Pohrana i obrada GPS podatka korištenjem prostorno vremenske baze podataka

Sugja-Jovetić, Luka

Undergraduate thesis / Završni rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Transport and Traffic Sciences / Sveučilište u Zagrebu, Fakultet prometnih znanosti**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:119:569102

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Repository / Repozitorij:

Faculty of Transport and Traffic Sciences - Institutional Repository

SVEUČILIŠTE U ZAGREBU

FAKULTET PROMETNIH ZNANOSTI

Luka Sugja-Jovetić

Pohrana i obrada GPS podatka korištenjem prostorno vremenske baze podataka

Završni Rad

Zagreb, rujan 2018.

SVEUČILIŠTE U ZAGREBU
FAKULTET PROMETNIH ZNANOSTI
ODBOR ZA ZAVRŠNI RAD

Zagreb, 30. ožujka 2018.

Zavod: **Zavod za inteligentne transportne sustave**
Predmet: **Algoritmi i programiranje**

ZAVRŠNI ZADATAK br. 4785

Pristupnik: **Luka Sugja-Jovetić (0036476842)**
Studij: Promet
Smjer: Informacijsko-komunikacijski promet

Zadatak: **Pohrana i obrada GPS podatka korištenjem prostorno vremenske baze podataka**

Opis zadatka:

Dani su podaci o prometnim segmentima unutar grada Zagreba koji sadrže geografske koordinate početka i kraja svakog prometnog segmenta. Zadane su i vrijednosti prosječnih brzina u 5-minutnim intervalima za prometne segmente. Izraditi grafičko sučelje koje će učitati informacije iz baze podataka i iscrtavati svaki prometni segment različitom bojom ovisno u prosječnoj vrijednosti brzina u bilo kojem odabranom vremenskom intervalu. Grafički prikaz potrebno je mijenjati u stvarnom vremenu oviseći o promatranom vremenskom intervalu koji korisnik može podešavati putem grafičkog sučelja. Potrebno je pokazati i objasniti kako se zadani podaci mogu učitati, interpretirati i obraditi koristeći programski jezik Ruby. Potrebno je opisati i objasniti algoritme za obradu podataka, iscrtavanja prometnih segmenata i promjene prikazanih informacija ovisno o korisničkom unosu. Objasniti kako su podaci pohranjeni te prednosti razvijenih algoritma u kontekstu grafičkog sučelja.

Mentor:

prof. dr. sc. Tonči Carić

Predsjednik povjerenstva za završni ispit:

SVEUČILIŠTE U ZAGREBU

FAKULTET PROMETNIH ZNANOSTI

Završni Rad

Pohrana i obrada GPS podatka korištenjem prostorno vremenske baze podataka

Storage and processing of GPS data using spatiotemporal databases

Mentor: prof. dr. sc. Hrvoje Gold

Student: Luka Sugja-Jovetić

JMBAG: 0036476842

Zagreb, rujan 2018.

Sažetak

Pohrana i obrada GPS podatka korištenjem prostorno vremenske baze podataka

Ovaj rad rješava praktični problem pohrane i obrade velikih količina GPS podataka izrađivanjem aplikacije koja interpretira te podatke i prezentira ih na vizualan, razumljiv način. Rad sadrži ideje kako postupiti izradi takve aplikacije i ponuđuje algoritme za realizaciju rješenja problema. Podaci su zasnovani na geografskom području grada Zagreba. Njih će se koristiti kao glavnu podatkovnu podlogu.

Ključne riječi: baze podataka; aplikacija; Ruby

Abstract

Storage and processing of GPS data using spatiotemporal databases

This thesis solves a practical problem of storing and processing large amounts of GPS data by building an application that interprets this data in a visual, user-friendly manner. This work contains ideas on how to approach the development of such an application, and offers algorithm for the realization of proposed solutions. The data is based on the geographic location of the city Zagreb. They will be used as the main data backbone for this thesis.

Key words: databases; application; Ruby

Sadržaj

1. Uvod 1

2. Osnove prostorno vremenskih baza podataka 4

3. Prikupljanje GPS podataka vozila tijekom projekta SORDITO 7

3.1. Općenito o projektu 7

3.2. Profili brzina 8

3.3. Način pohrane prikupljenih podataka 11

4. Obrada prikupljenih GPS podataka koristeći programski jezik Ruby . 14

5. Aplikacijsko rješenje vizualizacije prikupljenih GPS podataka putem programskog jezika Ruby 21

6. Mjerenje performansi učitavanja i prikazivanja GPS podataka iz baze podataka 28

6.1. Brzine pri učitavanju podataka 28

6.2. Brzine pri iscrtavanju podataka 29

7. Zaključak 31

Literatura 33

Popis slika 35

Popis grafikona 35

Popis kratica 36

Popis priloga 37

1. Uvod

Kako se društvo, tehnologija i urbana infrastruktura razvijaju, teži se prema organizacijskom usavršavanju svakodnevnih pojava ljudskog iskustva. Cilja se na efikasnost, efektivnost i preciznost pri razrađivanju organizacijskih i tehničkih problema, prikupljanju i obradi informacija, i konačnoj analizi kako bi se doprinijelo rješenje za svakodnevne logističke probleme. S rapidnim razvojem informatičkih grana znanosti, informacijski sustavi postali su duboko integrirani u sve logističke procese. Napredna tehnologija omogućila je prikupljanje enormne količine podataka o raznim aspektima društva. Ovi podaci pohranjuju se u velikim bazama podataka i obrađuju se kako bi se dalje studirali ili iskoristili. Iako drukčije situacije zahtijevaju unikatno rješenje, uvijek se započinje od problematike pohrane i obrade podataka.

Ovaj rad bavit će se konkretno tom problematikom, zatim će predložiti i izraditi jedno praktično rješenje za pohranu, obradu i u konačnici interpretaciju velikih količina GPS podataka, kako bi se najlakše ilustriralo stanje na prometnicama grada Zagreba. Rješenje će biti aplikacijskog oblika, gdje je nužno izraditi aplikaciju koja će primijeniti ponuđene ideje i rješenja, samim time iz prakse dokazati da li su zaista moguće i praktične. Kako bi se postigao taj rezultat, koristit će se programski jezik Ruby u kojemu će biti napisani svi algoritmi zaslužni za obradu i interpretaciju prikupljenih podataka. Aplikacija se temelji na principu vizualnog iscrtavanja prometnica grada Zagreba ovisno o prometnom opterećenju koje je prostorno i vremenski distribuirano. Podatkovna podloga potrebna za izvedbu ovog rada pružana je od strane projekta SORDITO-a (Sustav za optimizaciju ruta u dinamičkom transportnom okruženju) i u suradnji s voditeljem projekta je implementirana kao potpuna informacijska podloga ovoga rada.

Materija rada izložena je u 7 poglavlja:

1. Uvod,
2. Osnove prostorno vremenskih baza podataka;
3. Prikupljanje GPS podataka vozila tijekom projekta SORDITO;
4. Obrada prikupljenih GPS podataka koristeći programski jezik Ruby;
5. Aplikacijsko rješenje vizualizacije prikupljenih GPS podataka putem programskog jezika Ruby;
6. Mjerenje performansi učitavanja i prikazivanja GPS podataka iz baze podataka;
7. Zaključak.

Druga teza opisuje osnovne koncepte i modele prostorno vremenskih baza podataka koji će se primijeniti u svrhe realizacije vlastite baze podataka; kako bi služila kao glavna informacijska podloga za ispravan rad aplikacije.

Treći dio rada opisuje projekt SORDITO i postupke s kojima su se prikupili sirovi GPS podaci, zatim opisuje njihovu strukturu i način na koji su originalno pohranjeni.

Četvrta teza objašnjava osnovni koncept programskog jezika Ruby; njegovih prednosti i nedostatka, zatim opisuje korištene algoritme za obradu sirovih podataka i njihovu svrhu.

Peti dio rada bavi se izrađenom aplikacijom, njezinim sučeljem i korištenim algoritmima za vizualizaciju prikupljenih GPS podataka.

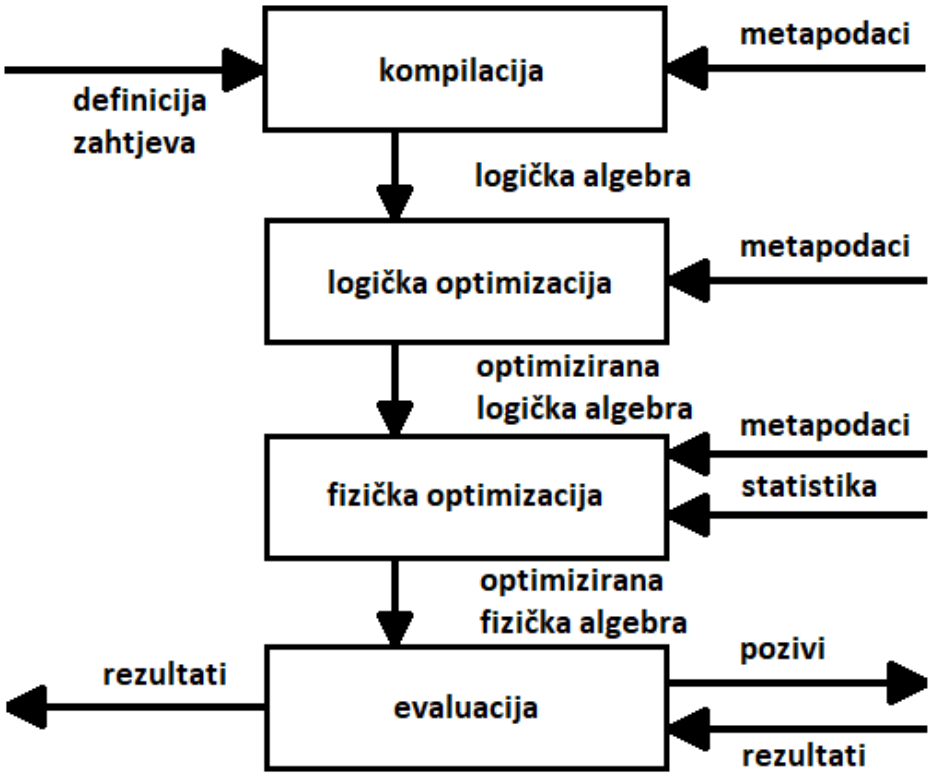
Šesta teza proučava potencijalne probleme koji bi utjecali na efikasnost rada aplikacije, zatim predlaže određena rješenja i nudi stvarne vremenske vrijednosti koje reprezentiraju izmjerene performanse rada aplikacije.

Svrha završnog rada je primjena predložene teoretske podloge i istraženih informacija kako bi se napravilo praktično, aplikacijsko rješenje za pohranu i obradu velike količine sirovih podataka, uz optimalno iskorištenje raspoloživih resursa.

