Izrada multimodalnog planera putovanja za područje grada Zadra

Galac, Donat

SVEUČILIŠTE U ZAGREBU
FAKULTET PROMETNIH ZNANOSTI

Donat Galac

IZRADA MULTIMODALNOG PLANERA PUTOVANJA ZA PODRUČJE GRADA ZADRA

DIPLOMSKI RAD

Zagreb, 2015.

Umjesto ove stranice uvezuje se zadatak diplomskog rada...
Obrazac ZADATAK ZAVRŠNOG RADA uvezuje se kao prva stranica završnog rada ispred potkorice.

Sveučilište u Zagrebu
Fakultet prometnih znanosti

DIPLOMSKI RAD

IZRADA MULTIMODALNOG PLANERA PUTOVANJA ZA PODRUČJE GRADA ZADRA

Mentor: prof. dr. sc. Tonči Carić
Neposredni voditelj: Mario Buntić, mag. ing. traff.
Student: Donat Galac, 0246026066

Zagreb, rujan 2015.

Predgovor

Prvotno bih se zahvalio neposrednom voditelju Mariu Buntiću na velikom trudu i entuzijazmu kojeg je pokazao, te količini vremena kojeg je uložio za mene u toku izrade rada. Njegova volja i pristupačnost je uvelike pomogla mojoj kreativnosti.

Zatim bih se zahvalio mentoru, profesoru Tončiju Cariću na pruženoj prilici.

Hvala.

Sažetak

U diplomskom radu je izrađeno i opisano rješenje bazirano na *Web* aplikaciji za multimodalno planiranje putovanja na području grada Zadra. Na području Republike Hrvatske ima jako malo ovakvih rješenja (primjer je ZET planiranje putovanja na području grada Zagreba), stoga je cilj prikazati mogućnosti primjene ovakvih usluga u stvarnom okruženju. Prvotno su opisani samo značenje i vrste multimodalnih planova putovanja. Nakon toga, uz tehnologije koje su se koristile za izradu planera putovanja, opisan je postupak sakupljanja podataka za planiranje putovanja te njihov način spremanja u bazu podataka. Opisana je i GTFS specifikacija tranzitnih podataka, kako je nastala te njena primjena u praktičnom dijelu rada. Vrlo je bitan opis algoritma koji se koristio u postupku rutiranja jer predstavlja temelj praktičnog dijela rada, stoga je većina truda uložena u opis i usporedbu algoritama koji bi bili dobri kandidati za izradu rješenja. Opisan je i postupak implementacije odabranog algoritma u praktičnom dijelu rada. Rad kao cjelina opisuje danu temu na teoretskoj i praktičnoj razini te kao rezultat daje gotovo rješenje multimodalnog planiranja putovanja baziranog na *Web* tehnologijama.

Ključne riječi: Multimodalno planiranje putovanja, GTFS, Tranzitni podaci.

Abstract

In the final paper the solution based on Web application was described and made for multimodal travel planning in the area of Zadar. In Croatia, there are very few such solutions (the example is ZET travel planning in the area of Zagreb). Because of that, the main goal is to demonstrate possibilities of applying such services in a real environment. First, there were described the meaning and types of multimodal travel planners. After that, with the technologies that were used for creation of travel planners, there were also described the process of collecting vital informations for travel planning and implementation of those informations in database. GTFS specification of transit data was also described with its creation and appliance in practical part of paper. It is very important to describe algorithm that was used in the process of routing because it represents the base of practical part of paper, so the most of effort was invested in description and comparison of algorithms which would be good candidates for construction of the solution. The process of implementation of the selected algorithm in practical work was also described. The paper as whole describes a given topic on the theoretical and practical level, and as a result provides finished solution of multimodal travel planning based on modern technologies.

Keywords: Multimodal travel planners, GTFS, Transit data.

Sadržaj

1 Uvod

Planiranje putovanja bitno je svakom putniku, bez obzira na prometni entitet kojeg koristi. Može bitno smanjiti vrijeme putovanja i resurse. Stoga su današnje tehnologije omogućile izradu planera putovanja koji omogućuju putniku da unese podatke o startnoj i odredišnoj adresi u planer putovanja, a planer mu vrati linije mogućih ruta i prikaže ih u obliku geografske karte na kojoj se nalaze putanje gibanja javnog prijevoza. Ulazne podatke (početna i ciljna lokacija, datum i vrijeme polaska) unosi sam putnik. U diplomskom radu će biti kreirano takvo rješenje i bit će opisan postupak njegove izrade sa teoretskog i praktičnog stajališta.

U većini europskih zemalja sustavi za planiranje putovanja su svakodnevica za svakog autoprijevoznika. Iskoristivost takvih sustava se pokazala velikom, a pogotovo u gradovima sa gušće naseljenim stanovništvom. Dosadašnjim istraživanjem može se zaključiti da još uvijek ne postoji dovoljno kvalitetnih servisa za izradu multimodalnih planova putovanja u ovoj regiji (Hrvatska, Italija, Mađarska, Slovenija i Austrija), a oni koji postoje odnose se pretežito na lokalna područja. Svaki autoprijevoznik je u mogućnosti vizualno prikazati svoje transportne usluge i planove prijevoza na način da kreira određenu aplikaciju kojom će se spojiti sa servisom kao što je *Google* ili *Bing maps*. Bitno je naglasiti da u hrvatskim gradovima postoji jako malo takvih rješenja, a može se za primjer spomenuti ZET planiranje putovanja grada Zagreba. Kod ovakvih rješenja bitna je multimodalnost koja uključuje više vrsta transporta (autobusni, osobni, zračni ili brodski prijevoz). U diplomskom radu je cilj kreirati prototip lokalnog multimodalnog planera putovanja koji će obuhvaćati područje grada Zadra, a uključivat će autobusni, brodski i zrakoplovni prijevoz. Rad nakon uvoda u drugom poglavlju sadrži definicije o tipovima multimodalnih planera putovanja, njihov opis te što to multimodalnost u prometu znači. Bit će također spomenuto koji modovi transporta postoje te koje su razlike između lokalnih, globalnih i *real-time* planova putovanja. Treće poglavlje uključuje opis modela koji najbolje opisuju

pojedini mod prijevoza koji su obuhvaćeni u planeru putovanja. Sljedeće poglavlje sadrži opis algoritma u postupku rutiranja. Ovo poglavlje predstavlja temelj diplomskog rada jer se ponašanje i izlazni podaci kreiranog rješenja temelje na odabiru algoritma za kreiranje rute. Poglavlje također sadrži informacije o postojećim algoritmima koji se mogu koristiti za rutiranje. Sljedeće, peto poglavlje, obuhvaća praktične postupke prikupljanja podataka za planiranje putovanja. Sadrži bitne informacije o GTFS-u (eng. *General Transit Feed Specification*), relacijskoj reprezentaciji GTFS podataka, opis mogućih načina prikupljanja podataka vezanih za oznake geografskih lokacija te opis načina pohranjivanja tih podataka u relacijskoj bazi podataka. Nadalje, šesto poglavlje sadrži opis cjelokupnog postupka izrade lokalnog planera putovanja, a on uključuje tehnologije koje su se koristile za izradu rješenja, korisničke zahtjeve, arhitekturu *Web* aplikacije i postupak generiranja grafa iz relacijske baze podataka. U zadnjem poglavlju slijedi zaključak koji ujedno predstavlja i završetak rada.

2 Podjela multimodalnih planera putovanja

Pitanje koje se prvo može postaviti jest što je to zapravo multimodalni plan putovanja. To je informacijski sustav za planiranje putovanja koji uključuje u jednoj kombinaciji više vrsta transportnih modova jednu za drugom ili istovremeno. Ti modovi mogu uključivati cestovni prijevoz, željeznički prijevoz, zračni ili vodni prijevoz. Drugim riječima, definicija multimodalnosti u ovom kontekstu je korištenje različitih transportnih modova u različitim fazama jednog putovanja. Glavna ideja je da se u putovanju ne koristi samo jedan transportni mod. Sustavno gledajući, multimodalnost predstavlja razvoj neprekidne mreže integriranih transportnih lanaca povezanih cestovnih, željezničkih i plovnih puteva. Takva integracija dovodi do povećane fleksibilnosti, kvalitete i efikasnosti te stimulira konkurentnost na tržištu autoprijevoznika istog transportnog moda umjesto između više različitih transportnih modova [10].

Rani sustavi za planiranje putovanja su se isključivo razvijali za planiranje i praćenje prometa velikih vrijednosti kao što je željeznički promet. Dobri primjeri takvih sustava su *Sabre, Amadeus, Galileo* i *Rail Journey Information System* razvijen od strane Britanskih Željeznica. Nakon što su računalni sustavi postali dostupniji, informacijski sustavi za planiranje putovanja su se počeli razvijati na osobnim računalima i mobilnim terminalnim uređajima. Razvojem Internet mreže omogućen je i razvoj planova putovanja baziranih na *Web*-u, a takvo je rješenje razvijeno u praktičnom dijelu diplomskog rada. Početkom 21. stoljeća počeli su se razvijati lokalni multimodalni planovi putovanja za metropole kao što je London. Potom je razvijena usluga *Traveline* koja nudi globalno multimodalno planiranje putovanja kroz cijelo Ujedinjeno Kraljevstvo. Mnogi entiteti, uključujući općinske vlasti, države i savezne vlade koje posluju i putem *Web* stranice, sada nude usluge za planiranje putovanja za velika gradska područja, ili čak za cijelu državu. Razne kompanije kao što su *EasyJet, National Rail Enquiries* ili *Deutsche Bahn* putem svojih *Web* stranica nude besplatne usluge planiranja putovanja da bi na taj način povećali svoju zaradu. Daljnji razvoj je

omogućio stvarnovremeno planiranje putovanja koje omogućuje ažuriranje informacija u realnom vremenu na način da uključuje kašnjenja i promjene rasporeda u putovanju [9].

Osnovna ideja praktičnog dijela diplomskog rada je da *Web* aplikacija omogući vizualni prikaz najkraće rute između početne i ciljne lokacije na *Google* karti. Taj postupak će u sljedećim poglavljima biti detaljnije objašnjen. Bitno je naglasiti da *Google* servis ne podržava sve vrste transportnih modova. U tablici 2.1 su prikazani oni koji su bitni za praktični dio rada. *Route Type* je broj koji predstavlja tip rute i koristi se kod povezivanja sa *Google* uslugom [5].

Tablica 2.1: Transportni modovi u praktičnom dijelu rada

Route Code (Šifra Rute)	*Route Type* (Vrsta Rute)
700	*Bus Service* (Autobusni prijevoz)
1000	*Water Transport Service* (Brodski prijevoz)
1100	*Air Service* (Zrakoplovni prijevoz)

U nastavku su objašnjene karakteristike lokalnih, globalnih i *real-time* planova putovanja.

2.1 Lokalni multimodalni planovi putovanja

Lokalni planovi putovanja uključuju lokalna područja kao što je grad ili županija. Negativna strana takve vrste planova je što ne obuhvaća geografsko područje šire od županije, putnik ga može koristiti tek kad dođe u to područje. S druge strane, pozitivna strana ove vrste plana putovanja je što može uključivati detaljniji prikaz podataka na karti. Neka točka na karti globalnog planera putovanja može predstavljati cestovno raskrižje čiji su detalji prikazani na lokalnom planeru putovanja. Ti detalji uključuju sve elemente raskrižja, od pješačkog prijelaza do semafora, koji nisu prikazani u globalnom planeru putovanja. Neki europski primjeri lokalnih multimodalnih planera putovanja su:

- ZET planiranje putovanja
- *Dublin Bus Route Planner*
- *London Traveline*

- *Metro Transport For West Yorkshire*

2.2 Globalni multimodalni planovi putovanja

Globalni planovi putovanja uključuju globalna područja kao što su države, regije ili cijeli svijet. Uz dodatak multimodalnosti pružaju putniku kvalitetne informacije o putovanju, ali negativne strane su te što ne pružaju detaljne informacije o prometu kao lokalni planovi putovanja. Poznati primjeri globalnih multimodalnih planova putovanja:

- *Rome2rio*
- *World Route Planner*
- *routeRANK*

2.3 *Real-time* multimodalni planovi putovanja

Planovi putovanja u stvarnom vremenu (eng. *real-time*) su sa razvojem tehnologije došli kasnije kao dodatak lokalnim i globalnim planovima putovanja. Tijekom razvoja i provedbe multimodalnih planiranja putovanja pružanje informacija putniku doživjelo je brzu promjenu i rast, posebno u odnosu na stvarnovremene informacije. Prometni stručnjaci se nadaju da će informiranjem putnika o razini zagušenja, dostupnosti parkinga, autobusnom ili željezničkom prijevozu ili upozorenjima o opasnim stanjima na ruti potaknuti putnike da donose kvalitetnije, sigurnije i točnije odluke. Očekivane prednosti su također smanjenje kašnjenja, smanjenje ukupnog vremena putovanja te odabir alternativnih, bržih ruta. Velike promjene u *real-time* planiranju putovanja je uzrokovao GPS (eng. *Global Positioning System*) koji snabdjeva putnike lokacijski baziranim podacima kao što je vrijeme do dolaska autobusa na određenu stanicu. Iako je napredak postignut u pružanju stvarnovremenih informacija za planiranje putovanja, još uvijek postoje tehnička i financijska ograničenja. Da bi ovakvi sustavi postigli maksimalnu prednost moraju biti točni, pouzdani, konzistentni te lako dostupni. Postizanje visoke razine kvalitete ovakvih podataka i dalje je posebno teško

i skupo, osobito za urbana područja. Pokušaji da se potpuno integriraju stvarnovremeni prometni podaci za *Web* i mobilne aplikacije su i dalje u početnim fazama razvoja [16].

3 Modeli za izračun multimodalnih ruta za planiranje putovanja

Rute, odnosno linije za planiranje putovanja, reprezentiraju se pomoću grafa. U nastavku će biti opisana teorija grafova te sam način reprezentacije linija pomoću grafa. Postoji više modela koji se koriste za izradu algoritama za multimodalno planiranje putovanja. Svaki od tih modela ima različite karakteristike zbog čega je nemoguće svima jednako pristupiti. Planer putovanja od kojeg se sastoji praktični dio rada obuhvaća cestovnu (javni prijevoz), pomorsku i zračnu mrežu, stoga će u nastavku biti objašnjeni modeli koji podržavaju svaku od navedenih mreža. Potom će u sljedećem, četvrtom poglavlju, biti opisani algoritmi koji se mogu koristiti za izradu ruta putovanja i njihove karakteristike vezane za multimodalnost i vremensku ovisnost pri rutiranju. Proces izvršavanja algoritama za planiranje putovanja općenito zahtjeva veliki kapacitet radne memorije i vremena, stoga su osmišljene određene tehnike ubrzanja kojima se smanjuje vrijeme izvršavanja algoritama. Osnovni primjeri tehnika ubrzanja koje se mogu primjeniti na spomenutim tipovima mreža su također opisani u četvrtom poglavlju.

3.1 Teorija grafova

U svrhu boljeg razumjevanja modela i algoritama opisanih u nastavku ovog poglavlja potrebno je razumjeti i određene definicije vezane uz graf i njegovu strukturu. Način rada svih algoritama rutiranja temelji se na grafovima. Također je za multimodalno planiranje putovanja potrebno znati temeljne pojmove vezane uz regularne jezike (eng. *regular languages*) i automat (eng. *automata*).

3.1.1 Definicije grafa

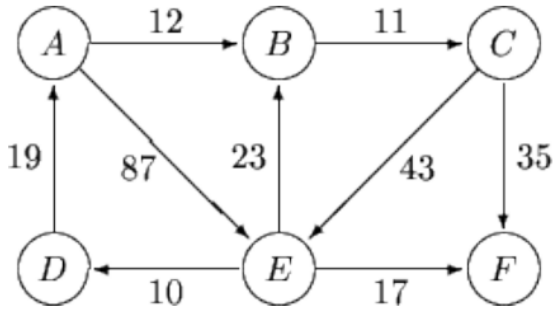
Definicije pojmova vezanih uz graf koji se spominju u nastavku rada [17]:

Definicija 1. ***Graf*** *$G = (V, E)$ se sastoji od skupa vrhova (čvorova) V i skupa bridova E. Svaki element iz skupa E je neusmjereni par vrhova. Neusmjereni bridovi imaju oznaku $\{u, v\}$ gdje vrijedi $u \in V$ i $v \in V$.*

Definicija 2. ***Jednostavan graf*** *je graf koji nema petlje ili višestruke bridove. U slučaju jednostavnoga grafa svaki se brid e iz skupa $E(G)$ može odrediti pomoću krajnjih vrhova u, v iz skupa $V(G)$. Petlje su slučajevi u grafu kad isti vrh predstavlja početnu i završnu točku jednog brida. Višestruki bridovi predstavljaju pojavu kad više bridova dijeli isti par početne i završne točke, odnosno vrha.*

Definicija 3. ***Težinski graf*** *je graf čijem se svakom bridu $e \in E$ pridruži njegova težina $w(e) \in \mathbb{R}_0^+$. Uređeni par (G, w) grafa G i funkcije w čine težinski graf.*

Definicija 4. ***Usmjereni graf*** *(digraf) je graf G ili skup vrhova V povezanih sa bridovima E kojima su dodjeljeni određeni smjerovi. Drugim riječima, digraf je par $G = (V, E)$ sa usmjerenim bridovima E.*



Slika 3.1: Usmjereni težinski graf.

