Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Analýza kvality cestnej siete

Bakalárska práca

2016
Peter Hraška

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Analýza kvality cestnej siete

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	RNDr. Tomáš Kulich, PhD.

Bratislava, 2016
Peter Hraška

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Peter Hraška
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Analýza kvality cestnej siete
Analysis of the Quality of Road Network

Cieľ: (1) Analyzovať dôležitosť ciest z hľadiska parametru Reach (http://research.microsoft.com/pubs/60764/tr-2005-132.pdf)
(2) Zostaviť štatistický ukazovateľ kvality pokrytia územia cestnou sieťou

Vedúci: RNDr. Tomáš Kulich, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: doc. RNDr. Daniel Olejár, PhD.

Dátum zadania: 29.10.2015

Dátum schválenia: 29.10.2015 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Rád by som sa poďakoval môjmu školiteľovi, za cenné rady a konzultácie a taktiež mojej rodine za hodnotné podnety pri písaní tejto práce.

Abstrakt

Práca prezentuje aplikovanie algoritmov pre vyhľadávanie najkratšej cesty v grafe na určenie stupňa dôležitosti ciest v mape cestnej siete. Popisuje v praxi použiteľný spôsob použitia mapových podkladov z projektu OpenStreetMap na vyhľadanie najkratšej cesty, určenie dôležitosti cesty a následné grafické zobrazenie dôležitosti cesty na týchto mapách.

Kľúčové slová: grafové algoritmy, cestné siete, OpenStreetMap

Abstract

In the following thesis the application of graph algorithms for the shortest paths problem, for determining the significance of each road inside the road network, is presented. The utilization of mapping data obtained from the OpenStreetMap project in order to find the shortest paths is described, the significance of each road in these maps is determined and finally, the maps visually containing this information are rendered.

Keywords: graph algorithms, road networks, OpenStreetMap

Obsah

Úvod **1**

1 Základné princípy a pojmy **3**
1.1 Dôležitosť cesty . . . 3
1.1.1 Dôležitosť cestného úseku . . . 3
1.1.2 Parameter reach . . . 4
1.1.3 Kvalita pokrytia cestnou sieťou . . . 6
1.2 Extrakcia údajov . . . 7

2 Technológie **8**
2.1 Mapové podklady . . . 8
2.1.1 Open Street Map . . . 9
2.1.2 Dátový model . . . 9
2.1.3 Detailnosť dát a cestná sieť v OSM . . . 10
2.1.4 Export dát . . . 11
2.2 Práca s mapovými dátami . . . 13
2.3 OMSCTools . . . 13
2.3.1 Osmfilter . . . 13
2.3.2 Osmconvert . . . 14
2.4 Programovací jazyk . . . 14
2.5 Knižnice . . . 14
2.5.1 JGraphT . . . 14
2.5.2 BasicOSMParser . . . 15
2.6 Vizualizácia . . . 15
2.6.1 Renderovacie nástroje . . . 15

3 Implementácia **18**
3.1 Parsovanie OSM XML súborov . . . 18
3.1.1 BasicOSMParser . . . 19
3.2 Grafová reprezentácia mapy . . . 20
3.2.1 Triedy knižnice JGraphT . . . 20

3.2.2 Identifikácia ciest v OSM XML 21
3.3 Výpočet dĺžok ciest 22
3.4 Výpočet hodnoty parametra reach 24
3.4.1 Floyd-Warshall algoritmus 24
3.4.2 Bellman-Fordov algoritmus 25
3.4.3 Dijkstrov algoritmus 25
3.5 Výkon algoritmov na reálnych dátach 26
3.6 Optimalizácie algoritmu 26
3.6.1 Aproximácia hodnoty parametra reach 26
3.6.2 Výber podmnožiny najkratších ciest 27
3.6.3 Optimalizácia algoritmu 29
3.7 Vizualizácia dôležitosti ciest 29
3.7.1 Dôležitosť cesty v OSM XML 30
3.8 Renderovanie máp 31
3.8.1 Renderovacie pravidlá 31
3.9 Ukazovateľ kvality pokrytia siete 33
3.10 Beh programu a zdrojové kódy 34

4 Výsledky a ich analýza **36**
4.1 Výpočet dôležitosti ciest 36
4.2 Vyrenderované mapy 37
4.3 Kvalita cestnej siete 39

5 Dôležosť pre veľké oblasti **43**
5.1 Sťahovanie mapových dát 43
5.2 Výpočet dôležitosti ciest 44
5.3 Alternatívne riešenie 46

6 Záver **47**
6.1 Práca do budúcna 47

Úvod

V dnešnej dobe vo vyspelých krajinách je štandardom, ak domácnosť vlastní aspoň jeden automobil. Na cestách je viac automobilov, než kedykoľvek v minulosti a keďže cestné siete nie sú všade rovnako rozvinuté, vznikajú často na cestách dopravné zápchy.

V tejto bakalárskej práci skúmame práve rozvinutosť cestných sietí rôznych oblastí. V cestných sieťach existujú úseky, po ktorých denne prejde väčšie množstvo automobilov, než po iných a môžeme konštatovať, že takéto úseky ciest sú dôležitejšie než iné. Naprogramovali sme preto aplikáciu, ktorá každému cestnému úseku určí jeho dôležitosť, pričom využívame algoritmy hľadajúce najkratšie cesty v grafe. Aby sme dokázali porovnávať kvalitu cestných sietí rôznych oblastí, využívame nami vypočítané dáta o dôležitostiach ciest a pre každú cestu tento údaj porovnávame s hustotou premávky, ktorú je táto cesta schopná zniesť.

Náš prístup určovania dôležitosti ciest využíva výpočet parametra reach, definovaný Ronom Gutmanom v jeho práci [15], ktorý o každom vrchole v grafe hovorí, aké najdlhšie trasy, ktoré sú zároveň najkratšími trasami medzi dvojicami rôznych bodov, cez tento vrchol prechádzajú. Keďže ľudia pri transporte medzi dvoma miestami takmer vždy hľadajú najkratšiu cestu, vieme vďaka parametru reach predpokladať, ktoré cestné úseky budú vyťaženejšie oproti ostatným a vďaka tomu dokážeme cestným úsekom priradiť ich dôležitosť v cestnej sieti.

Výsledkom našej bakalárskej práce je program, ktorý zo vstupného mapového súboru načíta cesty, vloží ich do grafovej dátovej štruktúry, vypočíta pre každú cestu jej dôležitosť a tento vypočítaný údaj vloží naspäť do máp, s ktorými pracoval. Okrem toho sme vytvorili spôsob, ako tieto mapy vyrenderovať tak, aby sa z nich dala čítať dôležitosť jednotlivých cestných úsekov.

V kapitole 1 tejto bakalárskej práce definujeme pojem dôležitosti ciest a kvality cestnej siete, vysvetlíme výpočet parametra reach, ktorý ja základom určovania dôležitosti ciest v našom algoritme a definujeme vlastné pojmy a označenia, ktoré v celej bakalárskej práci používame. V kapitole 2 sme vymenovali technológie, použité pri riešení

nášho problému a kapitola 3 sa venuje našej implementácii týchto technológií. Kapitola 4 sa venuje konkrétnym výsledkom a dátam, ktoré sme pri riešení nášho problému získali. V tejto kapitole ukazujeme taktiež množstvo máp, v ktorých sme vyzualizovali dôležitosť v nich obsiahnutých ciest, čo tvorí hlavným výsledkom našej bakalárskej práce. Kapitola 5 je bonusová a ide nad rámec nami stanovených cieľov. Ukazujeme v nej, ako dokážeme naše postupy určovania a vizualizovania zložitostí ciest aplikovať aj na obrovské mapové súbory napríklad o veľkosti celých štátov. Na záver sme v kapitole 6 zhrnuli, čo sa nám podarilo v práci dosiahnuť a čo je ešte možné vylepšiť.

Kapitola 1

Základné princípy a pojmy

V našej práci narábame s pojmami ako kvalita cestnej siete, dôležitosť cesty či kvalita pokrytia cestnou sieťou. V tejto kapitole vysvetľujeme, čo si pod týmito pojmami predstavujeme a zadefinujeme ich.

1.1 Dôležitosť cesty

Na začiatku sme potrebovali určiť, pre aké malé úseky ciest má zmysel jednotlivo počítať ich dôležitosť. Logicky sme dospeli k ďeleniu cesty vždy v mieste križovatky. Preto *cestným úsekom* v tejto práci označujeme ľubovoľnú časť cesty, ktorá vedie medzi práve dvoma križovatkami.

V texte často spomíname *najkratšiu cestu*. Myslíme ňou najkratšiu cestu, medzi práve dvoma bodmi (niekedy náhodnými dvoma bodmi) v grafe mapy, kde križovatky sú vrcholy a cesty medzi nimi hrany grafu. Hrany tohto grafu cestnej siete sú orientované a ohodnotené, pričom váhy jeho hrán sú rovné dĺžke cesty medzi jej dvoma vrcholmi.

1.1.1 Dôležitosť cestného úseku

Najprv vymenujeme, z akých vlastností cesty je možné intuitívne usudzovať, že sú dôležitejšie, než ostatné cesty v mape.

- Diaľnica je vo všeobecnosti dôležitejšia, než cesta spájajúca 2 dediny.
- Ak po ceste A denne prejde viac áut, ako po ceste B, cesta A by pre nás mala

byť dôležitejšia.

- Ak je pri trase z miesta X do miesta Y trasa vedúca cez A rýchlejšia alebo kratšia trasa vedúca cez B, trasa A by pravdepodobne mala byť dôležitejšia.
- Ak pri ceste z miesta X do miesta Y nemám inú možnosť, než využiť cestný úsek C, je tento cestný úsek dôležitý.

Naša výsledná metóda určovania dôležitosti ciest by mala generovať výsledky, ktoré čo najviac súhlalsia s našou intuitívnou predstavou o dôležitosti ciest.

Všetky tieto vlastnosti ciest, ktoré intuitívne pokladáme za dôležitejšie súvisia s výberom najkratšej trasy pri cestovaní medzi dvoma bodmi v mape, či grafe. Dospeli sme ku tomu, významným by pre nás mohol mať parameter reach, ktorý pre každý vrchol grafu určuje, aké dlhé najkratšie cesty cezeň prechádzajú.

1.1.2 Parameter reach

Reach parameter, je ukazovateľ, ktorý vieme vypočítať pre každý vrchol v grafe, pre náš prípad v grafe cestnej siete. Tento parameter vrcholov grafu má v sebe zakódovanú informáciu, ako významne napomáha konkrétny vrchol prepojiť dve rôzne oblasti pri cestovaní po grafe. Prvý krát ho popísal a ukázal jeho praktické využitie Ron Gutman v jeho práci [15]. Gutman v tejto práci riešil, ako sa dá zlepšiť výkon algoritmov vyhľadávajúcich najkratšie cesty v grafe cestných sietí, ak si môže tento graf vopred spracovať. Výsledkom jeho predspracovávania je práve parameter reach, ktorý priradil každému vrcholu v grafe.

Prvotná myšlienka o reach parametri hovorí, že by mal pre každý bod v grafe obsahovať istú formu informácie o dĺžkach najkratších ciest vedúcich cez tento bod. Aby mal bod vysoký reach, musí ležať na najkratšej ceste medzi dvoma bodmi, ktoré sú od neho ďaleko oboma smermi.

Konkrétna definícia hovorí. Majme:

- orientovaný ohodnotený graf $G = V, E$ v ktorom váhy hrán sú kladné,
- cestu P v grafe G, so začiatočným bodom s a koncovým t, pričom P je najlacnejšia cesta medzi s a t,
- bod v, ktorý leží na ceste P.

Potom reach pre v vzhľadom na cestu P je rovný kratšej z dvojice dĺžok sv a vt na ceste P. Reach pre v vzhľadom na celý graf G je rovný najväčšej hodnote reachu pre všetky najkratšie cesty Q v grafe G, ktoré obsahujú vrchol v.

Ak by sme teda postupne hľadali všetky najkratšie cesty v celom grafe, vieme o reach parametri povedať, že ho počítame nasledovne: dá sa predpokladať, že medzi dvoma miestami v mape existuje viacero trás, po ktorých sa dostaneme z jedného miesta do druhého. Práve najkratšia z týchto trás je však hodnotná natoľko, aby sme vrcholom, cez ktoré prechádza boli ochotní zvýšiť hodnotu ich parametra reach. Ak na tejto najrýchlejšej trase nájdeme vrchol, v ktorom je doteraz nájdený parameter reach menší, než ten, ktorý vyplýva z novo-nájdenej najkratšej trasy, prepíšeme parameter reach pre tento vrchol väčšou z týchto dvoch hodnôt.

V cestnej sieti existuje prirodzená hierarchia. Diaľnice sú stavané s cieľom urýchliť dopravu na väčšie vzdialenosti. Prirodzene sú preto dôležitejšie ako cesty mimo obcí a tie sú dôležitejšie, než cesty vedúce k parkoviskám alebo budovám. Nie každá diaľnica je však rovnako dôležitá a našim cieľom v tejto bakalárskej práci je práve vyčísliť dôležitosť každého cestného úseku zvlášť. Hierarchiu, ktorá prirodzene v cestných sietiach existuje pritom do úvahy neberieme.

Gutman v jeho práci rozobral aj problém, či ako cenu hrany v grafe cestnej siete uvažovať geografickú vzdialenosť týchto dvoch bodov, alebo čas potrebný na prejdenie tejto cesty. V jeho experimentoch sa ukázalo, že vo všeobecnosti produkovala vzdialenosť ako cena hrany lepšie výsledky, než čas potrebný na jej prejdenie. Aj v našej práci uvažujeme najmä vzdialenosť a informácie o maximálnej rýchlosti na každej hrane využívame pri analyzovaní kvality pokrytia cestnou sieťou.

Aby sme parameter reach vedeli vyčísliť presne, potrebujeme riešiť problém všetkých najkratších ciest grafu cestnej siete. Gutman si uvedomil zložitosť takéhoto problému a namiesto rátania presný hodnôt hľadá iba horné ohraničenie v každom bode. Takto predspracované údaje o grafe Gutman využíva pre zefektívnenie behu algoritmu A*, ktorý v grafe hľadá najkratšie cesty. Ďalšie optimalizácie a nové prístupy využitia tohto parametra uviedli Goldberg, Kaplan a Werneck v ich práci Reach for A* [14].

Naše riešenie berie do úvahy Gutmanovu definíciu parametra reach a na základe hodnôt reach bodov v grafe priraďujeme každému cestnému úseku dôležitosť na škále od 0 po 7, kde hodnota 7 označuje najdôležitejšie cestné úseky. Tieto najdôležitejšie cestné úseky budeme skrátene pomenúvať ako dôležité úseky. Pracujeme s rôzne veľkými objemami dát. Niekedy si môžeme dovoliť počítať presné hodnoty reachu, no pre väčšie oblasti vypočítame iba jeho spodné ohraničenie. O konkrétnych algoritmoch,

implementácii a optimalizáciach, ktoré sme vykonali, píšeme v kapitole 3.

1.1.3 Kvalita pokrytia cestnou sieťou

Predpokladajme, že dôležitosť každého cestného úzeku v grafe už máme vypočítanú. V kapitole 2 bližšie popisujeme, aké rôzne informácie nám mapové podklady poskytujú o jednotlivých cestných úsekov, no už teraz je dôležité spomenúť, že z týchto mapových dát vieme zistiť maximálnu rýchlosť a počet jazdných pruhov pre každý cestný úsek. Práve tieto informácie využívame pri určovaní kvality pokrytia oblasti cestnou sieťou.