2. Osnove prostorno vremenskih baza podataka

Prostorno vremenske baze podataka bave se pohranom i obradom informacija vremenskih i prostornih jedinica. Njihov najčešći primjer je u praćenju određenih pokretnih entiteta u sklopu usluga geolokacija. Bave se proučavanjem prostornih svojstava i njihovim odnosom s nekim promatranim vremenskim razdobljem. Važno je osigurati strukturu takve baze podataka gdje svojstva prostornih i vremenskih podataka međusobno surađuju pri realizaciji određene funkcije [1]. Ovaj rad bavi se konkretnim proučavanjem tih odnosa i konstrukcijom idealne baze podataka kako bi se osigurala funkcija vizualizacije prometnih segmenata i njihovo ponašanje na što efektivniji i efikasniji način. Iako se primarni koncept prostorno vremenskih baza podataka bavi pokretnim entitetima, one ne obrađuju isključivo samo takav scenarij. Kako tehnologija teži razvojem usluga praćenja položaja entiteta u stvarnom vremenu, lako je zaboraviti da postoje slučajevi gdje takvo ponašanje nije poželjno, naime u analizi i obradi prikupljenih podataka u svrhe optimizacije prostornog ponašanja i predviđanja potencijalnih trendova [2].

Manipulacija prostorno vremenskih podataka postaje važno područje istraživanja jer se ne usredotočuje isključivo na statičko stanje prostornih jedinica, već ih promatra kako se mijenjaju s vremenom. Stoga se prostorno vremenska podrška podataka smatra važnim smjerom istraživanja, budući da mnoga aplikacijska rješenja moraju obrađivati podatke koji se mijenjaju u vremenu [2]. Jedno od tih aplikacijskih rješenja je predstavljeno u ovom radu. Stoga je potrebno razmotriti problem određivanja, projektiranja i provođenja prostorno vremenskih baza podataka, očekujući da će se osigurati potpuna funkcionalnost nad prostornim i vremenskim podacima. Najvažnija odluka u ovom pristupu je na koji način treba izraditi podatkovni model [3].



Slika 1. Procesiranje zahtjeva nad podacima

Izvor: [2]

Iz Slike 1. vidi se da je potrebno imati definirane zahtjeve nad bazom podataka kako bi se mogao odrediti tok prikupljenih podataka. Na osnovnoj razini potrebno je kompilirati skup ponuđenih podataka kako bi ih se kroz softverske algoritme (logičku optimizaciju) unaprijedili za efikasniju manipulaciju. Stoga, važno je optimizirati fizičku razinu koja je u konačnici i podloga koja omogućuje izvršavanje traženih funkcija. Zatim, potrebno je imati sustav za evaluaciju izvršenih zahtjeva koji će moći prenijeti pohranjene i obrađene podatke korisniku, na razumljiv način. Ovaj pristup problemu pruža dobar teoretski model za izvedbu aplikacije u kontekstu ovog rada. Pošto je konačni cilj aplikacije obrada i vizualizacija pohranjenih podataka, u kasnijim tezama rada bit će pokazano kako se mogu primijeniti programski algoritmi za optimalno rješenje problema [2].

Kako bi se uspješno prikazali sadržaji prikupljenih podataka, mnogi suvremeni informacijski sustavi pružaju korisniku alate za vizualnu interpretaciju podataka. Standardni alati za prostorne baze podataka uključuju preglednike i karte. Većina tih, međutim, neadekvatno su prilagođene za prikaz dinamičkih ili vremenskih informacija. Stoga novi načini vizualnog prezentiranja podataka moraju se istražiti [4].

Potrebno je proučiti nekoliko dinamičkih varijabli potrebno za nadopunu statičkih karata [4]:

- vremenski prikaz,
- trajanje između dva susjedna vremenska intervala,
- frekvencija pojavljenja podataka u vremenskom intervalu,
- redoslijed vremenskih intervala,
- stopa promjene prikazanih podataka.

3. Prikupljanje GPS podataka vozila tijekom projekta SORDITO

3.1. Općenito o projektu

SORDITO projekt osnovan je s primarnim ciljevima da se izgradi dovoljan kapacitet za prometne tehnologije i da se usklade istraživanja s potrebama gospodarstva. SORDITO projekt počeo je 17.10.2014. i trajao je do 16.02.2016. [5] Kreće od ideje da se skupi znanje i iskustvo o primjeni optimizacijskih tehnika u transportu kao i da se razvijaju tehnologije digitalnih karata. Projekt je baziran na praćenju velikog broja vozila kako bi se prikupila dovoljna količina informacija za razvoj algoritma navigacije po prometnicama. S budžetom od 3.475.670,37 HRK, 30% investiralo se u opremu kako bi se polučili precizni rezultati [5].

Pratilo se ukupno preko 6.000 vozila s kojih se skupilo oko 500 [GB] sirovih podatka. Da se dođe do tih podataka, vršila se dubinska analiza GPS tragova vozila, pomoću lociranja mjesta zagušenja i zona usporavanja [5]. Ti podaci, skupa s raznim metodama, koriste se kako bi se razvili algoritmi za kreiranje profila brzina. Važno je bilo prikupiti dovoljno podataka kako bi se riješio problem usmjeravanja vozila. Transport obično predstavlja više od pola logističkih troškova, a smanjenje tih troškova bi ostvarilo bolju upotrebu prometnih resursa, gdje je cilj minimizacije troškova uz zadovoljavanje zadanih ograničenja [5].

Podaci su se prikupljali korištenjem [5]:

- radarskih detektora i
- GPS zapisnika.

Gdje su se radarski detektori koristili radi mogućnosti snimanja i postavljanja, pokrivenosti, potrošnje energije i utjecaja vremenskih prilika. Radarski detektori fiksirani su na određene prometne segmente, dok su GPS zapisnici korišteni za mjerenje vrijednosti na rutama u svrhu što veće prostorne pokrivenosti.

Konkretno, u kontekstu ovog rada, glavni fokus je na područje grada Zagreba. Stoga potrebni su zapisi linkova koji pokrivaju željeno područje, unutar kojega je pronađeno ukupno 28.496 GPS zapisa od mogućih 34.111 povezica. Zatim, promatrani linkovi u ovome radu odnosit će se na razdiobu identifikacijskog broja *198897 - 400427* [6].

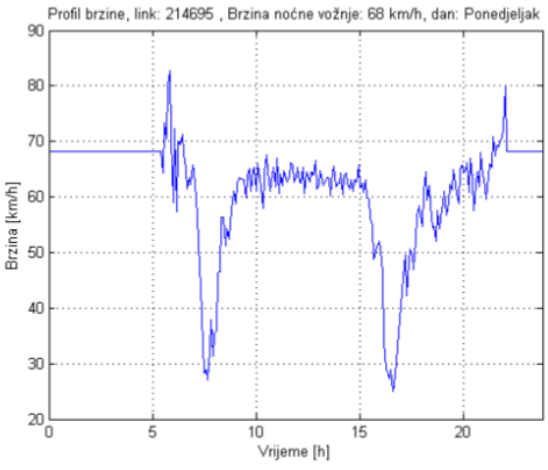
3.2. Profili brzina

Pomoću mjerne opreme i podataka pojedinih uređaja, moguće je kreirati profil brzine. Profil brzine je prikaz promjene brzina tijekom dana na određenom prometnom segmentu. Ovi profili se razvijaju kako bi se koristili kao podloga za algoritme za optimizaciju rutiranja [6]. Svrha tih algoritma je odrediti što efikasnije vrijeme putovanja transportnog sredstva prometnim segmentima, u bilo koje doba dana, te za bilo koji dan u tjednu. Podaci se dobivaju iz čisto izmjerenih jedinica GPS signala praćenih vozila. Prikupljeni podaci su prikaz realističnih prometnih brzina na promatranim prometnim segmentima [6].

Svaki prikupljeni zapis sadrži sljedeće [6]:

- vremenski trenutak uzorka signala,
- očitanu geografski širinu i dužinu,
- oznaku vožnje ili stajanja,
- očitanu brzinu vozila,
- orijentaciju gibanja vozila,
- prijeđenu udaljenost po prometnici od prethodnog uzorka signala.

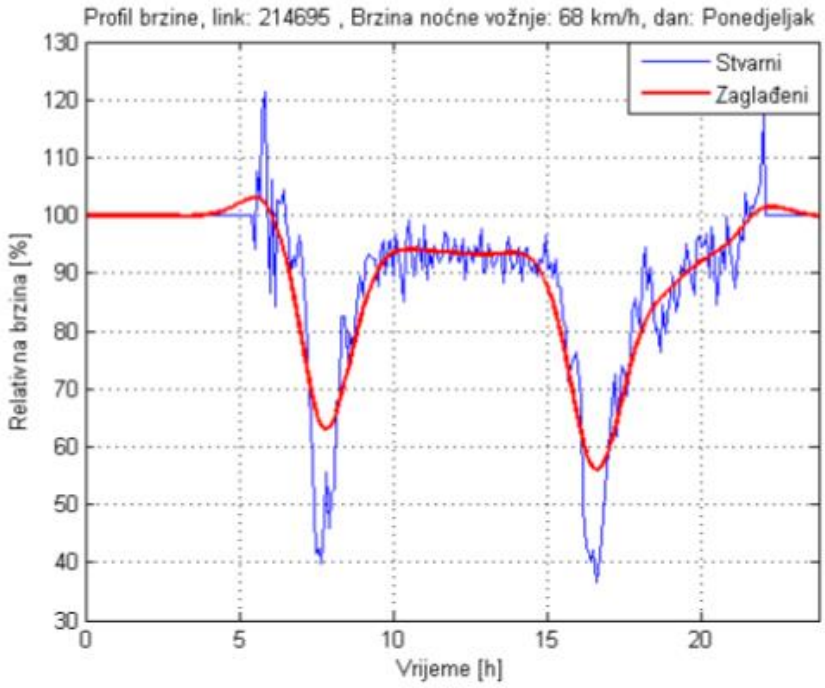
Prikupljane ovih podataka nudi pregled apsolutnog profila brzina na prometnim segmentima. On zaista pokazuje realističnu promjenu vrijednosti na prometnim segmentima tijekom dana [6]. Uzorci brzina zabilježeni su u 5-minutnim intervalima, gdje se promatraju dnevni sati od 5:30 - 22:00 [h] i noćni od 22:00 - 5:30 [h]. Svaki dnevni uzorak dobiva se uprosječenjem svih evidentirani brzina koje su se pojavile u tom intervalu, dok se noćni uzorci računaju uprosječenjem svih brzina koje su se pojavile po noći, i pridjeljuje im se samo jedna vrijednost brzine [6].



Grafikon 1. Apsolutni profil brzina

Izvor: [6]

Za optimalni prikaz podataka i vizualizacije promjene brzina kroz vrijeme, ovi profili se zaglađuju kako bi se pokazala kontinuirana promjena stanja na prometnim segmentima. U kontekstu ovog rada, apsolutne vrijednosti nam nisu nužno potrebne, jer njihova odstupanja od kontinuiranog toka grafikona pri prikazu u grafičkom sučelju su zanemariva [6].