Definicija 5. ***Podgraf*** *H je podgraf grafa G ako vrijedi $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$, a svaki brid iz H ima iste krajeve u H kao što ih ima u G. Podgraf se označava na sljedeći način $H \subseteq G$.*

Definicija 6. ***Put*** *u G je niz vrhova $v_0, v_1, v_2, ..., v_k$ umreženih tako da postoji brid između bilo koja dva susjedna vrha v_i, v_{i+1} u nizu. Ako postoje ponavljajući bridovi na putu, tada se radi o* ***stazi*** *na grafu, a ako postoje ponavljajući bridovi i vrhovi, radi se o* ***šetnji*** *grafom.*

Definicija 7. ***Ciklus*** *je jednostavan graf čiji se vrhovi mogu ciklički rasporediti na način da su dva vrha susjedna samo ako su neposredni u cikličkom raspoređivanju.*

Definicija 8. ***Povezani graf*** *je graf čija su svaka dva vrha povezana nekim putem.*

Definicija 9. ***Šetnja*** *je skup* $v_0, e_1, v_1, ..., e_k, v_k$ *vrhova i bridova takvih da ako vrijedi* $1 <= i <= k$*, onda brid* e_i *ima krajnje točke* v_{i-1} *i* v_i*.* ***Trag*** *je šetnja koja se sastoji od bridova koji se ne ponavljaju. Kada su prvi i zadnji vrhovi šetnje ili traga isti, za njih se kaže da su zatvoreni. Zatvoreni trag se zove* ***krug****.*

Slika 3.2: Grafički prikaz definicija vezanih uz graf.

Svi spomenuti grafovi u radu su usmjereni grafovi. Graf koji se dobije okretanjem smjerova svih bridova naziva se povratni graf (eng. *backward graph*) $\overleftarrow{G} := (V, \overleftarrow{E})$ gdje vrijedi $(u,v) \in \overleftarrow{E} \Leftrightarrow (v,u) \in E$.

Glavna razlika između vremenski ovisnog (eng. *time-dependent*) i vremenski neovisnog (*time-independent*) planiranja ruta je u vrstama težina bridova. U vremenski neovisnom planiranju ruta dovoljno je imati konstantne težine, stoga se taj koncept generalizira u periodične funkcije u svrhu prilagodbe različitim težinama bridova u različita doba dana. Sve funkcije povezane sa bridovima su elementi prostora funkcija $\mathfrak{T}$ koji se sastoji od funkcija $f: \mathbb{R}_0^+ \to \mathbb{R}_0^+$. Funkcija povezana sa bridom $e \in E$ ima oznaku f_e. U radu se koriste periodične funkcije sa vremenskim periodom Π, što znači da za sve $\tau \in \mathbb{R}_0^+$ mora vrijediti $f(\tau) = f(\tau \bmod \Pi)$. Nadalje, fokus je samo u funkcijama vremena putovanja (eng. *travel time functions*), što znači da je vrijednost funkcije $f(\tau)$ zapravo vremenski raspon. Iz tog razloga sve funkcije moraju ispuniti FIFO uvjet koji glasi: za bilo koja dva $\tau_1, \tau_2 \in \mathbb{R}_0^+$ sa $\tau_1 < \tau_2$ mora vrijediti $f(\tau_1) + \tau_1 < f(\tau_2) + \tau_2$. FIFO uvjet garantira da duž brida nije moguće krenuti kasnije iz polaznog čvora, a doći ranije na dolazni čvor. Ako za sve $\tau \in \mathbb{R}_0^+$ vrijedi $f(\tau) = c$ za neki $c \in \mathbb{R}_0^+$, tada se f zove konstantna funkcija. Neka su $f, g \in \mathfrak{T}$ dvije

funkcije, tada se izraz $f \oplus g := f + g \circ (f + id)$ definira kao operacija povezivanja (eng. *link operation*). *id* definira identifikacijsku funkciju. Ako su f i g konstantne funkcije, izraz tada glasi $f + g$. Ova se operacija koristi za kaskadiranje funkcija, odnosno za računanje vremena putovanja duž dva ili više uzastopnih bridova. Operacija minimuma (*minimum operation*) između dviju funkcija $f,g \in \mathfrak{T}$ koja ima oznaku min(f,g) dobiva se uzimanjem minimalne vrijednosti iz funkcija f i g za svaku ulaznu vrijednost $\tau \in \mathbb{R}^+$. Ova se operacija može nazivati i operacija spajanja (eng. *merge operation*) jer spaja dvije funkcije zajedno.

Donja granica funkcije f je minimalna vrijednost od f za bilo koji $\tau \in \mathbb{R}_0^+$, a ima oznaku $\underline{f} := min_{\tau \in \mathbb{R}_0^+} f(\tau)$. Gornja granica funkcije je maksimalna vrijednost svih njenih vrijednosti $\overline{f} := max_{\tau \in \mathbb{R}_0^+} f(\tau)$. Vremenski neovisna donja i gornja granica grafa $\underline{G}/\overline{G}$ može se dobiti iz G zamjenom svake funkcije brida sa donjom ili gornjom granicom.

Duljina puta P u grafu G je suma težina bridova duž puta i ima oznaku

$$duljina(P) := \sum_{i=1}^{k-1} f_{(v_i,v_{i+1})} = f_{(v_1,v_2)} \oplus f_{(v_2,v_3)} \oplus ... \oplus f_{(v_{k-1},v_k)}. \tag{3.1}$$

Duljina(P) daje funkciju koja se može interpretirati kao vrijeme putovanja duž puta za bilo koju danu vremensku točku τ. Međutim, ako su težine bridova duž puta konstantne, tada je i vrijednost duljina(P) konstantna. Često je bitno znati duljinu puta za određeno vrijeme polaska $\tau \in \mathbb{R}_0^+$. U tom slučaju se duljina(P,τ) definira na sljedeći način:

- Za $|P| = 1$: duljina(P,τ) := $f_{(v_1,v_2)}(\tau)$,
- za $|P| > 1$: duljina(P,τ) := duljina($P - v_k, \tau$) + $f_{(v_{k-1},v_k)}$(τ + duljina($P - v_k, \tau$)).

Drugim riječima, duljina(P,τ) se dobiva šetnjom duž puta s početkom u vremenu τ te mjerenjem svake funkcije brida za željeno vrijeme τ s dodatkom već prekrivene udaljenosti duž puta. Iz tog razloga je dobivena vrijednost duljina(P,τ) skalarna.

Udaljenost između dva čvora u,$v \in V$ koja ima oznaku daljina(P,τ) za dano vrijeme polaska τ je minimalna duljina svih puteva P od čvora u do v. Moguće je da postoji više od jednog puta najmanje udaljenosti od u do v. Put najmanje duljine između čvorova u i v u vremenu τ naziva se najkraći put (eng. *shortest path*) od u do v. Funkcija vremena putovanja koja predstavlja duljinu svih najkraćih puteva za sva doba dana ima oznaku

daljina*(u,v).

Za dva čvora se može reći da su "$u,v \in V$ povezani" ako postoji put od u do v. Ako je to točno za sve parove čvorova $u,v \in V$, cijeli graf se zove "povezani graf". Za nepovezani graf G, povezani podgraf $G' \subseteq G$ ima naziv "jako povezana komponenta" od G [21].

3.1.2 Jezici

Neka Σ bude konačan skup simbola koji se općenito zove abeceda (eng. *alphabet*). Slijed $w := [\sigma_1, \sigma_2, ..., \sigma_k]$ simbola iz Σ zove se riječ (eng. *word*). Duljina riječi je broj simbola od kojih se ona sastoji. Prazna riječ ima oznaku ε i duljinu jednaku 0. Za dvije riječi $w_1 := \sigma_1...\sigma_k$ i $w_2 := \sigma_{k+1}...\sigma_1$, ulančavanje (eng. *concatenation*) tih riječi $w := w_1 w_2$ dobiva se jednostavnim dodavanjem druge riječi prvoj, stoga vrijedi $w = \sigma_1...\sigma_k...\sigma_l$. Neobvezno konačan skup L riječi preko Σ zove se "jezik preko Σ" (eng. *language over* Σ). Sve operacije na skupovima kao što su unija, presjek i razlika također vrijede za jezike. Ako je L proizvoljan jezik, tada se i-ti "skup moći" (eng. *power set*) od L rekurzivno definira kao:

- ako $i = 0 : L^0 := \{\varepsilon\}$ i
- ako $i > 0 : L^i := \{wv \mid w \in L^{i-1}$ i $v \in L\}$.

U radu se koriste regularni (redovni) jezici za koje slijedi definicija.

Definicija 10. ***Regularni jezici (eng. regular languages)*** *- Neka* Σ *bude abeceda. Tada je jezik* L *preko* Σ *regularan samo ako ispunjava sljedeće uvjete izgradnje:*

- *Prazan jezik* $\oslash$ *je regularan.*
- *Za svaki* $\sigma \in \Sigma$ *jednočlani jezik* $\{\sigma\}$ *je regularan.*
- *Ako su* L_1 *i* L_2 *regularni jezici, tada su* $L_1 \cup L_2$, $L_1 \cdot L_2$ *i* L_1^* *također regularni jezici.*

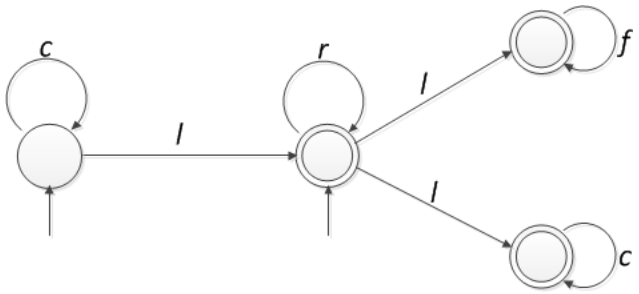
Osim korištenja regularnih izraza postoji drugi način da se opišu regularni jezici, a to je pomoću konačnog automata (eng. *finite automaton*).

3.1.3 Konačni automat

Neodređeni konačni automat $\mathcal{A} := (Q, \Sigma, \delta, S, F)$ sastoji se od konačnog skupa stanja Q, abecede Σ, funkcije prijenosa $\delta : Q \times \Sigma \to \mathcal{P}(Q)$, skupa početnih stanja S i od skupa konačnih stanja F. Konačni automat se često vizualno opisuje pomoću njegovog grafa prijenosa: stanja $q \in Q$ su zapisana kao čvorovi i za svako stanje $q \in Q$ i svaki simbol $\sigma \in \Sigma$ crta se brid od q prema q' označen sa σ samo ako vrijedi $q' \in \delta(q, \sigma)$. Početna stanja su obilježena dolazni-brid dojavom, dok su završna stanja dvostruko uokvirena. Slika 3.3 prikazuje jednostavan primjer konačnog automata koji je prikazan grafom prijenosa sa 4 stanja od kojih su 2 početna stanja, a 3 završna stanja.

Neka je $L \subseteq \Sigma^*$ proizvoljan jezik. Tada je riječ $w \in L$ prihvaćena od strane $\mathcal{A}$ ako postoji put u grafu prijenosa koji započinje u početnom stanju $q_0 \in S$ vodeći do konačnog stanja $q_f \in F$, i gdje su sljedeći bridovi na putu označeni sa sljedećim simbolima od w. Ako ne postoji put koji ispunjava ova svojstva, riječ se odbija. Ako za svaku riječ $w \in L$ vrijedi da je w prihvaćena od $\mathcal{A}$, kaže se da je jezik L prihvaćen od $\mathcal{A}$.

Svaki regularni jezik L može se opisati neodređenim konačnim automatom $\mathcal{A}$ u smislu da za svaku riječ $w \in \Sigma^*$ vrijedi da $\mathcal{A}$ prihvaća w samo ako je $w \in L$. S druge strane, vrijedi i da za svaki konačni automat $\mathcal{A}$ skup riječi prihvaćenih od $\mathcal{A}$ ima svojstva da bude regularan jezik. Zbog toga se termini regularnog jezika i konačnog automata mogu koristiti za isto značenje [21].



Slika 3.3: Primjer jednostavnog konačnog automata.

3.2 Modeli za izradu planera putovanja

Ranije je spomenuto da se planer putovanja sastoji od tri vrste prometnih mreža: mreža javnog cestovnog prijevoza (autobusni prijevoz), pomorska (brodske i trajektne linije) i zrakoplovna mreža. Svaka od tih mreža ima različite karakteristike. Zbog toga je potrebno koristiti različite pristupe prema svakoj vrsti prometne mreže. U nastavku će se predstaviti modeli od kojih će svaki predstavljati temelj za svoj tip mreže. Modeli za pomorsku i zrakoplovnu mrežu su u principu slični jer plovila kao i zrakoplovi imaju najčešće jednu odlaznu luku (aerodrom) i jednu dolaznu luku (aerodrom). Stoga su putanje njihovih grafova direktne i bez puno čvorišta. Posebni su slučajevi kad zrakoplovi presjedaju preko jednog ili više aerodroma da bi došli do odredišta, ali isto tako se i brodovi mogu vezati na više luka na putu prema odredišnoj luci. Poseban slučaj je cestovni javni prijevoz čiji se graf sastoji od velikog broja bridova i vrhova (čvorišta) jer na putu od polazišta do odredišta postoji veći broj stanica.

Da bi se izračunale multimodalne vrijednosti u planiranju putovanja algoritam mora iskoristiti više prometnih mreža istovremeno. Zbog toga različiti prometni modeli koji kreiraju različite grafove moraju biti međusobno "zalijepljeni". To se radi na način da se dodjele x i y koordinate svakom čvorištu u grafovima te se riješava "problem najbližeg susjeda" na podskupu parova čvorova. Problem najbližeg susjeda će biti objašnjen u nastavku kao i proces kombiniranja prometnih mreža. [21].

3.2.1 *Time-Independent* i *Time-Dependent* modeli

Svi modeli koji opisuju svoj tip prometne mreže temelje se na dva glavna, *time-independent* i *time-dependent* modela. Glavna razlika između ova dva modela je u dodjeljenim "težinama" bridova koji su konstante u *time-independent* modelu, a funkcije u *time-dependent* modelu.

Time-Independent modeli se najviše koriste u cestovnim prometnim mrežama. Kod ove vrste modela svakom se bridu e dodjeljuje konstantna vrijednost w koja može predstavljati vrijeme putovanja, geografsku udaljenost ili bilo koju drugu veličinu. Ako neki brid konstantne vrijednosti predstavlja neki dio rute, algoritam za planiranje putovanja će

uvijek izračunati istu vrijednost za taj dio rute. Postavlja se pitanje koliko je ovaj model realan ukoliko konstanta vrijednost koja se dodjeljuje pojedinom bridu predstavlja vrijeme putovanja jer vrijeme putovanja nije isto za vrijeme glavnog prometnog sata kao i u noći. Za rješavanje "problema najkraćeg puta" koji će kasnije biti objašnjen, kod ovog modela se može koristiti *Dijkstra* algoritam sa manjim modifikacijama [21].

Općenito u cestovnim prometnim mrežama odsutnost vremenske ovisnosti ne utječe toliko loše na izračune prometnih ruta koliko kod drugih tipova prometnih mreža. Ali ukoliko je ta cestovna prometna mreža zapravo autobusna mreža, ovisnost o vremenu mora biti od velike važnosti pri računanju prometnih ruta jer izračun najkraćeg puta može ovisiti o vremenu polaska autobusa sa polazne stanice. U diplomskom radu se za cestovni javni prijevoz zbog ovisnosti o vremenu koriste modeli temeljeni na *time-dependent* modelu. Izračun prometnih ruta kod pomorske i zrakoplovne prometne mreže također ovisi o vremenu, tako da se njihovi modeli također temelje na *time-dependent* modelu.

***Time-Dependent* modeli** ne sadrže konstantne težine dodjeljene bridovima. Bridovima se umjesto konstantnih težina dodjeljuju proizvoljne funkcije f iz nekog prostora funkcija $\Im$. Najkraći *s-t* put u vremenski ovisnom modelu tada ovisi o vremenu polaska τ_s sa izvornog čvorišta. Neka proizvoljna funkcija f bude linearna po dijelovima, a to znači da se sastoji od konačnog broja segmenata. Tada se f može opisati kao konačan $\mathfrak{B}$ skup interpolacijskih točki gdje se svaka interpolacijska točka $\mathfrak{p}_i \in \mathfrak{B}$ sastoji od vremena polaska τ_i i vezane funkcije vrijednosti $f(\tau_i)$. Tada se vrijednost funkcije f za proizvoljno vrijeme τ računa pomoću interpolacije. Računanje funkcije se obavlja na način da je prvo potrebno čekati polazak sljedećeg autobusa, broda ili zrakoplova. Nakon toga se dodaje čisto vrijeme putovanja bridu koji je povezan sa čvorištem polaska. Potom se za neku proizvoljnu vremensku točku τ koristi najbliža buduća interpolacijska točka τ_i i interpolira se prema formuli $f(\tau) = -\gamma(\tau_i - \tau) + f(\tau_i)$. $-\gamma$ predstavlja fiksni uspon funkcije f za kojeg vrijedi $\gamma \in [-1, 0]$, τ_i predstavlja vremensku točku polaska autobusa, broda ili zrakoplova, τ predstavlja proizvoljnu vremensku točku, a $f(\tau_i)$ predstavlja funkciju vremenske točke polaska [21].

3.2.2 Mreža javnog cestovnog prijevoza

Mreža javnog cestovnog prijevoza ovdje predstavlja autobusni prijevoz. Model ovog tipa prometne mreže se može gledati kao spoj dvaju segmenata: cestovna mreža i mreža javnog prijevoza. Za potpuno razumijevanje ovog modela potreban je opis jednog i drugog segmenta. Izračun ruta kod cestovnog prijevoza može se temeljiti na *time-independent* modelu kod kojeg se bridovima u grafu dodjeljuju konstantne vrijednosti vremena putovanja. Izračun ruta kod javnog gradskog prijevoza se temelji na *time-dependent* modelu, iako se on odvija na cesti. Razlog tomu je što kod autobusnog prijevoza postoji raspored vožnje, vrijeme dolaska i odlaska sa stanica. Stoga u ovom slučaju nije realno dodjeliti konstantne vrijednosti bridovima u grafu koji spajaju stanice jer te vrijednosti variraju.

