osm_id, benzinky.nazev, benzinky.popis
FROM (SELECT ST_buffer(geom,500) AS buffer,silnice.osm_id AS osm_id
FROM shortest_path('SELECT osm_id AS id,source,target,length AS cost
                    FROM b12.silnice',1099,8316,false,false) AS path
JOIN silnice ON path.edge_id=silnice.osm_id) AS A
JOIN benzinky ON ST_intersects(A.buffer,benzinky.geom);

   osm_id   |             nazev             | popis
------------+-------------------------------+-------
  339888577 |                               | fuel
  206170391 |                               | fuel
  310134088 | OMV                           | fuel
  339888196 |                               | fuel
 1602073144 |                               | fuel
  305559726 | Makro                         | fuel
  457470416 |                               | fuel
  457484639 | U devíti křížů                | fuel
...
(52 rows)
```



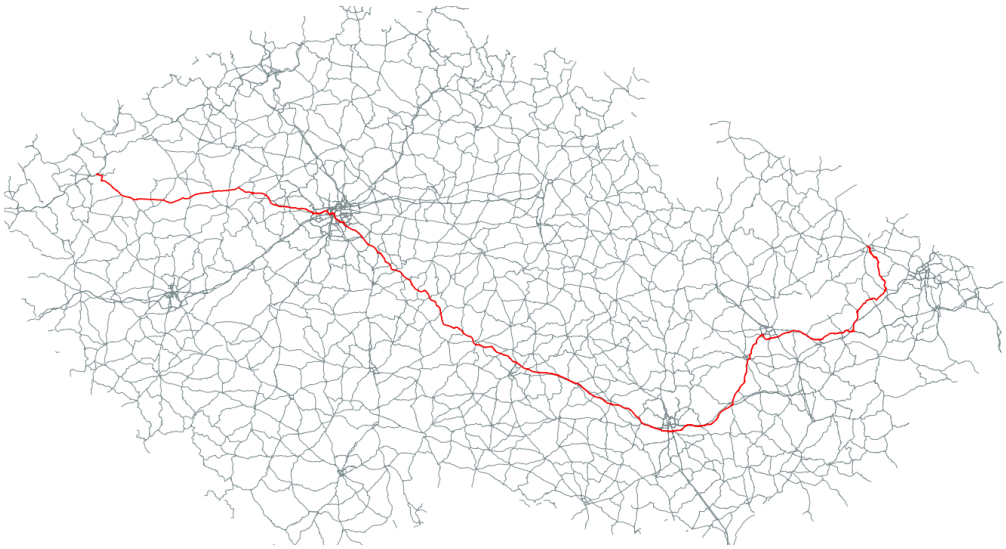
2. Kterými obcemi se projiždí na nejkratší trase mezi těmito dvěma body? Seřaďte podle abecedy.

```
SELECT DISTINCT nazev_eng
FROM (SELECT geom, silnice.osm_id AS osm_id
FROM shortest_path_astar('SELECT osm_id AS id,source,target,length AS cost,x1,y1,x2,y2
                          FROM silnice', 1099,8316, false, false) AS path
JOIN silnice ON path.edge_id=silnice.osm_id) AS A
JOIN obce ON ST_intersects(A.geom,obce.geom) order by nazev_eng;

    nazev_eng
-----------------
 Archlebov
 Bernartice
 Blizkov
 Brno
 Cestlice
 Choratice
 Damborice
 Dekanovice
...
(96 rows)
```



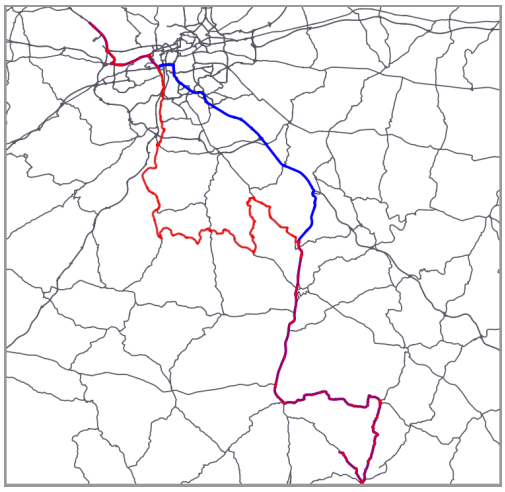
3. Jaká je nejkratší cesta z Karlových Varů do Opavy? Řešte postupně pomocí algoritmů Djistkra, A-Star a Shooting Star.