Grafikon 2. Zaglađeni profil brzina

Izvor [6]:

Grafikon 2. prikazuje profil brzina koji će se dalje u radu i u razvijenoj aplikaciji koristiti u svrhe iscrtavanja prometnog stanja na promatranim prometnicama.

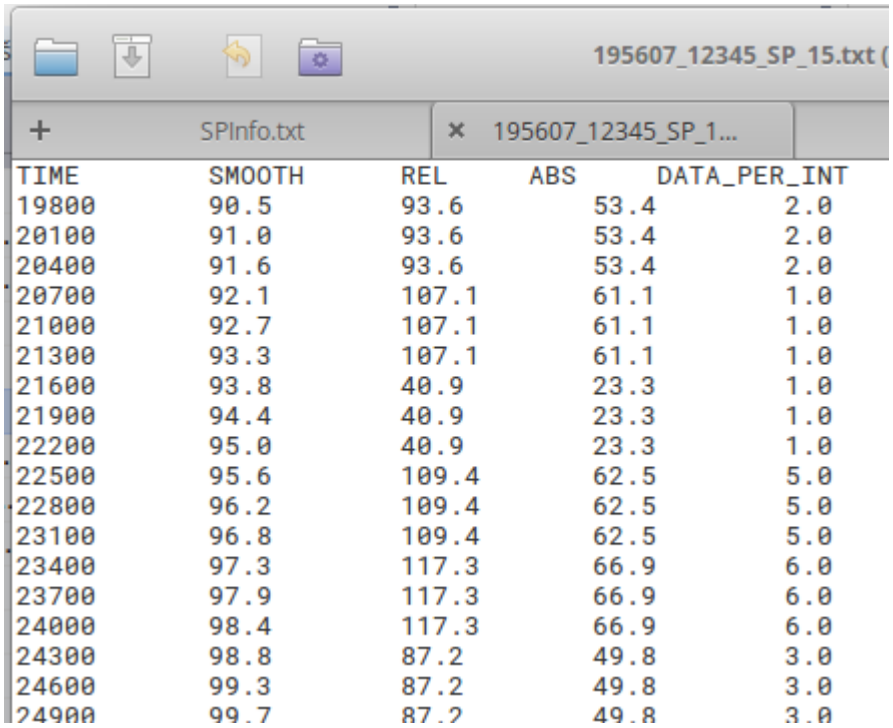
3.3. Način pohrane prikupljenih podataka

Prikupljeni podaci vremenskih uzoraka za prometne segmente bilježe se u tabličnom obliku u pojedinačne datoteke koje sadrže identifikacijski broj svakog linka (prometnog segmenta) kao i vrijednost intervala uzimanja uzoraka i dan promatranja, izražen kao numerička vrijednost.

Svaka datoteka sadrži:

- vrijeme uzimanja uzorka (izraženo u sekundama),
- vrijednosti zaglađenog profila brzina (izraženo kao postotak prosječno postignute brzine na tom prometnom segmentu),
- vrijednosti relativnog profila brzina (izraženo kao postotak prosječno postignute brzine na tom prometnom segmentu),
- vrijednosti apsolutnog profila brzina (izraženo kao postotak prosječno postignute brzine na tom prometnom segmentu),
- broj podataka skupljeno u svakom intervalu.

Stvarne brzine uzoraka potrebno je izračunati ovisno o evidentiranim prosječnim vrijednostima zabilježene u datoteci koja sadrži svu ključnu informaciju, zatim se te vrijednosti množe s postotcima izraženih profila u vremenskim intervalima koje su pohranjene u datotekama profila brzine.

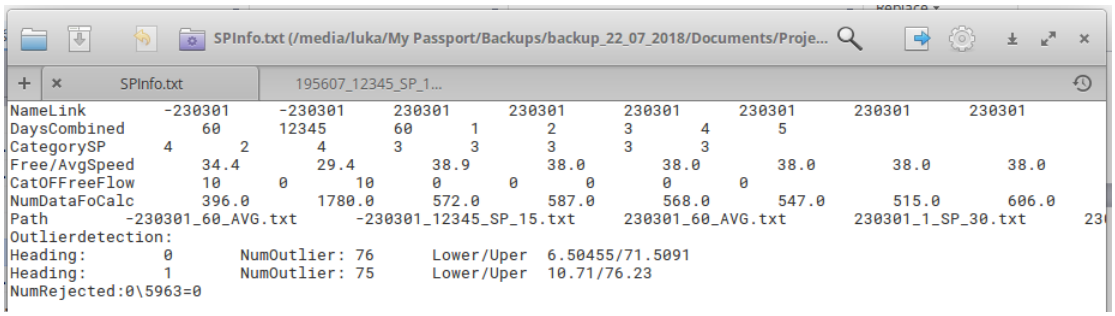


Slika 2. Prikaz pohranjene informacije za prometni segment

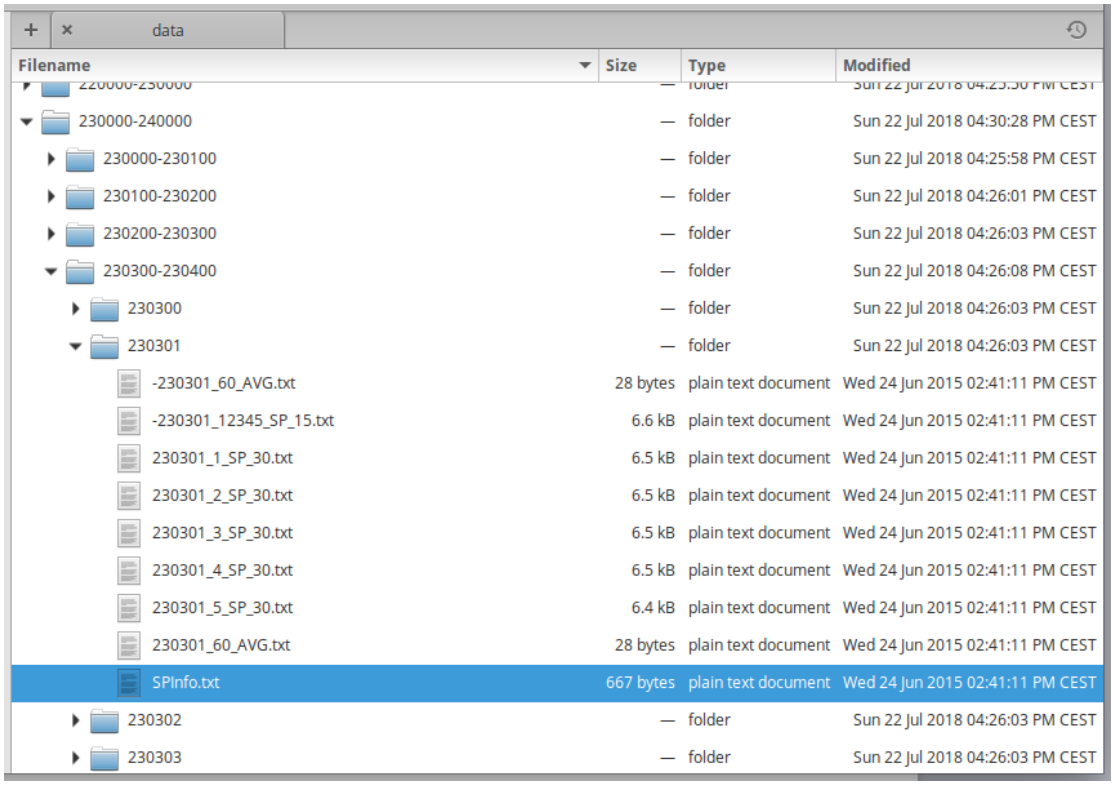
Datoteke s profilima brzina prometnih segmenata grupiraju se po identifikacijskom broju linka radi lakše organizacije i navigacije kroz bazu podataka. Uz individualne datoteke s profilima prometnog segmenta, nalazi se i *SPInfo.txt* datoteka koja je zadužena za pravilno usmjeravanje aplikacije prema točnim informacijama.

Ona sadrži:

- podatke o prosječnim vrijednostima brzina,
- ime datoteka prikupljenih podataka s obzirom na promatrani dan,
- identifikacijski broj linka radi adresiranja prometnog segmenta.



Slika 3. Sadržaj datoteke SPInfo.txt



Slika 4. Stablo mapa pohranjenih datoteka

4. Obrada prikupljenih GPS podataka koristeći programski jezik Ruby

Ruby je objektno-orijentiran, interpretiran i dinamičan programski jezik, najčešće korišten u svrhe razvijanja back-end infrastruktura servera i drugih tehnologija računalstva u oblaku (engl. cloud computing). Ruby se bazira na API-u (engl. Application Programming Interface) i podlozi programskog jezika C, s fokusom na fleksibilnost u sintaksi i funkcionalnosti. Inspiracije sintakse proizlaze iz programskih jezika Perl, Lisp i Smalltalk, zadržavajući familijarnost za programere koji imaju prethodno iskustvo s programskim jezikom C ili Java [7]. Ovakav pristup prilagođene i pojednostavljene sintakse, a u istom trenu i zadržavanja starijih pravopisa sintakse, čini ovaj programski jezik atraktivnim za nove i postojeće programere. Sintaksa je primarno dizajnirana kako bi glavni fokus bio razvijanje i implementiranje programskih algoritma, gdje se jezik ne bori protiv programera i omogućuje mu uredan tok rada [8].

Važno je imati na umu da je Ruby kompletno objektno-orijentiran, što znači da se svaki element unutar sustava tretira kao objekt. To se odnosi i na same tipove podataka i elemente sintakse jezika [7]. Zatim, velika fleksibilnost ovog programskog jezika omogućuje programeru da nadograđuje svaki element (tj. svaki objekt) unutar samog jezika, pružajući veliki potencijal prilagodbe jezika i njegovih sustava kako bi se osigurao najbolji rad programa kojeg se piše koristeći ovaj jezik. Ovakav pristup programiranju omogućuje:

- prilagodbu tipova podataka u optimalni oblik u kontekstu prikupljenih sirovih GPS podataka,
- skalabilnost za optimalno baratanje velikih količina podataka.

S time je osigurano da u slučaju nedostatka funkcionalnosti programskog jezika, moguće je nadograditi osnovne elemente (objekte) za svrhe aplikacije ovog rada. Bitno je poznavati ove koncepte kako bi programer bio upoznat s potencijalnim dobitcima, a isto tako i nedostacima samog jezika. Sva prethodno navedena fleksibilnost samih tipova podataka, sintakse i općenita fleksibilnost programskog jezika dolazi s problemom optimizacije korištenih hardverskih resursa [9]. Vrlo je lako izgubiti se u sintaksi i napisati kod koji vizualno (a vjerojatno i funkcionalno) dojmi apstraktno, minimalistično i "magično", bez da se uzme u obzir opterećenje procesora i radne memorije koje takav pristup može imati na mašini koja pokreće program. Ovaj problem postaje sve više očit u trenutku kada se radi o velikoj količini podataka koju je potrebno na neki način interpretirati. Naime, važno je konzervativnije pristupiti problemu i osigurati iskorištenje resursa samo kada je apsolutno nužno [8]. Dodatna proširenja elemenata može biti djelotvorno; ali isto tako, ako se neodgovorno koristi, može pružati smetnje pri izvedbi i korištenju konačnog produkta.