Cestovna mreža kao model je najjednostavniji jer je reprezentacija njegovog grafa neposredna. To znači da brid $e = u,v$ između dva čvora $u,v \in V$ postoji jedino ako postoji i dio puta koji spaja u i v u cestovnoj mreži. Ako je u realnosti put koji spaja u i v dvosmjeran, tada se i dva brida *(u,v)* i *(v,u)* implementiraju u graf. Težine koje se dodjeljuju bridovima u grafu cestovne mreže su konstantne vrijednosti i predstavljaju prosječno vrijeme putovanja na određenom segmentu puta. Prosječno vrijeme putovanja se računa na način da se prvo uzme prosječna brzina vozila na određenom cestovnom segmentu. Nakon toga se prosječna brzina računa sa geografskom dužinom gledanog cestovnog segmenta. Dobivena vrijednost predstavlja prosječno vrijeme putovanja. [21].

Mreža javnog prijevoza predstavlja tranzitnu mrežu za koju postoji više različitih modela koji se mogu koristiti za izračun grafa. Svi se ti modeli temelje na osnovnom *time-dependent* modelu. Prije svega je potrebno razumjeti vremenske tablice (eng. *timetables*) koje su temelj za modele koji opisuju tranzitnu mrežu.

Osnova svih modela kojima se može opisati tranzitna mreža je vremenska tablica na temelju koje se konstruira neka vrsta grafa na kojemu se računa najkraći put. Vremenska tablica je skup $(\mathcal{C}, \mathcal{B}, \mathcal{Z}, \Pi)$ u kojemu je $\mathcal{B}$ skup stanica, $\mathcal{Z}$ skup vozila, Π periodičnost tablice i $\mathcal{C}$ je skup elementarnih veza. Elementarna veza iz skupa $\mathcal{C}$ je definirana kao $c := (Z, S_1, S_2, \tau_1, \tau_2)$ i interpretira se kao vozilo $Z \in \mathcal{Z}$ koje putuje od stanice $S_1 \in \mathcal{B}$ do stanice

$S_2 \in \mathcal{B}$ tako da odlazi od stanice S_1 u vremenskom trenutku $\tau_1 < \Pi$ i dolazi na stanicu S_2 u vremenskom trenutku $\tau_2 < \Pi$. Za jednu elementarnu vezu vozilo mora putovati od stanice S_1 do stanice S_2 bez postojanja međustanica. Stoga se neko vozilo koje putuje nekom rutom sastoji od više elementarnih veza iz vremenske tablice. Prilikom računanja vremena putovanja elementarne veze c bitno je obratiti pozornost na to je li τ_2 veći od τ_1. Ako je veći, tada je vrijeme putovanja veze c jednostavna razlika između τ_2 i τ_1. Ako nije veći (u slučaju ako vozilo kreće prije ponoći sa S_1, a dolazi na S_2 nakon ponoći), tada se vrijeme putovanja računa kao zbroj vremena prestalog do ponoći i vremena od ponoći do dolaska na stanicu S_2. S obzirom na periodičnost Π vrijedi jednadžba [21]:

$$\Delta(\tau_1, \tau_2) := \left\{ \begin{array}{cc} \tau_2 - \tau_1 & \text{ako je } \tau_2 \geq \tau_1, \\ \Pi - \tau_1 + \tau_2 & \text{inače.} \end{array} \right\} \tag{3.2}$$

Iako postoji više različitih modela kojima se može opisati tranzitna mreža, u radu je odabran **stvaran vremenski ovisan** (*realistic time-dependent*) **model sa konstantnim vremenom prijenosa**. Da bi se lakše razumio takav model, u nastavku su opisane jednostavna i stvarna (realistična) verzija modela.

Jednostavna verzija. Kod jednostavne verzije ovog modela skup čvorišta je jednak skupu stanica, a vezni brid između dva čvorišta (vrhova) u i v se umetne samo ako postoji bar jedna veza između u i v u vremenskoj tablici. Ovdje bridovi postaju vremenski ovisni, a za vrstu funkcije brida se koristi ranije spomenuta linearna funkcija po dijelovima. Za svaku vezu c = (Z,S_1, S_2, τ_1, τ_2) u vremenskoj tablici dodaje se interpolacijska točka $p := (\tau_1, \Delta(\tau_1, \tau_2))$ funkciji f koja pripada bridu između S_1 i S_2. To se može zamisliti kao korespodencija interpolacijskih točki odlascima na određenom bridu prometne mreže. Stoga, ukoliko se analizira funkcija f na jednoj od interpolacijskih točki τ_i, vrijednost funkcije $f(\tau_i)$ će biti jednaka vremenu putovanja i-tog vozila na tom segmentu mreže. Ukoliko se analizira brid na ranijoj točki $\tau < \tau_i$, potrebno je za odlazak čekati vozilo na stanici S_1. Zaključno tome, težina brida $f(\tau)$ sastoji se od zbroja vremena putovanja $f(\tau_i)$ i vremena čekanja. To dovodi do jednadžbe 3.3 u kojoj vrijednost $(\tau_i - \tau)$ predstavlja vrijeme čekanja, a $f(\tau_i)$ predstavlja

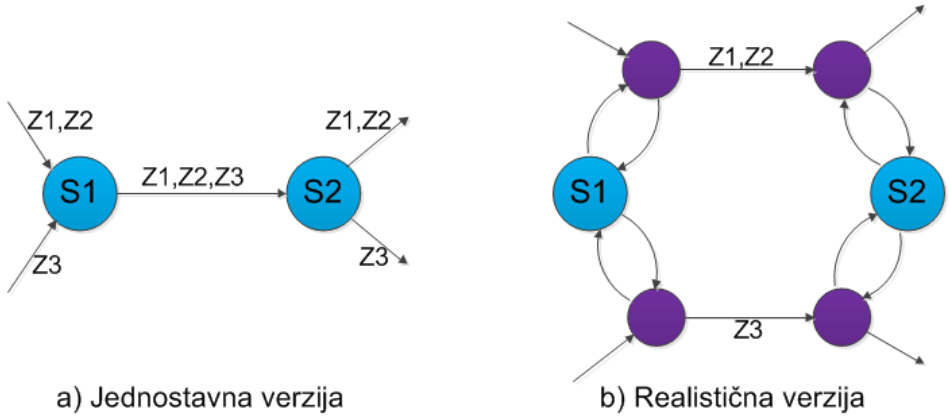
vrijeme putovanja [21].

$$f(\tau) = (\tau_i - \tau) + f(\tau_i) \tag{3.3}$$

Realistična verzija. Objašnjena jednostavna vezija modela ne vrijedi za stvarna vremena prijenosa. Zbog toga je jednostavna verzija unaprijeđena na realističnu verziju u kojoj se vremenska tablica proširuje funkcijom prijenosa: $\mathcal{B} \rightarrow \mathbb{R}_0^+$ s ciljem da se svakoj stanici dodjeli fiksno vrijeme prijenosa. Tada se graf realistične verzije *time-dependent* modela kreira tako da se za svaku stanicu $S \in \mathcal{B}$ u grafu uvodi "super-čvor" nazvan stanični čvor (eng. *station node*). Bitna razlika u odnosu na jednostavnu verziju je ta da stanični čvorovi nisu direktno povezani u realističnoj verziji modela. Ideja je da se u graf uvedu i dodatne vrste čvorišta nazvane čvorovi rute (eng. *route nodes*) i da vozila koja se kreću istim rutama zapravo putuju uzastopnim čvorovima rute. Čvorovi rute su potom povezani sa odgovarajućim staničnim čvorovima preko odgovarajućih težina koje odgovaraju vremenu prijenosa. Drugim riječima, skup vozila $\mathcal{Z}$ se dijeli na vozila rute (eng. *vehicle routes*). Skup vozila rute se označava sa $\mathbb{R}$, a svako vozilo rute $R \in \mathbb{R}$ je maksimalan podskup od $\mathcal{Z}$ koji sadrži samo vozila koja prate isti slijed stanica $[S_1, S_2, ..., S_k]$. To znači da ako dva vozila Z_1 i Z_2 prate istu rutu, može ih se smatrati ekvivalentnima. Neka skup $[S_1, S_2, ..., S_k]$ bude slijed stanica koje pripadaju nekoj tranzitnoj ruti $R \in \mathbb{R}$. Tada se za svaku stanicu $S_i \in R$ umetne čvor rute r_i u graf. Odatle se spajaju sljedeći čvorovi rute sa vremenski ovisnim bridovima $e = (r_i, r_{i+1})$. Interpolacijske točke funkcije f_e brida e kreiraju se kao i za jednostavan model: Za svaku elementarnu vezu c koja pripada vozilu Z koristeći rutu R unosi se interpolacija $p = (\tau_1, \Delta(\tau_1, \tau_2))$. Fiksni gradijent svih funkcija iznosi -1. Da bi se dopustila komutacija vozila između različitih ruta unose se dodatni bridovi prijenosa (eng. *transfer edges* - bridovi koji ostvaruju zahtjev za održavanjem minimalnog vremena prijenosa) u graf. Neka se za neku stanicu S uzme svaki od čvorišta rute koji pripada stanici S. Tada se umetnu dva dodatna brida: jedan brid *(r,S)* iz čvorišta rute prema staničnom čvoru sa konstantnom težinom koja je jednaka 0 (to modelira izlazak iz vozila što ne oduzima nikakvo vrijeme), a drugi brid *(S,r)* se unosi da bi se modelirao ulazak u vozilo. Težina ovog brida ima vrijednost prijenosa *(S)*. Ovaj pristup se zove pristup konstantnog vremena prijenosa (eng.

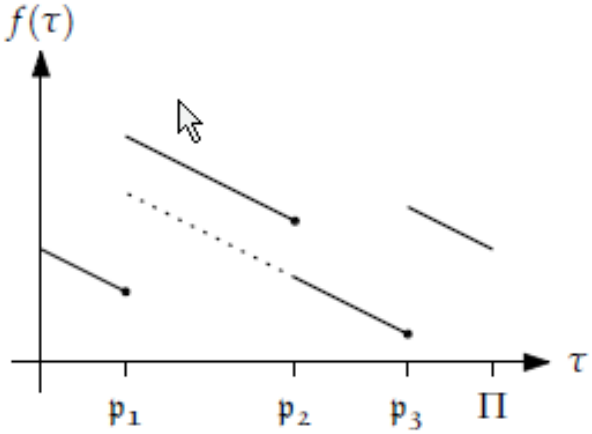
constant transfer time) s obzirom da se ukrcavanje u sva vozila na stanici obavlja sa istom vrijednosti prijenosa *(S)* [21].

Na slici 3.4 je prikaz razlike realistične verzije vremenski ovisnog modela u odnosu na jednostavnu na primjeru segmenta vremenski ovisnog modela u kojem se nalaze dvije stanice, S_1 i S_2. Stanični čvorovi su označeni plavom, a čvorovi rute su označeni ljubičastom bojom. Vozila Z_1 i Z_2 su ekvivalentni (koriste istu rutu), dok vozilo Z_3 koristi različitu rutu. U realističnoj verziji modela prelazak iz jednog vozila u drugo koje koristi različitu rutu zahtjeva prolazak kroz stanični čvor. Zbog toga je vrijeme prijenosa naplaćeno preko bridova koji spajaju stanicu i čvorove rute [21].



Slika 3.4: Prikaz razlike realistične verzije *time-dependent* modela u odnosu na jednostavnu na primjeru segmenta *time-dependent* modela.

Vremenski ovisan model ima restrikciju koju je bitno naglasiti. Kod ovog modela problem najkraćeg puta postaje kompliciran za rješavanja ukoliko funkcije dodjeljenih težina bridova u grafu ne ispunjavaju FIFO (*First Come First Serve*) uvjet. FIFO je svojstvo koje glasi da za svaku funkciju f mora postojati uvjet $f(\tau_1) + \tau_1 < f(\tau_2) + \tau_2$. To znači da ne smiju postojati dva vozila $Z_1, Z_2 \in R$ za koja između dvije uzastopne stanice S_1 i S_2 vozilo Z_1 kreće nakon vozila Z_2 sa S_1, ali stiže na S_2 stanicu prije vozila Z_2. Ovaj problem je prikazan na slici 3.5. Međutim, ovaj problem se može izbjeći na način da se ruta R podjeli na minimalan skup ruta u kojem svaka od ruta više ne sadrži konfliktna vozila [21].



Slika 3.5: Primjer narušavanja FIFO svojstva u *time-dependent* funkciji brida.

Na slici 3.5 je prikaz narušavanja FIFO svojstva u *time-dependent* funkciji brida u kojem treće vozilo prestiže drugo. Međutim, za neke τ između $\mathfrak{p}_1$ i $\mathfrak{p}_2$ za procjenu funkcije f (isprekidana crta) isplatilo bi se koristiti $\mathfrak{p}_3$ umjesto korištenja $\mathfrak{p}_2$ [21].

Stvaran vremenski ovisan model sa konstantnim vremenom prijenosa daje točne rezultate izračuna za problem najkraćeg puta u vremenski ovisnom okruženju. I u jednostavnoj i u realističnoj verziji sa konstantnim vremenom prijenosa izvorni i ciljni čvorovi su unaprijed poznati. Korištenjem vremenski ovisnog modela u izračunu ruta veličina grafa je manja nego u većini ostalih modela koji se koriste u tranzitnim mrežama, ali nedostatak je što graf nije vremenski neovisan. To povećava količinu memorije koja je potrebna za spremanje funkcija. Međutim, ovi nedostaci ne premašuju prednost grafa manje veličine, a samim time su i vremena izračuna puta manja [21].

3.2.3 Mreža zračnog prometa

Model koji se koristi za opis mreže zračnog prometa je vrlo sličan modelu javnog cestovnog prijevoza (tranzitne mreže). Međutim, potrebne su određene izmjene, jer korištenjem istog modela za opis mreže zračnog prometa nastaje graf sa nepotrebno velikim brojem čvorova i bridova. Stoga se uvodi model klase leta (eng. *flight-class model*) detaljnije objašnjen u nastavku. Ovaj model se također temelji na vremenskoj ovisnosti. Također su bitne promjene vezane za vremensku tablicu, za ovu mrežu se zapravo koriste vremenske tablice

letova (eng. *flight timetables*).

Kreiranje grafa zračne mreže zahtjeva vremensku tablicu letova iz koje ovakav graf nastaje. Vremenska tablica letova ima sličnu strukturu kao i ona od tranzitne mreže, ali umjesto stanice koristi se termin zračna luka, a umjesto vozila se koristi termin let. Vremenska tablica letova je skup $(\mathcal{C}, \mathcal{A}, \mathcal{F}, \zeta, \Pi)$ u kojem $\mathcal{C}$ predstavlja skup elementarnih veza (letova), $\mathcal{A}$ predstavlja skup zračnih luka, $\mathcal{F}$ je skup letova, a Π je vremenski period. Dodan je ζ koji svaku zračnu luku mapira vremenskoj zoni kojoj ona pripada. Vremenske zone se reprezentiraju kao UTC (*Coordinated Universal Time*) pomak od UTC+0 vremenske zone. Elementarna veza $c \in \mathcal{C}$ je skup $c = (F, A_1, A_2, \tau_1, \tau_2)$ koji se reprezentira kao let $F \in \mathcal{F}$ koji započinje od zračne luke $A_1 \in \mathcal{A}$ u vremenu τ_1 i završava u zračnoj luci $A_2 \in \mathcal{A}$ u vremenu τ_2. τ_1 i τ_2 su vremenske točke koje ovise o vremenskoj zoni zračnih luka A_1 i A_2 [21].

Za računanje duljina letova također se koristi Δ spomenut u modelu za tranzitnu mrežu, ali korištenje samo Δ vrijednosti dovelo bi do krivih rezultata jer je potrebno uračunati i vremenske zone. Ako se uzme neki let između dvije zračne luke A_1 i A_2 sa vremenom polaska τ_1 i vremenom dolaska τ_2, pravo vrijeme trajanja leta se može izračunati tako da se τ_1 i τ_2 konvertiraju u UTC+0 vrijeme. To se može postići na način da se pomak vremenske zone ζ oduzme od vremena τ_1 i τ_2 kao u jednadžbi 3.4 [21]:

$$\tau_1' := \tau_1 - \zeta(A_1) \bmod \Pi \quad \text{i} \quad \tau_2' := \tau_2 - \zeta(A_2) \bmod \Pi \tag{3.4}$$

Koristeći jednadžbu izračuna se τ_1' i τ_2', te se na kraju dobije duljina leta $\Delta(\tau_1', \tau_2')$.

Jednostavna verzija. Iako su velike sličnosti između vremenskih tablica tranzitne mreže i zrakoplovne mreže, ipak postoje bitne razlike vezane za rute i procedure koje se odvijaju u zračnim lukama, što model tranzitne mreže čini nepodobnim. Jednostavna verzija vremenski ovisnog modela tranzitne mreže se samo djelomično može prilagoditi vremenskim tablicama zrakoplovne mreže. Čvorišta mogu predstavljati zračne luke, a vremenski ovisni bridovi se mogu umetnuti između dvije zračne luke, A_1 i A_2, samo ako postoji najmanje jedan let između A_1 i A_2. Interpolacijske točke za funkcije bridova se kreiraju identično kao i kod funkcija tranzitne mreže. Iako ovaj model na prvi pogled može dovesti do pravog

rješenja za problem najkraćeg puta, to tako zapravo nije jer putnik obično mora potrošiti više od jednog sata na razne procedure u zračnoj luci prije polaska. Zbog toga jednostavna verzija vremenski ovisnog modela nije dovoljno dobra [21].