```
-- Djistkra
 SELECT * FROM shortest_path(
 'SELECT osm_id AS id,source,target,length AS cost FROM b12.silnice',
 (SELECT id FROM find_node_by_nearest_link_within_distance(
 (SELECT ST_AsText(geom) FROM obce_bod WHERE nazob_eng = 'Karlovy Vary'),5000,'silnice')),
 (SELECT id FROM find_node_by_nearest_link_within_distance(
 (SELECT ST_AsText(geom) FROM obce_bod WHERE nazob_eng = 'Opava'),5000,'silnice')),false,false);

  vertex_id |  edge_id  |       cost
-----------+-----------+------------------
     30847 |   5083061 | 431.150051875099
     30896 |  24234704 | 253.945043431098
     30882 |  32846520 | 188.526925928259
...
     27555 | 100658157 | 235.295351260029
     27251 |  49302669 | 1036.98283951956
     27744 |        -1 |                0
(508 rows)


-- A-Star
 SELECT * FROM shortest_path_astar(
 'SELECT osm_id as id,source,target,length as cost,x1,y1,x2,y2 FROM silnice',
 (SELECT id FROM find_node_by_nearest_link_within_distance(
 (SELECT st_astext(geom) FROM obce_bod where nazob_eng = 'Karlovy Vary'),5000,'silnice')),
 (SELECT id FROM find_node_by_nearest_link_within_distance(
 (SELECT st_astext(geom) FROM obce_bod where nazob_eng = 'Opava'),5000,'silnice')),false, false);

  vertex_id |  edge_id  |       cost
-----------+-----------+------------------
     30847 |   5083061 | 431.150051875099
     30896 |  24234704 | 253.945043431098
     30882 |  32846520 | 188.526925928259
...
     27555 | 100658157 | 235.295351260029
     27251 |  49302669 | 1036.98283951956
     27744 |        -1 |                0
(508 rows)


-- Shooting Star
 SELECT * FROM shortest_path_shooting_star(
 'SELECT osm_id as id,source,target,length as cost,x1,y1,x2,y2,rule,to_cost FROM silnice',
 (SELECT * FROM find_nearest_link_within_distance(
 (SELECT st_astext(geom) FROM obce_bod WHERE nazob_eng = 'Karlovy Vary'),5000,'silnice')),
 (SELECT * FROM find_nearest_link_within_distance(
 (SELECT st_astext(geom) FROM obce_bod WHERE nazob_eng = 'Opava'),5000,'silnice')),false,false);

  vertex_id |  edge_id  |       cost
-----------+-----------+------------------
     21009 |   5046149 | 2040.77321110597
     21052 |   5083061 | 431.150051875099
     21131 |  24234704 | 253.945043431098
     21104 |  32846520 | 188.526925928259
...
     14864 | 100648740 | 293.613349261483
     14865 | 100658157 | 235.295351260029
     14326 |  49302669 | 1036.98283951956
(508 rows)
```



4. O kolik delší bude cesta z bodu 5626 do 12073, jestliže nepojedeme po dálnici?

```
SELECT (b.delka-a.delka)/1000 AS rozdil_km FROM
(SELECT SUM(length) AS delka
 FROM shortest_path('SELECT osm_id AS id, source, target, length AS cost
 FROM silnice', 5626, 12073, FALSE, FALSE) AS cesta2
JOIN silnice ON cesta2.edge_id = silnice.osm_id) AS a
JOIN
(SELECT SUM(length) AS delka
 FROM shortest_path('(SELECT osm_id AS id, source, target, length AS cost FROM silnice
 WHERE popis NOT IN (''motorway'', ''motorway_link''))', 5626, 12073, FALSE, FALSE) AS cesta
JOIN silnice ON cesta.edge_id = silnice.osm_id) AS b
ON a.delka != b.delka;

    rozdil_km
------------------
 47.8367686689323
(1 row)
```



5. Jakým směrem se vydáte, pokud pojedete po cestě z průmyslové zóny Papcel do Opavy?