Velika zamka ovog rada je prilagoditi podatke za brzo učitavanje i trajno zadržavanje u radnoj memoriji radi lakše manipulacije podataka, bez da se naruši stabilnost aplikacije i opterećenje hardverskih resursa.

U ovakvom slučaju gdje je potrebno izraditi aplikacijsko rješenje koje će biti u stanju obrađivati veliku količinu sirovih podataka i prikazivati je korisnicima u lako razumljivom obliku, nije dovoljno napraviti rješenje koje radi u apstraktnom ili najosnovnijem smislu, već je veoma bitno napraviti sustav koji će najefektivnije raspolagati s resursima kako bi se osigurao optimalni rad čitave aplikacije [10].

Konceptualno, najlakše rješenje bi bilo svaki put ponovno referencirati datoteke koje sadrže informacije o linku kojega promatramo ili pokušavamo prikazati u aplikaciji. Na neki način, ovo je čak i ispravna pretpostavka. U početku su na raspolaganju samo sirovi podaci. Ovi podaci zahtijevaju da ih se na neki način u sustav učita, kako bi aplikacija imala točne podatke na raspolaganju. Tu se stvara problem, naime dolazi do ograničene sposobnosti samog programskog jezika. Operacije nad pohranjenim datotekama (kao što su u ovom slučaju) nisu veoma brze [8]. Trajanje otvaranja pojedinačne datoteke i učitavanja sadržaja je izraženo u vremenskim jedinicama manjima od milisekunda, što na prvi pogled ne zvuči toliko kritično. Naime, ako se uzme u obzir da je u ovom radu potrebno obrađivati stotine tisuća datoteka, može se pretpostaviti da akumulacija velikih vremena potrebna za učitavanje tih podataka će narušavati performanse same aplikacije. Poanta je olakšati krajnjem korisniku pregled prikupljenih podataka, što znači da napraviti sustav u kojemu se treba čekati približno 10 minuta da se svi podaci obrade (a tako i svaki put kada korisnik pokuša imati neku interakciju s aplikacijom) ne ide u korist korisnika niti programera.

Potrebna su rješenja ovome problemu, a ima ih mnogo. Ovaj rad predlaže sljedeće:

- ako je jedan od ograničavajućih faktora količina datoteka koja čeka na red za učitavanje, dio rješenja bi bio eliminirati silnu količinu podataka i sve kompresirati u jednu datoteku (rješavajući problematiku količine podataka);

- ako iščitavanje, pretvorba i obrada sirovih podataka pohranjena u datoteci pridodaje vremenskoj problematici, potrebno je formatiranje podataka tijekom kompresije (rješavajući problematiku iščitavanja podataka);

- ako pri svakoj interakciji korisnika s aplikacijom, i promjenom stanja na zaslonu je potrebno nanovo dohvatiti podatke, ti podaci mogu unaprijed biti spremljeni u radnu memoriju (rješavajući problematiku brzine pri dohvaćanju podataka).

Ova predložena rješenja su najlakša za izvesti jer je potrebno razviti samo jedan algoritam koji će jednom proći kroz sve sirove podatke i prilagoditi ih za potrebe aplikacije. Taj algoritam će generirati kompresiranu datoteku koju će aplikacija dalje koristiti kako bi se osigurale optimalne performanse. Pošto se prikupljeni podaci ne mijenjaju u stvarnom vremenu i koriste se analitički, ovaj pristup nije neispravan.

Uz prethodno navedene prijedloge, dolazi se do zaključka da taj algoritam treba napraviti sljedeće [11]:

- učitati sirove podatke za svaki link unutar promatranog područja;

- formatirati, konvertirati i enkapsulirati učitane podatke za lakšu manipulaciju;

- pridodati ispravan link ID obrađenim podacima,

- sekvencijalno pohraniti podatke za individualni link ID unutar iste datoteke;

- kompresirati konačnu datoteku da se smanji potrebna memorija za pohranu datoteke.

Iz prethodne teze, poznato je da za područje Zagreba, promatraju se samo linkovi s ID-om *198897* - *400427*. Isto tako, poznato je formatiranje mapa, datoteka i podataka koji su u njima pohranjeni. Prvi zadatak je pronalaženje točnih datoteka za svaki odgovarajući link, pri čemu se koristi sljedeće:

```
# dobiva imenik podataka
limits = [[(id/10000.0).floor*10000,(id/10000.0).ceil*10000],[(id/100.0).floor*100,(id/100.0).ceil*100]]

dir = "data/#{limits[0][0]}-#{limits[0][1]}/#{limits[1][0]}-#{limits[1][1]}/#{id}"

if File.safeDirectory?(dir) && File.safeExist?("#{dir}/SPInfo.txt")
    File.open("#{dir}/SPInfo.txt", 'rb') {|f|
```

Slika 5. Kôd za otvaranje tablice sadržaja podataka po linku

Ovaj isječak definira stablo mapa u kojoj algoritam treba pokušati naći glavnu datoteku koja sadrži kontrolnu tablicu datoteka koje korespondiraju s prostorno-vremenskim podacima dodijeljenim svakom linku. Kada je taj postupak izvršen, jedino što preostaje u vezi učitavanja podataka je samo iščitavanje vrijednosti brzina pohranjenih u odgovarajućim datotekama [12].

Pošto su sirovi podaci formatirani u obliku tablica, najjednostavnije je kreirati objekte u obliku dvodimenzionalnih tablica unutar algoritma i jednostavno prebacivati podatke iz originalne datoteke u novo generiranu tablicu:

```
inter = f.read.split("\r\n")
# racuna na kojem indeksu se pocinje
x = inter[0].split(/\s/).reject{|e| e.nil? || e == "" }.length - 1
y = 7
# generira novu tablicu
table = Array2D.new(x, y)

# ubacuje podatke u tablicu
for ky in 0...y
    sspl = inter[ky].split(/\s/).reject{|e| e.nil? || e == "" }
    for kx in 1...sspl.length
        table[kx - 1,ky] = sspl[kx]
    end
end
```

Slika 6. Kôd za prebacivanje informacija iz datoteke u novu tablicu

Novo generirane tablice postaju Ruby objekti, kojima se pridružuje link ID kako bi ih se moglo kasnije lakše referencirati. Učitana informacija se prebacuje u tip podataka float, kako bi se sačuvala informacija o decimalnim zarezima. Prilikom obrade podataka, geografsko područje se pridružuje vremenskim uzorcima, kako bi se prometni segmenti mogli iscrtavati ovisno o poziciji i željenom promatranom vremenu.

Preostali koraci su konverzija ostatka sirovih podataka u točan tip, i konačna pohrana obrađenih podataka u proizvoljnu, izlaznu datoteku:

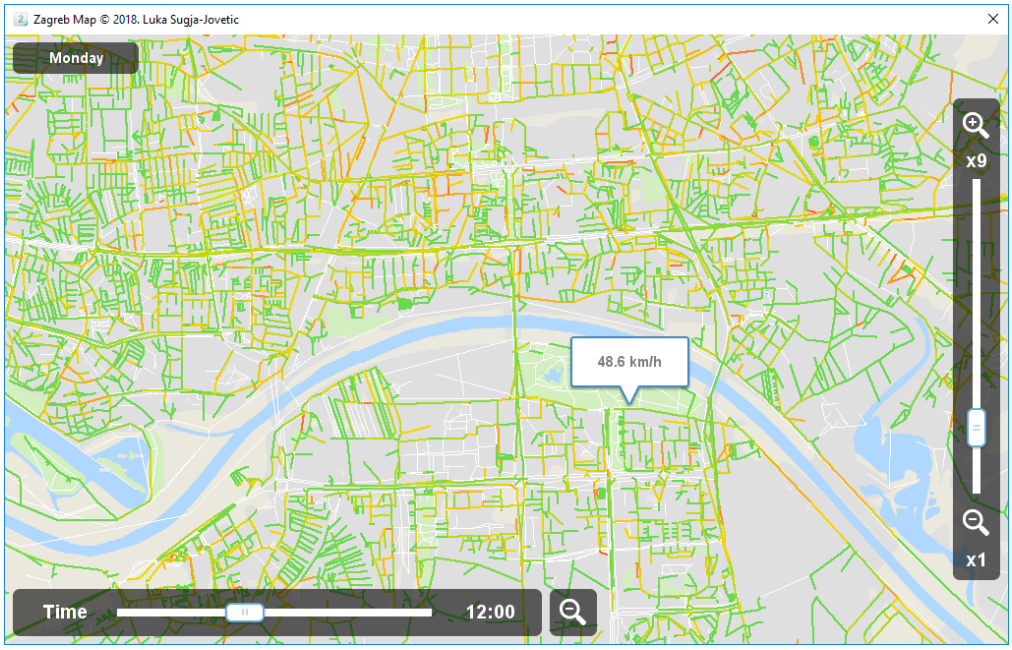
```
# pohrana brzina
$dataPoint[id].setDay(speeds)

# pohrana slojeva karte
k = $dataPoint[id].detailed ? 2 : 0
k = 0 if args[5].to_i == 0
$keys[k].push(id)

# failsafe
if Time.now > tm + 5
    tm = Time.now
    Graphics.update
    GC.start
end

Marshal.dump($dataPoint[id], compiled)
```

Slika 7. Kôd za pohranu obrađenih podataka

Korištene metode kompresiraju podatke tijekom njihove pohrane u glavnu datoteku koristeći *ZLib* biblioteku [13]. Ovaj algoritam dalje više nije potreban, pošto je svrha bila jednokratno prilagoditi veliku količinu podataka kako bi se maksimiziralo učitavanje i manipulacija podataka tijekom rada aplikacije. Osim što je osiguran optimalan rad same aplikacije, ovim postupkom je 1.090.015 datoteka sveukupne veličine od 2.5 [GB] kompresirano u jednu datoteku od 79.2 [MB]. Ova datoteka postaje glavna prostorno vremenska baza podataka iz koje se kasnije vuku informacije.