Realistična verzija. S druge strane, realistična verzija vremenski ovisnog modela je dobar način za integraciju realističnih vremena prijenosa u model. To zahtjeva definiranje funkcije prijenosa $\mathcal{A} \to \mathbb{R}_0^+$ koja mapira svaku zračnu luku nekoj konstantnoj vrijednosti vremena prijenosa. Usprkos tome, i ova verzija ima nedostatke prilikom povezivanja sa vremenskim tablicama letova. Jedan od nedostataka je taj što vremenska tablica letova nema ruta jer ovdje sve rute imaju duljinu jednaku 1. I kad postoje letovi u rasporedima letova koji imaju presjedanja, odnosno međustanice, ti se letovi gledaju kao direktne veze između startne i ciljne zračne luke. Ovaj nedostatak se može izbjeći na način da se za svaku zračnu luku A uvede čvor rute kojemu pristupa bar jedan let i čvor rute od kojeg kreće bar jedan let. Drugim riječima, može se dogoditi da neki broj čvorova rute po zračnoj luci može biti povezan sa duplo većim brojem susjeda zračne luke A. To dovodi do još jednog problema, a to je veliki broj susjeda što uzrokuje velikim brojem čvorišta po zračnoj luci. Kod ove verzije vremenski ovisnog modela problem je također u tome što nije realna pretpostavka da su vremena prijenosa konstantna. Obično kad se putnik ukrcava u zrakoplov polazne zračne luke, on tada mora potrošiti više vremena na proceduru kao što je provjera prtljage i sigurnosna provjera nego onda kada presjeda u nekoj posrednoj zračnoj luci u kojoj mora samo proći iz jednog ulaza u drugi. Zbog tog razloga modeliranje realističnih procedura zračnih luka zahtjeva najmanje dvije vrste vremena po zračnoj luci, a to su vrijeme prijave (eng. *check-in time*) i vrijeme prijenosa (eng. *transfer time*). Vrijeme prijave se odnosi na vrijeme potrošeno na cijelu proceduru kao što je provjera prtljage, sigurnosna provjera i čekanje pred ulazom u zrakoplov. Vrijeme prijenosa predstavlja samo ono vrijeme potrebno za presjedanje između zrakoplova u posrednoj zračnoj luci. Postoji i još jedna vrsta vremena koja se odnosi na izlazak iz odredišne zračne luke. Ta vremena se mogu modelirati na način da se uvedu dodatni bridovi prijenosa između svih čvorova rute. To pak dovodi do velikog broja bridova u grafu. Stoga je za modeliranje zrakoplovne mreže potrebno uvesti posve nove modele od kojih je jedan već spomenuti model klase leta [21].

Za razumijevanje modela klase leta prvo je potrebno detaljnije opisati određene termine, a također je bitno znati da je vremenska tablica leta $(\mathcal{C}, \mathcal{A}, \mathcal{F}, \zeta, \Pi)$ temelj ovog modela.

- Vrijeme prijave ($\tau^{checkin} : \mathcal{A} \rightarrow \mathbb{R}_0^+$) predstavlja cijeli proces od dolaska putnika u zračnu luku do polaska zrakoplova. To se vrijeme sastoji od prijave putnika, prolaska sigurnosnih provjera, vrijeme čekanja na ulazak u zrakoplov te samo vrijeme ulaska u zrakoplov.

- Vrijeme odjave ($\tau^{checkout} : \mathcal{A} \rightarrow \mathbb{R}_0^+$) je zapravo obrnuti proces od vremena prijave koji se sastoji od izlaska iz zrakoplova, prolaska kroz carinu, te vremena potrebnog za prihvat prtljage.

- Vrijeme prijenosa ($\tau^{transfer} : \mathcal{A} \rightarrow \mathbb{R}_0^+$) je vrijeme potrebno za prelazak iz jednog zrakoplova u drugi u posrednoj zračnoj luci. Obično samo uključuje izlazak iz zrakoplova, prelazak kroz vrata prema drugom zrakoplovu te ulazak u njega.

- Terminalni čvor (eng. *terminal node*) je "super-čvor" koji u grafu zrakoplovne mreže predstavlja jednu zračnu luku A

- Odlazni čvor (eng. *departure node*) modelira odlazak leta sa terminalnog čvora A. To je zapravo čvor polaska iz zračne luke.

- Dolazni čvor (eng. *arrival node*) modelira dolazak leta na terminalni čvor A. To je zapravo čvor dolaska u zračnu luku.

- Brid prijave (eng. *check-in edge*) se u grafu implementira od terminalnog do odlaznog čvora, a njegova težina iznosi $\tau^{checkin}(A)$.

- Brid odjave (eng. *check-out edge*) predstavlja vezu od dolaznog čvora do terminalnog čvora, a njegova težina iznosi $\tau^{checkout}(A)$.

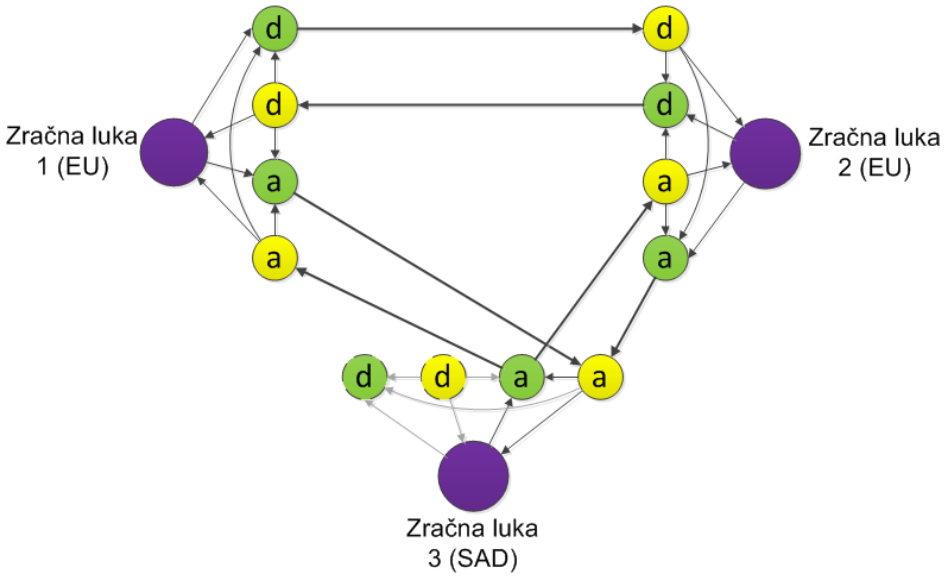
- Brid prijenosa (eng. *transfer edge*) predstavlja vezu od dolaznog čvora do odlaznog čvora, njegova težina iznosi $\tau^{transfer}(A)$.

- Brid leta (eng. *flight edge*) predstavlja pravi let od zračne luke A_1 do zračne luke A_2. On postoji samo ako postoji najmanje jedna elementarna veza između A_1 i A_2 [21].

***Flight-class* model.** Obično se pretpostavi da su sva tri vremena uvijek istih vrijednosti, ali to nije dovoljno dobro. Stoga je uveden model klase leta u kojem se pretpostavlja da zračna luka A sadrži domaće letova i međunarodne letove. Vrijeme prijave je poprilično manje ukoliko putnik ide na domaći let jer su sigurnosne prijave slabije i kratkotrajnije u odnosu na vremena prijave međunarodnih letova. Ova razlika također vrijedi za prijelaze iz domaćih letova u međunarodne i obrnuto. Zato se vremena prijave $\tau^{checkin}$, odjave $\tau^{checkout}$ i prijenosa $\tau^{transfer}$ proširuju da bi podržali različite klase letova. Ta proširenja se tada implementiraju u model klase letova. Slično kao i kod ruta u realističnom vremenski ovisnom modelu tranzitne mreže, skup letova $\mathcal{F}$ se dijeli na različite klase letova. Skup klase letova ima oznaku $\mathfrak{C}$. Neka su dva leta F_1 i F_2 slična ($F_1 \sim F_2$) ukoliko pripadaju istoj klasi, odnosno ako su odlazni i dolazni let u istoj državi (bez prelaska preko carine). Sa definiranim klasama leta vremenske funkcije se proširuju na sljedeći način: vrijeme prijave se proširuje na $\tau^{checkin} : \mathcal{A} \times \mathfrak{C} \to \mathbb{R}_0^+$, vrijeme odjave se proširuje na $\tau^{checkout} : \mathcal{A} \times \mathfrak{C} \to \mathbb{R}_0^+$, dok se vrijeme prijenosa proširuje na $\tau^{transfer} : \mathcal{A} \times \mathfrak{C} \times \mathfrak{C} \to \mathbb{R}_0^+$ tako da odgovara prijenosu između letova proizvoljnih parova klasa [21].

Da bi se integrirala proširenja vremenskih funkcija, model zrakoplovne mreže s konstantnim vremenima se proširuje na sljedeći način: Neka $A \in \mathcal{A}$ bude zračna luka. Umjesto jednog odlaznog i dolaznog čvora umetne se $k = |\mathfrak{C}|$ odlaznih i dolaznih čvorova - jedan za svaku klasu leta $c_i \in \mathfrak{C}$. Odlazni i dolazni čvorovi se vežu sa terminalnim čvorom preko bridova prijave i odjave. Za težine bridova koristi se $\tau^{checkin}(A, c_i)$ i $\tau^{checkout}(A, c_i)$ za svaku klasu. Za modeliranje prijenosa između proizvoljnih klasa letova, za svaki par c_i, c_j klase leta umetne se prijenosni brid iz dolaznog čvora klase c_i u odlazni čvor klase c_j. Prijenosnom bridu se dodjeli težina $\tau^{transfer}(A, c_i, c_j)$. Naposljetku se vremenski ovisni bridovi leta umetnu između dvije zračne luke A_1 i A_2 ovisno o klasi, npr. ako je let klase c, odlazni čvor koji pripada c-u u zračnoj luci A_1 koristi se kao rep, dok se dolazni čvor iste klase u zračnoj luci A_2 koristi kao glava brida leta. Interpolacijske točke funkcija bridova leta se kreiraju na isti način kao u objašnjenom modelu javnog cestovnog prijevoza, dok se ovdje samo elementarne veze tražene klase leta uzimaju u obzir. Da bi se izbjegli nepotrebni čvorovi, u svakoj zračnoj luci se mogu izostaviti odlazni i dolazni čvorovi koji pripadaju klasi leta koja ne sadrži nijednu odlaznu i dolaznu vezu prema/od zračne luke A. Slika 3.6 prikazuje

primjer modela klase letova koji sadrži jedan let međunarodne klase leta (EU - SAD) i dva leta domaće klase letova (unutar EU) od kojih domaća klasa predstavlja domaće letove (čvorovi označeni sa *d*), a međunarodna klasa predstavlja međunarodne letove (čvorovi označeni sa *a*). Odlazni čvorovi su zelene boje, a dolazni čvorovi su žute boje. Ukoliko postoje neki odlazni i dolazni čvorovi iste klase leta, mora postojati i brid leta koji će spajati te dvije vrste čvora. Na slici se pretpostavlja da zračna luka 3 nema dolaznih i odlaznih međunarodnih letova, stoga su ti čvorovi iscrtkani, a bridovi posivljeni. Ti čvorovi i bridovi se ne bi kreirali u realnom grafu. [21].



Slika 3.6: Primjer *flight-class* modela sa 3 zračne luke.

Može se zaključiti da *realistic-time dependent* model korišten u mreži javnog cestovnog prijevoza nije dovoljno dobar za modeliranje zračne mreže. Zbog toga je uveden *flight-class* model koji koristi konstantna vremena. Mogu se koristiti i promjenjiva vremena, ali ona su previše općenita, dok kod konstantnih vremena duljine trajanja prijave, odjave i prijenosa uopće ne ovise o pojedinim letovima nego su postavljena na nepromjenjivu vrijednost ovisno o klasi kojoj svaki let pripada.

3.2.4 Mreža pomorskog prometa

Ranije je spomenuto da je mreža pomorskog prometa slična mreži zračnog prometa. Model kojim se može opisati ova mreža se također temelji na osnovnom *time-dependent* modelu.

Flight-class model zračne mreže uz neke dodatne izmjene može biti koristan za opis pomorske mreže. Također je za kreiranje grafa ove mreže pogodna i vremenska tablica zračne mreže, ali je potrebna dodatna modifikacija tablice.

Neka se za konstrukciju grafa mreže pomorskog prometa koristi vremenska tablica plovidbe (eng. *sail timetable*). Neka se umjesto termina let ovdje koristi termin plovidba, a termin pomorska luka neka zamjeni zračnu luku. Vremenska tablica plovidbe neka bude skup $(\mathcal{C}, \mathcal{V}, \mathcal{S}, \Pi)$ u kojem $\mathcal{C}$ predstavlja skup elementarnih veza (plovidbi), $\mathcal{V}$ je skup luka, $\mathcal{S}$ je skup linija, a Π je vremenski period tablice. Ovdje ζ nije potreban jer pomorska mreža u diplomskom radu ne obuhvaća više vremenskih zona. Elementarna veza $c \in \mathcal{C}$ je skup $c := (S, V_1, V_2, \tau_1, \tau_2)$ koji se prikazuje kao plovidba $S \in \mathcal{S}$ koja počinje u luci $V_1 \in \mathcal{V}$ u vremenu $\tau_1 < \Pi$, a završava u luci $V_2 \in \mathcal{V}$ u vremenu $\tau_2 < \Pi$. Vremenske točke τ_1 i τ_2 u pomorskoj mreži pripadaju samo jednoj vremenskoj zoni. Za računanje duljine trajanja plovidbe koristi se već spomenuti Δ koji je u ovom modelu jednak tranzitnoj mreži jer nema vremenskih zona kao u zračnom prometu. Stoga se Δ računa jednostavnom razlikom između τ_2 i τ_1. Ukoliko je vrijeme τ_2 manje vrijednosti od τ_1, koristi se izračun:

$$\Delta(\tau_1, \tau_2) = \Pi - \tau_1 + \tau_2. \tag{3.5}$$

Jednostavna verzija. Ako bi se mreža pomorskog prometa promatrala kroz jednostavnu verziju vremenski ovisnog modela tranzitne mreže, uvidjele bi se neke sličnosti. Čvorišta, slično modelu zrakoplovne mreže, mogu predstavljati luke, a vremenski ovisni bridovi se također mogu umetnuti između dvije luke, V_1 i V_2, samo ako postoji najmanje jedna plovidba između V_1 i V_2. Kao i kod tranzitne mreže, za svaku vezu $c := (S, V_1, V_2, \tau_1, \tau_2)$ u vremenskoj tablici dodaje se interpolacijska točka $p := (\tau_1, \Delta(\tau_1, \tau_2))$ funkciji f koja pripada bridu između V_1 i V_2.

Realistična verzija. Jednostavna verzija vremenski ovisnog modela, kao i u ostalim prometnim mrežama, nije dovoljno dobra jer ona ne uključuje vrijeme prijenosa. Stoga se ponovno razmatra realistična verzija ovog modela. U mreži pomorskog prometa vrijeme prijenosa je bitno jer je potrebno određeno vremensko razdoblje za ukrcavanje putnika

i/ili vozila u trajekt ili brod u polazišnoj luci, a isto tako je potrebno vrijeme za iskrcavanje istih u dolazišnoj luci. Stoga je ponovno potrebno definirati funkciju prijenosa $\mathcal{V} \rightarrow \mathbb{R}_0^+$ koja mapira svaku luku nekoj konstantnoj vrijednosti vremena prijenosa. Ovakav način je zasad dovoljno dobar jer procedura pri ukrcaju i iskrcaju u plovilo zahtjeva jednako vremena. Unatoč tome, realistična verzija modela i u ovoj mreži ima nedostatke prilikom povezivanja sa vremenskim tablicama plovidbi. Ove vremenske tablice također imaju sve vrijednosti ruta jednake 1 jer nema posrednih luka, linije su direktne. Ovaj nedostatak se može izbjeći na isti način kao i kod mreže zračnog prometa tako da se uvedu dodatni čvorovi rute, što bi također uzrokovalo velikim brojem čvorova.

Postoji još jedan nedostatak kod ovog modela. Brodske i trajektne linije u diplomskom radu uključuju i međunarodne linije (primjer je linija Zadar - Ancona) čije trajanje procedura u lukama može varirati zbog prolaska putnika i/ili vozila kroz carinu. Ovaj nedostatak zahtjeva uvođenje nove vrste modela sličnog *flight-class* modelu koji se može nazvati *sail-class model*. Taj model temeljen na klasi plovidbe može modelirati mrežu pomorskog prometa ovisno o tome je li plovidba međunarodna ili unutar RH. Zbog vremena prijenosa koji u međunarodnoj plovidbi nisu konstantnih vrijednosti modeliranje realističnih procedura pomorskih luka zahtjeva ponovno uvođenje više vrsta vremena. Osim tipova vremena, opisani su dodatni termini koji se odnose na ovaj model:

- Vrijeme ukrcaja ($\tau^{board} : \mathcal{V} \rightarrow \mathbb{R}_0^+$) koje predstavlja proces čekanja u redu za ulazak u brod ili trajekt te sam proces ukrcavanja.
- Vrijeme iskrcaja ($\tau^{disembark} : \mathcal{V} \rightarrow \mathbb{R}_0^+$) koje se sastoji od čekanja na red na izlazak iz broda ili trajekta te sam proces iskrcavanja iz broda.
- Terminalni čvor (eng. *terminal node*) ovdje predstavlja jednu pomorsku luku V grafa pomorske mreže.
- Odlazni čvor (eng. *departure node*) modelira odlazak plovila sa terminalnog čvora V.
- Dolazni čvor (eng. *arrival node*) modelira dolazak plovila na terminalni čvor V.
- Brid ukrcaja (eng. *board edge*) u grafu se implementira od terminalnog do odlaznog čvora, njegova težina je vrijednosti τ^{board}.