Náš pohľad na kvalitu pokrytia cestnou sieťou je nasledovný:

- kvalitná cestná sieť by mala spoľahlivo zvládať nepredvídané udalosti v premávke (teda dôležité cestné úseky budú mať viac jazdných pruhov, aby sa dala napríklad jednoducho obchádzať dopravná nehoda)
- kvalitná cestná sieť by nemala obsahovať malé množstvo veľmi dôležitých ciest
- za ten istý čas by dôležitejšími cestami malo byť schopné prejsť väčšie množstvo áut než menej dôležitými

Dôležitosť ciest určujeme pomocou hľadania najkratších ciest v grafe. Čím dôležitejšia cesta, tým väčšie množstvo áut bude cez ňu chodiť, pretože ľudia si pri transporte prirodzene vyberajú najkratšie cesty vedúce k ich cieľom. Ak však na dôležitej ceste, ktorá je plne vyťažená nastane napríklad nehoda, ktorá zablokuje jeden z dvoch jazdných pruhov, znamená to približne zdvojnásobenie času potrebného na prejdenie tohto úseku. V prípade, že je cestná sieť vybudovaná kvalitne, malo by obmedzenie v jazdných pruhoch znamenať iba minimálne zdržanie pre autá, ktoré ním prechádzajú. Minimalizácia zdržania môže spočívať v rezerve objemu áut, ktoré cestný úsek za jednotku času prepustí alebo v existencii podobne dôležitej cesty v blízkosti.

Pred tým, než definujeme ukazovateľ kvality pokrytia cestnou sieťou, zadefinujeme si priepustnosť cestného úseku. Pod priepustnosťou cestného úseku rozumieme počet áut, ktoré vedia daným úsekom prejsť za jednotku času. Ak maximálna rýchlosť úseku je v_m a počet jazdných pruhov na tomto úseku je p, tak priepustnosť cestného úseku vyčíslujeme ako $v_m \cdot p$.

Kvalitná cestná sieť je taká, kde dôležitosť cestných úsekov je priamo úmerná ich priepustnosti. Všeobecne povedané, v kvalitnej cestnej sieti dôležitosť a priepustnosť cestných úsekkov navzájom korelujú.

1.2 Extrakcia údajov

Pri extrakcii údajov o cestných sieťach z mapových súborov používamé takzvané syntaktické analyzátory. Ide o programy, ktoré rozumejú niektorej konkrétnej dátovej štruktúre dát a vedia z nej napríklad vyberať konkrétne informácie alebo kontrolovať správnosť štruktúry týchto dát. Toto pomenovanie nám však príde zbytočne dlhé a preto v práci používame pre syntaktický analyzátor jeho anglický názov - *parser*.

Kapitola 2

Použité technológie

V tejto kapitole sa venujeme technológiam, ktoré sme pri riešení nášho problému využili, aké alternatívy sme zvažovali a prečo sme sa rozhodli ich nepoužiť. Podľa potreby rozpíšeme o konkrétnej technológii a jej alternatívach viac detailov, o ostatných píšeme podrobnejšie v ďalších kapitolách.

2.1 Mapové podklady

Pre zisťovanie dôležitosti ciest v cestnej sieti prirodzene potrebujeme mapové dáta, v ktorých túto dôležitosť budeme počítať. Mohli by sme pracovať s dátami, ktoré by sme sami vygenerovali, no implementovať naše algoritmy priamo na existujúce mapové dáta je omnoho zaujímavejšie. V reálnej cestnej sieti prirodzene existuje hierarchia ciest, od príjazdových ciest až po diaľnice a bude preto zaujímavé vidieť vypočítanú dôležitosť ciest na reálnych dátach, alebo dokonca na mapách oblastí, ktoré osobne poznáme.

Ako nutnú podmienku, ktorú mapy, s ktorými sa rozhodneme pracovať, musia spĺňať, sú ich licenčné zmluvy, ktoré nám umožnia tieto dáta voľne sťahovať a manipulovať s nimi. Druhotnými, no taktiež dôležitými podmienkami sú detailnosť, presnosť a aktuálnosť týchto mapových údajov, schopnosť jednoducho extrahovať údaje o konkrétnych cestách a globálny rozsah pokrytia týchto máp.

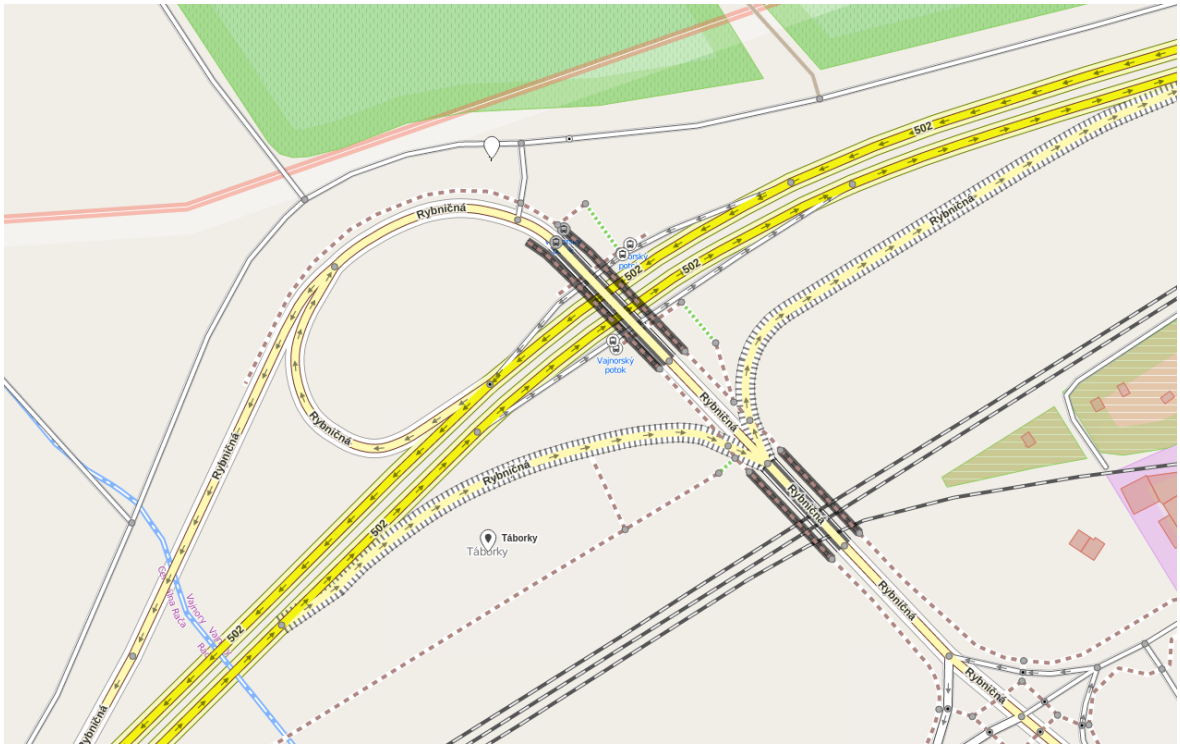
Dá sa povedať, že iba jediný projekt splnil všetky tieto podmienky a tým predčil svojich konkurentov. Týmto projektom je Open Street Map.

2.1.1 Open Street Map

Open Street Map [6], skrátene OSM, je kolaboratívny projekt na vytvorenie voľne dostupnej editovateľnej mapy sveta. Tento projekt založil Steve Coast v roku 2004 a inšpiroval ho úspech kolaboratívnej encyklopédie Wikipedia. Tvorcami mapových dát v OSM sú používatelia po celom svete a tieto dáta sú prístupné pod licenčnými podmienkami Open Database License [5], ktoré umožňujú používateľom voľne zdieľať, upravovať a modifikovať dáta v databáze, zatiaľ čo ich zmeny sú pod rovnakými podmienkami prístupné ostatným používateľom.

Voľné licenčné podmienky a užívatelia z celého sveta sú dôvodom, prečo dnes pokrývajú tieto dáta celú zemeguľu. Dáta nemusia byť upravované iba používateľmi. Kde je to možné, využívajú sa aj rôzne verejné zdroje zo štátnych organizácií. Na Slovensku je napríklad veľa dát preberaných priamo z Katastrálneho portálu Slovenska.

Ukážku vizualizovaných dát z databázy OSM môžete vidieť na obrázku 2.1



Obr. 2.1: Ukážka komplikovanej križovatky v OSM

2.1.2 Dátový model

Na reprezentáciu a uchovanie dát používa OSM vlastný dátový model, ktorý pozostáva z týchto primitívnych dátových prvkov (elementov):

- **node** (bod) - reprezentuje bod na mape a uchováva jeho zemepisnú šírku a výšku vo formáte WGS84 [4]. Môže reprezentovať samostatný malý objekt ako strom alebo lavičku, môže byť taktiež využitý na tvarovanie objektov typu *way*.
- **way** (plocha/cesta) - obsahuje najmä zoznam *node*-ov, ktoré spolu tvoria uzavretú plochu alebo vyznačujú neuzavretú cestu v mape. Obsahuje taktiež pomocné informácie, ktoré určujú o aký typ objektu sa jedná (napríklad či ide o budovu, jazero, diaľnicu a pod.).
- **relation** (vzťah) - multifunkčná dátová štruktúra združujúca ľubovoľný počet iných objektov. Vie napríklad niesť informáciu o zákaze odbočenia alebo po akých prvkoch typu *way* premáva konkrétny spoj hromadnej dopravy.
- **tag** (značka) - všetky doteraz spomenuté prvky môžu obsahovať takzvané značky (tagy). Tie napríklad určia pre prvky *way*, či ide o jazero, budovu alebo cestu, aká je maximálna rýchlosť na danej ceste a iné konkrétne informácie. Značky obsahujú vždy práve dva údaje: kľúč a hodnotu. Napríklad značka *<tag k="highway"v="motorway»* s kľúčom *highway* a hodnotou *motorway*, ktorá sa nachádza vo vnútri prvku *way* hovorí, že daný *way* reprezentuje cestu (highway) typu diaľnica (motorway).

Každý dátový element nesie taktiež unikátny číselný identifikátor, informácie o jeho poslednej zmene, konkrétne čas, v ktorom bola zmena vykonaná a meno používateľa, ktorý zmenu vykonal a koľko krát bol prvok upravovaný.

2.1.3 Detailnosť dát a cestná sieť v OSM

Štruktúra, v ktorej sú dáta v OSM uchovávané umožňuje mapovať prakticky akýkoľvek objekt. Od psích parkov až po verejné stojany na bicykle. Informácie o objektoch môžu byť detailné až do takej miery, že je možné z mapy vyčítať, aké náročné je prejsť konkrétnym lesným cestným úsekom na bicykli. Množstvo a detailnosť týchto dát závisí iba od vôle a sily komunity, ktorá do projektu OSM prispieva.

My pracujeme s cestnými sieťami. Zaujímalo nás preto, ktoré informácie z OSM budú pre nás užitočné a ako detailne sú zmapované. O samotných cestách v mapách vieme vyčítať nasledovné údaje:

- kategória cesty (od poľných po diaľnice)
- maximálna rýchlosť
- počet jadzdných pruhov

- smer jazdy pre každý jazdný pruh
- kvalita cestnej vozovky (od kamenistej po asfaltovú)

Okrem údajov o samotných cestách nás môžu zaujímať aj nasledovné objekty:

- semafory
- zákazy odbočenia
- dopravné značenie určujúce prednosť v jazde
- kruhové objazdy
- súkromné cesty

Semafóry, kruhové objazdy a prednosti vjazdu značne vplývajú na plynulosť prechodu cestami, na ktorých sa nachádzajú, preto by bolo zaujímavé pracovať aj s týmito informáciami. Pri identifikácii zaujímavých dát sme si pomohli dokumentáciou OSM pre vývojárov navigácií [11].

2.1.4 Export dát

Mapa sveta reprezentovaná primitívnymi dátovými typmi OSM je uložená v jedinej, takzvanej hlavnej PostgreSQL databáze. Pre účely zdieľania týchto dát je kompletná databáza pravideľne exportovaná v dvoch primárnych formátoch. Jeden z nich je XML, teda rozšíriteľný značkovací jazyk, ktorý je čitateľný aj ľudmi a druhým je PBF [9], formát vytvorený spoločnosťou Google, ktorý obsahuje tie isté, no skomprimované dáta. Formát PBF bol vytvorený primárne na efektívny prenos objemných XML dát po sieti. Jeho využitie v projekte OSM je opodstatnené, kedže kompletný export dát celej planéty, nazývaný tiež *planet.osm* [10], zaberá vo formáte XML niečo cez 666GB a iba 31GB vo formáte PBF. Tento súbor je voľne prístupný na stiahnutie, no existuje veľa projektov, ktoré umožňujú z tohoto súboru stiahnuť iba konkrétnu požadovanú časť.

Malé časti máp vieme získať priamo na domovskej stránke projektu OSM [6]. Po zvolení záložky *export* vieme pomocou webovej aplikácie na mape označiť obdĺžnikovú oblasť, ktorej dáta chceme stiahnuť. Táto metóda však umožňuje sťahovať iba limitovaný objem dát. Na tento limit sme narazili už pri pokuse stiahnuť mapu celej Bratislavy.

Pre väčšie objemy dát je odporúčané využiť Overpass API [8], ktorá umožňuje každému užívateľovi denne stiahnuť až 5GB dát. Dáta z nej získavame skonštruovaním URL adresy pre jej HTTP API rozhranie, v ktorej špecifikujeme súradnice rohov obdĺžnika, v ktorom sa nachádza územie, ktorého dáta chceme stiahnuť. Server nám následne pošle požadovaný mapový súbor vo formáte XML.

V kapitole 5 sme dosiahli i hranice možností Overpass API. Potrebovali sme pracovať s oblasťou, ktorá výsledne zaberala 60GB. Riešením pre nás bolo využiť služby spoločnosti Geofabrik, ktorá zdarma poskytuje všetky dáta OSM v členení po jednotlivých krajinách sveta. Z ich serverov sme stiahli dáta krajín, ktorých územie zasahovalo do oblasti, ktorú sme potrebovali analyzovať. Tieto dáta sme pomocou nástrojov *osmctools*, o ktorých píšeme ďalej v tejto kapitole, spojili do jedného veľkého súboru a následne z neho extrahovali iba požadované územie.