5. Aplikacijsko rješenje vizualizacije prikupljenih GPS podataka putem programskog jezika Ruby

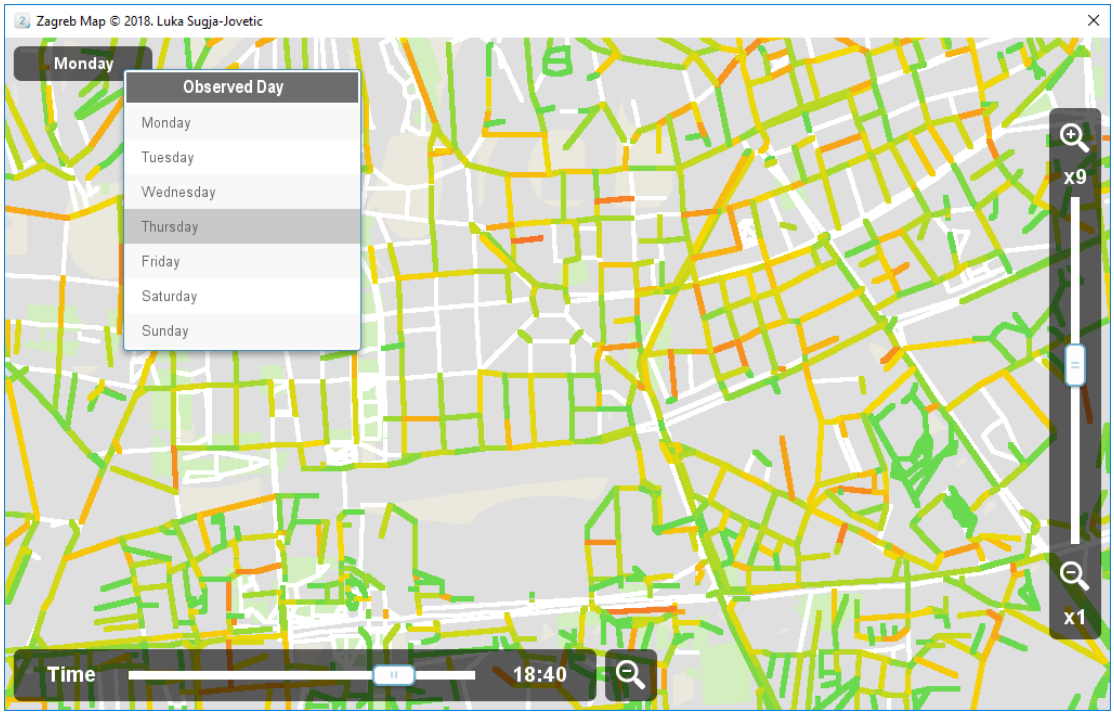
Kruti podaci, bili oni pravilno formatirani ili ne, nude ograničenu korisnost ako ne postoji aktivni sustav koji će znati te podatke interpretirati. Ovakva ogromna količina podataka zahtjeva dodatne korake kako bi se složila koherentna priča koju bi ljudski um mogao u cijelosti pojmiti. Zatim je važno izraditi aplikaciju i vizualno rješenje, koje će tu količinu podataka moći korisniku prenijeti na najrazumljiviji način. U konačnici, korisnik nema nikakav dobitak od te silne količine informacije ako nema alat koji će mu pomoći je razumjeti [14]. Ovaj rad nudi takvo rješenje u obliku Ruby aplikacije koja iscrtava prometnice na bazi geografskih koordinata zabilježenih za svaki link unutar grada Zagreba i vrijednosti brzina koje se mijenjaju vremenski. Grafičko sučelje aplikacije je dizajnirano kako bi bilo što poznatije korisnicima, ako su ikada prije koristili navigacijske usluge ili aplikacije za prikaz karte.



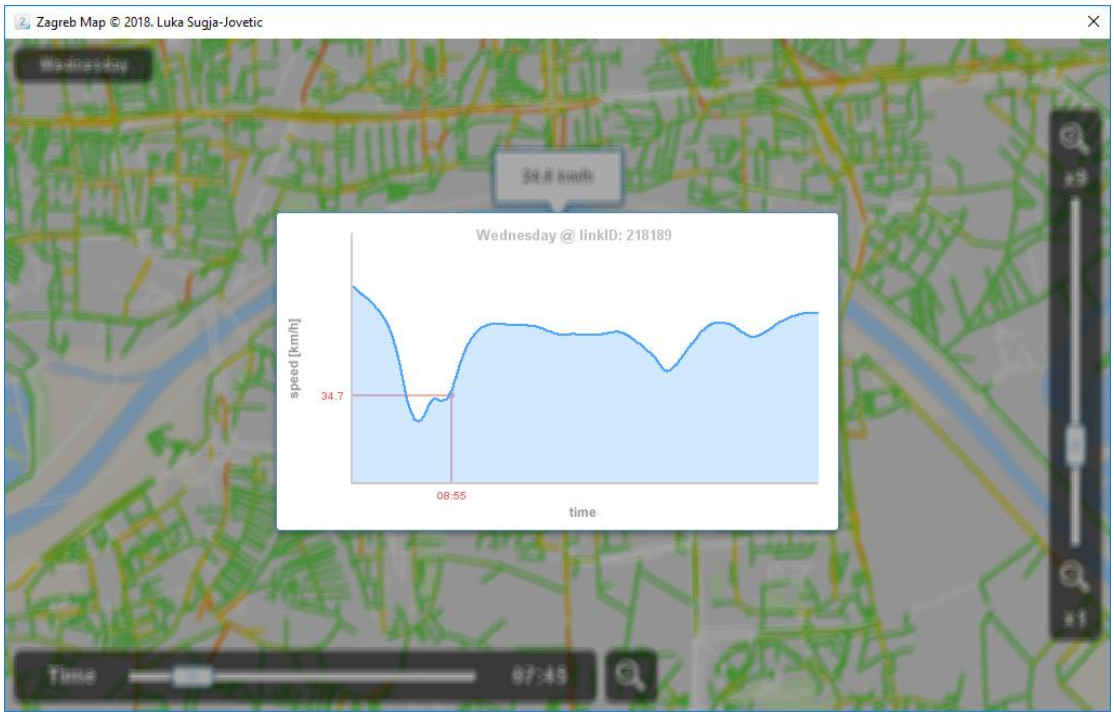
Slika 8. Aplikacija za iscrtavanje prometnog opterećenja u Zagrebu

Cilj aplikacije je bio što preglednije prezentirati što veću količinu informacija a uz to i osigurati sljedeće [14]:

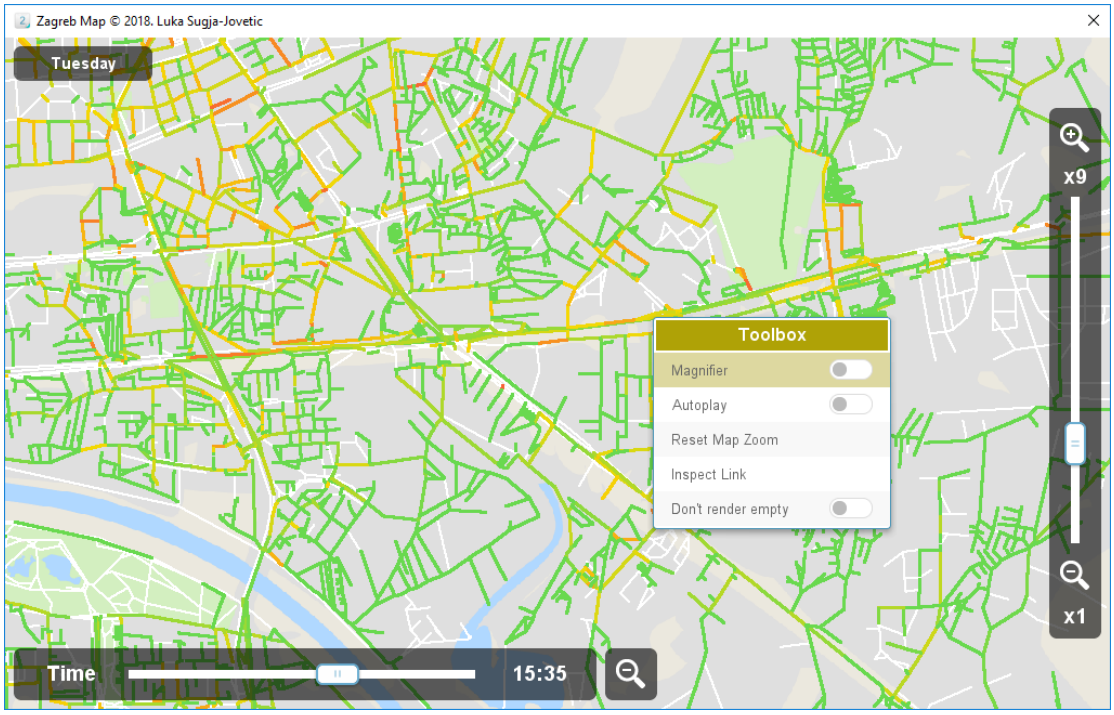
- jednostavnu navigaciju po grafičkom sučelju,
- jednostavan pregled prometnica unutar grada Zagreba,
- reprezentaciju vrijednosti brzina jednostavnim bojama,
- kontrolu nad promatranim vremenskim intervalima,
- kontrolu nad promatranim danom u tjednu,
- brzi pregled vrijednosti brzina na odabranim prometnicama.



Slika 9. Opcija za odabir promatranog dana



Slika 10. Pregled profila brzina tijekom dana



Slika 11. Kontekstualni meni s dodatnim opcijama

Za grafički prikaz unutar aplikacije, pojavljuje se novi problem: stvara se potreba za pretvorbom stvarnih geografskih informacija u neki kompatibilni oblik za iscrtavanje na zaslonu. Geografske koordinate se ne mogu jednostavno pretočiti bez da ih se prvo ne obradi. Naime, potrebno je moći pretvoriti geografski koordinatni sustav u pixel koordinatni sustavi koje koristi grafičko sučelje. , .

Potrebno je prvo izračunati maksimalnu duljinu i širinu geografskog područja, kao i samog sučelja. Zatim se pomoću apsolutnih vrijednosti geografskih koordinata trebaju izračunati relativne koordinate u pixel koordinatnom sustavu. Vrijedi da je:

$$x_1 = \frac{x_{ID} - x_{min}}{x_{max} - x_{min}} * w, \tag{1}$$

$$y_1 = h - \frac{y_{ID} - y_{min}}{y_{max} - y_{min}} * h, \tag{2}$$

$$x_2 = \frac{x_{2ID} - x_{min}}{x_{max} - x_{min}} * w, \tag{3}$$

$$y_2 = h - \frac{y_{2ID} - y_{min}}{y_{max} - y_{min}} * h, \tag{4}$$

gdje je:

- x_1 - početna horizontalna pixel koordinata,
- x_2 - završna horizontalna pixel koordinata,
- y_1 - početna vertikalna pixel koordinata,
- y_2 - završna vertikalna pixel koordinata,
- x_{ID} - početna horizontalna geografska koordinata,

- x_{2ID} - završna horizontalna geografska koordinata,
- y_{ID} - početna vertikalna geografska koordinata,
- y_{2ID} - završna vertikalna geografska koordinata,
- x_{min} - najmanja horizontalna geografska koordinata,
- x_{max} - najveća horizontalna geografska koordinata,
- y_{min} - najmanja vertikalna geografska koordinata,
- y_{max} - najveća vertikalna geografska koordinata,
- *w* - širina grafičkog sučelja,
- *h* - visina grafičkog sučelja.