- Brid iskrcaja (eng. *disembark edge*) predstavlja vezu od dolaznog čvora do terminalnog čvora, njegova težina je vrijednosti $\tau^{disembark}$.

- Brid plovidbe (eng. *sail edge*) u grafu predstavlja pravu plovidbu od luke V_1 do luke V_2. Postoji samo ako postoji najmanje jedna elementarna veza između V_1 i V_2.

***Sail-class* model.** Neka se skup plovidbi $\mathcal{C}$ dijeli na različite klase plovidbi, a neka skup klase plovidbi slično kao i u zrakoplovnoj mreži ima oznaku $\mathcal{K}$. Neka su dvije plovidbe C_1 i C_2 slične ($C_1 \sim C_2$) ukoliko pripadaju istoj klasi, odnosno ukoliko su odlazna i dolazna plovidba unutar RH. Sa definiranim klasama plovidbi vremenske funkcije se proširuju na način: vrijeme ukrcaja se proširuje na $\tau^{board} : \mathcal{V} \times \mathcal{K} \to \mathbb{R}_0^+$, a vrijeme iskrcaja se proširuje na $\tau^{disembark} : \mathcal{V} \times \mathcal{K} \to \mathbb{R}_0^+$.

Da bi se i u ovom modelu integrirala proširenja vremenskih funkcija, model pomorske mreže s konstantnim vremenima proširuje se na sljedeći način: neka $V \in \mathcal{V}$ bude pomorska luka. Umjesto jednog odlaznog i dolaznog čvora umetne se $k = |\mathcal{K}|$ odlaznih i dolaznih čvorova - jedan za svaku klasu plovidbe $k_i \in \mathcal{K}$. Odlazni i dolazni čvorovi vežu se sa terminalnim čvorom preko bridova ukrcaja i iskrcaja. Za težine bridova koristi se $\tau^{board}(V,k_i)$ i $\tau^{disembark}(V,k_i)$ za svaku klasu. Tu nema potrebe za umetanjem prijenosnog brida jer nema posrednih luka niti prijenosa između klasa plovidbi. Potom se vremenski ovisni bridovi plovidbi umetnu između dvije luke V_1 i V_2 ovisno o klasi. Ako je plovidba klase k, odlazni čvor koji pripada toj klasi u luci V_1 koristi se kao rep, dok se dolazni čvor iste klase u luci V_2 koristi kao glava brida plovidbe. Interpolacijske točke funkcije bridova plovidbi također se kreiraju na isti način kao kod modela zrakoplovne mreže.

Model pomorske mreže opisan je sa stajališta da je većim dijelom sličan modelu mreže zračnog prometa. Razlike su u tome što se graf kreira na temelju vremenske tablice koja nema različitih vremenskih zona, što je sličnije karakteristici vremenski ovisnom modelu tranzitne mreže. Ova mreža također ima vrijednosti rute jednake 1, a vremena prijenosa se očituju kroz vremena ukrcaja i iskrcaja koja su konstantna. Zbog mogućnosti plovidbe u inozemstvo uveden je model klase plovidbi po uzoru na model zračnog prometa, a od njega se razlikuje po tome što ne postoji $\tau^{transfer}$ jer u ovom slučaju ne postoji posredna

luka između polazišne i odredišne luke.

3.2.5 Kombiniranje mreža

Izračun najkraćeg puta na grafu uključuje mrežu javnog cestovnog prometa, mrežu zračnog prometa i mrežu pomorskog prometa u isto vrijeme. Stoga je potrebna kombinacija navedenih mreža u multimodalnu mrežu (eng. *multimodal network*). Neka se pretpostavi da su grafovi tranzitne (cestovne), zračne i pomorske mreže već kreirani. Kombinacijom tih grafova nastaje graf multimodalne mreže koji ima oznaku ***G = (V,E)***.

Za unos veza između parova čvorova različitih mreža koje su geografski jedna do druge uvodi se već spomenuti termin problema najbližeg susjeda (eng. *Nearest Neighbor Problem*). Neka je $\mathbb{R}^n$ n-dimenzionalan vektorski prostor od $\mathbb{R}$, a $P \subset \mathbb{R}^n$ je konačni skup vektora koji se zove kandidatne točke (eng. *candidate points*). Neka $d : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ bude metrika na $\mathbb{R}^n$.

Definicija 11. ***Problem najbližeg susjeda*** *- ako se u upit kao ulazne vrijednosti predaju metrički prostor (*$\mathbb{R}^n$,d*), skup kandidatnih točaka* P *na* $\mathbb{R}^n$ *i skup* Q *točaka upita na* $\mathbb{R}^n$*, kao izlazna vrijednost traži se mapa* $f : Q \to P$ *sa svojstvom:*

$$f(q) := p \Leftrightarrow \forall p' \in P : p \neq p' \Rightarrow d(p', q) \geq d(p, q) \tag{3.6}$$

Drugim riječima, za svaku točku upita $q \in Q$ pokušava se pronaći najbliža kandidatna točka $p \in P$ [21].

K-dimensional trees **algoritam.** Postoji više algoritama za rješavanje problema najbližeg susjeda, a jedan od njih je *k-dimensional trees* algoritam. To je struktura podataka dizajnirana za algoritme za geometrijsko pretraživanje. Glavna ideja kod ovog algoritma je generalizacija stabla binarnog pretraživanja u k dimenzija, što mu omogućuje sadržavanje k-dimenzionalnih vektora. Upiti k-dimenzionalnih točaka na taj način mogu biti riješeni u prosječnom logaritamskom vremenu. Algoritam djeluje u dvije faze. Prvo se k-dimenzionalno stablo kreira na temelju svih kandidatnih točaka P. Nakon toga se za svaku upitnu točku $q \in Q$ iskazuje upit na strukturi podataka, što vraća najbližeg susjeda od q. S obzirom da se

na svaki upit može odgovoriti u prosječnom vremenu $\mathcal{O}(log|P|)$, trajanje djelovanja algoritma se smanjuje na $\mathcal{O}(|Q|log|P|)$, za što je potrebno svega par sekundi na najsloženijim podatkovnim strukturama. Algoritam 1 prikazuje proces *k-dimensional trees* algoritma [21].

Algorithm 1 *k-d-Tree* pretraga

Podaci: Konačan skup kandidatnih točaka P i točaka upita Q. Metrika d.
Rezultat: Mapa $f: Q \rightarrow P$ koja svaku točku upita dodjeljuje njezinom najbližem susjedu.

```
1  T ← new k-d-stablo
2  T.Build (P)
3  for all q ∈ Q do
4      f(q) ← T.Query(p)
```

Nakon definiranja *k*-dimensional trees algoritma, potrebno je kombinirati mreže $G_{cestovna}$, $G_{zračna}$ i $G_{pomorska}$ u multimodalnu mrežu **G**. To se radi operacijama spajanja (eng. *merge*) i povezivanja (eng. *link*). Operacija povezivanja se može obavljati više puta i ona osigurava da se umetnu odgovarajući bridovi za pravilno povezivanje mreža, dok operacija spajanja ujedinjuje skupove čvorova i bridova od više grafova. Neka se pretpostavi da je svaka mreža opremljena funkcijama $coord_x : V \rightarrow \mathbb{R}$ i $coord_y : V \rightarrow \mathbb{R}$ koje mapiraju svaki čvor njegovoj geografskoj lokaciji danoj u x i y koordinatama. Te koordinate predstavljaju vrijednosti geografske širine i dužine [21].

Operacija spajanja funkcionira tako da se dobiveni unimodalni grafovi G_1,...G_n sa skupovima čvorova i bridova $G_i = (V_i, E_i)$ spajaju u multimodalni graf **G = (V,E)** u kojemu su skupovi čvorova i bridova zapravo unije skupova čvorova i bridova ulaznog grafa. Po tome za čvorove vrijedi $\mathbf{V} := V_i \cup ... \cup V_n$, a za bridove $\mathbf{E} := E_i \cup ... \cup E_n$. Također je potrebno svakom čvoru i bridu dodjeliti funkcijsku oznaku $V \cup E \rightarrow \mathfrak{L}_{čvor} \cup \mathfrak{L}_{brid}$ u svrhu određivanja tipa čvora u rezultnom grafu. Oznake čvorova mogu biti cestovni_čvor, zračni_čvor i pomorski_čvor, dok oznake bridova mogu biti cestovni_brid, zračni_brid, pomorski_brid te povezni_brid novonastalog brida. Dodatno se uvodi zastavica (eng. *flag*) *dij*, koja pokazuje je li se određeni čvor $v \in \mathbf{V}$ može koristiti kao izvorni ili ciljni čvor za *Dijkstra* algoritam (ili bilo koji drugi algoritam za izračun najkraćeg puta). U tranzitnoj mreži, kao u pomorskoj i zračnoj mreži, svaki je čvor koji predstavlja stanicu, zračnu ili pomorsku luku

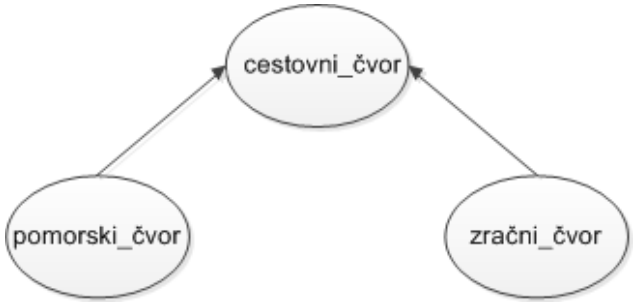
(terminalni čvor) pravovaljan da bude izvorni ili ciljni čvor u grafu za izračun najkraćeg puta. Ostali čvorovi u grafu imaju vrijednost dij zastave jednak *false*. *Dij* oznaka je bitna za operaciju povezivanja jer se povezni bridovi unose jedino između čvorova koji imaju vrijednost dij zastave jednak *true*. Prilikom kombiniranja više unimodalnih grafova, operacija spajanja se početno obavlja nad svim ulaznim grafovima. To rezultira multimodalnim grafom **G**. Sljedeći korak je opetovano primjenjivanje operacije povezivanja koja povezuje dva podgrafa različitih tipova u **G** umetanjem poveznih bridova [21].

Operacija povezivanja predstavlja proces umetanja veza između različitih mreža koje čine **G**. To se radi umetanjem veznih bridova. Ove veze određuju gdje je moguće presjedanje između mreža. Jedno pokretanje operacije povezivanja veže najviše dva tipa mreže. Stoga za multimodalnu mrežu koja se sastoji od više od jednog tipa mreže operacija povezivanja se mora pokretati više puta. Pošto se ovdje radi o tranzitnoj (cestovnoj), pomorskoj i zračnoj mreži, multimodalni graf koji se sastoji od tih mreža zahtjeva dvije operacije povezivanja: povezivanje zračne mreže sa tranzitnom mrežom i povezivanje pomorske mreže sa tranzitnom mrežom. Ovdje povezivanje zračne i pomorske mreže nije potrebno jer se sav promet odvija preko tranzitne (cestovne) mreže. Ako se uzme za primjer povezivanje pomorske mreže sa tranzitnom mrežom, za svaku pomorsku luku koja postoji u mreži cilj je pronaći čvor u tranzitnoj mreži koji je najbliži luci. Tada se može umetnuti veza između ta dva čvora. Pronalazak najbližeg čvora tranzitne mreže za određeni čvor pomorske mreže točno odgovara problemu najbližeg susjeda. Skup kandidatnih čvorova P se sastoji od svih čvorova $v \in \mathbf{V}$ sa $\mathfrak{L}_{čvor}(v)$ = cestovni_čvor, dok je skup upitnih točaka Q skup svih pomorskih čvorova (luka). Stoga je $v \in \mathbf{V}$ sa $\mathfrak{L}_{čvor}(v)$ = pomorski_čvor, a dij(v) = *true*.

Formalnije, ukoliko je potrebno povezati mrežu tipa T_1 sa mrežom tipa T_2, definira se $Q := \{v \in \mathbf{V} \mid \mathfrak{L}_{\text{čvor}}(v) = \text{T}_1 \text{ i dij}(v = \textit{true})\}$. Rješavanjem dvodimenzionalnog problema najbližeg susjeda na (P,Q,d) sa odgovarajućim d za svaki čvor $q \in Q$ tipa T_1 dobije se najbliži čvor p tipa T_2 iz P. Moguće je da u stvarnim grafovima ne postoji veza između dva čvora (ne postoji veza između pomorske i zračne luke), stoga se odbacuje par (q,p) ako je $d(q,p) > d_{max}$. d_{max} parametar koji predstavlja maksimalnu udaljenost do koje se veze između dva

čvora još uvijek mogu umetati. Na kraju se umetnu 2 brida $e_1 = (p,q)$ i $e_2 = (q,p)$ u graf [21].

Vrlo je bitan smjer bridova u grafu jer nije praktično umetati iste bridove u drugom smjeru koji povezuju pomorsku luku sa najbližom stanicom tranzitne mreže. Stoga se u ovom slučaju samo traže najbliže stanice tranzitne mreže za svaku luku, a ne traži se za svaku stanicu tranzitne mreže najbliža luka. Zato je potrebno obratiti pozornost na smjerove bridova na slici 3.7 koja pokazuje odabrani pristup pri povezivanju različitih tipova mreža prilikom kreiranja multimodalne mreže. Svaka strelica na slici predstavlja operaciju povezivanja između odgovarajućih čvorova. Stoga je bitno obratiti pozornost na smjer strelica.



Slika 3.7: Odabrani pristup pri povezivanju različitih tipova mreža prilikom kreiranja multimodalne mreže.

4 Algoritmi rutiranja pri izradi planera putovanja

Za razumijevanje algoritama koji se koriste za izradu planera putovanja prvo je potrebno razumjeti koncept problema najbržeg dolaska (eng. *earliest arrival problem*) i pokazati da je taj problem ekvivalentan problemu najkraćeg puta (eng. *shortest path problem*) koji je također bitan za algoritme rutiranja. Nakon problema najbržeg dolaska, u nastavku će biti opisano rješenje jednostavnog problema najkraćeg puta na unimodalnim vremenski neovisnim (eng. *time-independent*) mrežama i vremenski ovisnim (eng. *time-dependent*) mrežama koje je bazirano na *Dijkstra* algoritmu. Potom će se objasniti složeniji problem najkraćeg puta na multimodalnim mrežama i kako se može koristiti *Dijsktra* algoritam za rješavanje takvog problema. Rješenja tih problema bit će opisana na multimodalnoj mreži koja se sastoji od objašnjene pomorske, zračne i tranzitne mreže.

4.1 Problem najkraćeg puta i problem najbržeg dolaska

Prilikom planiranja putovanja postoji više kriterija po kojima se odabire najbolja ruta. Prva pomisao je na minimiziranje vremena putovanja, ali postoje drugi kriteriji kao što su jeftinija ruta (manja potrošnja goriva), ruta sa što manje prelaska između modova putovanja, najzanimljivija ruta i slično. Također je moguće kombinirati različite kriterije za izradu ruta, kao što je kombinacija najjeftinijeg i najbržeg. U radu su najbitniji kriteriji vrijeme putovanja i prostorna udaljenost između početne i ciljne stanice.

Definicija 12. ***Problem najbržeg dolaska*** *- uzimajući u obzir vremenski neovisnu i vremenski ovisnu mrežu, izvorišnu točku* s *i odredišnu točku* t *u mreži te vrijeme polaska* $\tau < \Pi$*, traži se ruta u mreži sa sljedećim karakteristikama:*

1. *Ruta mora počinjati od točke* s*.*

2. *Vrijeme polaska sa točke* s *je* τ.

3. *Ruta završava u točki* t.

4. *Vrijeme putovanja svih drugih ruta koje ispunjavaju gornje uvjete mora biti veće ili bar jednako odabranoj ruti.*

Drugi uvjet se ne mora poštovati ukoliko je osnovna mreža vremenski neovisna [21].

Drugim riječima, od svih mogućih ruta u mreži od ishodišta *s* do odredišta *t* (počevši u vremenu τ), traži se ruta koja dolazi u točku *t* prva.

Problem najkraćeg puta je od osnovnog interesa jer se najviše pojavljuje u stvarnim grafovima.

Definicija 13. ***Problem najkraćeg puta*** *- uzimajući u obzir težinski, usmjereni vremenski neovisni graf, vremenski ovisni ili mješoviti graf* G = (V,E), *izvorišni čvor* s $\in$ V, *odredišni čvor* t $\in$ V *i vrijeme dolaska* $\tau < \Pi$, *traži se put* P = *[*v_1, ..., v_2*] sa sljedećim svojstvima:*

1. *Put počinje u točki* s, *stoga je* v_1 = s.

2. *Put završava u točki* t, *stoga je* v_k = t.