```
SELECT ST_Azimuth(
(SELECT ST_Point(ST_X(vertices_tmp.the_geom),ST_Y(vertices_tmp.the_geom)) FROM silnice
  JOIN vertices_tmp ON silnice.source=vertices_tmp.id
  WHERE osm_id IN
```

```
 (SELECT edge_id FROM shortest_path('SELECT osm_id AS id,source,target,length AS cost FROM silnice',
 (SELECT id FROM find_node_by_nearest_link_within_distance(
 (SELECT st_astext(st_centroid(geom)) FROM prum_zony where nazev = 'Papcel'),5000,'silnice')),
 (SELECT id FROM find_node_by_nearest_link_within_distance(
 (SELECT st_astext(geom) FROM obce_bod where nazob_eng = 'Opava'),5000,'silnice')), false,false) limit 1)),
(SELECT ST_Point(ST_X(vertices_tmp.the_geom),ST_Y(vertices_tmp.the_geom)) FROM silnice
 JOIN vertices_tmp ON silnice.target=vertices_tmp.id
 WHERE osm_id in
 (SELECT edge_id FROM shortest_path('SELECT osm_id AS id,source,target,length AS cost FROM silnice',
 (SELECT id FROM find_node_by_nearest_link_within_distance(
 (SELECT st_astext(st_centroid(geom)) FROM prum_zony where nazev = 'Papcel'),5000,'silnice')),
 (SELECT id FROM find_node_by_nearest_link_within_distance(
 (SELECT st_astext(geom) FROM obce_bod where nazob_eng = 'Opava'),5000,'silnice')),false,false) limit 1)))
/(2*pi())*360 AS azimut;

      azimut
------------------
 9.85564390434798
(1 row)
```

8 Závěr

Náš cíl vytvořit tématické vrstvy, nad nimiž bude možné kromě atributových a prostorových dotazů provádět síťové analýzy, jsme splnili. Je však nutné podotknout, že pro reálnější výsledky dotazů by bylo třeba mít úplnější data nebo provést nad daty z *OSM* rozsáhlé topologické úpravy.

Při práci na projektu jsme se potýkali s problémy transformace dat a kompatibilitou různých verzí PostGISu a pgRoutingu, které jsme vyřešili s pomocí Ing. Martina Landy. Největším problémem zůstalo řešení síťových analýz - tedy především vyhledání správných počátečních a koncových uzlů či hran. V dokumentaci pgRoutingu se nám bohužel nepodařilo nalézt žádné elegantní řešení. Pro jednoduché vyhledávání cesty mezi dvěma místy by bylo třeba přidat do tabulky *silnice* uzlové body pro každé konkrétní místo, z něhož bychom síťové analýzy chtěli provádět. Vzhledem k časové náročnosti a objemu dat jsme nové uzlové body nepřidávali.

Výsledkem našeho projektu jsou tématické vrstvy s validní geometrií v databázi *pgis_uzpd* a několik ukázkových atributových, prostorových a síťových analýz, jež lze nad nimi provádět.

Reference

[1] *153UZPD Úvod do zpracování prostorových dat* [citováno 2012-05]
Dostupné z: <http://geo101.fsv.cvut.cz/gwiki/153UZPD>

[2] *PostGIS 2.0.0 Manual* [citováno 2012-05]
Dostupné z: <http://postgis.org/documentation/manual-2.0/>

[3] *OpenStreetMap project* [citováno 2012-05]
Dostupné z: <http://www.openstreetmap.org/>

[4] *pgRouting Documentation: Open Source Routing Library* [citováno 2012-05]
Dostupné z: <http://www.pgrouting.org/docs/1.x/index.html>

[5] [*pgRouting Repository* [citováno 2012-05]
Dostupné z: <https://github.com/pgRouting/pgrouting/blob/master/core/sql/matching.sql>

[6] *Free and Open Source GIS Ramblings: pgRouting* [citováno 2012-05]
Dostupné z: <http://underdark.wordpress.com/tag/pgrouting/>

9 Přílohy

9.1 Zdrojový kód funkce *find_nearest_link_within_distance()*