Ove jednadžbe potrebno je prilagoditi u pravilni sintaktički oblik programskog jezika Ruby. Kôd prikazan u Slici 12. dalje se koristi kao način izračuna točnih početnih i završnih koordinata koje se koriste za iscrtavanje prometnih segmenata na grafičkom sučelju.

```
# racunica za x i y koordinate
x1 = ((info.getData("x1") - $dataPoint["x"])/($dataPoint["width"]/@zoom)) * Graphics.width - @offset_x
y1 = Graphics.height*@zoom - ((info.getData("y1") - $dataPoint["y"])/($dataPoint["height"]/@zoom)) * Graphics.height - @offset_y
x2 = ((info.getData("x2") - $dataPoint["x"])/($dataPoint["width"]/@zoom)) * Graphics.width - @offset_x
y2 = Graphics.height*@zoom - ((info.getData("y2") - $dataPoint["y"])/($dataPoint["height"]/@zoom)) * Graphics.height - @offset_y
```

Slika 12. Izračun početnih i završnih koordinata prometnih segmenata

Uzevši u obzir da se ubacuju dodatne varijable unutar jednadžbe kako bi se osiguralo da linkovi se pravilno iscrtavaju na bilo kojem faktoru povećanja karte, isto kao i tijekom bilo kojeg pomaka karte u svrhe navigacije po sučelju. Osim početne i završne koordinate svakog linka, potrebna nam je i boja kojom će se iscrtati prometni segment. Ovisno o promatranom danu i promatranom vremenskom intervalu (koji se podešavaju pomoću samog grafičkog sučelja), uzima se omjer trenutne brzine i maksimalne brzine postignute tog dana u tom prometnom segmentu. Taj omjer se pretvara u postotak i koristi se kako bi se ekstrahirala boja iz gradijenta grafikona spremljenog u vanjskoj datoteci. Ta boja predstavlja prometno zagušenje tako da uspoređuje trenutno postignute brzine s maksimalno postignutim brzinama tog dana.

```
# provjera detalja segmenta
if info.detailed
    spd = info.getRel(@day,@timeline) # za regularne brzine
    # dohvaca boju iz spremljne datoteke
    c = gradient.get_pixel(99*spd,2)
    # dodjeljuje debljinu linije
    t = @zoom
end


t = 1 if t < 1
# iscrtavanje segmenta
@sprites["map#{m}"].bitmap.drawLine(x1,y1,x2,y2,c,t.to_i)
```

Slika 13. Metoda crtanja prometnih segmenata

Iscrtavanje grafikona promjene brzina slična je problematika iscrtavanju prometnih segmenata, samo se koristi drukčiji set informacija iz kojega se vuku relativne pozicije početnih i završnih točaka vremenskih intervala. Brzine se očitavaju iz pohranjene baze podataka, uspoređuju s maksimalno postignutom brzinom tijekom promatranog dana za specifični prometni segment, i nakon toga se crtaju na definiranom koordinatnom platnu.

```
def drawGraph
    for t in 19800..78600
        next if t%300 != 0
        px1 = 430*(t - 19800.0)/59100.0
        px2 = 430*((t+300) - 19800.0)/59100.0
        py1 = 230 - 180*(@data.getDay(@day,t)/@data.max(@day))
        py2 = 230 - 180*(@data.getDay(@day,t+300)/@data.max(@day))
        @sprites["graph"].bitmap.drawLine(px1,py1,px2,py2,Color.new("#4DA6FF"),2)
    end
    for x in 0...@sprites["graph"].bitmap.width
        for h in 0...@sprites["graph"].bitmap.height
            y = @sprites["graph"].bitmap.height - h
            pixel = @sprites["graph"].bitmap.get_pixel(x,y)
            break if pixel.alpha > 0
            @sprites["graph"].bitmap.set_pixel(x,y,Color.new(77,166,255,64))
        end
    end
end
```

Slika 14. Funkcija zadužena za crtanje grafikona profila brzine u promatranom danu

Važna petlja je prva prikazana; ona se bavi iteracijom vremenskih odsječaka u razdoblju od 5:30 - 22:00 [h]. Petlja se iterira samo u 5-minutnim intervalima (svakih 300 [s]), kako su to dostupni podaci unutar baze.

6. Mjerenje performansi učitavanja i prikazivanja GPS podataka iz baze podataka

6.1. Brzine pri učitavanju podataka

Kako bi se osigurao optimalan rad aplikacije, važno je pratiti koliko vremena treba proći da algoritmi unutar aplikacije obave svoje funkcije. U slučaju da aplikacija ne zadovoljava vremenske zahtjeve pri izvršavanju svojih funkcija, nužno je ponovo se obratiti tim algoritmima kako bi se dalje mogli optimizirati. Bilo koje akcije koje trebaju previše vremena da se izvedu su neprihvatljive. Zatim se pri razvijanju svakog algoritma moraju ispitivati vremena potrebna za odvijanje funkcija [15]. U prethodnim tezama se ustanovilo da jedan od nedostataka programskog jezika, zatim i aplikacije, je da je proces otvaranja i iščitavanja lokalnih datoteka spor kada ga se gleda u kontekstu količine podataka koju je potrebno obratiti. Rješenje za to je bilo razviti novi algoritam kako bi se sve datoteke od baze kompresirale u jednu, za lakše i brže dohvaćanje podataka.

Prva kompresija sirovih podataka trajala je oko 10 [min]. Ovo bi bilo potrebno vrijeme pri svakom pokretanju aplikacije da se svi potrebni podaci obrade i pohrane u radnu memoriju. Za potrebe korisnika, takvo vrijeme čekanja je neprihvatljivo. Zato se primjenjuju vlastiti algoritmi zaslužni za pohranu i obradu svih sirovih podataka unutar jedne ispravno formatirane datoteke. Ovaj pristup omogućuje da se početno vrijeme učitavanja podataka u radnu memoriju smanji na 8,419 [s]. Podaci se sada znatno brže učitavaju u radnu memoriju, a učitavanje u radnu memoriju dalje pomaže pri manipulaciji podataka ili iščitavanja, kako ne bi došlo do ponovnog usporenja funkcije.

6.2. Brzine pri iscrtavanju podataka

Drugi problem koji se prikazuje usko je vezan uz količinu informacije koja se treba iscrtati na zaslonu. Iako je problem manipulacije podataka riješen i više ne predstavljaju problem za daljnje korištenje aplikacije, ako se algoritam za iscrtavanje ne optimizira opet pada razina kvalitete aplikacije u neprihvatljivo područje. Interakcije koje korisnik ima s aplikacijom ne smiju "štekati" i zahtijevati da korisnik dugo čeka.

Važno je definirati koje segmente karte se treba nanovo crtati, ovisno o trenutno podešenim parametrima. Prvi put kada se karta generira, potrebno ju je iscrtati u potpunosti, premda takvo iscrtavanje nije nužno kad se radi o promjeni određenih prometnih segmenta zbog korisničkog unosa. Treba se uzeti u obzir sljedeće:

- ako prometni segment ne sadrži zabilježene podatke, njega se kao dinamični dio sustava odbacuje, dodjeljuje mu se bijela boja i permanentno je fiksiran kao iscrtani element pozadine - samim time on se u vremenu ne mijenja,

- ako je razlika iz između novog i starog stanja prometnog segmenta neuočljiva sa strane korisnika, onda se ta promjena isto tako može zanemariti,

- ako prometni segment ima kontinuiranu vrijednost tijekom dana, on se takav može zanemariti.

Uzevši te stavke u obzir, može se uočiti da nije nužno potrebna promjena stanja svih prometnih segmenata. Nema koristi od bespotrebnog iskorištavanja resursa. Prazni prometni segmenti iscrtavaju se na vlastitom

sloju grafičkog sučelja, a segmenti koji mijenjaju svoje vrijednosti brzina ne crtaju se u kompletnosti iz početka. To olakšava veliku količinu posla, samim time poboljšava krajnje iskustvo od korisnika. Da bi se karta u potpunosti iscrtala treba 1,073 [s] za svaki prometni segment. Kad se primjene prethodno navedene optimizacije, to vrijeme se smanji na 0,366 [s] što znači da nam je zaista potrebna samo jedna trećina očekivanog vremena. Dodatni čimbenici za optimizaciju su:

- manje vidljivih segmenata pri povećavanju karte samim time je potrebno manje prometnih segmenata iscrtavati,

- tijekom navigacije po karti drugačiji prometni segmenti ulaze u fokus na zaslonu, što znači da nije uvijek fiksni broj prometnih segmenata prisutan za crtanje.

7. Zaključak

Razvojem tehnologije, dolazi do velike količine prikupljenih podataka nad kojima se izvršava obrada i analiza kako bi se mogli prikazati postojeći trendovi i utvrditi budući. Ovaj rad bavio se velikom količinom prikupljenih GPS podataka od strane projekta SORDITO, koji su služili kao informacijska podloga aplikacije. Cilj je bio smisliti rješenje za obradu i interpretaciju te velike količine podataka, kako bi se kasnije moglo prezentirati potencijalnim korisnicima. Utvrdilo se da u tom kontekstu je potrebno prilagoditi bazu podataka kako bi najefektivnije pohranila informaciju koju aplikacija kasnije interpretira. Isto tako, utvrdilo se da je potrebno prilagoditi algoritme obrade i interpretacije podataka na način na koji bi se osiguralo brzi rad same aplikacije. Ovakva velika količina podataka zahtjeva pametno smišljen pristup i treba ga se izvršavati u svrhu usavršavanja prethodno definiranih zahtjeva i ciljeva aplikacije.

Ovakav praktični pristup dozvoljava programeru da prilagodi aplikaciju podacima, a podatke aplikaciji, osiguravajući optimalan rad cijelog sustava. Kroz razne algoritme, pokazalo se kako uopće pristupiti problemu i kako najbolje optimizirati sustav. Nažalost nije dovoljno da aplikacija samo radi, nego mora raditi najbolje moguće prema definiranim zahtjevima. Važno je opaziti prednosti i nedostatke programskog alata za izvedbu ovakvih rješenja, uzevši ih u obzir kako bi se formiralo rješenje prihvaćajući snaga i izbjegavanja slabosti korištenih alata. Konačni aplikacijski produkt je bio jedan koji je zaista funkcionirao, naime i u potpunosti izvršio namijenjenu funkciju.