3. *Za sve putove* P' *koji imaju gore navedena svojstva mora se poštovati uvjet da duljina(*P',τ*)* $\geq$ *duljina(*P,τ*).*

Ako je ***G*** *vremenski neovisan* τ *se ignorira, a duljina puta* P *se dobije od vrijednosti duljina(*P*) [21].*

Postoji više verzija problema najkraćeg puta koje se mogu pojaviti prilikom kriranja grafa:

- *Many-to-many*-problem najkraćeg puta - umjesto jednog čvora *s* i *t* postoji skup izvorišnih čvorova $S \subseteq V$ i skup odredišnih čvorova $T \subseteq V$. Na temelju tih skupova čvorova traži se najkraći put $P_{s,v}$ za svaki par $(s,t) \in S \times T$. U multimodalnom rutiranju koristi se ova verzija problema najkraćeg puta.

- *One-to-all*-problem najkraćeg puta - poseban slučaj many-to-many-problema najkraćeg puta u kojem je *S* jednočlani skup koji se sastoji od jednog izvornog člana *s*, a $T = V$ je skup svih čvorova. Tu se traže najkraći putevi P_v prema svakom čvoru $v \in V$.

- *All-pairs*-problem najkraćeg puta - verzija many-to-many-problema najkraćeg puta kod koje su S i T kompletan skup čvorova V u grafu. Ako se riješi ova verzija problema najkraćeg puta automatski se mogu riješiti sve druge navedene verzije ovog problema u grafu. Nedostatak rješavanja ovog problema je velika konzumacija memorije i vremena izvršavanja [21].

Svaki od tri verzije problema najkraćeg puta se može riješiti pomoću *Dijsktra* algoritma. Za svaku od opisanih mreža u diplomskom radu nadalje se primjenjuje problem najkraćeg puta u grafu koji je kreiran na temelju odabranih modela za svaku mrežu.

Mreža javnog cestovnog prijevoza. U vremenski ovisnoj tranzitnoj mreži (sa i bez realističnih vremena prijenosa) najkraći put rute se pronalazi na sljedeći način: svaki put kad se vezni brid koristi u mreži iz točke interpolacije koja se koristi za procjenu brida dobije se veza iz vremenske tablice za taj brid. S obzirom da je jedini način da se dođe iz jedne stanice do druge tako da se koristi vezni brid redoslijed dobivenih veza iz vremenske tablice formira ispravnu rutu. Ako je put u grafu najkraći put sa nekim vremenom polaska τ iz izvorišne autobusne stanice ta ruta je rješenje za problem najkraćeg puta sa tim vremenom polaska τ [21].

Mreža zrakoplovnog prometa. Za rješavanje problema najkraćeg puta koristi se isti dokaz istinitosti kao i kod realističnog vremenski ovisnog modela tranzitne mreže. S obzirom da su dvije zračne luke A_1 i A_2 povezane vremenski ovisnim bridom leta u grafu jedino ako postoji let u vremenskoj tablici između A_1 i A_2, put u grafu uvijek daje pravilnu rutu. Kao i u grafu tranzitne mreže pravi let na bridu leta se može dobiti iz interpolacijske točke koja je procjenjena na tom bridu leta. Osim toga, čvorovi i bridovi zračnih luka su modelirani tako da odgovaraju ograničenjima vremena prijave, odjave i prijenosa svake zračne luke. S obzirom da ova vremenska ograničenja vrijede također za sve rute ove mreže najkraći put se na kraju dobije iz rute koja zapravo rješava problem najkraćeg puta i obrnuto [21].

Mreža pomorskog prometa. Model ove mreže je sličan modelu mreže zrakoplovnog prometa, stoga su ovdje dvije pomorske luke V_1 i V_2 povezane vremenski ovisnim bridom

plovidbe u grafu jedino ako postoji plovidba u vremenskoj tablici između V_1 i V_2 i zato ovdje također put u grafu uvijek daje pravilnu rutu. Prava plovida brida plovidbe se može dobiti iz interpolacijske točke tog brida plovidbe, a čvorovi i bridovi pomorskih luka su ovdje modelirani tako da odgovaraju ograničenjima vremena ukrcaja i iskrcaja svake luke. Rute također moraju odgovarati ovim ograničenjima i zato ruta koja je najkraća, a poštuje ova ograničenja, daje rješenje problema najkraćeg puta.

4.2 Unimodalno rutiranje

Algoritam 2 prikazuje Dijkstra algoritam za rješavanje problema najkraćeg puta unimodalne mreže sa težinskim grafom *G = (V,E)*. U nastavku je algoritam objašnjen kroz vremenski neovisno i vremenski ovisno rutiranje.

Algorithm 2 *Dijkstra* algoritam

Podaci: Težinski graf *G = (V,E)*, čvor $s \in V$, i skup čvorova $T \subseteq V$.
Rezultat: Najkraći putevi od čvora s prema svim čvorovima $t \in T$.

```
1   Q ← prioritetni red čvorova
2   Q.insert(s, 0)
3   obrađeni-ciljevi ← ∅
4   while not Q.isEmpty() do
5       v ← Q.dequeue()
6       if v ∈ T then
7           obrađeni-ciljevi ← obrađeni-ciljevi ∪ {v}
8           if obrađeni-ciljevi = T then
9               stop ;                          /* najkraći putevi pronađeni */
10      for all izlazne bridove e = (v,w) do
11          if w je novi čvor then
12              Q.insert (w, daljina_s(v) + w(e))
13              preth(w) ← v
14          else
15              if daljina_s(v) + w(e) < daljina_s(w) then
16                  Q.decreaseKey(w,daljina_s(v) + w(e))
17                  preth(w) ← v
18  stop ;                                  /* niti jedan najkraći put pronađen */
```

***Time-independent* rutiranje.** U algoritmu se poziva čvor $v \in V$ odabran od algoritma iz prioritetnog reda čekanja (linija 5). Odabrani čvor se više ne smije vraćati u red čekanja, što znači da je svaki čvor procesuiran najviše jednom. Skup svih procesuiranih čvorova u algoritmu se naziva prostor pretrage (eng. *search space*). Brid koji je dodirnut od strane algoritma (linija 11) naziva se opušteni brid (eng. *relaxed edge*). Oznaka daljina$_s$(v) svakom čvoru dodjeljuje njegovu udaljenost od izvornog čvora s za vrijeme računanja algoritma. Oznaka preth(v) putem postavlja prethodni čvor od v. Na početku, za sve čvorove $v \in V$, ove dvije oznake moraju biti postavljene na vrijednosti daljina$_v = \infty$ i preth(v) = *null*. Nakon što algoritam završi računanje, daljina$_s(t)$ će sadržavati duljinu najkraćeg s-t puta [21].

Dijkstra algoritam rješava sve spomenute verzije problema najkraćeg puta. Kod verzija s jednim izvorom algoritam prestaje računanje čim su svi ciljni čvorovi $t \in T$ procesuirani (ovisno o skupu T to može biti jedan ciljni čvor, skup ciljnih čvorova, ili cijeli graf). Kod verzija sa skupinama izvornih čvorova u kojima postoji skup S izvornih čvorova, uzastopno se izvršava algoritam za svaki izvorni čvor te se tako dobije najkraći put od s-a prema svim ciljnim čvorovima T u jednom ciklusu algoritma.

Što se tiče verzija problema najkraćeg puta sa velikim brojem izvorišnih i odredišnih čvorova, često se potraga fokusira na samo jedan put minimalne duljine od skupine čvorova S do skupine čvorova T. Zbog toga je dovoljan samo jedan ciklus izvođenja *Dijkstra* algoritma. Linija 2 algoritma 2 se zamjenjuje tako da se ubacuju svi čvorovi iz skupa S u prioritetni red čekanja sa ključem 0, a umetanje se zaustavlja čim su svi ciljni čvorovi iz skupa T procesuirani. Željeno rješenje se dobije odabirom puta sa minimalnom udaljenosti od svih ciljnih čvorova. U ovom algoritmu je bitno da su vrijednosti svih bridova veće ili jednake 0. U suprotnom se mogu dobiti negativne vrijednosti puteva, što nije realno [21].

***Time-dependent* rutiranje.** Kod ove vrste rutiranja *Dijkstra* algoritam se proširuje tako da može savladati vremensku ovisnost. Uvedene su dvije varijante vremenske ovisnosti: vremenski upiti (eng. *time queries*) za računanje najkraćeg puta za fiksno vrijeme dolaska τ, i profilni upiti (eng. *profile queries*) kod kojih je fokus na pronalasku najkraćeg puta u svako doba dana.

Prilikom računanja vremenskih upita vremenski ovisna verzija *Dijkstra* algoritma je skoro identična vremenski neovisnoj verziji. Usprkos tome, bitno je da sve funkcije bridova ulaznog grafa ispunjuju FIFO svojstvo. Jedine promjene koje u algoritmu trebaju biti napravljene su:

- Potrebno je kao dodatnu ulaznu vrijednost uvesti vrijeme polaska τ.
- Da bi se procjenile težine bridova, potrebno je uzeti u obzir trenutna vremena nailaska na pojedini brid. Neka e = *(v,w)* bude brid čija se težina mora procjeniti. Tada je vrijeme u kojemu se procjenjuje funkcija f_e brida e zapravo vrijeme dolaska τ plus duljina puta do v (dostupna u algoritmu preko vrijednosti $\text{daljina}_s(v)$). Odatle *w(e)* mora biti zamjenjen sa $f_e(\tau + \text{daljina}_s)$ za sva pojavljivanja u algoritmu 2 [21].

Verzija sa vremenskim upitima računa samo najkraće puteve za jedno određeno vrijeme τ. Za razliku od te verzije, kod verzije sa profilnim upitima nije samo fokus na najkraćem putu u jednom vremenu nego u svako doba dana. Ako se uzme za primjer autobusni prijevoz neka autobusna linija kreće u $\tau = 8$ sati i treba joj do odredišta 2 sata. Neka postoji druga autobusna linija koja kreće u $\tau = 9$ sati, a treba joj 35 minuta do istog odredišta. Stoga bi bilo dobro pružiti putniku informacije o vremenu putovanja za svako vrijeme odlaska $\tau < \Pi$. Drugim riječima, rezultat upita bi trebao biti linearna funkcija po dijelovima f gdje svaka interpolacijska točka prezentira najkraći put za pripadajuće vrijeme. Profilni upiti su bitni za tehnike ubrzanja algoritama.

Profilni *s-t* upit u *Dijsktra algoritmu* se računa na sljedeći način: Neka $daljina_s^*(v)$ označava profilnu funkciju koja sadrži udaljenost od s do v za sva doba u danu. Tada se algoritam modificira koristeći $daljina_s^*(v)$ kao oznake udaljenosti na čvorovima. Kao ključ prioritetnog reda koristi se donja granica $\underline{daljina_s^*(v)}$ odgovarajuće oznake udaljenosti f. Na početku je izvorni čvor s umetnut sa konstantnom nultom funkcijom f_0 kao oznakom, dok su svi drugi čvorovi inicijalizirani sa f_∞. Tada se za svaki izvučeni čvor v sa oznakom udaljenosti $f_v := daljina_s^*(v)$ privremena funkcija $f_w^{nova} := f_v \oplus f_e$ iterira preko svih izlaznih bridova e = (v,w) i računa (operacija povezivanja) gdje f_e predstavlja funkciju dodjeljenu bridu e. Ako $f_w^{nova} \geq f_w$ ne vrijedi, tada je f_w^{nova} poboljšanje za bar jedno vrijeme polaska u danu. Potom se w unosi u prioritetni red sa oznakom $\min(f_w, f_w^{nova})$ (operacija spajanja).

Ovdje je potrebno dvije stvari naznačiti: nedodirnuti čvor se uvijek ubacuje u prioritetni red jer f_w^{nova} nikad ne vrijedi za $f_w = \infty$. Osim toga moguće je da se čvor u red ubacuje više puta za vrijeme jednog ciklusa algoritma, ali potrebno je uvesti kriterij prestanka ulaska u red. Stoga se prestaje kada najniži ključ u prioritetnom redu bude veći od gornje granice funkcije f_t dodjeljene ciljnom čvoru t. Tada je sigurno da ne postoje neistraženi putevi prema čvoru t [21].

Gornji algoritam koristi i operaciju povezivanja i operaciju spajanja. Problem započinje onda kada graf sadrži vremenski ovisne bridove koristeći linearnu funkciju po dijelovima sa različitim gradijentom. Vremenski ovisni bridovi koji modeliraju tranzitnu, pomorsku i zračnu mrežu imaju gradijent -1 dok grafovi koji se kreiraju sadrže i konstantne bridove koji se reprezentiraju kao konstantne funkcije koje imaju gradijent jednak 0. Povezivanje bridova različitih tipova nije problematično dok njihovo spajanje daje nehomogene linearne funkcije po dijelovima gdje nisu svi segmenti istog gradijenta. Rješenje za to postoji. Algoritam 3 predstavlja modifikaciju algoritma za profilne upite koji se zove ***multi label correcting*** algoritam. Umjesto jedne oznake po čvoru svakome čvoru se dodjeljuje skup oznaka. Za svaki procesuirani čvor uspoređuje se privremena funkcija f_w^{nova} sa svim oznakama dodjeljenim ciljnom čvoru w (linija 8), te se ponovno ubacuje čvor ako ne postoji oznaka dodjeljena čvoru w koja potpuno dominira nad f_w^{nova}. Ako nova oznaka f_w^{nova} potpuno dominira nad već postojećom oznakom f_w, tada se f_w može obrisati. Kriterij za zaustavljanje algoritma se mora modificirati tako da je potrebno uzeti maksimum od svih gornjih granica dodjeljenih ciljnim čvorovima t. Profilna funkcija na ciljnom čvoru t može se izračunati spajanjem svih oznaka odgovarajućeg ciljnog čvora [21].

Algorithm 3 *Multi label correcting* algoritam

Podaci: Težinski vremenski ovisan graf $G = (V,E)$, čvor $s \in V$, i skup čvorova $T \subseteq V$.
Rezultat: Profili od čvora s prema svim čvorovima $t \in T$.

```
1  Q ← prioritetni red čvorova
2  Q.insert((s, f_0), 0) ;        /* Ubaci izvor sa nultom funkcijom kao oznaku */
3  while not Q.IsEmpty() or not Q.isFinished() do
4      (v, f) ← Q.dequeue()
5      for all izlazne bridove e = (v,w) do
6          f_w^nova ← f ⊕ f_e ;   /* Izračunaj privremenu novu oznaku na čvoru w */
7          ins ← true
8          for all oznake f_w na w do
9              if f_w ≤ f_w^nova then
10                 ins ← false
11             else if f_w ≥ f_w^nova then
12                 ukloniOznaku (w,f_w)
13         if ins is true then
14             Q.insert((w, f_w^nova), f_w^nova) ; /* Koristi donju granicu funkcije f_w^nova
               kao ključ */
15 stop
```

Izlazna vrijednost svakog algoritma je profilna funkcija f_t koja daje vrijeme putovanja $f_t(\tau)$ od s do t za sve $\tau < \Pi$. Za određenu vremensku točku τ pravi najkraći put može se dobiti tako da se na τ koristi vremenski ovisna verzija algoritma 2. Informacije o putu se mogu integrirati u operaciju povezivanja dviju funkcija. Kad su dvije funkcije povezane informacije o putu se dodjeljuju interpolacijskim točkama, što se onda može koristiti za raspakiranje puta za danu interpolacijsku točku $\mathfrak{p}$. Što se tiče vremena izvršavanja algoritma čvorovi se više puta ubacuju u prioritetni red za procesuiranje što uzrokuje veću potrošnju vremena u odnosu na vremenske upite. Dodatno tome, operacija povezivanja je skupa, a kriterij zaustavljanja izvršavanja je kompleksan. Broj unosa čvorova u red ovisi o prirodi funkcija bridova, a ovdje je milijunski broj operacija ponovnog unosa u prioritetni red po *s-t* profilnom upitu. Sa povećanom vremenskom kompleksnosti može se zaključiti da je *multi label correcting* algoritam izvediv samo na jako malim mrežama [21].

4.3 Multimodalno rutiranje

U prijašnjem odlomku objašnjen je osnovni postupak pri rješavanju problema najkraćeg puta na *time-dependent* i *time-independent* unimodalnim mrežama. U ovom odlomku bit će objašnjeno rješenje za isti problem, ali u multimodalnom slučaju. Ovdje se zbog multimodalnosti koncept problema najkraćeg puta proširuje tako da se pomoću oznaka uvode uvjeti koje najkraći putevi moraju poštovati, inače će kao rezultat dati nepoželjne rute. Zbog toga se uvodi *label constrained shortest path problem* (problem najkraćeg puta sa uvjetnim oznakama) koji je zapravo proširenje klasičnog problema najkraćeg puta. Neka graf bude multimodalni graf $G = \{V, E\}$ koji se sastoji od vremenski ovisnih težina bridova (nema vremenski neovisnih težina jer se radi o vremenski ovisnim modelima mreža). Funkcija oznaka svakom čvoru i bridu dodjeljuje odgovarajući tip (cestovni_čvor, zračni_čvor ili pomorski_čvor te cestovni_brid, zračni_brid ili pomorski_brid). Skupovi čvorova i bridova imaju oznake $\mathfrak{L}_{čvor}$ i $\mathfrak{L}_{brid}$.

Definicija 14. ***Label constrained shortest path problem.*** *Neka se kao ulazne vrijednosti uzmu abeceda* Σ*, jezik* $\mathrm{L} \subset \Sigma^*$*, težinski usmjereni graf* $\boldsymbol{G = V,E}$ *sa bridovima označenim sa* Σ*, izvorni i ciljni čvorovi* $\mathrm{s,t} \in \Sigma$*, a kao izlazna vrijednost traži se najkraći put* P *od* s *do* t*, gdje redoslijed oznaka duž bridova puta formira riječ* L*. Tako dobiven* P = $[v_1, ..., v_k]$ *mora poštovati uvjet*

$$oznaka((v_1, v_2))oznaka((v_2, v_3))...oznaka((v_{k-1}, v_k)) \in \mathrm{L} \tag{4.1}$$

U radu se koristi regularni jezik koji je dovoljan za modeliranje razumnih ograničenja puteva, a to je REGL-CSPP *(label constrained shortest path problem restricted to regular languages)*, odnosno problem najkraćeg puta sa uvjetnim oznakama ograničen na regularne jezike. [21].