```
 CREATE TYPE link_point AS (id integer, name varchar);

CREATE OR REPLACE FUNCTION find_nearest_link_within_distance(point varchar,
distance double precision, tbl varchar)
RETURNS INT AS
$$
DECLARE
    row record;
    x float8;
    y float8;

    srid integer;

BEGIN

    FOR row IN EXECUTE 'SELECT ST_srid(geom) AS srid FROM '||tbl||'
     where osm_id = (SELECT min(osm_id) FROM '||tbl||')' LOOP
    END LOOP;
srid:= row.srid;

    -- Getting x and y of the point

    FOR row in EXECUTE 'SELECT st_x(ST_GeometryFromText('''||point||''', '||srid||')) AS x' LOOP
    END LOOP;
x:=row.x;

    FOR row in EXECUTE 'SELECT st_y(ST_GeometryFromText('''||point||''', '||srid||')) AS y' LOOP
    END LOOP;
y:=row.y;

    -- Searching for a link within the distance

    FOR row in EXECUTE 'SELECT osm_id, st_distance(geom,
     ST_GeometryFromText('''||point||''', '||srid||')) AS dist FROM '||tbl||'
     where st_setsrid(''BOX3D('||x-distance||' '||y-distance||', '||x+distance||'
      '||y+distance||')''::BOX3D, '||srid||')&&geom order by dist asc limit 1'
    LOOP
    END LOOP;

    IF row.osm_id IS NULL THEN
--RAISE EXCEPTION 'Data cannot be matched';
RETURN NULL;
    END IF;

    RETURN row.osm_id;

END;
$$
LANGUAGE 'plpgsql' VOLATILE STRICT;
```

9.2 Zdrojový kód funkce *find_node_by_nearest_link_within_distance()*

```
CREATE OR REPLACE FUNCTION find_node_by_nearest_link_within_distance(point varchar,
distance double precision, tbl varchar)
RETURNS link_point AS
$$
DECLARE
    row record;
```

```
    link integer;
    d1 double precision;
    d2 double precision;
    field varchar;
    res link_point;

    srid integer;
BEGIN

    FOR row IN EXECUTE 'SELECT ST_srid(geom) AS srid FROM '||tbl||'
     where osm_id = (SELECT min(osm_id) FROM '||tbl||')' LOOP
    END LOOP;
srid:= row.srid;


    -- Searching for a nearest link

    FOR row in EXECUTE 'SELECT id
     FROM find_nearest_link_within_distance('''||point||''', '||distance||', '''||tbl||''') AS id'
    LOOP
    END LOOP;
    IF row.id is null THEN
        res.id = -1;
        RETURN res;
    END IF;
    link:=row.id;

    -- Check what is nearer - source or target

    FOR row in EXECUTE 'SELECT st_distance((SELECT ST_StartPoint(geom) FROM '||tbl||'
     where osm_id='||link||'), st_geometryFROMtext('''||point||''', '||srid||')) AS dist'
    LOOP
    END LOOP;
    d1:=row.dist;

    FOR row in EXECUTE 'SELECT st_distance((SELECT ST_EndPoint(geom) FROM '||tbl||'
     where osm_id='||link||'), st_geometryFROMtext('''||point||''', '||srid||')) AS dist'
    LOOP
    END LOOP;
    d2:=row.dist;

    IF d1<d2 THEN
field:='source';
    ELSE
field:='target';
    END IF;

    FOR row in EXECUTE 'SELECT '||field||' AS id, '''||field||''' AS f FROM '||tbl||'
     where osm_id='||link
    LOOP
    END LOOP;

    res.id:=row.id;
    res.name:=row.f;

    RETURN res;


END;
$$
LANGUAGE 'plpgsql' VOLATILE STRICT;
```