Iako aplikacija zadovoljava definirane zahtjeve, ne znači da je sustav savršen. S obzirom na to da je u pitanju obrada velike količine GPS podataka i iscrtavanja istih, Ruby nije najprilagođeniji programski jezik za riješiti problem. Veliki nedostatak jezika pri izradi aplikacije je bio manjak brzine nad izvršavanjem određenih funkcija. Najsporiji procesi bili su otvaranje i učitavanje datoteka, kao i iscrtavanje velikog broja prometnica na zaslonu. Koristeći jezik srednje razine, poput C++ -a, mogao bi se zaobići problem slabijih performansi. Zatim, upravljanje memorije bilo je problematično, pošto Ruby nema striktna pravila niti sposobnosti napredne kontrole radne memorije. Neki jezik koji bi pružao programeru potpunu kontrolu nad memorijom aplikacije bi omogućio bolje iskorištavanje resursa, samim time pospješio efikasnost sučelja. Međutim, koristeći Ruby i navedene algoritme, postigla se optimalna ravnoteža između kompleksnosti razvoja sustava i dobivenih rezultata. Aplikacija je uspješno pohranila, obradila i prikazala veliku količinu GPS podatke uz minimalni gubitak performansi.

Literatura

[1] Bazlur R., Anwar Hossain A., (2012.) Challenging Issues of Spatio-Temporal Data Mining. Computer Engineering and Intelligent Systems 3(4):55-63

[2] Paton N. W., Fernandes A. A., Griffiths T., (2000.) Spatio-Temporal Databases: Contentions, Components and Consolidation. Int. Workshop on Advanced Spatial Databases (ASDM), 11th DEXA Workshop, IEEE Press, pp. 851-855

[3] Mahmood N., Burney A., Shah A., Rizwan K., Bakar A. A., Bari S. A. K., (2016.) Spatio-temporal Database Design Architecture (STDDA): A Conceptual Framework. Science International –(Lahore)). 28(1):4333-4338.

[4] Tamas A., Roddick J.F., (1999.) Survey of spatio-temporal databases. GeoInformatica, Springer, 3(1):61-99

[5] Presentations from the First conference. Preuzeto sa: http://www.fpz.unizg.hr/sordito/presentations-from-the-first-conference/ (pristupljeno: kolovoz 2018.)

[6] Erdelić T., Vrbančić S., Rožić L., (2015.) A Model of Speed Profiles for Urban Road Networks Using G-means Clustering. Information and Communication Technology Electronics and Microelectronics (MIPRO) 2015 38th International Convention on, IEEE, pp. 1081-1086

[7] Cooper P.; The Ruby Programming Language; O'Riley, California; 2008.

[8] Dwarampudi V., Dhillon S. S., Shah J., Sebastian N. J., Kanigicharla N., (2010.) Comparative study of the Pros and Cons of Programming languages Java, Scala, C++, Haskell, VB .NET, AspectJ, Perl, Ruby, PHP & Scheme - a Team 11 COMP6411-S10 Term Report CoRR abs/1008.3431

[9] Ruby pioneers come clean on the language's shortcomings. Preuzeto sa: https://www.infoworld.com/article/2870966/ruby/ruby-pioneers-come-clean-on-languages-shortcomings.html (pristupljeno: kolovoz 2018.)

[10] Cooper P.; Beginning Ruby: From Novice to Professional; Apress, New York; 2017.

[11] Borkar V.R., (2013.) A Common Compiler Framework for Big Data Languages: Motivation, Opportunities, and Benefits. IEEE Data Eng. Bull., 36(1):56-64

[12] Fulton A., Arko A.; The Ruby Way: Solutions and Techniques in Ruby Programming, Portable Documents; Addison-Wesley, New Jersy; 2015.

[13] Ruby Core Reference Documentation. Preuzeto sa: http://ruby-doc.org/core-1.9.2/ (pristupljeno: kolovoz 2018.)

[14] Principles of User Interface Design. Preuzeto sa: http://bokardo.com/principles-of-user-interface-design/ (pristupljeno: kolovoz 2018.)

[15] Majumdar R., Gupta R., Singh A. (2018.) Software Performance Measuring Benchmarks. International Conference on Wireless, Intelligent, and Distributed Environment for Communication, WIDECOM 2018. Lecture Notes on Data Engineering and Communications Technologies, vol 18. Springer, Cham

Popis slika

Slika 1. Procesiranje zahtjeva nad podacima 5
Slika 2. Prikaz pohranjene informacije za prometni segment 12
Slika 3. Sadržaj datoteke SPInfo.txt 13
Slika 4. Stablo mapa pohranjenih datoteka 13
Slika 5. Kôd za otvaranje tablice sadržaja podataka po linku 18
Slika 6. Kôd za prebacivanje informacija iz datoteke u novu tablicu 19
Slika 7. Kôd za pohranu obrađenih podataka 20
Slika 8. Aplikacija za iscrtavanje prometnog opterećenja u Zagrebu 21
Slika 9. Opcija za odabir promatranog dana 22
Slika 10. Pregled profila brzina tjekom dana 23
Slika 11. Kontekstualni meni s dodatnim opcijama 23
Slika 12. Izračun početnih i završnih koordinata linkova 25
Slika 13. Metoda crtanja prometnih segmenata 26
Slika 14. Funkcija zadužena za crtanje grafikona profila brzine u promatranom danu 27

Popis grafikona

Grafikon 1. Apsolutni profil brzina 9
Grafikon 2. Zaglađeni profil brzina 10

Popis kratica

SORDITO - Sustav za optimizaciju ruta u dinamičkom transportnom okruženju

API – Application Programmin Inteface

GPS – Global Positioning System

Popis priloga

Prilog 1. Metoda za kompilaciju sirovih podataka

```
def compileLargeChunks
    # creates temporary variables
    $keys = [[],[],[]]
    $dataPoint = {
        "idTable" => Array2D.new(Graphics.width, Graphics.height)
    }
    # starts IO for compiled data
    compiled = File.open("data/compiled.dat", 'wb')
    tm = Time.now
    # variables
    lines = []
    days = ["sunday","monday","tuesday","wednesday","thursday","friday","saturday"]
    x_min = 1000
    x_max = 0
    y_min = 1000
    y_max = 0
    # reads podaci.txt
    File.open("data/podaci.txt", 'rb') {|f| lines = f.read.split("\r\n"); lines.delete_at(0) }
    # saves the number of traffic segments
    Marshal.dump(lines.length, compiled)
    # processes each line individually
    for line in lines
        args = line.split(";")
        # canvas size determination
```

```
            x_min = [x_min,args[1].to_f,args[3].to_f].min
            x_max = [x_max,args[1].to_f,args[3].to_f].max
            y_min = [y_min,args[2].to_f,args[4].to_f].min
            y_max = [y_max,args[2].to_f,args[4].to_f].max
            # creates DataPoint object for each traffic segment
            id = args[0].to_i
            $dataPoint[id] = DataPoint.new(*args[0..5])
            speeds = {}
            # gets the data directory
            limits =
[[(id/10000.0).floor*10000,(id/10000.0).ceil*10000],[(id/100.0).floor*100,(id/100.0).ceil*100]]
            dir = "data/#{limits[0][0]}-#{limits[0][1]}/#{limits[1][0]}-#{limits[1][1]}/#{id}"
            if File.safeDirectory?(dir) && File.safeExist?("#{dir}/SPInfo.txt")
                # opens file if able
                File.open("#{dir}/SPInfo.txt", 'rb') {|f|
                    inter = f.read.split("\r\n")
                    # calculates the index at which to start
                    x = inter[0].split(/\s/).reject{|e| e.nil? || e == "" }.length - 1
                    y = 7
                    # generates a table (swapped for easier reading)
                    table = Array2D.new(x, y)
```

```
# inserts data into table
for ky in 0...y
	sspl = inter[ky].split(/\s/).reject{|e| e.nil? || e == "" }
	for kx in 1...sspl.length
		table[kx - 1,ky] = sspl[kx]
	end
end
# formats tabled data
for mx in 0...x
	next if table[mx,0].include?("-")
	dsel = []
	jump = false
	# interprets the file
	for my in 1...y
		case my
		when 1
			for v in table[mx,my].scan(/./)
				dsel.push(days[v.to_i])
			end
		when 2
			jump = true if table[mx,my] == "4" || table[mx,my] == "5"
		when 3
			savg = table[mx,my]
		when 6
			path = table[mx,my]
		else
			next
		end
```

```
                    end
                    # logs data into main hash
                    file = File.open("#{dir}/#{path}", 'rb') if !jump && File.safeExist?("#{dir}/#{path}")
                    for md in dsel
                        speeds[md] = [savg]
                        if file
                            flines = file.read.split("\r\n")
                            if flines.length > 2
                                for j in 1...flines.length
                                    time = flines[j]
                                    entries = time.split(/\s/)
                                    entries.delete("")
                                    speeds[md].push(entries[1]) # pushes SMOOTH
                                end
                            end
                        end
                    end
                    file.close if file
                    file = nil if file
                end
            }
            # failsafe for hanging script
            if Time.now > tm + 5
                tm = Time.now
                Graphics.update
                GC.start
            end
        end
```

```
        # registers the speed hash
        $dataPoint[id].setDay(speeds)
        # registers key layers
        k = $dataPoint[id].detailed ? 2 : 0
        k = 0 if args[5].to_i == 0
        $keys[k].push(id)
        # failsafe for hanging script
        if Time.now > tm + 5
            tm = Time.now
            Graphics.update
            GC.start
        end
        Marshal.dump($dataPoint[id], compiled)
end
# saves canvas size comparisons
$dataPoint["x"] = x_min
$dataPoint["y"] = y_min
$dataPoint["width"] = (x_max - x_min)
$dataPoint["height"] = (y_max - y_min)
# dumps data
Marshal.dump($dataPoint["x"], compiled)
Marshal.dump($dataPoint["y"], compiled)
Marshal.dump($dataPoint["width"], compiled)
Marshal.dump($dataPoint["height"], compiled)
Marshal.dump($dataPoint["idTable"], compiled)
Marshal.dump($keys, compiled)
compiled.close
```

```
        msgbox_p "Compilation complete! The Application needs to restart to apply changes."
        exit
end
```

Prilog 2. Objekt formatiranih podataka

```
class DataPoint
    attr_reader :speed
    attr_accessor :erased
    # main constructor
    def initialize(*loaded)
        @points = ["id","x1","y1","x2","y2","cnt"]
        @data = {}
        @speed = {}
        @erased = false
        @detailed = false
        for i in 0...@points.length
            key = @points[i]
            n = loaded[i]
            if key == "id" || key == "cnt"
                @data[key] = n.nil? ? nil : n.to_i
            else
                @data[key] = n.nil? ? nil : n.to_f
            end
        end
    end
```

```
# returns the max speed in a given day
def max(day)
	return 0 if !self.detailed
	max = @speed[day][0].to_f
	for i in 1...@speed[day].length
		val = @speed[day][i].to_f * @speed[day][0].to_f/100.0
		max = [val, max].max
	end
	return max
end
# checks if there is data to extrapolate from
def detailed
	return !@speed.empty?
end
# adds speed data (depreciated)
def addSpeed(*points); end
# gets information from key
def getData(point)
	return @data[point] if @data.keys.include?(point)
	return nil
end
# registers data throughout a day
def setDay(speed)
	@speed = speed
end
```

```
    # gets the recorded speed data
    def getDay(day,time)
        index = (time - 19800)/300 + 1
        return 0 if !self.detailed
        return @speed[day][0].to_f if @speed[day][index].nil?
        return @speed[day][index].to_f/100.0 * @speed[day][0].to_f
    end
    # gets the relative percentage
    def getRel(day,time)
        rel = 0 if !self.detailed
        rel = self.getDay(day, time) / self.max(day)
        rel = 1.0 if rel > 1.0
        return rel
    end
    # returns all the key names
    def keys
        return @data.keys
    end
end
```