REGL-CSPP. Algoritam za rješavanja ove vrste problema najkraćeg puta izvršava se na produkcijskoj mreži ulaznog grafa **G** i konačnog automata koji opisuje jezik *L*. Ovaj algoritam se koristi kao rješenje za vremenski ovisne multimodalne mreže. Nadalje se skup oznaka bridova $\mathfrak{L}_{brid}$ identificira sa Σ.

Pošto se algoritam izvršava na produkcijskoj mreži, potrebna je definicija takve vrste mreže.

Definicija 15. ***Produkcijska mreža.*** *Ako se kao ulazne vrijednosti uzmu graf* $\boldsymbol{G}$ = **V,E** *sa* Σ *oznakama i neodređeni konačni automat* $\mathcal{A}$ = (Q,Σ,δ,S,F)*, produkcijska mreža* $G^x = (V^x, E^x)$ *se definira na sljedeći način:*

- *Skup čvorova se sastoji od produkcijskih čvorova* (v,q) $\in V^x$ *gdje vrijedi* v $\in$ **V** *i* q $\in$ Q*.*
- *Brid* $e^x = (v_1, q_1), (v_2, q_2)$ *je uključen između 2 produkcijska čvora u* E^x *samo ako vrijedi* e = $(v_1, v_2) \in$ **E** *i ako postoji oznaka* $\sigma \in \Sigma$ *za koju se nalazi prijenos* $q_2 \in \delta(q_1, \sigma)$ *u automatu. Težina od* e^x *je postavljena na težinu od* e*, a oznaka*(e^x) *je postavljena na* σ*.*

Rezultirajući graf je unimodalan [21].

REGL-CSPP za graf **G** = **V,E** sa Σ oznakama od izvora $s \in$ **V** do cilja $t \in$ **E** i sa regularnim jezikom $L \subseteq \Sigma^*$ može se reducirati na klasičan problem najkraćeg puta u svrhu lakšeg pronalaska rješenja, a to se može napraviti na sljedeći način:

1. Konstruira se neodređeni konačan automat $\mathcal{A}$ = (Q,Σ,δ,S,F) koji opisuje L.
2. Konstruira se produkcijska mreža G^x = **G** $\times \mathcal{A}$.
3. Rješava se *many-to-many* problem najkraćeg puta na G^x sa izvornim čvorom i skupom ciljnih čvorova
$$S := \bigcup_{q_s \in s} (s, q_s) \qquad i \qquad T := \bigcup_{q_f \in f} (s, q_f). \tag{4.2}$$
4. Od svih rezultantnih puteva odabire se onaj koji ima najmanju duljinu.

Neka $P = [(v_1, q_1), ..., (v_k, q_k)]$ bude najkraći put dobiven algoritmom u gornjim koracima. Potom neka duljina puta u **G**-u bude ista kao vrijednost duljina(P) u G^x. Pravi put u **G**-u može se dobiti izostavljanjem dijela produkcijskih čvorova koji su vezani za automat te se tako dobiju $[v_1, ..., v_k]$. S druge strane, riječ koja je u skladu sa L duž puta može se dobiti ulančavanjem oznaka bridova [21]:

$$riječ(P) := oznaka((v_1, q_1), (v_1, q_2))...oznaka((v_{k-1}, q_{k-1}), (v_k, q_k)). \tag{4.3}$$

Algoritam 4 pokazuje finalnu metodu izračuna najkraćeg puta. Iako se radi o mrežama koje imaju vremenski ovisne težine prikazuje se algoritam sa isključivo vremenski neovisnim težinama zbog svoje jednostavnosti. U ovom algoritmu, za razliku od prijašnjih algoritama, prioritetni red ima oznaku PQ (*priority queue*) da se izbjegne zabuna u odnosu na skup stanja konačnog automata Q. Ovaj algoritam je sličan algoritmu 2 osim razlika koje slijede: PQ sadrži produkcijske čvorove koji su sastavljeni od izvornog čvora s te bilo kojeg od početnih skupova konačnih automata (linija 2). U liniji 7 nije dovoljno smjestiti ciljni čvor nego je potrebno da automat bude u završnom stanju. Linije 11 i 12: za trenutni produkcijski čvor (v,q) koji je uzet u obzir simultano se iterira preko odlaznih bridova od v u **G**-u i odlaznih prijenosa u $\mathcal{A}$ iz stanja q označenog sa oznakom brida oznaka(e). Na taj način se simulira šetnja duž brida $e^x = ((v,q),(w,'q))$ produkcijske mreže. S obzirom na kriterij stopiranja izvršavanja algoritma (linija 7) najkraći put do t se dobije čim je prvi produkcijski čvor (t,q_f) smješten. Razlog tomu je što je potreban samo najkraći put minimalne duljine između svih parova izvornih i ciljnih čvorova koji se odabire u produkcijskoj mreži G^x [21].

Algorithm 4 *Time-independent Dijkstra* algoritam multimodalne mreže

Podaci: Multimodalni graf ***G = (V,E)***, čvor $s \in V$, skup čvorova $T \subseteq V$ i konačan automat $\mathcal{A}$ = (Q,Σ,δ,S,F) koji reprezentira regularan jezik $L \subset \Sigma^*$.
Rezultat: Najkraći put od čvora s prema čvoru t pronađen na temelju regularnog jezika L.

```
1  PQ ← prioritetni red produkcijskih čvorova
2  for all q_s ∈ S do
3      PQ.insert((s, q_s), 0)
4  obrađeni-ciljevi ← ∅
5  while not PQ.IsEmpty() do
6      (v, q) ← PQ.dequeue()
7      if v ∈ T and q ∈ F then
8          obrađeni-ciljevi ← obrađeni-ciljevi ∪ {v}
9          if obrađeni-ciljevi = T then
10             stop ;                          /* pronađeni svi najkraći putevi */
11     for all izlazne bridove e = (v,w) do
12         for all stanja q' ∈ δ(q,oznaka(e)) do
13             if (w,q') je novi produkcijski čvor then
14                 PQ.insert (w,q', daljina_s(v,q) + w(e))
15             else
16                 if daljina_s(v,q) + w(e) < daljina_s(w,q') then
17                     PQ.decreaseKey((w, q'),daljina_s(v,q) + w(e))
18 stop ;                                      /* nisu pronađeni svi najkraći putevi */
```

4.4 Tehnike ubrzanja

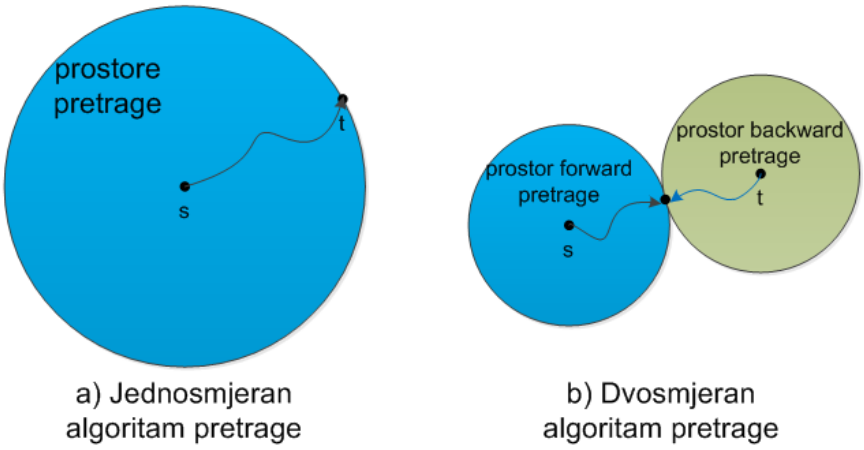
U prethodnim odlomcima opisani su osnovni algoritmi rutiranja za rješavanje problema najkraćeg puta na odabranim modelima mreže. Ti algoritmi se temelje na *Dijsktra* algoritmu. Glavni nedostatak tih algoritama je vrijeme izvršavanja, tek u zadnjih par godina računalni sustavi su se razvili dovoljno da bi mogli podržati velike mreže kao što su one u planiranju ruta. Čak i danas su vremena izvršenja algoritama prevelika. Iz tog razloga je istraživanje vezano za algoritme rutiranja dovelo do razvoja tehnika ubrzanja (eng. *speed-up techniques*) za *Dijsktra* algoritam koje za cilj imaju smanjenje prostora pretrage, dok pritom još uvijek daju optimalne rezultate. Tehnike ubrzanja temelje se na tri osnovna primjera: dvosmjerna

pretraga (eng. *bi-directional search*), pretraga usmjerena cilju (*goal-directed search*) i skupljanje (eng. *contraction*).

4.4.1 *Bi-directional* search

Prilikom računanja najkraćeg *s-t* puta koristeći *Dijkstra* algoritam pretraga počinje od s čvora i slijedno se smještaju čvorovi v dok ciljni čvor t nije smješten. Kad se čvor v jednom procesuira više se ne procesuira ponovno, stoga je $\text{daljina}_s(v)$ najkraća udaljenost od s do v i za sve čvorove u koji su smješteni prije v vrijedi $\text{daljina}_s(u) \leq \text{daljina}_s(v)$. Ovo računanje najkraćeg puta može se zamisliti balon koji raste oko čvora s tokom izvršavanja algoritma [21].

Ideja *bi-directional* rutiranja je da se uz izvršavanje *s-t* upita u G-u također izvršava upit od čvora t do s u grafu suprotnog smjera $\overleftarrow{G}$. *S-t* pretraga u G-u se naziva pretraga naprijed (eng. *forward search*), a *t-s* pretraga u $\overleftarrow{G}$-u se naziva pretraga unazad (*backward search*). Algoritam prekida izvršavanje čim *forward* ili *backward* pretraga smjesti čvor koji je već smješten suprotnom pretragom. Neka v bude čvor u kojem se *forward* i *backward* pretraga sastaju. Tada je najkraći put P od s do t u G-u zapravo sastav od posebno izračunatog najkraćeg puta od s do v (*forward* pretraga) i od posebno izračunatog *t-v* puta (*backward* pretraga). Bridovi *t-v* puta u $\overleftarrow{G}$-u moraju biti okrenuti u svrhu održavanja točnog puta. Slika 4.1 prikazuje usporedbu veličine prostora pretrage između jednosmjernog i dvosmjernog algoritma. Na slici je vidljivo kako prostor pretrage raste kao balon oko s i t čvorova. Prilikom istovremene pretrage od s i t prostor pretrage se može smanjiti jer algoritam može prestati sa računanjem čim je traženi čvor istovremeno u prostorima *forward* i *backward* pretrage.



Slika 4.1: Usporedba jednosmjerne sa dvosmjernom pretragom.

Algoritam upita je meta-algoritam koji se sastoji od dva *Dijkstra* algoritma: jedan za *forward s-t* pretragu, a drugi za *backward t-s* pretragu. Svaki algoritam mora označiti zastavicom čvorove koji pripadaju odgovorajućim prostorima pretrage, a drugi algoritmi moraju imati slobodan pristup tom čvoru. Meta-algoritam tada kontrolira *forward* i *backward* pretragom tako da naizmjence izvršava iteraciju u svakom algoritmu. Algoritam prekida izvršavanje čim se prostori pretrage susretnu. To proizvodi redukciju u prostoru pretrage jer je suma individualnih prostora *forward* i *backward* pretrage općenito manja od prostora pretrage samo jedne *forward* pretrage [21].

Uvođenje dvosmjerne pretrage u vremenski ovisne mreže stvara poteškoće što se tiče vremena upita. To se događa zbog činjenice da vrijeme dolaska na ciljni čvor t nije unaprijed poznato. Zbog toga se ne može pokrenuti *backward* pretraga u $\overleftarrow{G}$-u. Usprkos tome, postoji mogućnost da *backward* pretraga može barem asistirati *forward* pretrazi. *Forward* pretraga je vremenski ovisna sa vremenom polaska τ, dok se *backward* pretraga od t izvodi na vremenski neovisnom grafu donje granice $\underline{\overleftarrow{G}}$ od čvora t. Neka $\mathcal{S}$ bude skup čvorova unutar prostora *forward* pretrage, a $\overleftarrow{\mathcal{S}}$ neka bude skup čvorova otkrivenih *backward* pretragom. Tada, čim se dva prostora pretrage sastanu ($\mathcal{S} \cap \overleftarrow{\mathcal{S}} \neq \oslash$) računa se početni vremenski ovisan najkraći put do t. Neka v bude čvor u kojem se dva prostora pretrage

sastaju. Početni *s-t* put P_{prel} se dobije ocjenom *v-t* puta induciranog *backward* pretragom na vremenski neovisan način sa poštivanjem vremena dolaska na čvor v dobivenog *forward* pretragom. Duljina P_{prel} je samo gornja granica prave udaljenosti od s do t. U sljedećem koraku se nastavljaju vremenski ovisna *forward* pretraga i vremenski neovisna *backward* pretraga tako da se *backward* pretraga može zaustaviti samo ako postoji čvor $v \in \overleftarrow{S}$ sa vrijednosti $\overleftarrow{daljina_t}(v) \geq$ duljina P_{prel}. U tom slučaju može se sa sigurnošću zaključiti da je najkraći *s-t* put sadržan u $\mathcal{S} \cap \overleftarrow{S}$. Na kraju se isključivo *forward* pretraga nastavlja dok t nije smješten [21].

Prilagodba dvosmjerne pretrage u multimodalnim mrežama. *Forward* i *backward* pretraga su instance multimodalnog algoritma upita. Neka **G** bude multimodalni graf, $\mathcal{A}$ neka bude konačan automat i S,$T \subseteq \mathbf{V} \times Q$ neka bude skup izvornih i ciljnih produkcijskih čvorova. Kao i u unimodalnom slučaju, *forward* pretraga izvodi jednostavnu *S-T* pretragu, dok *backward* pretraga izvodi *T-S* pretragu. Algoritam može prestati izvršavanje čim se produkcijski čvor *(v,q)* smjesti od strane obje pretrage. Da bi se dobili pravilni putevi koji odgovaraju regularnom jeziku L reprezentiranom sa $\mathcal{A}$, *backward* pretraga se ne treba samo izvršavati na *backward* grafu $\overleftarrow{\mathbf{G}}$, nego i na inverznom automatu $\overleftarrow{\mathcal{A}}$. Konačni automat je $\mathcal{A}$ = (Q,Σ,δ,S,F), a inverzni automat $\overleftarrow{\mathcal{A}}$ = (Q,Σ,$\overleftarrow{\delta}$, S, F) se definira na sljedeći način: početna i završna stanja su okrenuta, stoga vrijedi $\overleftarrow{S} := F$ i $\overleftarrow{F} := S$, a funkcija prijenosa se invertira po pravilu [21]:

$$q' \in \overleftarrow{\delta}(q, \sigma) \quad \Leftrightarrow \quad q \in \delta(q', \sigma). \tag{4.4}$$

Ako je multimodalni graf **G** potpuno vremenski neovisan to dovodi do točnog rješenja za multimodalni problem najbržeg dolaska sa poštivanjem jezika L. Usprkos tome, u ovom slučaju **G** je i vremenski neovisan i vremenski ovisan. Stoga se koriste jednake restrikcije kao i u unimodalnom slučaju: vrijeme dolaska se ne zna unaprijed i zbog toga se *backward* potraga može samo koristiti za asistiranje *forward* pretrazi [21].

Može se zaključiti da je vremenski ovisna dvosmjerna pretraga jako kompliciranа, čak i sporija od jednostavnog vremenski ovisnog *Dijkstra* algoritma. Zbog toga je *bi-directional search* algoritam najbolje koristiti samo u vremenski neovisnim dijelovima mreže. S obzirom

da su opisani modeli mreže samo vremenski ovisni ovaj algoritam nema neku primjenu.

4.4.2 *Goal-directed search*

Jedan od algoritama pretrage usmjerene cilju zove se ALT (*A* with landmarks*). To je kombinacija A^* pretrage i poboljšanja korištenjem simbola (eng. *landmarks*) i trokutne nejednakosti (eng. *triangle inequality*). A^* spada u pretrage usmjerene cilju u smislu da prvo traži čvorove koji su bliže cilju. To se radi na način da se mijenja prioritet pojedinih čvorova u skladu sa potencijalnom funkcijom koja može modificirati redoslijed u kojemu se oni smještaju *Dijkstra* algoritmom. Na taj način se potraga gura prema cilju.

Neka $\pi : V \to \mathbb{R}$ bude potencijalna funkcija iz skupa čvorova grafa. Težine od G mijenjaju se u svrhu smanjenja troškova i to na način da se π dodaje težinama bridova: $w_\pi(u,v) := w(u,v) + \pi(v) - \pi(u)$. Graf čiji su svi bridovi izmjenjeni ima oznaku G_π. Tada se duljina bilo kojeg puta (uključujući najkraći put) $P = [v_1, ..., v_k]$ u G-u mijenja u $\text{duljina}_\pi(P) = \text{duljina}(P) + \pi(v_k) - \pi(v_1)$ jer na sljedećim bridovima potencijali poništavaju jedan drugog. Pošto *Dijsktra* algoritam funkcionira samo na pozitivnim težinama bridova, proizvoljne potencijalne funkcije nisu dozvoljene. Potencijalna funkcija nek se zove izvediv (eng. *feasible*) π ako je $\text{duljina}_\pi(u,v) \geq 0$ za sve puteve od u do v čvora i ako su proizvoljni $u,v \in V$. Pronalazak najkraćeg puta u G-u je ekvivalentan pronalasku najkraćeg puta u G_π [21].