Stiahnuté dáta vo formáte XML môžu vyzerať nasledovne:

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap 0.0.2">
    <node id="1831881213" version="1" changeset="12370172" lat="54.0900666"
        lon="12.2539381" user="lafkor" uid="75625">
        <tag k="name" v="Neu Broderstorf"/>
        <tag k="traffic_sign" v="city_limit"/>
    </node>
    ...
    <way id="26659127" user="Masch" uid="55988" visible="true" version="5">
        <nd ref="292403538"/>
        ...
        <tag k="highway" v="highway"/>
    </way>
    ...
</osm>
```

V tejto ukážke sa nachádzajú dva mapové objekty. Prvý objekt, *node*, reprezentuje dopravnú značku, ktorá označuje hranicu mesta a druhým je *way*, ktorý reprezentuje diaľnicu.

2.2 Práca s mapovými dátami

V projektoch, ktoré využívajú OSM mapy, je najbežnejšie najprv vložiť mapové dáta do PostgreSQL databázy a pracovať s nimi až v takejto podobe. Veľa nástrojov na ich manipuláciu, spájanie alebo vizualizáciu počíta s tým, že dáta, s ktorými majú pracovať, sú uložené v PostgreSQL databáze. V našom prípade potrebujeme v týchto dátach hladať obrovské množstvo najkratších ciest, čo je algoritmicky náročný problém. Uchovávanie dát v databáze je z toho dôvodu pre nás nevýhodné. Program by bol zbytočne spomalený veľkým množstvom dopytov na databázu.

Našim plánom preto bolo dáta počas potrebných výpočtov udržiavať v operačnej pamäti (RAM). Jedinou obavou bolo, či kapacitné možnosti bežného osobného počítača budú postačujúce. Kompletná mapa Bratislavy exportovaná do XML súboru zaberá iba niečo cez 270MB a takýto súbor sa určite do operačnej pamäti bežného osobného počítača zmestí. O tom, ako naše výpočty možno aplikovať aj na omnoho väčšie súbory píšeme v kapitole 5.

2.3 OMSCTools

OSMCtools je balíčkom nástrojov pre príkazový riadok unixových systémov. Umožňuje modifikáciu mapových XML súborov. Využili sme z neho dva nástroje: Osmfilter a Osmconvert.

Najčastejšie používaným balíčkom s podobnými schopnosťami je Osmosis [7], ten však vie pracovať iba s dátami uloženými v PostgreSQL databáze. Siahli sme preto po jeho alternatíve.

2.3.1 Osmfilter

Osmfilter umožňuje z XML súboru vytvoriť nový súbor, ktorý obsahuje iba primitívne prvky spĺňajúce nami definované podmienky a všetky primitívne prvky, ktoré tieto vyselektované prvky obsahujú. To znamená, že ak chceme z XML súboru vybrať iba prvky *way*, ktoré reprezentujú diaľnice, ponechá Osmfilter okrem týchto prvkov *way* aj všetky prvky *node*, ktoré určujú tvar týchto diaľníc.

2.3.2 Osmconvert

Osmconvert umožňuje hlavne konverziu mapových súborov medzi rôznymi mapovými formátmi. Okrem toho však dokáže aj spájať viacero súborov do jediného, pričom vynecháva duplicitné informácie. Taktiež vie zo súboru vybrať a ponechať iba dáta, ktoré sa nachádzajú vo vnútri nami definovanej oblasti alebo napríklad vypísať rôzne štatistické údaje o tomto mapovom súbore.

2.4 Programovací jazyk

Pri rozhodovaní sa, ktorý programovací jazyk použijeme sme uvažovali o jazykoch Java a C++. Žiaden z nedávnych testov porovnávajúcich výkon jazykov nenaznačujú, že by jeden z nich výkonovo výrazne predčil druhý. Pretože s jazykom Java mám o niečo väčšie skúsenosti, rozhodli sme sa naše riešenie programovať v tomto jazyku.

2.5 Knižnice

Pri riešení rôznych podproblémov nášho problému sme využili rôzne programátorské knižnice. Uvádzame preto prehľad týchto knižníc a zdôvodnenie, prečo sme nepoužili niektorú z ich alternatív.

2.5.1 JGraphT

Ako sme už spomenuli, pri určovaní dôležitosti ciest potrebujeme prehľadať veľké množstvo najkratších ciest. Potrebovali sme teda dáta o reálnych cestných sietiach reprezentovať ako graf tvorený vrcholmi a hranami, aby sme na ne vedeli aplikovať algoritmy hľadajúce najkratšie cesty. Najprv sme používali vlastnú dátovú štruktúru a vlastné algoritmy, neskôr nám však prišlo šikovnejšie použiť robustnejšiu dátovú štruktúru v podobe knižnice a sústrediť sa viac na samotný problém dôležitosti ciest.

Jedným z kandidátov na knižnicu, ktorá by sa postarala o grafovú reprezentáciu dát má názov JUNG (Java Universal Network/Graph Framework). Vývoj tejto knižnice sa však v roku 2010 zastavil a neponúka také možnosti ako knižnica JGraphT, ktorá v súčasnosti ponúka implementácie väčšieho počtu rôznych algorimtov na hľadanie najkratšej cesty, čo sa nám hodilo pri experimentoch s rôznymi algoritmami, ktoré v grafe vyhľadávajú najkratšie cesty.

2.5.2 BasicOSMParser

Za účelom vkladania prvkov z mapových súborov do našej grafovej štruktúry potrebujeme mapy *sparsovať*. Ako sme už spomenuli, mapy z OSM vieme sťahovať vo formáte XML, ktorý je navrhnutý tak, aby sa z neho informácie dali pohodlne čítať.

Metód parsovania XML súborov je veľa, pre každú situáciu je vhodné použiť inú metódu. Pre manipuláciu s veľkými súbormi je najčastejšie využívaná SAX parsovacia metóda, ktorá vstupný súbor číta riadok po riadku zhora nadol. Takýto prístup zabezpečuje čítanie XML súborov v lineárnom čase a pre naše potreby je ideálny, keďže potrebujeme z XML súbora iba jednoducho načítať prvky reprezentujúce cesty do našej grafovej dátovej štruktúry. Nezáleží nám pritom na poradí v akom prvky čítame. Nevyhovovala nám napríklad DOM metóda parsovania, ktorá potrebuje najskôr celý XML súbor načítať do operačnej pamäte a až potom z neho číta. Pri väčších XML súboroch sa takýto prístup ukázal ako problematický.

Pre spracovanie XML súborov v našom programe preto využívame open source nástroj BasicOSMParser [2], ktorý sme mierne upravili podľa našich potrieb. Okrem uloženia dát z OSM XML súborov umožňuje tento nástroj ich následný export do súboru vo formáte CSV. Túto schopnosť však nevyužívame.

2.6 Vizualizácia

Najdôležitejším výsledkom našej práce je mapa, v ktorej graficky znázorňujeme dôležitosť jednotlivých cestných úsekov. Pôvodne sme si mysleli, že bude potrebné naprogramovať vlastnú aplikáciu na takúto vizualizáciu, keďže ide o nami vygenerované dáta. Existuje však veľké množstvo aplikácií, ktoré vedia OSM XML dáta renderovať, pričom vieme ovládať, ako a ktoré mapové údaje sa majú vykreslovať.

Aby sme dôležitosť ciest dokázali v externom programe vykresliť, naprogramovali sme triedu, ktorá tieto dáta vkladá priamo do OSM XML súboru, v ktorom sme dôležitosť ciest počítali. Tieto dáta do štruktúry XML vkladáme pre každý element *way*, ktorého dôležitosť sme vypočítali, ako nový *tag* s kľúčom *reach* a hodnotou je vypočítana dôležitosť danej cesty. O konkrétnej implementácii píšeme v kapitole 3.

2.6.1 Renderovacie nástroje

Pri výbere sme zvážili nasledovné nástroje:

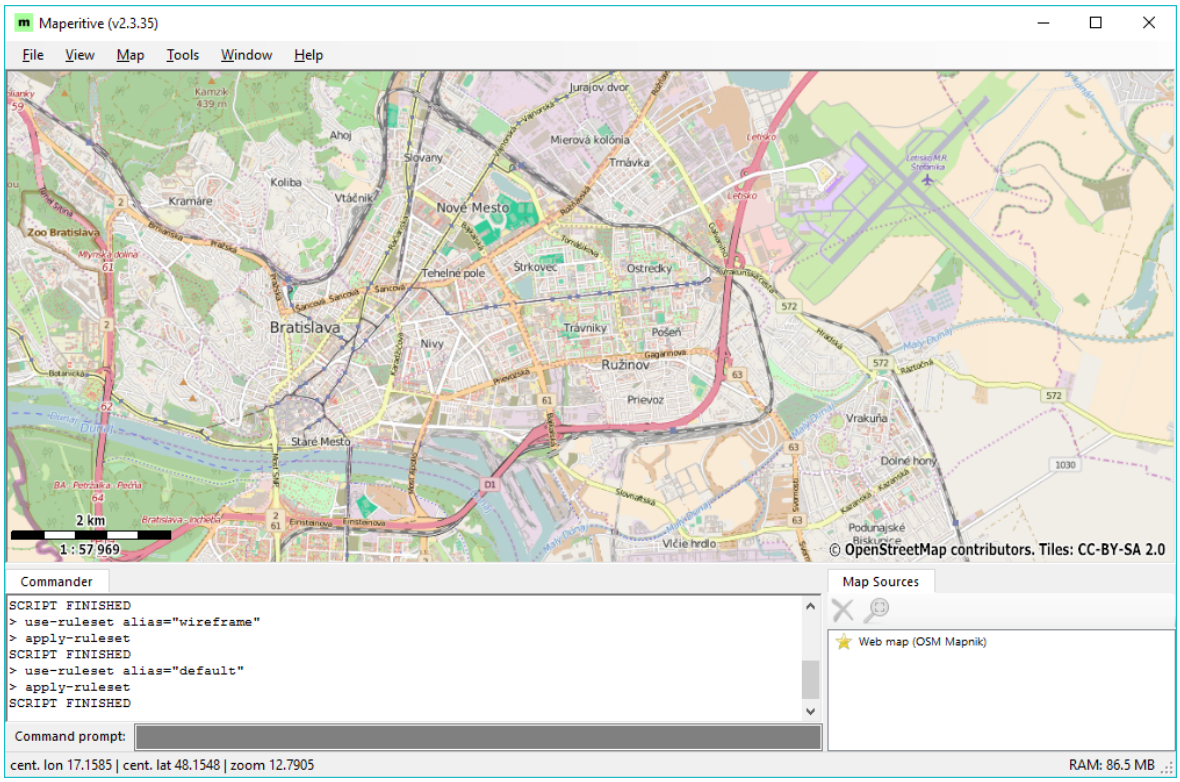
1. **Mapnik** - open source projekt, ktorý je v súčasnosti využívaný na vykresľovanie máp na domovskej stránke OSM. Je naprograomvaný v C++, zvláda rýchlo vykreslovať veľké množstvo dát, preferuje však čítanie dát z PostgreSQL databázy. Vizualizáciu dát z XML súborov zvláda iba v obmedzenom rozsahu a ovláda sa z príkazového riadku.

2. **Osmarander** - v minulosti využívaný na vykresľovanie máp na domovskej stránke projektu OSM, v súčasnosti už nie je aktívne vyvíjaný. Zvláda renderovanie máp vo formáte SVG.

3. **Maperitive** - nástupca populárneho renderovacieho nástroja Kosmos. Ide o desktopovú aplikáciu s vlastným grafickým užívateľským rozhraním, ktorá je implementovaná na operačné systémy Windows, Linux aj Mac OS. Umožňuje sťahovanie mapových dát priamo cez jej grafické rozhranie, renderovanie dát vo formátu PNG a SVG.

Všetky tri nástroje umožňujú ovládať spôsob renderovania štýlovacími súbormi napísanými vždy v ich vlastnom formáte. Z týchto kandidátov sme sa rozhodli používať Maperitive [3]. Medzi jeho nespomenuté výhody patrí automatické znovunačítanie pozmenených XML alebo štýlovacích súborov počas behu programu a dobre zdokumentovaný postup na vytváranie vlastných štýlovacích pravidiel. Pri bakalárskej práci striedavo používame operačné systémy Windows a Linux, preto sa nám hodila aj podpora oboch týchto operačných systémov.

Ukážku grafického rozhrania programu Maperitive môžete vidieť na obrázku 2.2.



Obr. 2.2: Pohľad na Bratislavu v aplikácii Maperitive

Kapitola 3

Implementácia

V tejto kapitole sa zaoberáme konkrétnymi metódami, ktoré sme pri hľadaní dôležitosti ciest využili. Hlavným problémom bol čas behu programov, keďže vstupné súbory pre náš program zaberali rádovo gigabajty pamäťe a obsahovali desiatky tisícov vrcholov a hrán. Od ich parsovania, cez počítanie dôležitosti ciest až po vykresľovanie sme neustále museli hľadať riešenia, ktoré budú fungovať aj pre takto veľké vstupy.

3.1 Parsovanie OSM XML súborov

Existuje mnoho metód parsovania XML súborov. Najrozšírenejšími typmi parsrov sú DOM a SAX parsre.

- **DOM parsre** načítajú celý súbor do operačnej pamäte naraz a až potom vedia z tohoto súboru dáta čítať alebo ich meniť. Tento prístup je pri aktívnej práci s malým súborom ideálny. V každom momente sa vieme pozrieť na ľubovoľné miesto v XML súbore a meniť ho. Zistili sme však, že už pri práci so súbormi o veľkosti v rádoch desiatok megabajtov funguje časovo veľmi neefektívne.
- **SAX parsre** prechádzajú XML súbory riadok po riadku, pamätajúc si, aké párové značky doteraz ešte v XML štrutúre neboli uzatvorené, teda rozumejú kontextu každého elementu v danom XML súbore. Časová zložitosť takejto metódy závisí lineárne od veľkosti vstupu. Neumožňuje však kedykoľvek skočiť na náhodné miesto v XML štruktúre.

V tabuľke 3.1 porovnávame výkon týchto dvoch metód parsovania na našich dátach. Uvádzame v nej oblasti, na ktorých sme parsovanie testovali, veľkosť súborov týchto

Tabuľka 3.1: Porovnanie DOM a SAX parsovania