Prilog 3. Metoda za učitavanje podataka iz baze u radnu memoriju

```
def loadData
    # calls a graphics update
    Graphics.update
    tm = Time.now
    # reads from file
    $dataPoint = {}
    File.open("data/compiled.dat", 'rb') {|f|
        points = Marshal.load(f)
        points.times do
            point = Marshal.load(f)
            i = point.getData("id")
            $dataPoint[i] = point
            # failsafe for hanging script
            if Time.now > tm + 5
                tm = Time.now
                Graphics.update
            end
        end
        $dataPoint["x"] = Marshal.load(f)
        $dataPoint["y"] = Marshal.load(f)
        $dataPoint["width"] = Marshal.load(f)
        $dataPoint["height"] = Marshal.load(f)
        $dataPoint["idTable"] = Marshal.load(f)
        $keys = Marshal.load(f)
    }
```

```
    # call to prevent script hang
    Graphics.update
    # registeres all data points in main link state table
    for key in $dataPoint.keys
        next if !$dataPoint[key].is_a?(DataPoint)
        info = $dataPoint[key]
        next if !info.detailed
        # calculates x and y coordinates of links
        x1 = ((info.getData("x1") - $dataPoint["x"])/$dataPoint["width"]) * Graphics.width
        y1 = Graphics.height - ((info.getData("y1") - $dataPoint["y"])/$dataPoint["height"]) *
Graphics.height
        x2 = ((info.getData("x2") - $dataPoint["x"])/$dataPoint["width"]) * Graphics.width
        y2 = Graphics.height - ((info.getData("y2") - $dataPoint["y"])/$dataPoint["height"]) *
Graphics.height
        # checks if coordinates are in screen range
        next if x1 > Graphics.width || y1 > Graphics.height || x2 < 0 || y2 < 0
        # records route in main table
        analyzeLines(x1,y1,x2,y2,2,info.getData("id"))
    end
end
```

Prilog 4. Metoda analize prikupljenih podataka

```
def analyzeLines(x1, y1, x2, y2, width, parameter)
    # Return if width is less than or 0
    return if width <= 0
    # Reverse all parameters sent if 2 x is less than the first x
    x1, x2, y1, y2 = x2, x1, y2, y1 if x2 < x1
    # Get S (1/2 width)
    s = width / 2.0
    # If X Coordinates are equal
    if x1 == x2
        # Registers coordinates in table
        if !parameter.nil? && $dataPoint
            for i in (x1 - s).to_i...(x1 - s + width).to_i
                for j in ([y1,y2].min).to_i...(y2).to_i
                    $dataPoint["idTable"][i,j] = parameter
                end
            end
        end
    # If Y Coordinates are equal
    elsif y1 == y2
        # Registers coordinates in table
        if !parameter.nil? && $dataPoint
            for i in (x1).to_i...(x2).to_i
                for j in (y1 - s).to_i...(y1 - s + width).to_i
                    $dataPoint["idTable"][i,j] = parameter
                end
            end
        end
```

```
    end
    # Get Length
    length = x2 - x1 < (y2 - y1).abs ? (y2 - y1).abs : x2 - x1
    # Get Increments
    x_increment, y_increment = (x2 - x1) / length.to_f, (y2 - y1) / length.to_f
    # Get Current X and Y
    x, y = x1, y1
    # While Current X is less than end X
    while x < x2
        # Registers coordinates in table
        if !parameter.nil? && $dataPoint
            for i in (x - s).to_i...(x - s + width).to_i
                for j in (y - s).to_i...(y - s + width).to_i
                    $dataPoint["idTable"][i,j] = parameter
                end
            end
        end
        # Increment X and Y
        x += x_increment
        y += y_increment
    end
end
```

Prilog 5. Metoda za iscrtavanje prometnih segmenata

```
def refresh(full = false,transition = true)
        tm = Time.now
        # clears current map bitmap if full refresh is needed
        if full
            @dragged_x = 0 if transition && @zoom == 1
            @dragged_y = 0 if transition && @zoom == 1
            self.loadAnim(transition)
            for i in 0...$keys.length
                @sprites["map#{i}"].bitmap.clear
                @sprites["map#{i}"].x = 0
                @sprites["map#{i}"].y = 0
                @sprites["map#{i}"].zoom_x = 1
                @sprites["map#{i}"].zoom_y = 1
            end
            @sprites["detail"].x = Graphics.width/2
            @sprites["detail"].y = Graphics.height/2
            @sprites["detail"].zoom_x = 1
            @sprites["detail"].zoom_y = 1
            # clears details bitmap
            map = @sprites["detail"].bitmap
            map.clear
            # re-draws details bitmap
            det = Bitmap.new("data/details.png")
            large_x = (det.width.to_f/Graphics.width.to_f)/@zoom
            large_y = (det.height.to_f/Graphics.height.to_f)/@zoom
            drec                                                    =
Rect.new(@offset_x*large_x,@offset_y*large_y,map.width*large_x,map.height*large_y)
```

```
            map.stretch_blt(map.rect,det,drec)
            det.dispose
        end
        # caches the colour levels
        gradient = Bitmap.new("img/gradient.png")
        c1 = Color.new(255,255,255)
        # calculates the relations between canvas and screen
        l = 0
        m = 0
        for keys in $keys
            for key in keys
                next if @withDetails && m < 2
                next if !$dataPoint[key].is_a?(DataPoint) # skips if data linked with key is of
incorrect format
                info = $dataPoint[key]
                next if !full && !info.detailed # skips if no speed data
                next if info.getData("cnt") < @threshold && m != 0 # skips if below threshold
                # resets status
                info.erased = false
                # calculates x and y coordinates of links
                x1 = ((info.getData("x1") - $dataPoint["x"])/($dataPoint["width"]/@zoom)) *
Graphics.width - @offset_x
                y1 = Graphics.height*@zoom - ((info.getData("y1") -
$dataPoint["y"])/($dataPoint["height"]/@zoom)) * Graphics.height - @offset_y
                x2 = ((info.getData("x2") - $dataPoint["x"])/($dataPoint["width"]/@zoom)) *
Graphics.width - @offset_x
                y2 = Graphics.height*@zoom - ((info.getData("y2") -
$dataPoint["y"])/($dataPoint["height"]/@zoom)) * Graphics.height - @offset_y


                # checks if coordinates are in screen range
```

```
                    next if !self.lineInScreen?(x1,y1,x2,y2)
                    # determines colour
                    c = c1
                    # checks the mode parameter
                    det = ["night_speed","min_speed","avg_speed","speed_drop"][@mode]
                    # gets thickness of line
                    t = @zoom - 1
                    # checks if speed information is present
                    if info.detailed
                        spd = info.getRel(@day,@timeline) # external function to calculate relative speed
                        # gets colour from cached levels
                        c = gradient.get_pixel(99*spd,2)
                        # gets thickness of line
                        t = @zoom
                    end
                    t = 1 if t < 1
                    # draws routes
                    @sprites["map#{m}"].bitmap.drawLine(x1,y1,x2,y2,c,t.to_i)
                end
                # increment
                m += 1
            end
            gradient.dispose
            # refreshes magnifying lens
            self.refreshLens
            GC.start
            self.finishAnim(transition) if full
        end
```

Prilog 6. Metoda dohvaćanja profila brzine

```
def getSpeed
        return nil if !$mouse.over?(Rect.new(16,16,Graphics.width-32,Graphics.height-32))
        det = ["night_speed","min_speed","avg_speed","speed_drop"][@mode]
        avg = 0
        c = 0
        # for easier use, adds a radius around the mouse to check for average speed
        r = (@zoom > 1) ? 0 : 1
        r = 1 if self.magnifier
        # coordinates to check for (takes into account offsets and zooms)
        mx = ($mouse.x)/@zoom + @offset_x/@zoom
        my = ($mouse.y)/@zoom + @offset_y/@zoom
        # range to account for radius
        x1 = [mx.floor - r, 0].max
        x2 = [mx.ceil + r, Graphics.width].min
        y1 = [my.floor - r, 0].max
        y2 = [my.ceil + r, Graphics.height].min
        # main iterative loop
        if mx > 0 && my > 0
            # iterates through the radius range
            for x in x1...x2
                for y in y1...y2
                    # grabs ID from cursor position
                    next if $dataPoint.nil? || $dataPoint["idTable"].nil?
                    x = (x*1280.0/Graphics.width).to_i; x = 0 if x < 0; x = 1279 if x > 1279
                    y = (y*720.0/Graphics.height).to_i; y = 0 if y < 0; y = 719 if y > 719
                    id = $dataPoint["idTable"][x,y]
                    # failsafe
```

```
                    next if id.nil? || id == 0 || $dataPoint[id].nil?
                    next if $dataPoint[id].erased
                    next if !$dataPoint[id].detailed
                    @linkID = id
                    avg += $dataPoint[id].getDay(@day,@timeline)
                    c += 1
                end
            end
        end
        # returns speed values
        return nil if c == 0
        return 100 - (100*avg/c).round(1) if det == "speed_drop"
        return (avg/c).round(1)
end
```

Sveučilište u Zagrebu
Fakultet prometnih znanosti
10000 Zagreb
Vukelićeva 4

IZJAVA O AKADEMSKOJ ČESTITOSTI I SUGLASNOST

Izjavljujem i svojim potpisom potvrđujem kako je ovaj završni rad isključivo rezultat mog vlastitog rada koji se temelji na mojim istraživanjima i oslanja se na objavljenu literaturu što pokazuju korištene bilješke i bibliografija.

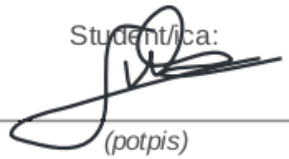
Izjavljujem kako nijedan dio rada nije napisan na nedozvoljen način, niti je prepisan iz necitiranog rada, te nijedan dio rada ne krši bilo čija autorska prava.

Izjavljujem također, kako nijedan dio rada nije iskorišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

Svojim potpisom potvrđujem i dajem suglasnost za javnu objavu završnog rada pod naslovom **Pohrana i obrada GPS podatka korištenjem prostorno vremenske baze podataka** na internetskim stranicama i repozitoriju Fakulteta prometnih znanosti, Digitalnom akademskom repozitoriju (DAR) pri Nacionalnoj i sveučilišnoj knjižnici u Zagrebu.

U Zagrebu, 9/7/2018

Student/ica:



(potpis)