oblasť	**veľkosť [MB]**	**vrcholy**	**hrany**	**čas (DOM) [m:s]**	**čas (SAX) [s]**
Limbach	2.3	344	356	00:13	<0.5
Allentown	9.6	19151	22193	00:48	<0.5
Abuja	13.7	32629	35000	02:35	1
Pezinok	19	2404	2545	04:11	2
Brighton	26	14641	15868	58:58	3

oblastí, počet vrcholov a hrán, ktoré sme z nich vyextrahovali a nakoniec čas trvania samotného parsovania pre obe metódy.

Z týchto dát je zrejmý nedostatok výkonu DOM parsrov pri práci s väčšími súbormi. Mapové OSM XML súbory, ktoré potrebujeme parsovať dosahujú veľkosti rádovo až stovky megabajtov, kde sa použitie DOM parsra ukázalo ako nemožné.

Spomenuli sme už, že pre určenie dôležitosti ciest potrebujeme v našej mape hľadať veľké množstvo najkratších ciest. Za týmto účelom dáta o cestách a križovatkách reprezentujeme v grafovej štruktúre, do ktorej dáta z OSM XML súborov potrebujeme dostať. Nepotrebujeme teda aktívne pracovať so samotnými XML súbormi, stačí nám ich jediný krát prejsť, vybrať potrebné prvky a vložiť ich do nášho grafu ako vrcholy a hrany. Žiadne z výhod DOM parsrov nám teda pri použití SAX parsra chýbať nebudú.

3.1.1 BasicOSMParser

Knižnicu BasicOSMParser sme už spomenuli, uvedieme teda aj ako funguje a v čom sme ho pre naše potreby upravili.

BasicOSMParser je knižnica, ktorú v jazyku Java naprogramoval Adrien Pavie a jej zdrojové kódy ponúkol pod licenciou GNU GPL všetkým, ktorí by tento nástroj chceli využiť alebo upraviť podľa vlastných požiadaviek.

Samotná knižnica prechádza vstupný súbor a ukladá si informácie o prvkoch *node*, *way* a *relation* ako inštancie tried *Node*, *Way* alebo *Relation*, podľa toho o aký prvok sa jedná. Jej výstupom je hashovacia tabuľka, kde kľúčom tejto tabuľky je ID prvku z OSM XML dát a hodnotou je smerník na inštanciu triedy daného prvku.

My si potrebujeme o každom z prvkov pamätať nasledovné:

- **node** - ID, zemepisná výška a šírka (kvôli výpočtu dĺžky ciest), zoznam tagov, ktoré k danému prvku prislúchajú

- **way** - ID, zoznam prvkov *node*, ktoré tvoria daný *way*, zoznam tagov, ktoré k danému prvku prislúchajú
- **relation** - ID, zoznam prvkov, ktoré tento prvok *relation* združuje, zoznam tagov, ktoré k danému prvku prislúchajú

Pôvodne si BasicOSMParser o každom prvku uchovával aj veľké množstvo ďalších informácií ako napríklad posledná zmena prvku, prezývka používateľa, ktorý zmeny vykonal atď., tieto sme však nikdy nevyužili. Tento parser sme preto upravili tak, aby informácie tohto druhu ignoroval a zefektívnili sme tým jeho výkon pre naše potreby.

3.2 Grafová reprezentácia mapy

Údaje o križovatkách a cestách reprezentujeme pre riešenie problému najkratšej cesty ako graf. Naša pôvodná implementácia využívala vlastný dátový model obsahujúci samostatné triedy pre orientovaný ohodnotený graf, pre hrany a pre vrcholy tohto grafu. Na takomto grafe sme najkratšie cesty hľadali pomocou Dijkstrovho algoritmu [21], ktorý dokáže nájsť najkratšie cesty z jediného vrchola do všetkých ostatných vrcholov grafu. Ten sme implementovali ako ďalšiu triedu programu.

Pri počítaní dôležitosti ciest sme v neskoršej fáze riešenia tohto problému chceli porovnať výkon rôznych algoritmov riešiacich problém najkratšej cesty. Najrozumnejšie preto bolo použiť robustnejšiu grafovú štruktúru, ktorá ponúka väčšie množstvo implementovaných algoritmov na hľadanie najkratšej cesty v grafe. Použili sme preto už spomenutú knižnicu tried a algoritmov JGraphT.

3.2.1 Triedy knižnice JGraphT

Pri práci s knižnicou JGraphT je potrebné najskôr vytvoriť inštanciu vhodnej grafovej štruktúry a následne do nej vkladať inštancie tried vrcholov a hrán.

O grafe,ktorý má reprezentovať cestnú sieť existujúcu v reálnom svete nevieme vysloviť žiadne predpoklady. Môže obsahovať viacnásobné hrany medzi dvoma vrcholmi a môže taktiež obsahovať hrany, ktorých počiatočný a koncový vrchol je identický. Kvôli jednosmerným cestám, ktoré sa v cestnej sieti často vyskytujú musíme rátať s tým, že hrany nemusia byť obojsmerné. To znamená, že graf musí mať orientované hrany. Váhy hrán v mape nie sú rovnaké, ale vypočítame ich ako dĺžku úsekov ciest, ktoré hranu reprezentujú. Potrebujeme preto vytvoriť orientovaný, ohodnotený graf, ktorý môže

obsahovať násobné hrany a slučky. Trieda spĺňajúca tieto naše požiadavky je zároveň najvšeobecnejším typom grafu spomedzi všetkých tried grafov implementovaných v JGraphT. Názov tejto triedy je *DirectedWeightedPseudograph*.

Existuje iba jediná trieda hrany, ktorú môžeme do takéhoto grafu pridávať. Volá sa $DefaultWeightedEdge$. Vrcholy nemajú triedy, medzi ktorými by sme museli vyberať, pridávajú sa volaním metódy grafu.

3.2.2 Identifikácia ciest v OSM XML

Po sparsovaní XML dokumentu a vytvorení dátovej štruktúry, ktorá vie cestnú sieť reprezentovať ako graf, je potrebné v prvom rade identifikovať, ktoré dáta v mapách sú cesty a v druhom rade pre ktoré cesty chceme počítať ich dôležitosť.

Každý *way* v štruktúre OSM, ktorý je zároveň cestou, musí byť označený prvkom *tag* s kľúčom *highway*. Môže ísť pritom napríklad o cyklistickú cestu, chodník, schody alebo diaľnicu. Rôznych typov ciest je veľa, no ak po danej ceste môžu jazdiť aj automobily, dokumentácia OSM stanovuje, že hodnota tohoto tagu musí byť jednou z nasledovných:

(Čísla v zátvorke informujú, aké percento všetkých ciest z tohoto zoznamu sú daného typu. Tieto údaje sme čerpali z projektu Taginfo [12], ktorý generuje rôzne štatistiky o OSM mapách.)

1. *motorway* (0.93%) - diaľnice, ktoré majú väčšinou spodný aj horný rýchlostný limit, teda v niektorých prípadoch nie sú prístupné chodcom alebo pomalším dopravným prostriedkom
2. *trunk* (1.01%) - podobné cestám typu *motorway*, avšak môžu obsahovať jednoduché križovatky, teda úseky, v ktorých je možné dosiahnuť maximálnu rýchlosť sú kratšie
3. *primary* (2.5%) - cesty prvej triedy spájajúce mestá, s maximálnou rýchlosťou medzi 90-100 km/h, no iba 30-50 km/h ak prechádzajú rezidenčnou oblasťou
4. *secondary* (3.68%) - cesty podobné cestám prvej triedy, avšak spájajúce menej významné oblasti, napríklad obce
5. *tertiary* (5.88%) - cesty podobné cestám druhej triedy, avšak spájajúce menej významné oblasti, napríklad menšie dediny
6. *motorway_link* (0.74%) - nájazdy a výjazdy z diaľnice

7. *trunk_link*, *primary_link*, *secondary_link* a *tertiary_link* (1.06%) - označujú veľmi krátke úseky prepájajúce dôležitejšie cesty, napríklad priama odbočka vpravo tesne pred kruhovým objazdom, ktorá umožňuje automobilom vyhnúť sa kruhovému objazdu

8. *residential* (46.64%) - rezidenčné cesty v rezidenčných oblastiach s maximálnou rýchlosťou zväčša medzi 30-50 km/h, tieto cesty často obsahujú retardéry a iné prvky, ktoré na nich skľudňujú premávku

9. *living$_s$treet* (1.01%) - cesty, v ktorých má absolútnu prednosť chodec, alebo pomalšie dopravné vozidlo, vodiči automobilov sú vyslovene odrádzaní využívať tieto cesty

10. *service* (20.28%) - označuje súkromné cesty, parkoviská alebo cesty umožňujúce prístup k budovám (príjazdové cesty)

11. *track* (15.48%) - cesty pre poľnohospodárske účely

12. *pedestrian* (0.57%) - cesty v peších zónach, ktoré za určitých okolností umožňujú vstup automobilov, napríklad za účelom zásobovania

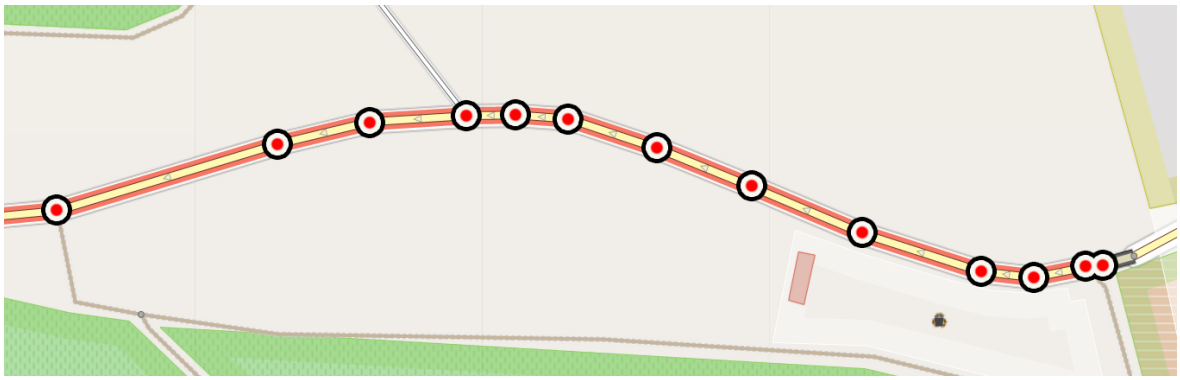
V rôznych situáciách nás môže zaujímať dôležitosť rôznych ciest, preto je náš program navrhnutý tak, aby sa zoznam typov ciest, ktoré chceme vo výpočtoch zahrnúť dal jednoducho meniť.

V našich výpočtoch sme však chceli zahrnúť práve tie typy ciest, ktoré sú určené na bežnú premávku automobilov, teda tie, v ktorých automobily nie sú obmedzované prísnymi znevýhodňujúcimi dopravnými pravidlami alebo neudržiavaným povrchom vozovky. Preto v našich výpočtoch zahŕňame a zároveň dôležitosť ciest vyhodnocujeme pre cesty vymenované v prvých ôsmich odrážkach tohoto zoznamu. Ide vždy o diaľnice, rýchlostné cesty, cesty prvej až tretej triedy, ich spojovacie cesty, výjazdy, nadjazdy a cesty rezidenčných oblastí.

3.3 Výpočet dĺžok ciest

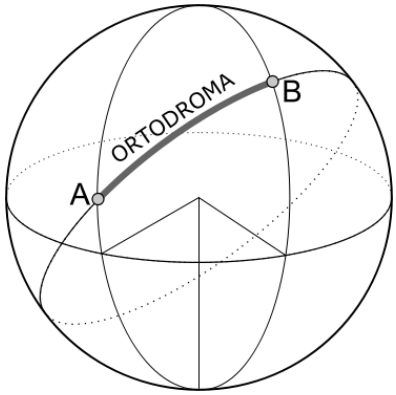
Aby sme vedeli správne hľadať najkratšie cesty v grafe, potrebujeme poznať váhy jeho jednotlivých hrán. Na to potrebujeme poznať reálnu dĺžku cestných úsekov, ktorú si OSM nikde neukladá, teda ju potrebujeme vypočítať. Dôležitý je poznatok, že formát OSM XML neponúka žiadnu formu reprezentácie zaoblených hrán, pracuje iba s úsečkami medzi prvkami *node* v jej štruktúre.

Pre reálne cesty je výnimočné, ak sú dokonale rovné, v OSM XML sú preto reprezentované zoznamom bodov a ich geografickými súradnicami, ktoré sú v miestach s rôznorodým zaoblením hustejšie. Príklad jediného prvku *way* reprezentovaného dokopy trinástimi prvkami *node*, ktoré definujú tvar zaoblenia cesty znázorňujeme na obrázku 3.1



Obr. 3.1: Body reprezentujúce zaoblenie cesty

Body, ktoré cesty v štruktúre OSM XML obsahujú, sú vo vnútri prvkov *way* uvedené v tom istom poradí, v akom za sebou nasledujú na ceste. Dĺžku celej cesty vieme vypočítať ako súčet vzdialeností všetkých susedných dvojíc týchto bodov a každú z týchto vzdialeností vypočítame ako dĺžku ortodrómy medzi týmito dvojicami.



Obr. 3.2: Ortodróma medzi bodmi A a B

Ortodróma, zobrazená na obrázku 3.2, je definovaná ako najkratšia spojnica dvoch bodov na guľovej ploche, v našom prípade na povrchu Zeme. Pre výpočet dĺžky ortodrómy používame "Haversine"vzorec, ktorý je často využívaný v automobilových navigáciách. Tento vzorec spomínajú J. J. Mewmezi a Youfang Huang v ich článku, ktorý sa venuje sférickým vzdialenostiam [18]. Implementáciu tohoto vzorca sme prebrali zo stránok Movable Type Scripts [16], v ktorej zo zemepisných súradníc $lat1$, $lon1$, $lat2$ a $lon2$ vypočítame ich vzdialenosť pomocou nasledovnej funkcie:

```
public static double getDistance(double lat1, double lon1, double lat2, double lo
```

```
    double lon_diff = lon2 - lon1;
    double lat_diff = lat2 - lat1;
    ... //prevedenie všetkých uhlov na radiány
    double rad = 6378137;
    double a = Math.sin(lat_diff/2)*Math.sin(lat_diff/2) +
    Math.cos(lat1)*Math.cos(lat2) *
    Math.sin(lon_diff/2)*Math.sin(lon_diff/2);
    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
    return c*rad;
}
```

kde lon_diff a lat_diff sú rozdiely zemepisných výšok a šírok v radiánoch, rad je polomer zeme v metroch, a je druhá mocnina polovice dĺžky tetivy medzi 2 bodmi a c je veľkosť stredového uhla medzi týmito bodmi v radiánoch.

3.4 Výpočet hodnoty parametra reach

Už sme naznačovali, že pri práci na tomto probléme bojujeme kvôli objemu dát, s ktorým narábame, s časom trvania všetkých operácií. Sťahovanie dát a následnú extrakciu údajov o cestnej sieti z týchto dát sme vyriešili dostatočne efektívne na to, aby sme vedeli narábať napríklad s mapou celej Bratislavy. Najkritickejšou časťou celého procesu určovania dôležitostí ciest je však výber vhodných algoritmov, ktoré hľadajú najkratšie cesty v grafe cestnej siete. V tejto časti textu sa preto venujeme algoritmom na hľadanie najkratších ciest v grafe a optimalizáciam, ktoré sme vykonali, aby sme pomocou týchto algoritmov časovo oveľa evektívnejšie získali dostatočne presné výsledky.

3.4.1 Floyd-Warshall algoritmus

Tento algoritmus slúži na nájdenie najkratších ciest medzi úplne všetkými dvojicami bodov v grafe. Jeho v súčasnosti najrozšírenejšiu podobu publikoval Robert Floyd a je založená na algoritme hľadajúcom tranzitívny uzáver, ktorý publikoval Stephen Warshall. Základnou charakteristikou tohto algoritmu je trojica navzájom vnorených for-cyklov. Horný odhad časovej zložitosti tohto algoritmu je $\Theta(V^3)$ a horný odhad pamäťovej zložitosti je $\Theta(|V|^2)$ kde V je počet vrcholov v grafe.

Aplikovanie Floyd-Warshallovho algoritmu na riešenie nášho problému sa však ukázalo ako neefektívne. Pri aplikovaní tohoto algoritmu na malé dediny beh programu

trval vždy rádovo najviac desiatky sekúnd, avšak pri pokuse o nájdenie všetkých najkratších ciest pre väčšie oblasti, napríklad pre Bratislavu, sa beh programu prerušil pre nedostatok operačnej pamäte.

Prišli sme preto s novým prístupom. Namiesto hľadania najkratších ciest medzi všetkými vrcholmi grafu súčasne ich budeme hľadať postupne. Pre každý bod v našom grafe spustíme algoritmus pre hľadanie najkratších ciest do všetkých ostatných bodov v grafe. Vďaka tomuto prístupu sa nám podarilo hľadať najkratšie cesty naprieč mapami celej strednej Európy.

Najrozšírenejšími algoritmami na riešenie problému najkratších ciest do všetkých bodov grafu z jediného počiatočného bodu sú Bellman-Fordov algoritmus a Dijkstrov algoritmus.

3.4.2 Bellman-Fordov algoritmus

Tento algoritmus slúži práve na hľadanie najkratších ciest v orientovanom ohodnotenom grafe. Funguje aj na grafoch s negatívnymi hodnotami cien hrán.

Jeho časová zložitosť tohoto algoritmu je $\Theta(V \cdot E)$, kde V je počet vrcholov a E počet hrán v grafe.

3.4.3 Dijkstrov algoritmus

Existujú rôzne implementácie tohoto algoritmu. Jeho základným princípom je udržiavanie si zoznamu vrcholov, ktoré sme ešte neprehľadali, ktoré však susedia s už prehľadanými vrcholmi. Pri každej iterácii tohoto algoritmu navštívime zatiaľ neprehľadaný vrchol, do ktorého vedie najlacnejšia cesta zo všetkých neprehľadaných vrcholov. Pre každý navštívený vrchol tak zároveň zistíme, aká je najlacnejšia cesta, ktorou sa doň vieme z počiatočného vrchola dostať. Ak nám stačí nájsť cestu z vrcholu A do vrchola B, spustíme ho s počiatkom vo vrchole A a algoritmus zastavíme v momente, keď prvý raz navštívime vrchol B. Ak algoritmus neprerušíme a necháme ho graf prehľadávať, až kým nie sú prehľadané všetky vrcholy, nájde najlacnejšiu cestu do úplne všetkých vrcholov grafu.

Ceny hrán v grafe cestnej siete nebudú za žiadnych okolností záporné. Dĺžka ciest, ani čas potrebný na prejdenie cesty nikdy nebudú záporné. Preto nepotrebujeme pracovať s Bellman-Ford algoritmom a môžeme použiť Dijkstrov algoritmus, ktorého horný odhad časovej zložitosti je rýchlejší, než u Bellman-Ford algoritmu.

Jeho časová zložitosť je $\Theta(E + V \log V)$.

Pri implementácii Dijkstrovho algoritmu v našom programe najprv využívame jeho formu, ktorá vyhľadá najkratšie cesty do všetkých vrcholov z jediného počiatku, pri optimalizáciach pre väčšie územia používame jeho formu, ktorá zastaví prehľadávanie grafu po nájdení cieľového vrchola.

3.5 Výkon algoritmov na reálnych dátach

Na otestovanie našich algoritmov sme si pripravili sadu testovacích súborov menších miest z rôznych častí sveta. Konkrétne Allentown, Abuju a Brighton uvedené v tabuľke 3.1. Avšak pri výbere týchto miest sme boli príliš optimistickí. Napríklad mapa Brightonu, obsahuje 214.4 milióna rôznych dvojíc bodov a medzi všetkými potrebujeme nájsť najkratšiu bodov. Pri testovaní sme zistili, že vyhľadanie najkratších ciest medzi úplne všetkými dvojicami bodov by trvalo približne hodinu. Naplnenie nášho cieľa, určiť dôležitosti ciest pre mapy rozsahu Bratislavy, ktorá obsahuje 6-násobne väčšie množstvo vrcholov a hrán, než mapa Brightonu, by nebolo kvôli časovej zložitosti Dijkstrovho algoritmu reálne. Náš algoritmus sme na základe týchto zistení testovali na ešte menších vstupoch, ktoré sú uvedené v tej istej tabuľke. Na mape obce Limbach trvalo presné vyčíslenie parametra reach pre každý bod v mape 1 sekundu a pre mapu Pezinka to bola minúta a pol.

3.6 Optimalizácie algoritmu

Aby sme dokázali určovať dôležitosť ciest aj pre väčšie oblasti, musí dojsť ku kompromisom medzi kvantitiou a kvalitou. Nemôžeme hľadať najkratšie cesty medzi všetkými dvojicami bodov. Ukrojením počtu hľadaných ciest znížime presnosť parametra reach, ktorý pre každý vrchol v grafe vypočítame, no budeme hľadať spôsob, ako ukrojiť čo najväčší počet týchto najkratších ciest, aby sme stratili čo najemenej z informácie o dôležitosti ciest.

3.6.1 Aproximácia hodnoty parametra reach

Ak nebudeme hľadať všetky najkratšie cesty v grafe, nemôžeme hovoriť, že hľadáme parameter reach pre všetky vrcholy grafu. Čo v skutočnosti pri tejto optimalizácii budeme hľadať, bude spodné ohraničenie hodnoty parametra reach pre každý vrchol.

Pri hľadaní najkratších ciest si teda pre každý bod pamätáme iba najvyššiu hodnotu doteraz nájdeného potenciálneho parametra reach. Píšeme potenciálneho, pretože parameter reach vieme zistiť iba prehľadaním všetkých najkratších ciest.

Ak hľadáme spodné ohraničenie reach parametra pe každý vrchol grafu, na začiatku hľadania najkratších ciest priradíme každému vrcholu ako reach parameter hodnotu 0. Vždy, keď podľa definície reach parametra z kapitoly 1 nájdeme pre niektorý bod väčšiu hodnotu tohto parametra, prepíšeme tú doterajšiu novou, väčšou hodnotou. Pri hľadaní ľubovoľnej najkratšej cesty medzi dvoma vrcholmi grafu z definície reach parametra vyplýva, že dôležitejším vrcholom prejde náhodne zvolená najkratšia cesta s väčšou pravdepodobnosťou, než menej dôležitým bodom. Pri hľadaní spodného ohraničenia reach parametra to znamená, že dôležitejším bodom určíme reach parameter presnejšie, než menej dôležitým bodom. Taktiež sa môže stať, že v žiadnej z najkratších ciest (keďže nebudeme prehľadávať všetky) sa niektorý vrchol nikdy nevyskytne, aj keď nie je izolovaný. Vtedy jeho spodné ohraničenie reach parametra bude 0. Keďže menej dôležité vrcholy sa však vyskytujú v najkratších cestách menej často, je pravdepodobné, že takáto situácia nastane iba pre menej dôležité body.

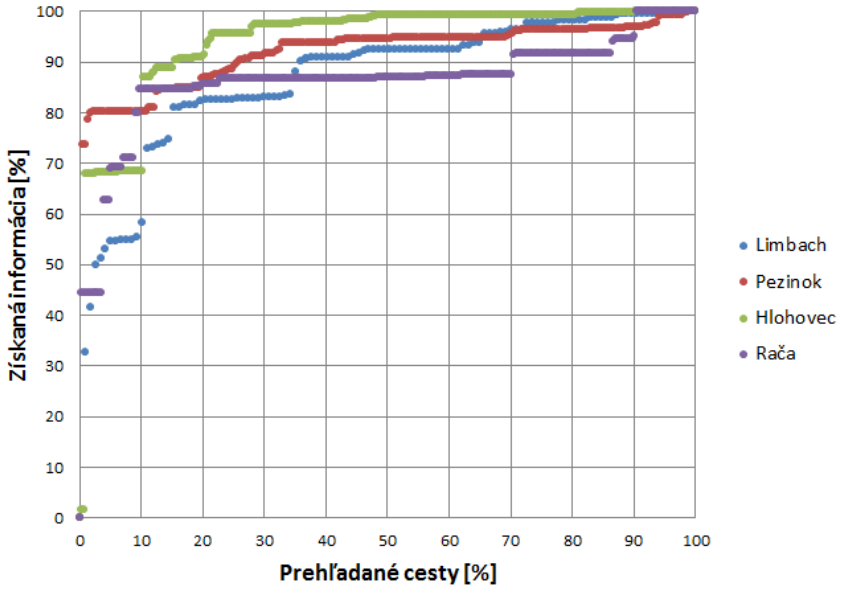
Takýto výsledok budeme považovať za dostatočne presný. Najviac nám totiž pri určovaní kvality pokrytia cestnou sieťou záleží na identifikovaní najdôležitejších ciest.

3.6.2 Výber podmnožiny najkratších ciest

Keďže nehľadáme najkratšie cesty medzi všetkými dvojicami bodov v grafe, ale iba ich podmnožinu, zistili sme, že záleží na tom, aká veľká táto podmnožina je a akým spôsobom ju vyberáme.

Aby sme dokázali testovať, ako množstvo ciest a ich výber ovplyvňuje kvalitu výsledných hodnôt, pri hľadaní všetkých najkratších ciest v celom grafe sme v pravidelných intervaloch vypisovali súčet spodných ohraničení všetkých vrcholov. To sme urobili pre štyri rôzne oblasti s rôznymi charakteristikami cestnej siete, napríklad jedno malé mesto alebo viacero dedín, ktoré sú navzájom poprepájané cestami, pretože hľadanie parametra reach sa v nich vyvíja rôzne. Pre prepojené dediny môžeme predpokladať, že najdôležitejšie budú práve cesty, ktoré tieto dediny spajajú a pre mestá sú to zväčša cesty prechádzajúce ich stredom.

Výsledky testov ukazujeme v grafe na obrázku 3.3, kde na x-ovej osi zobrazujeme percentá prehľadaných najkratších ciest spomedzi všetkých možných a na y-ovej osi aké percentá z celkovej sumy hodnôt parametrov reach sa práve nachádzajú v súčte všetkých spodných ohraničení tejto hodnoty.



Obr. 3.3: Presnosť parametra reach vzhľadom na počet hľadaných ciest

Z grafu je možné vyčítať, že vždy stačilo prejsť 20% všetkých najkraších ciest grafu, aby sme získali 80% z kompletnej informácie o parametroch reach všetkých bodov grafu.

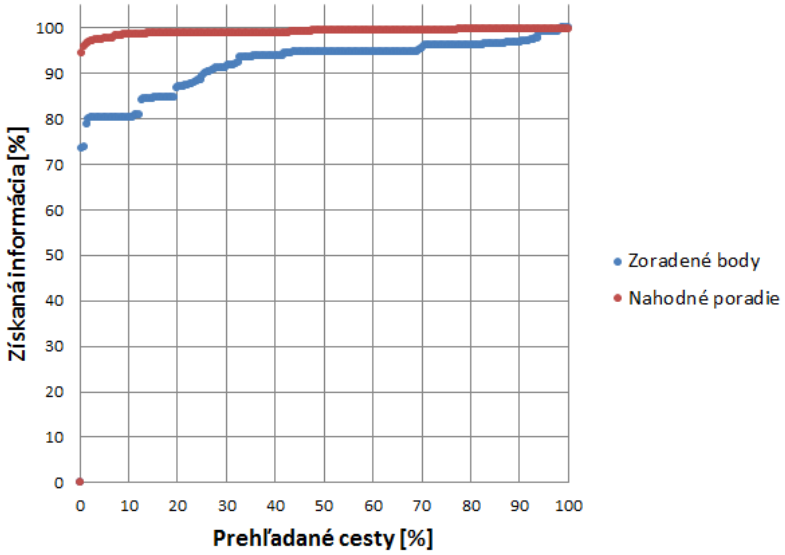
V našej implementácii hľadania hodnoty reach pritom postupne prechádzame zoznam všetkých vrcholov nášho grafu v rovnakom poradí, v akom sme ich načítali z OSM XML štruktúry a pre každý bod hľadáme najkratšiu cestu do všetkých ostatných bodov a to znova v poradí, v akom sme ich načítali z máp.

Pri hľadaní najkratších ciest medzi vrcholmi v tom istom poradí, v akom sú uvedené v mapových súboroch, sú často dva po sebe idúce body súčasťou tej istej cesty. Taktiež sme si pri práci s týmito dátami všimli, že cesty, ktoré sú v geografickej blízkosti, sú blízko seba aj OSM XML súbore. Dáta sme sťahovali z rôznych zdrojov a neexistujú pravidlá, ktoré určujú, v akom poradí musia byť dáta v OSM XML súboroch uvedené, hovoríme preto iba o pozorovaní a nie o fakte. Zistili sme však, že ak neustále hľadáme najkratšie cesty, ktoré vychádzajú z podobnej oblasti, dosahujeme horšie výsledky, ako keď počiatočné a koncové body, medzi ktorými hľadáme najkratšie cesty vyberáme náhodne.

Výsledky našich testov ukazujeme na obrázku 3.4.

Z týchto testov vyplynulo, že ak najkratšie cesty vyberáme náhodne, stačí nájsť menej, než 0,5% všetkých najkratších ciest v grafe, aby sme získali v súčte spodných ohraničení parametra reach aspoň 90% zo súčtu presných hodnôt tohoto parametra.

Na základe týchto testov vo finálnej implementácii hľadáme iba jedinú najkratšiu



Obr. 3.4: Porovnanie prehľadávania ciest v zoradenom a v náhodnom poradí

cestu pre každý vrchol v grafe. Konkrétne z každého vrcholu v grafe hľadáme najkratšiu cestu do náhodného iného vrcholu. Týmto zaručíme, že všetky najkratšie cesty budú odlišné, pretože existuje práve jedna najkratšia cesta z ľubovoľného bodu, ktorú hľadáme a zároveň hľadanie najkratšej cesty z bodu A do bodu B nie je v orientovanom grafe to isté, ako hľadanie najkratšej cesty z bodu B do bodu A, preto nie je na závadu ak niekedy nastane prípad, že hľadáme najkratšie cesty medzi dvoma bodmi v oboch poradiach. Pri takejto implementácii je súčet spodných ohraničení hodnôt parametra reach všetkých vrcholov vždy blízko 90%. To je pre nás úplne dostatočný pomer medzi počtom ciest, ktoré potrebujeme prehľadať a výsledkom, ktorý z nich získame.

3.6.3 Optimalizácia algoritmu

Pri hľadaní najkratších ciest využívame (ako sme už spomínali) Dijkstrov algoritmus, ktorý hľadá najkratšie cesty do všetkých vrcholov z jediného, počiatočného vrcholu. Vysvetlili sme aj ako presne funguje a že existuje jeho implementácia, ktorá zastaví prehľadávanie v momente, keď nájde koncový vrchol. Keďže pre každý bod hľadáme iba cestu do jediného z cieľov, najväčšie urýchlenie spočíva práve v ukončení prehľadávania grafu po nájdení cieľového bodu.

3.7 Vizualizácia dôležitosti ciest

Pri tejto práci sme vizualizáciu dôležitostí ciest vnímali ako dôležitý cieľ. Čítať informácie o dôležitosti jednotlivých cestných úsekov z mapy nám poskytne oveľa viac

informácií a v lepšom kontexte, než keby sme mali túto informáciu čítať ako text.

Pri rozhodovaní sa, ako budeme dáta vizualizovať sme zvažovali možnosť vytvoriť vlastnú aplikáciu na vykresľovanie máp s našim dodatočným údajom o dôležitosti ciest. Zároveň sme chceli však informáciu o dôležitosti každého úseku dostať naspäť do OSM XML súborov, aby sa s nimi dalo narábať aj mimo programu, ktorý sme vytvorili my. Keďže po celú dobu pracujeme s mapovými dátami projektu OSM, je najrozumnejšie naše výsledné dáta reprezentovať v tej istej štruktúre, v akej sme ich získali.

V kapitole 1 sme spomenuli, že sme našli množstvo už existujúcich kvalitných metód na renderovanie máp do obrázkov, ktoré taktiež umožňujú kontrolovať, akým spôsobom sa tieto dáta renderujú. To nám umožní vytvoriť vlastné pravidlá pre vykresľovanie dôležitostí ciest priamo v mapách. Vybrali sme preto aplikáciu Maperitive spomínanú v prvej kapitole a naštudovali sme si syntax pravidiel, ktoré v tejto aplikácií ovládajú vykresľovanie máp.

3.7.1 Dôležitosť cesty v OSM XML

Prvý krok potrebný pre samotným vykreslením máp je určiť, ako a aké dáta budeme vkladať naspäť do máp, v ktorej sme dôležitosti určovali. Náš algoritmus podľa definície parametra reach určil dôležitosť všetkých bodov, ktoré v mape reprezentujú cestu. Pre dátovú štruktúru, v akej si OSM mapy uchováva, dáva však väčší zmysel, než vizualizovať dôležitosť týchto bodov práve vizualizácia dôležitosti celých ciest, na ktorých tieto body ležia. V cestnej sieti vieme cesty logicky rozdeliť na cestné úseky na základe križovatiek, ktoré môžu určovať ich začiatok a koniec. Pre jednotlivé prvky *way*, ktoré v OSM reprezentujú cestné úseky však nič podobné neplatí. Môžu byť v podstate ľubovoľne dlhé a môžu prechádzať cez väčšie množstvo križovatiek, no v praxi pozostáva prvok *way* maximálne rádovo z desiatok prvkov *node*.

Prvky *way* preto nedelíme na menšie časti a dôležitosť týchto prvkov vyčíslime ako maximálnu hodnotu spomedzi hodnôt reach všetkých prvkov *node*, ktoré tento prvok *way* tvoria. Tieto hodnoty reach počítame z cien hrán v grafe a ceny hrán v grafe sú rovné dĺžkam ciest v reálnom svete s presnosťou v metroch. Hodnoty parametra reach v malej obci preto nadobúdajú hodnoty v intervale od 0 až po desiatky tisícov.

Dôležitosti ciest chceme na mape odlíšiť farebne. Napríklad dôležitejšia cesta by mala mať tmavší odtieň tej istej farby, akú má menej dôležitá cesta. Nebudeme však používať niekoľko tisíc odtieňov tej istej farby, v záujme prehľadnosti použijeme iba osem rôznych odtieňov. Do máp preto budeme vkladať vypočítané spodné ohraničenie parametrov reach predelené vhodnou konštantou, rovnou osmine najväčšej nájdenej

hodnoty reach.

Pre vkladanie dôležitosti ciest do štruktúry OSM XML sme sa rozhodli dôležitosť každého prvku *way*, ktorého dôležitosť sme vypočítali, doň vložiť ako jeho nový prvok *tag*, ktorého kľúč bude *reach* a hodnota bude rovná nami vypočítanej hodnote parametra reach. Výsledná cesta v OSM XML štruktúre s pridanou informáciou o jej dôležitosti vyzerá nasledovne:

```
<way id="232671283" timestamp="2013-08-07T16:26:12Z" user="Chrobák">
    <nd ref="26424110"/>
    <nd ref="2410080871"/>
    <nd ref="2410080869"/>
    <tag k="highway" v="residential"/>
    <tag k="name" v="Potočná"/>
    <tag k="reach" v="2"/>
</way>
```

Za účelom vkladania nových prvkov do OSM XML štruktúry sme využili podobnú metódu, akú sme použili na jej čítanie. Za pomoci knižnice SAX [1] programovacieho jazyka Java sme naprogramovali vlastnú triedu, ktorá ako vstup dostane zoznam ID prvkov *way*, ktorých dôležitosť sme našim algoritmom vypočítali a ku každému ID taktiež vypočítanú dôležitosť ako číslo z rozsahu 0 až 7. Táto trieda prejde celý OSM XML súbor a vždy, keď nájde prvok *way*, ktorého ID je aj v zozname, ktorý sme jej poskytli, vloží do vnútra tohto prvku spomínaný *tag*.

3.8 Renderovanie máp

Na renderovanie dát používame aplikáciu Maperitive, ktorá je dostupná zdarma. Táto aplikácia renderuje dáta v reálnom čase, môže renderovať iba tú časť mapy, ktorá sa práve nachádza vo vnútri zobrazovacieho okna, dokáže však vyrenderovať aj celú mapu naraz a exportovať ju do obrázkových formátov PNG a SVG.

3.8.1 Renderovacie pravidlá

Pravidlá, ktorými sa aplikácia Maperitive riadi pri renderovaní z mapových dát sú uchované vo vlastných súboroch s koncovkou $*.mrules$. V závislosti od toho, aké informácie chceme z vykreslovaných máp aktuálne čítať môžeme voliť, ktorý súbor s renderovacími pravidlami sa má použiť.

Pre účely vykresľovania dôležitosti ciest do máp sme vytvorili úplne nové, vlastné pravidlá. Maperitive poskytuje na svojej webovej stránke základnú dokumentáciu ktorá vysvetluje, ako tieto pravidlá fungujú a ako sa vytvárajú [17]. Pre lepšie pochopenie ich syntaxe sme si preštudovali aj rôzne, už existujúce renderovacie pravidlá pre túto aplikáciu.

Pri vytváraní renderovacích pravidiel sme si dali za cieľ nasledovné veci:

- malo by byť zrejmé, aké oblasti sa na mape nachádzajú, vypíšeme preto ich názvy
- vyššiu dôležitosť cesty vizuálne odlíšime podobnou farbou v tmavších odtieňov
- cesty, ktorých dôležitosť neurčujeme, pretože po nich nemôžu prechádzať autá vykreslíme menej výrazne ako cesty, ktorých dôležitosť sme určovali
- aby bol jasnejší kontext ciest v cestnej sieti, vykreslíme aj všetky budovy, prírodné plochy (lesy, parky...) a vodné plochy (rieky, jazerá, moria...)
- pre najdôležitejšie cestné úseky celej mapy vypíšeme aj popisné číslo týchto ciest
- šípkami znázorníme, ktoré cesty sú jednosmerné

V týchto pravidlách vieme upresniť, aké údaje sa vykresľujú alebo nevykresľujú pri určitom priblížení. Napríklad pri pohľade na celú krajinu chceme zobrazovať iba názov tejto krajiny a nie názvy všetkých dedín v nej, no pri priblížení nás zaujímajú práve názvy jednotlivých menších oblastí. V našich pravidlách sme taktiež využili dedenie vlastností zo všeobecnejších prvkov na konkrétnejšie. Napríklad všetkým cestám, ktoré majú definovanú dôležitosť sme definovali rovnakú šírku, ohraničenie a hodnota dôležitosti danej cesty určuje už iba farbu, akou sa má daná cesta vykresliť.

Na obrázku 3.5 ukazujeme farebnú škálu, ktorú sme vytvorili pre rôzne dôležitosti ciest. Hodnota 0 reprezentuje najmenej dôležitý a 7 je najdôležitejší cestný úsek v mape, ktorú zobrazujeme.

Obr. 3.5: Farebná škála pre rozlíšenie dôležitosti cestných úsekov

Ukážky obrázkov, ktoré boli vyrenderované pomocou našich pravidiel uvádzame v kapitole 4.

3.9 Ukazovateľ kvality pokrytia siete

Cieľom tejto bakalárskej práce je aj analýza kvality cestnej siete. Chceme dosiahnuť porovnanie kvality rôznych cestných sietí vzhľadom na rozloženie dôležitosti ciest. Najjednoduchší spôsob, ako porovnať dve oblasti, je túto kvalitu cestnej siete vyjadriť jediným číslom, ktoré zahŕňa kvalitu celej cestnej siete.

Dosiahli sme to vyčíslením koeficientu korelácie medzi dôležitosťou a priepustnosťou všetkých cestných úsekov. Čo myslíme pod priepustnosťou sme vysvetľovali v kapitole 1. Zopakujme, že týmto pojmom myslíme množstvo áut, ktoré je teoreticky schopné daným cestným úsekom prejsť. Priepustnosť podľa nás ovplyvňujú semafóry, intervaly týchto semaforov, kruhové objazdy, kvalita vozovky, maximálna rýchlosť a počet jazdných pruhov. Do úvahy v našich výpočtoch však berieme iba počet jazdných pruhov a maximálnu rýchlosť, pretože o ostatných spomenutých parametroch cesty vieme vysloviť iba veľmi nepresné predpoklady a ich zakomponovanie vo vyčísľovaní priepustností môže spôsobiť jej väčšiu nepresnosť. Priepustnosť cestného úseku teda vyčíslime ako súčin maximálnej rýchlosti a počet jazdných pruhov tohoto úseku.

Kvalitu cestnej siete určitej oblasti potom vyčíslime ako Pearsonov korelačný koeficient [19] medzi priepustnosťou a dôležitosťou cestného úseku. Z našej definície dôležitosti ciest vyplýva, že čím je cesta dôležitejšia, tým väčší počet áut bude chcieť po tejto ceste prejsť. Preto by dôležitosť všetkých cestných úsekov mala korelovať s ich priepustnosťou. Koeficient korelácie zrkadlí, na akom množstve cestných úsekov a ako výrazne tieto dva parametre navzájom korelujú.

Aby sme mohli vypočítať priepustnosť cestných úsekov, potrebujeme z mapovej štruktúry OSM získať ich maximálnu rýchlosť a počet jazdných pruhov. Oba sú v štruktúre OSM uložené ako *tag* vo vnútri prvku *way*, ktorého sa týkajú. Pre počet jazdných pruhov sú to tagy s kľúčom *lanes* a ich hodnotami sú prirodzené čísla, ktoré jednoducho prečítame. Maximálnu rýchlosť cestných úsekov vieme získať z tagov *maxspeed*, ktorých hodnota môže byť prirodzené číslo, napríklad 50, čo znamená, že táto rýchlosť je uvedená v kilometroch za hodinu, alebo môže byť hodnotou napríklad "30 mph", čo značí maximálnu rýchlosť v míľach za hodinu. Z týchto tagov preto parsujeme ich číselnú hodnotu, s ktorou ďalej narábame.

Oba tieto údaje sú voliteľnými tagmi v dátovej štruktúre OSM, záleží preto iba od schopností a možností komunity projektu OSM, koľko percent ciest tieto údaje obsahuje. Všeobecne platí, že vo vyspelejších krajinách (a najmä v mestách) sú tieto údaje zmapované kvalitnejšie, než v iných oblastiach. Rozhodovali sme sa, ako narábať

s cestami, ktorým jeden alebo oba údaje chýbajú. Keďže naše dáta testujeme najmä na mestách, ktoré sú na území Slovenskej republiky, rozhodli sme sa chýbajúci údaj o počte jazdných pruhov nahradiť konštantou 1 a chýbajúcu maximálnu rýchlosť nahradiť konštantou 50. V prípade potreby je možné takéto cesty z výpočtov korelácie úplne vynechať.

3.10 Beh programu a zdrojové kódy

Za účelom väčšej prehľadnosti bakalárskej práce neuvádzame časti zdrojového kódu v texte. Namiesto toho kompletné zdrojové kódy prikladáme ako CD prílohu a doplnili sme do nich podrobné komentáre umožňujúce naše zdrojové kódy pochopiť. Odkazujeme sa pritom iba na princípy popísane v tejto bakalárskej práci.

Ukážka výpisu behu programu vyzerá nasledovne:

```
Parsing: pezinokjur.osm

= Elements statistics =
* Nodes: 121641
* Ways: 18026
* Relations: 243
=========================
= Graph statistics =
# of vertices: 4301
# of oneway edges: 831
# of twoway edges: 3707
Total # of edges: 4538
=========================

Finding shortest paths with Dijkstra to determine reach:
Searched 0/4301 paths. (# of veritces: 4301)
current reach sum: 0
Searched 1000/4301 paths. (# of veritces: 4301)
current reach sum: 7395462
Searched 2000/4301 paths. (# of veritces: 4301)
current reach sum: 8278419
Searched 3000/4301 paths. (# of veritces: 4301)
current reach sum: 8653532
```

```
Searched 4000/4301 paths. (# of veritces: 4301)
current reach sum: 8982966

Total number of paths searched: 4301/4301

Inserting REACH into each OSM way
Pearson coefficient: 0.17283799553381368
Total time: 10254 [ms]
```

Kapitola 4

Výsledky a ich analýza

V tejto kapitole ukazujeme všetky dôležité alebo zaujímavé výsledky, ktoré sme získali pri početných testoch a pokusoch, ktoré sme vykonali. Najmä však ukazujeme samotné mapy, v ktorých sme vypočítali a vizualizovali dôležitosť cestných úsekov, ktoré sú najdôležitejším výsledkom našej práce. Zamýšlame sa taktiež nad nedostatkami našich metód, uvažujeme, čo sa dá vylepšiť a navrhneme, akým spôsobom by sa dali dosiahnuť lepšie, či presnejšie výsledky.

4.1 Výpočet dôležitosti ciest

Výpočtu dôležitosti ciest predchádzalo sťahovanie údajov a parsovanie OSM XML súborov. V kapitole 3 sme sa už venovali časovej zložitosti parsovania XML súborov, ďalšie zaujímavé údaje sme získali pri počítaní dôležitosti ciest v mapách.

V tabuľke 4.1 uvádzame oblasti, v ktorých nás zaujímali dôležitosti cestných úsekov, koľko miesta zaberajú ich OSM XML súbory na disku, koľko vrcholov a hrán sme z nich extrahovali a dva časy behu programu. Jeden čas ukazuje, ako dlho trvalo nájsť najkratšie cesty medzi všetkými dvojicami vrcholo v grafe a druhý reprezentuje trvanie behu programu, ktorý hľadá iba jedinú najkratšiu cestu pre každý z vrcholov grafu.

Z tejto tabuľky vieme vyčítať, že veľkosť mapového súboru príliš nesúvisí s dĺžkou výpočtov, ktoré na nich počítame, závisí iba od počtu hrán a vrcholov, ktoré z máp vyextrahujeme.

Tabuľka 4.1: Testovanie algoritmu

oblasť	**veľkosť [MB]**	**vrcholy**	**hrany**	**reach [m:s]**	**opt. reach [m:s]**
Limbach	2,3	344	356	00:01	<00:01
Svätý Jur	19	2404	2545	00:28	00:03
Pezinok	27	4301	4538	-	00:10
B. Bystrica	50	8753	9129	-	00:29
Brighton	26	14641	15868	-	01:37
Allentown	9,6	19151	22193	-	02:55
Bratislava	278	55820	58798	-	27:59

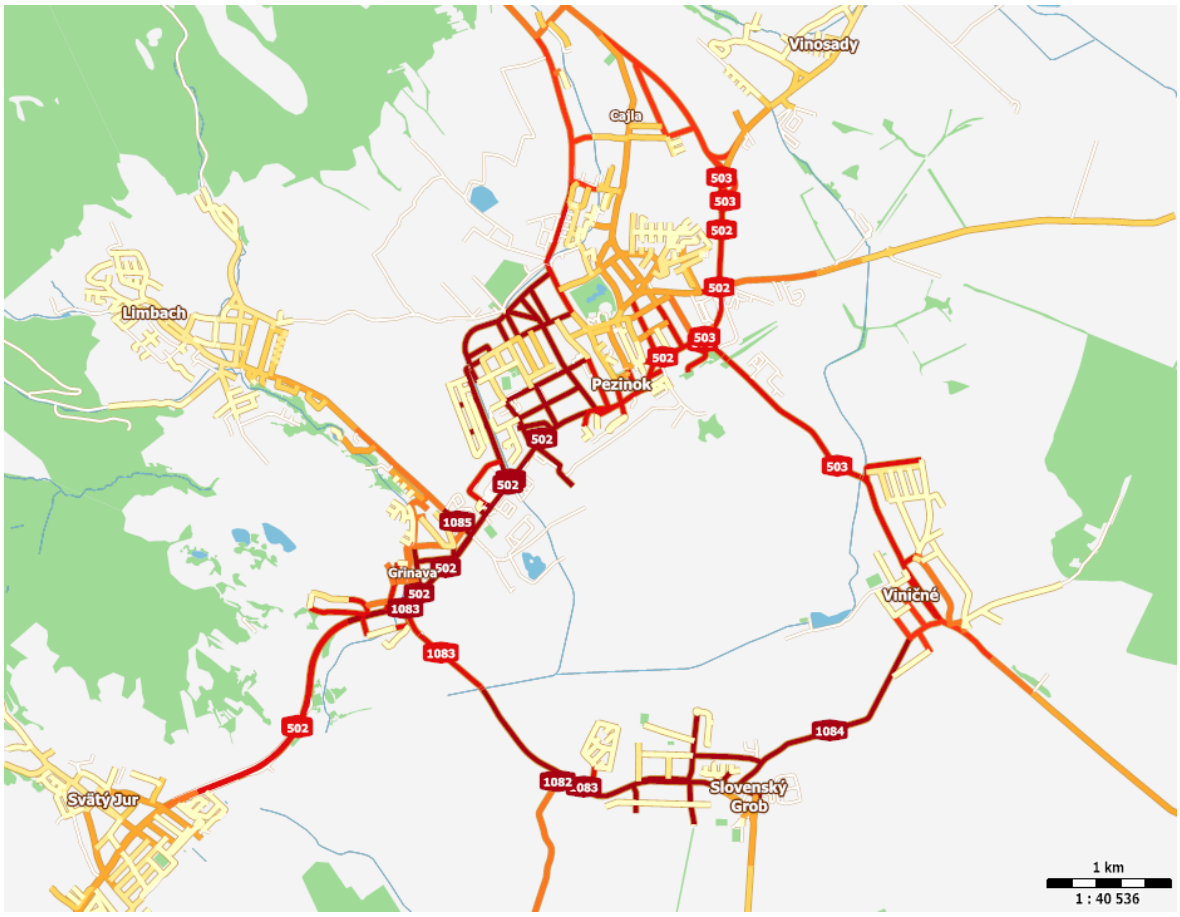
4.2 Vyrenderované mapy

Na obrázkoch 4.1 a 4.2 sú zámerne vyrenderované mapy oblastí, ktoré poznáme aj v reálnom svete a vieme vďaka tomu o nich povedať, že cesty, ktoré náš algoritmus vyhodnotil ako najdôležitejšie, sú skutočne tými, po ktorých denne prejde najväčšie množstvo áut. Zároveň tieto najdôležitejšie cesty sú tými, na ktorých najčastejšie vznikajú zápchy. Konkrétne pre obrázok 4.1 je to cesta 502, ktorú využívajú ľudia pri ceste do/z Bratislavy. V obrázku 4.2 sme si všimli napríklad cesty D2 a opäť cestu 502 ako dôležité pre náš algoritmus a zároveň problematické čo sa dopravných zápch týka.

Na všetkých vyrenderovaných mapách je vidno jeden nedostatok nášho algoritmu, ktorý je najviditeľnejší na obrázku 4.3. Definícia parametra reach, podľa ktorého dôležitosť ciest počítame, priradí väčšiu dôležitosť tým vrcholom v najkratších cestách, ktoré sú ďalej od oboch koncových vrcholov tejto najkratšej cesty. Tým pádom cestné úseky, ktoré sa nachádzajú pri kraji cestných sietí, ktoré skúmame, majú všetky veľmi podobnú dôležitosť. Tým pádom vieme o dôležitostiach ciest na kraji našich máp menej, ako o tých bližšie k stredu cestnej siete.

Na obrázku 4.1 nie je tento nedostatok na závadu napríklad pre dedinu Limbach, pretože ide o koncovú dedinu, cez ktorú autá nemajú dôvod prechádzať, pri ceste za iným cieľom. Je to však problém, ak sa pozrieme na mesto Svätý Jur, cez ktorú prechádza cesta z Pezinka do Bratislavy a táto cesta by určite mala byť vyhodnotená ako dôležitá.

Riešenie takéhoto problému je jednoduché. Stačí náš algoritmus spustiť na širšom okolí oblasti, ktorá nás zaujíma, nie iba na oblasti samotnej. Napríklad, ak nás zaujímajú dôležitosti ciest v Pezinku, obrázok 4.1 nám poskytne lepšie informácie, než keby sme do úvahy brali iba cesty, ktoré ležia vo vnútri Pezinka.



Obr. 4.1: Mapa dôležitostí ciest pre Pezinok a okolité dediny

Pri našich výpočtoch dôležitosti ciest v podstate simulujeme pohyb áut po cestnej sieti a z toho identifikujeme cesty, ktoré budú autami využívanejšie než ostatné. Nemáme dáta o reálnom pohybe áut, ani o miestach, ktoré sú väčšinou cieľmi ciest ľudí, preto generujeme trasy medzi úplne náhodnými bodmi v celej mape, ktorú analyzujeme. Nášmu algoritmu by preto pomohli reálne dáta o pohybe áut alebo informácie o navštevovanosti rôznych oblastí v mapách, aby sme vedeli pohyb áut v našom algoritme simulovať presnejšie.

Prácou do budúcna by preto mohla byť identifikácia bodov záujmu v mape, napríklad verejných budov, pracovísk, parkov, nákupných centier a podobných miest a oblastí, ktoré sú najčastejšie cieľmi ciest ľudí. Vedeli by sme potom predpokladať hustejší transport ľudí z/do týchto bodov záujmu a spresniť tak simuláciu pohybu áut.

Obr. 4.2: Mapa dôležitostí ciest pre Bratislavu a jej mestské časti

4.3 Kvalita cestnej siete

Kvalitu cestnej siete sme porovnali medzi oblasťami z obrázka 4.1, na ktorom je Pezinok a okolité dediny a oblasti z obrázka 4.2, na ktorom je Bratislava.

Náš algoritmus vypočítal koeficient kvality cestnej siete pre Bratislavu ako 0.38 a pre Pezinok 0.14. Z týchto dvoch čísel vieme povedať, že Bratislava má lepšie vybudovanú cestnú sieť, než Pezinok. Aby sme sa na výsledky pozreli lepšie, vložili sme dáta, ktorých Pearsonovu koreláciu počítame, do grafov, v ktorých sme zároveň vykreslili ich trendové čiary.

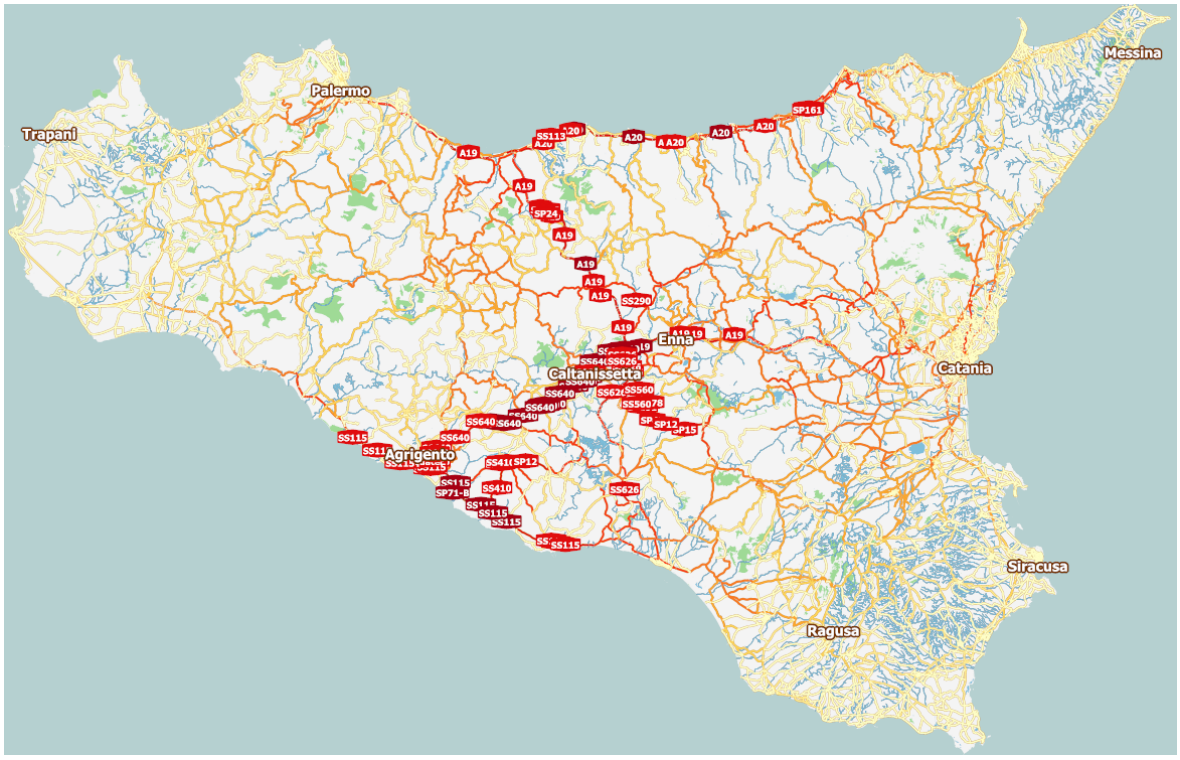
Z grafu na obrázku 4.7, na ktorom sú dáta o Bratislave, sa dajú veľmi dobre čítať informácie o cestách, ktorých priepustnosť je 50, 100 alebo 150. O týchto cestách sa dá predpokladať, že ide o jedno, dvoj a trojprúdové cesty s maximálnou rýchlosťou 50 km/h. Z grafu vidíme, že v Bratislave majú menej dôležité cesty s väčšou najväčšou pravdepodobnosťou priepustnosť 50 a dôležitejšie cesty majú pravdepodobnejšie priepustnosti 100 alebo 150, než 50.

Z grafu, ktorý patrí Pezinku, na obrázku 4.6 vidno, že drvivá väčšina ciest v tejto obci je jednoprúdových s maximálnou rýchlosťou 50 km/h. Pezinok je oveľa menší, než Bratislava a tým pádom nepotrebuje sofistikovanejšiu, alebo premyslenejšiu cestnú sieť, keďže množstvo áut, ktoré touto obcou prejde, je oveľa menšie. Aj keď cesty majú všetky veľmi podobnú priepustnosť bezohľadu na ich dôležitosť, nemá Pezinok

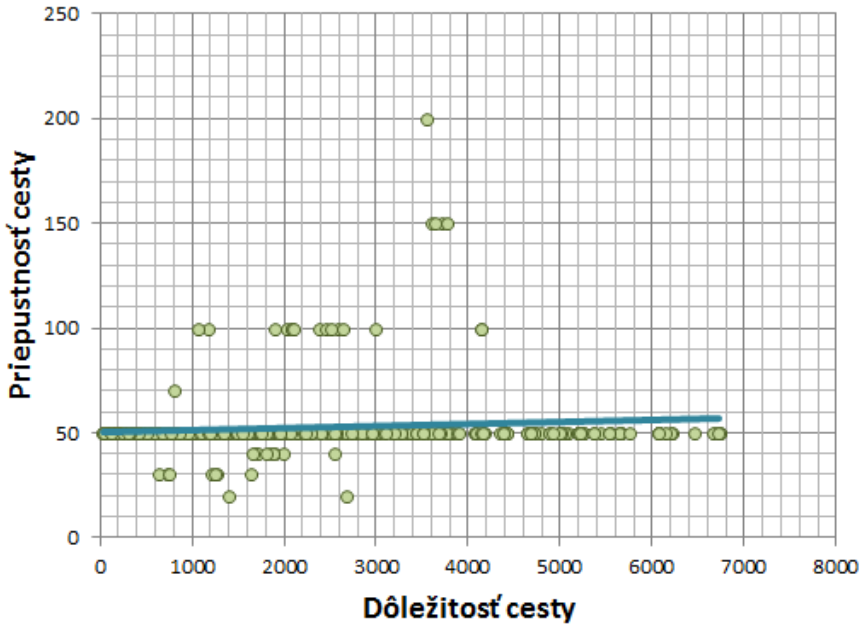
Obr. 4.3: Mapa dôležitostí ciest pre dedinu Limbach

veľký problém s dopravnými zápchami. Takáto cestná sieť je napriek jej nerozvinutosti pre túto obec dostatočná. Jediným problematickým úsekom v Pezinku je práve cesta, ktorú náš algoritmus vyhodnotil ako jednu z najdôležitejších. Je ňou cesta 502, ktorá je po dlhú dobu dvojprúdová, no približne v strede obce sa zúži na jediný prúd a okrem toho prechádza kruhovým objazdom. V čase dopravných špičiek práve na tejto ceste preto vznikajú kolóny, čo náš algoritmus dokáže predpovedať, ak si necháme vypísať priepustnosť iba najdôležitejších ciest a nájdeme v nich veľmi malé hodnoty priepustnosti.

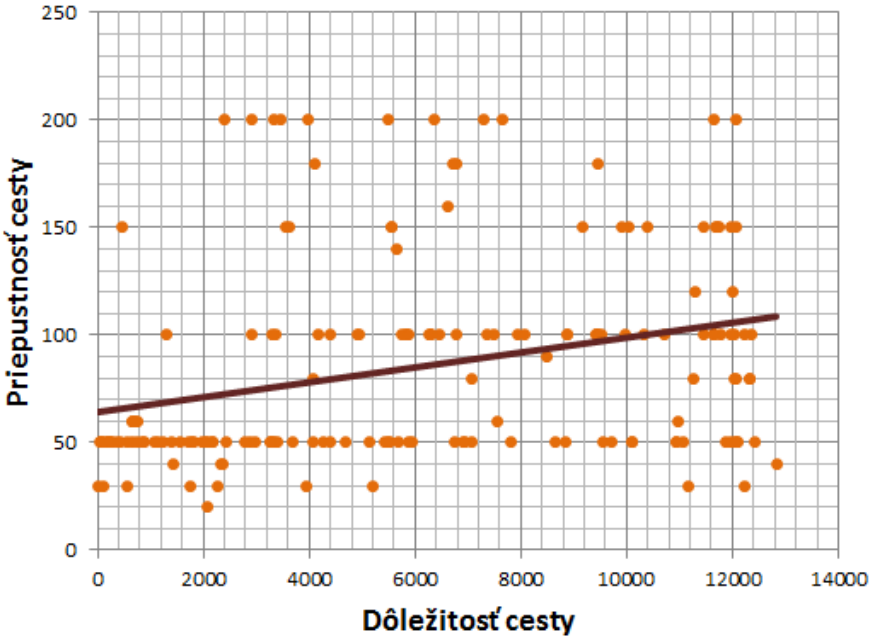
Obr. 4.4: Pohľad na detaily: pôdorysu budov, zastavané oblasti, lesnaté oblasti, rieka a jednosmerné cesty



Obr. 4.5: Ukážka dôležitosti ciest pre kompletnú cestnú sieť ostrova Sicília



Obr. 4.6: Kvalita cestnej siete pre Pezinok



Obr. 4.7: Kvalita cestnej siete pre Bratislavu

Kapitola 5

Dôležitosť ciest pre veľké oblasti

Popísali sme postupy a algoritmy, ktorými sme komfortne schopní analyzovať mapy miest, ktoré sú veľkosťou podobné nanajvýš Bratislave. Takéto dáta vieme stiahnuť, parsovať, určiť v nich dôležitosť a následne ich vyrenderovať. Ako bonus nás zaujímalo, čo je potrebné spraviť, aby sme naše algoritmy vedeli aplikovať na mapu celého Slovenska a aké kompromisy budeme musieť spraviť, aby sme to dokázali.

5.1 Sťahovanie mapových dát

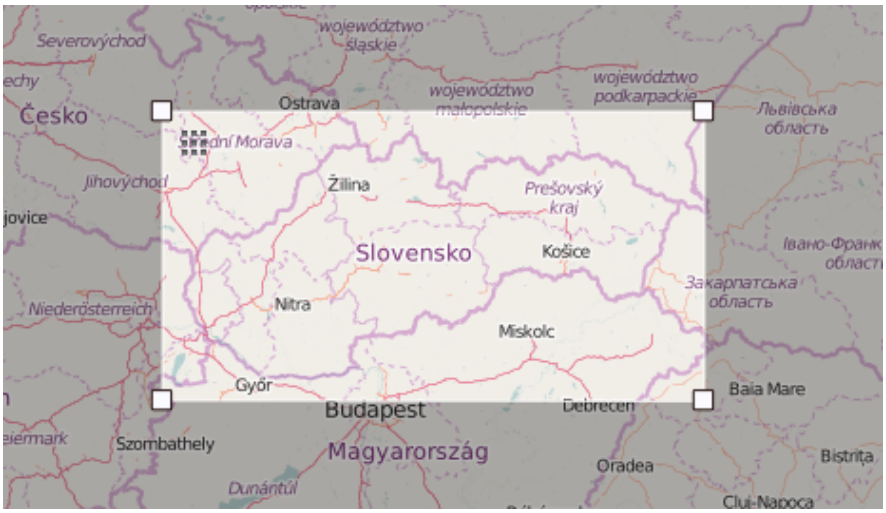
Všetky mapové dáta, ktorých výsledky sme doteraz prezentovali, sme sťahovali pomocou Overpass API, ktorú sme spomínali v kapitole 2. Umožňuje každému užívateľovi denne sťahovať približne 5GB dát, čo však na stiahnutie mapy Slovenska nestačí.

V kapitole 4 sme vysvetlili, že získame viac informácií o dôležitostiach ciest istej oblasti, ak náš algoritmus spustíme na širšom okolí tejto oblasti a nie iba na tejto oblasti samotnej. Ak nás zaujíma dôležitosť ciest na Slovensku, je vhodné algoritmus na počítanie týchto dôležitostí spustiť pre mapu, v ktorej budú okrem Slovenska aj časti jeho susedných štátov.

Využili sme preto stránku Geofabrik [13], ktorá poskytuje sťahovanie OSM máp po celých kontinentoch alebo jednotlivých štátoch. Pomocou tejto stránky sme stiahli skomprimované mapy Slovenska, Rakúska, Ukrajiny, Poľska, Česka a Maďarska. Všetky mapy sme extrahovali ako OSM XML súbory a pomocou nástroja Osmconvert, ktorý sme spolu s nástrojom Osmfilter spomínali v kapitole 2, sme mapy všetkých týchto štátov spojili do jediného OSM XML súbora, ktorý na disku zaberá 60GB.

Náš algoritmus si celé mapové dáta načítava do operačnej pamäti, no k dispozícii sme mali počítač nanajvýš s 16GB RAM. Tento problém sa dá vyriešiť úpravou časti kódu, ktorou parsujeme OSM XML súbory tak, aby do operačnej pamäti ukladala iba dáta nevyhnutné na výpočet dôležitostí ciest. Praktickejším prístupom je však nepotrebné dáta z OSM XML súboru odstrániť ešte pred tým, než v ňom rátame dôležitosť ciest. To je možné vďaka nástroju Osmfilter. V našom OSM XML súbore ponecháme iba údaje o cestách tých typov, ktoré nás pri výpočte dôležitosti ciest zaujímajú. Aké typy ciest to sú, sme uviedli v kapitole 3.

Mapy, ktoré sme takto získali obsahovali okrem celého Slovenska aj kompletné mapy všetkých susedných štátov. Nám sa však pre výpočty hodilo túto oblasť zmenšiť. Preto sme znova využili nástroj Osmconvert pomocou ktorého sme z mapy vystrihli iba oblasť vyznačenú na obrázku 5.1.



Obr. 5.1: Oblasť Európy, ktorú sme použili pre výpočet dôležitostí ciest na Slovensku

Všetky výpočty dôležitostí ciest sme vykonali na počítači s klasickým HDD diskom, pretože mal k dispozícii veľkú operačnú pamäť. Sťahovanie, extrahovanie, spájanie a orezávanie máp Slovenska a jeho susedných štátov sme však vykonali na inom počítači s SSD diskom, za účelom rýchlejšieho behu týchto operácií, ktoré vo veľkom čítajú a zapisujú na disk. Dokopy tieto operácie zabrali počítaču približne hodinu a štvrť.

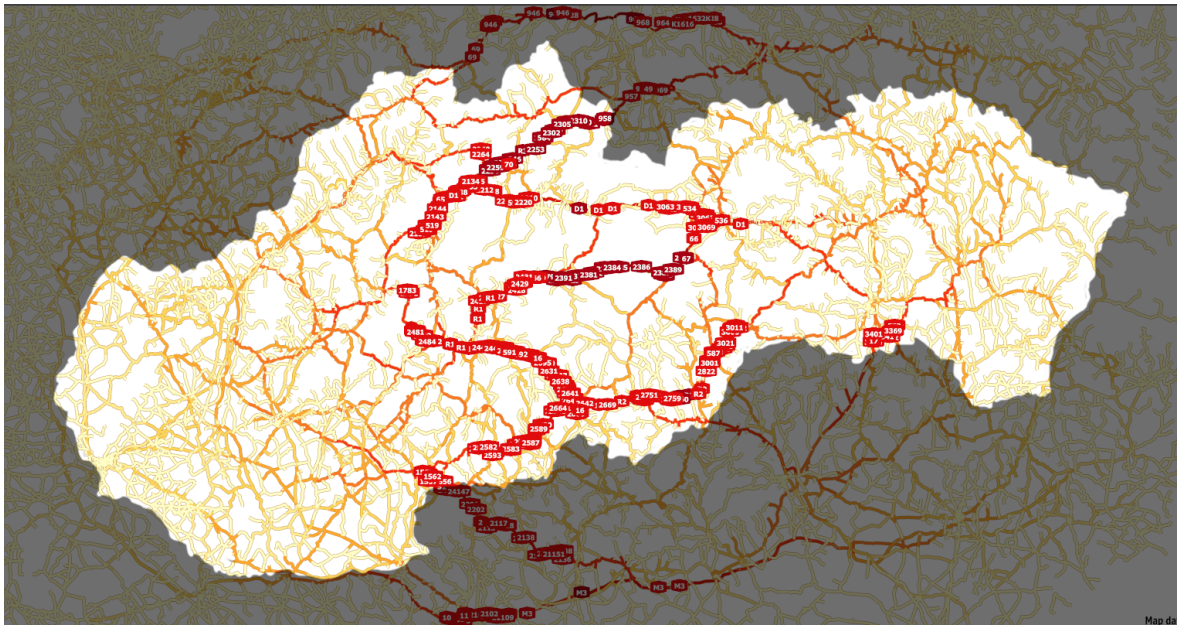
5.2 Výpočet dôležitosti ciest

Časová zložitosť Dijkstrovho algoritmu pre hľadanie najkratšej cesty v grafe vieme vyčísliť ako $\Theta(E + V \log V)$, kde E je počet hrán a V počet vrcholov grafu. Aby sme boli v rozumnom čase schopní dopočítať dôležitosť ciest celého Slovenska, boli sme nútení pristúpiť na dva veľké kompromisy.

V kapitole 3 sme uviedli, aké typy ciest v našich algoritmoch zahŕňame a taktiež koľko percent ktorých typov ciest sa v dátach OSM nachádza. Takmer polovica všetkých ciest v OSM, po ktorých môžu prechádzať automobily sú rezidenčnými cestami, ktoré najčastejšie vedú k budovám alebo parkoviskám. Obsahujú retardéry a iné prvky, ktoré skľudnujú premávku na týchto cestách a maximálna rýchlosť na týchto úsekoch je často nižšia, než na okolitých cestách. Rozhodli sme sa preto tieto cesty nezahrnúť do výpočtov dôležitostí ciest celého Slovenska, keďže nemajú slúžiť na skracovanie ciest medzi rôznymi oblasťami, ale iba ako prístupové cesty napríklad ku domom. Takýto kompromis znížil počet hrán a vrcholov grafu cestnej siete Slovenska približne o polovicu.

Druhým kompromisom bol nižší počet najkratších ciest, ktoré náš algoritmus hľadal. Z mapy Slovenska a jej susedných štátov sme vyextrahovali dokopy 837995 vrcholov a 849097 hrán (jednosmerných a obojsmerných dokopy). Neprichádzalo preto do úvahy hľadať jednu náhodnú najkratšiu cestu pre každý vrchol grafu. Namiesto toho sme hľadali iba 3000 náhodných najkratších ciest v celom grafe, čo trvalo približne 32 minút. Dôsledok tohoto kompromisu je oveľa menšia presnosť nášho algoritmu pre menej dôležité cesty. S takýmto postupom sa nám však podarí identifikovať väčšinu najdôležitejších ciest, keďže najdôležitejšie cesty sú prave tie, ktoré sa nachádzajú na najväčšom počte najkratších ciest v celom grafe. Aj s takýmto kompromisom, teda dokážeme identifikovať najdôležitejšie cesty na Slovensku, čo je zároveň naším cieľom.

Výslednú mapu s vyrenderovanými dôležitosťami ciest na území Slovenska prezentujeme na obrázku 5.2.



Obr. 5.2: Dôležitosť ciest na území celého Slovenska

Za povšimnutie stojí, že medzi cestami, ktoré náš algoritmus vyhodnotil ako najdôležitejšie sú diaľnice a rýchlostné cesty D1, R1, R2 a taktiež maďarská diaľnica M3.

5.3 Alternatívne riešenie

Naša prvotná idea ako vypočítať dôležitosti ciest pre oblasti o veľkosti celého Slovenska, bola inšpirovaná algoritmickou metódou "Rozdeľuj a panuj". Keďže sa nám podarilo určiť dôležitosti ciest pre celú Bratislavu, ktorej rozloha je oproti rozlohe Slovenska nezanedbateľná, bolo našim plánom rozdeliť celé Slovensko na množstvo malých mapových časti a vypočítať dôležitosti ciest pre každú časť individuálne a následne údaje o dôležitostiach ciest spojiť.

Touto metódou by sme však veľa nových informácií nezískali. Tie isté dôležitosti ciest by sme vypočítali, ak by sme náš algoritmus aplikovali na jednotlivé malé časti Slovenska zvlášť. Jediný rozdiel by bol v možnosti vykresliť dôležitosti ciest všetkých malých častí Slovenska súčasne na jedinej mape. Počítanie dôležitostí ciest pre celé Slovensko bolo väčšou výzvou a prinieslo zaujímavejšie výsledky.

Kapitola 6

Záver

V tejto bakalárskej práci sme demonštrovali prácu s mapami reálneho sveta, v ktorých sme za pomoci algoritmov hľadajúcich najkratšie cesty určili dôležitosť všetkým cestám v zadanej cestnej sieti. Vypočítanú dôležitosť ciest sme pre každú cestu porovnali s premávkou, ktorú je daná cesta schopná uniesť a na základe tohoto porovnania sme určili, ako kvalitne je cestná sieť danej oblasti vybudovaná.

Pracovali sme s mapami Open Street Map a ukázali sme ako efektívne sťahovať a narábať aj s obrovskými mapovými dátami. Naprogramovali sme aplikáciu, ktorá pomocou grafových algoritmov vypočítala dôležitosť ciest v cestnej sieti a vytvorili sme vlastné renderovacie pravidlá pre aplikáciu Maperitive, vďaka ktorým dokážeme dôležitosť ciest vizualizovať na reálnych mapách. Kvôli veľkosti dát, s ktorými sme pritom narábali, sme museli bojovať s našimi metódami, aby nevykročili mimo hraníc možností hardvéru a softvéru.

6.1 Práca do budúcna

Presnú dôležitosť ciest dokážeme s naším algoritmom zistiť iba prehľadaním najkratších ciest medzi všetkými dvojicami vrcholov v grafe cestnej siete. Preto sme po celú dobu pracovali iba s algoritmami, ktoré sú vhodné práve pre hľadanie všetkých najkratších ciest. Experimentálne sme však ukázali, že aj veľmi malá podmnožina najkratších ciest vo väčšine prípadov stačí na vygenerovanie dostatočne presných výsledkov. Pre hľadanie najkratšej cesty medzi dvoma náhodnými bodmi v grafe existujú efektívnejšie algoritmy, než Dijkstrov algoritmus, ktorý používame my. Ak by sme pri ďaľšej práci nepotrebovali vedieť presnú dôležitosť všetkých ciest v mape, bolo by vhodné použiť efektívnejší algoritmus na hľadanie najkratších ciest, než ten náš. Príkladom takého

algoritmu je A* [20].

Dôležitosť ciest sme počítali simuláciou pohybu áut po mape, no počiatočné a koncové body trás, medzi ktorými sa autá pohybovali sme generovali náhodne. Zaujímavé by bolo do nášho algoritmu zakomponovať heuristické metódy, ktoré by tieto trasy áut generovali dôveryhodnejšie. Viac o tejto myšlienke píšeme v kapitole 4 pri analýze výsledkov, ktoré sme dosiahli.

Dodatok A

Obsah priloženého CD

Na CD priloženom ku tejto bakalárskej práci sa nachádzajú:

- zdrojové kódy aplikácie, ktorú sme naprogramovali,
- súbor reach.mrules, v ktorom sú definované renderovacie pravidlá pre aplikáciu Maperitive, ktoré správne vyrenderujú mapu, ktorá je výstupom našej aplikácie,
- vyrenderované mapy z kapitoly 4 v ich plnej veľkosti.

Literatúra

[1] Sax - simple api for xml. http://www.saxproject.org/about.html [Dátum posledného prístupu 16.5.2016].

[2] Panier Avide. Basicosmparser. https://github.com/PanierAvide/BasicOSMParser [Dátum posledného prístupu 16.5.2016].

[3] Igor Brejc. Maperitive. http://maperitive.net/ [Dátum posledného prístupu 16.5.2016].

[4] Atlassian Confluence. World geodetic system 1984 (wgs84). https://confluence.qps.nl/pages/viewpage.action?pageId=29855173 [Dátum posledného prístupu 16.5.2016].

[5] Open Knowledge Foundation. Open data commons open database license (odbl). http://opendatacommons.org/licenses/odbl/ [Dátum posledného prístupu 16.5.2016].

[6] OpenStreetMap Foundation. Open street map. www.openstreetmap.org [Dátum posledného prístupu 16.5.2016].

[7] OpenStreetMap Foundation. Osmosis. http://wiki.openstreetmap.org/wiki/Osmosis [Dátum posledného prístupu 16.5.2016].

[8] OpenStreetMap Foundation. Overpass api. http://overpass-api.de/ [Dátum posledného prístupu 16.5.2016].

[9] OpenStreetMap Foundation. Pbf format. [Dátum posledného prístupu 16.5.2016].

[10] OpenStreetMap Foundation. Planet.osm. http://wiki.openstreetmap.org/wiki/Planet.osm [Dátum posledného prístupu 16.5.2016].

[11] OpenStreetMap Foundation. Routing. [Dátum posledného prístupu 16.5.2016].

[12] OpenStreetMap Foundation. Taginfo. https://taginfo.openstreetmap.org/ [Dátum posledného prístupu 16.5.2016].

[13] Geofabrik GmbH and OpenStreetMap Contributors. Openstreetmap data extracts. http://download.geofabrik.de/ [Dátum posledného prístupu 16.5.2016].

[14] Andrew V Goldberg, Haim Kaplan, and Renato F Werneck. Reach for a: Efficient point-to-point shortest path algorithms. In *Proceedings of the Meeting on Algorithm Engineering & Expermiments*, pages 129–143. Society for Industrial and Applied Mathematics, 2006.

[15] Ronald J Gutman. Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In *Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics, New Orleans, LA, USA, January 10, 2004*, pages 100–111, 2004.

[16] Movable Type Ltd. Calculate distance, bearing and more between latitude/longitude points. http://www.movable-type.co.uk/scripts/latlong.html [Dátum posledného prístupu 16.5.2016].

[17] Maperitive. Rendering rules introduction. [Dátum posledného prístupu 16.5.2016].

[18] Jovin J Mwemezi and Youfang Huang. Optimal facility location on spherical surfaces: algorithm and application. *Logistics Research Center, Shanghai Maritime University*, 2011.

[19] Philip Sedgwick et al. Pearson's correlation coefficient. *BMJ*, 345:e4483, 2012.

[20] Wikipedia. A* search algorithm. [Dátum posledného prístupu 16.5.2016].

[21] Wikipedia. Dijkstra's algorithm. [Dátum posledného prístupu 16.5.2016].