Pri traženju najkraćih s-t puteva korištenje donje granice od svakog čvora v do ciljnog čvora t donosi izvedivu potencijalnu funkciju. Za potencijalnu funkciju nije potrebno da bude fiksna za sve upite, ali može biti ukoliko je ovisna o s-t upitu. U originalnom A^* algoritmu donja granica koja se koristi zapravo je direktna geografska udaljenost $\text{daljina}_{geo}(v)$ iz bilo kojeg čvora v do t. To funkcionira samo ako je mjerna jedinica u grafu geografska udaljenost, ali pošto se ovdje kao mjerna jedinica koristi vrijeme putovanja, to nije moguće. ALT algoritam koji se temelji na A^* uvodi znakove koji su konačni skup namjenskih čvorova iz kojeg se donje granice računaju pomoću trokutne nejednakosti [21].

Za točne donje granice moguće je unaprijed izračunati udaljenosti između svih parova čvorova, ali taj postupak je preskup što se tiče vremena predprocesuiranja i zauzeća memorije. Stoga se odabire samo mali podskup $\mathcal{L} \subset V$ simbola, te se računa točna tablica

udaljenosti za sve čvorove v prema ili od svakog simbola $\ell \in \mathcal{L}$. Broj simbola je obično skup između 16 i 64. Udaljenosti na grafu G formiraju mjernu jedinicu, stoga za instance trokutne nejednakosti vrijedi

$$daljina(v,t) + daljina(t,\ell) \geq daljina(v,\ell) \quad i \tag{4.5}$$

$$daljina(\ell,v) + daljina(v,t) \geq daljina(\ell,t). \tag{4.6}$$

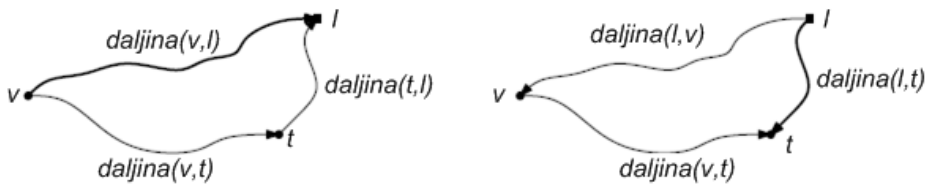
Ovdje daljina(u,v) za dva čvora u,$v \in V$ predstavlja udaljenost najkraćeg puta od u do v u G-u. Vizualni prikaz jednadžbi prikazan je na slici 4.2. Rješenjem obje jednažbe za daljinu(v,t) dobije se izvediva funkcija donje granice:

$$\pi_\ell(v) := max\{daljina(v,\ell) - daljina(t,\ell), daljina(\ell,t) - daljina(\ell,v)\} \leq daljina(v,t). \tag{4.7}$$

Najbolja donja granica π može se dobiti korištenjem simbola koji donosi najveću donju granicu, prema jednadžbi:

$$\pi(v) := max_{\ell \in \mathcal{L}} max\{daljina(v,\ell) - daljina(t,\ell), daljina(\ell,t) - daljina(\ell,v)\}. \tag{4.8}$$

Sa vrijednostima daljina(v,ℓ) i daljina(ℓ,v) unaprijed izračunatim za svaki simbol $\ell \in \mathcal{L}$ i svaki čvor $v \in V$, smanjeni trošak grafa G_π se tada računa implicitno mijenjajući ključ čvora v u prioritetnom redu u vrijednost daljina(s,v) + $\pi(v)$ [21].



a) Vizualni prikaz jednadžbe 3.11 b) Vizualni prikaz jednadžbe 3.12

Slika 4.2: Primjer primjene trokutne nejednakosti.

Slika 4.2 pokazuje primjenu trokutne nejednakosti na temelju jednadžbi 3.11 (slika

a) i 3.12 (slika b). Duži put je prikazan tanjim grafom, a kraći put je prikazan debljim grafom [21].

Predprocesuiranje se obavlja u dva koraka. Prvo se "dobar" skup simbola $\mathcal{L}$ odabire iz V, a potom se računa tablica udaljenosti za $\mathcal{L}$. Računanje tablice udaljenosti je lakši dio predprocesuiranja i može se riješiti koristeći *Dijsktra* algoritam, dok je odabir "dobrog" skupa znakova teži dio. Dobar simbol treba po mogućnosti donijeti visoku donju granicu tijekom izvršavanja upita za što više *s-t* upita i što više v čvorova [21].

Algoritam upita. Jedina promjena u primjeni algoritma upita u odnosu na algoritam 3 (slika 3.9) je da se umjesto korištenja vrijednosti daljina$_s(v)$ kao ključeva u prioritetnom redu ovdje koristi funkcija troška smanjenja udaljenosti daljina$_s(v) + \pi(v)$. Međutim, po jednadžbi 3.13 može se zaključiti da računanje donje granice uz poštivanje svih simbola producira previše opterećenja prilikom izvođenja upita. Zbog toga se jednadžba 3.11 izvodi samo na podskupu $\mathcal{L}_{aktivan} \subseteq \mathcal{L}$ aktivnih simbola. Obično se ograniči kardinalnost od $\mathcal{L}_{aktivan}$ na vrijednost 2. Odabir aktivnih simbola ovisi o upitu i određuje se na početku koristeći $\pi_\ell(s)$. Nadalje, svakom k iteracijom algoritma ažurira se skup aktivnih simbola tako da se ponovno provjerava koji simboli daju najbolju donju granicu za trenutno smještene čvorove [21].

Vremenska ovisnost. Adaptacija ALT algoritma u vremenski ovisnim mrežama zahtjeva samo prilagodbu predprocesuiranja algoritma. Upit se ne mijenja (osim onog dijela u kojem se ocjenjuju težine bridova na temelju vremena odlaska τ) dok god potencijalna funkcija nije izvediva za sva moguća doba dana τ. Iz tog razloga tablica udaljenosti za skup $\mathcal{L}$ odabranih simbola računa se na grafu donje granice $\underline{G}$. Tako vrijednosti daljina(v,ℓ) i daljina(ℓ,v) daju samo udaljenost od v do ℓ (i obrnuto) za najbolje moguće vrijeme τ tokom dana. Međutim, s obzirom da za sva druga vremena τ' u danu vrijedi daljina(v,ℓ,τ') $\geq$ daljina(v,ℓ) i daljina(ℓ,v,τ') $\geq$ daljina(ℓ,v), izračunate udaljenosti mogu se promatrati kao donje granice donjih granica. Ovisno o razlikama između donjih i gornjih granica funkcija težina bridova u G-u, kvaliteta prethodno izračunatih udaljenosti može biti loša. Zbog toga ALT algoritam nije toliko dobar u vremenski ovisnim mrežama kao u vremenski neovisnim

mrežama [21].

Prilagodba multimodalnom rutiranju funkcionira tako da u multimodalnom grafu **G** za svaki jezik L i svaki s-t put P koji zadovoljava L mora vrijediti:

$$duljina(P) \geq duljina(P_{\text{uni}}) \tag{4.9}$$

gdje je duljina(P_{uni}) duljina unimodalnog najkraćeg s-t puta u **G**-u. Drugim riječima, uvođenje ograničenja u najkraće puteve zapravo nikad ne donosi kraće puteve. Zbog toga se ALT algoritam može adaptirati bez ikakvih ograničenja. Predprocesuiranje se obavlja na $\underline{\mathbf{G}}$-u donoseći važeće donje granice za tablice udaljenosti. Te donje granice su općenito lošije jer se za daljinu(v,ℓ) (i obrnuto) ne koristi samo donja granica s obzirom na vrijeme polaska nego i s obzirom na modove transporta. Dobra strana je ta što algoritam upita nije potrebno prilagođavati, nastavlja se korištenje funkcije smanjenog troška kao ključa prioritetnog reda kao što se radi i u unimodalnoj verziji algoritma [21].

ALT se može lako prilagoditi vremenski ovisnom i multimodalnom rutiranju. Međutim, primjena ALT algoritma kao jednosmjerne tehnike ubrzanja ne donosi neka velika ubrzanja. Izvođenje dvosmjernog ubrzanja sa 2 neovisna ALT algoritma može dovesti do krivih rezultata. Glavni razlog tomu je što *forward* i *backward* pretraga ne koriste iste smanjene troškove na bridovima. Kao posljedica, čvor u kojem se dvije pretrage sastaju nije nužno i najkraći put. ALT algoritam je najbolje koristiti kao jednosmjernu tehniku ubrzanja zbog lakog prilagođavanja multimodalnom vremenski ovisnom rutiranju bez obzira na to što ne donosi velika ubrzanja.

4.4.3 *Contraction*

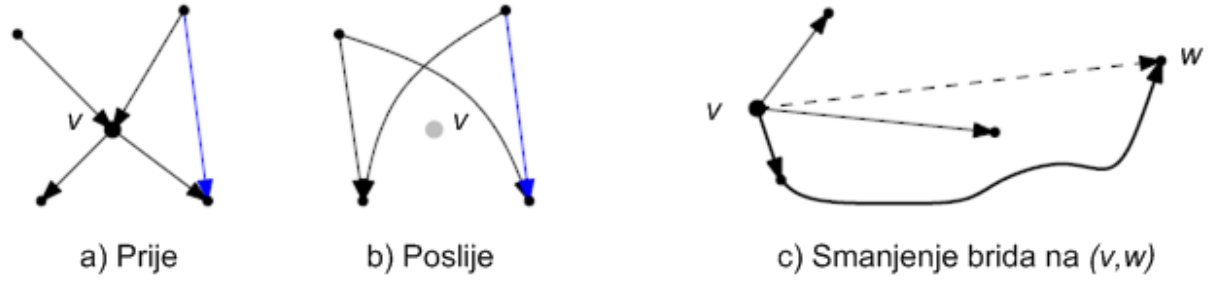
Treći primjer tehnika ubrzanja je *contraction*, odnosno tehnika skupljanja. Pod skupljanjem se misli na neku vrstu smanjena što se tiče veličine grafa, odnosno broja čvorova i veličine bridova. Tehnika skupljanja se može dobro kombinirati sa dvosmjernom pretragom i pretragom usmjerenom cilju. Također je korisna za multimodalno rutiranje jer podrazumijeva samo marginalna ograničenja na konačnom automatu tijekom upita. Postoje dva segmenta

ove tehnike ubrzanja, a to su smanjenje čvorova (eng. *node-reduction*) i smanjenje broja bridova (eng. *edge-reduction*). Kad se oba segmenta apliciraju samo jednom, dobije se mali graf naziva jezgra (*core*). Jezgreni graf je značajno manji od originalnog grafa. Upit se na jezgrenom grafu može smanjiti na *many-to-many* pretragu najkraćeg puta u kojem se druga tehnika ubrzanja može primjeniti ortogonalno. Na taj način funkcioniraju algoritmi ove tehnike ubrzanja. Druga prednost primjene tehnika ubrzanja na manjim grafovima je što se vrijeme predprocesuiranja i dodatno potreban prostor mogu značajno smanjiti [21].

Node-reduction. Kad se smanjenje čvorova jednom primjeni, on podijeli graf na dva dijela: jezgreni dio i komponentu. Na početku se za sve čvorove smatra da pripadaju jezgri. Tada se uzastopno zaobilaze čvorovi na način da se izvlače iz jezgre i na kraju pripadaju komponenti grafa dok nijedan čvor više ne bude zaobilazan.

Neka $G = (V,E)$ bude graf, a $v \in V$ bude čvor koji pripada jezgri grafa. Tada, za svaki dolazni brid $e_{ul} = (u, v)$ i svaki odlazni brid $e_{izl} = (v, w)$ unosi se brid prečaca $e := (u,w)$ od čvora u do čvora w. Težina brida e je postavljena na $w(e) := w(e_{ul}) + w(e_{izl})$. Ako bi umetanje brida e dovelo do većeg broja bridova od u do w postavlja se težina već postojećeg brida e' = (u,w) na minimum $w(e') := \min\{w(e),w(e')\}$. Na kraju se čvor v i njegovi slučajni bridovi brišu iz grafa. Smanjenje broja čvorova održava točne udaljenosti između dva proizvoljna čvora jezgre. Slika 4.3 prikazuje operaciju zaobilaska. Dobiveni jezgreni graf ima definiciju $G_{jezgra} = (V_{jezgra}, E_{jezgra})$, dok se komponenta definira kao $G_{komp} = (V_{komp}, E_{komp})$ gdje je $V_{komp} := V / V_{jezgra}$ i $E_{komp} = E / E_{jezgra}$ [21].

Edge-reduction. Izvođenje smanjenja bridova potencijalno ubacuje mnogo prečaca u jezgru što može biti nepotrebno za očuvanje točnih udaljenosti. Smanjenje bridova funkcionira na sljedeći način: za svaki čvor jezgre $v \in V_{jezgra}$ i za svaki brid $e = (v,w)$ izvršava se upit za najkraćim v-w putem. Ako je daljina$(v,w) < w(e)$ (što je uobičajeno za prečace), brid e se odstranjuje iz jezgre. Sada su istovremeno izračunate vrijednosti daljina(v,w) za sve susjede w i v, te se može nastaviti sa odstranjivanjem svih nepotrebnih odlaznih bridova čvora v. Primjenom smanjenja bridova održavaju se točne udaljenosti između čvorova jezgre u G_{jezgra}. Slika 4.3 pod c prikazuje koncept smanjenja bridova u pogledu jednog brida (v,w) [21].



Slika 4.3: Primjer operacije zaobilaženja.

Slika 4.3 (a i b) prikazuje operaciju zaobilaženja tijekom smanjenja bridova. Za svaki par dolaznih i odlaznih bridova na čvoru v umetne se prečac. Ako je prečac e već sadržan u grafu (plavi brid), težina na prečacu je postavljena na minimalnu težinu od e i na težinu brida koji bi se ubacio na to mjesto. Slika c pokazuje smanjenje brida, deblje označen put od v do w je kraći od brida $e = v,w$, stoga se e može obrisati (isprekidana crta) [21].

Predprocesuiranje tehnike skupljanja sastoji se od primjene smanjenja broja čvorova sve dok niti jedan čvor ne bude zaobilazan. Nakon toga se obavlja smanjenje bridova na jezgrenom grafu G_{jezgra}. Zbog jednostavnosti neka algoritam upita koristi samo jedan graf. Graf $G := G_{jezgra} \cup G_{komp}$ dobiva se ujedinjavanjem jezgrenog grafa sa komponentom. Jezgreni čvorovi označeni su sa zastavicom [21].

Algoritam upita. Tehnika skupljanja zahtjeva složeniji algoritam upita. Kao ulaz koristi se dobiveni graf $G = (V,E)$ sa određenim jezgrenim čvorovima. Za s-t upit algoritam radi u dvije faze. Prva faza djeluje na komponentnom dijelu grafa, dok druga faza djeluje samo na jezgri. Prva faza instancira *bi-directional* pretragu na komponenti od G-a. To se postiže tako da se ne opuštaju bridovi koji se nalaze u jezgri (npr. bridovi $e = (u,v)$ za koje u i v imaju postavljene zastavice jezgre). Na ovaj način se zapravo smještaju jezgreni čvorovi, a pretraga se zaustavlja čim se ustanovi da su s ili t jezgreni čvorovi (*forward* i *backward* pretraga se zaustavlja trenutno). Skup S jezgrenih čvorova koji su pogođeni *forward* pretragom zove se skup jezgrenih-ulaznih-čvorova (eng. *core-entry-nodes*), dok se skup T jezgrenih čvorova koji su pogođeni *backward* pretragom zove skup jezgrenih-izlaznih-čvorova (eng. *core-exit-nodes*). Prva faza završava ako se ispuni jedan od sljedećih

uvjeta.

1. Prioritetni redovi *forward* i *backward* pretrage su prazni.

2. Pronađen je *s*-*t* put za kojeg vrijedi

$$duljina(P) \leq daljina(s, v_{ulaz}^{min}) + daljina(v_{izlaz}^{min}, t) \tag{4.10}$$

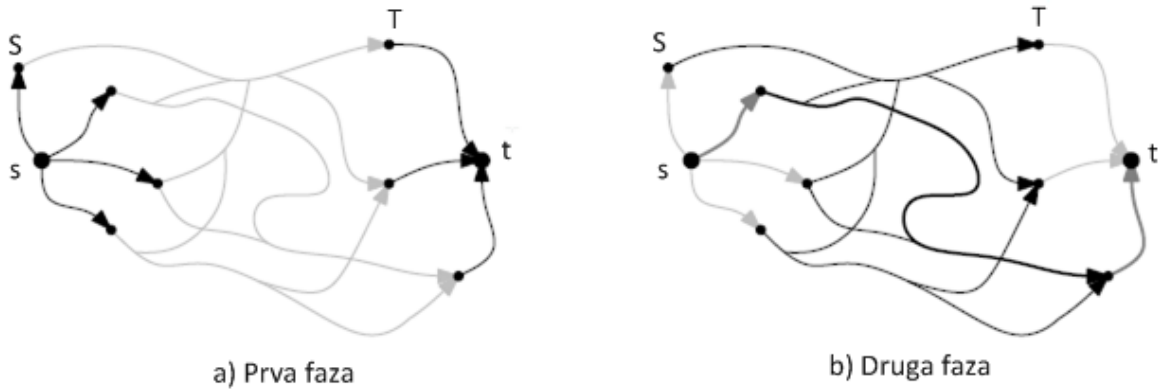
   gdje $v_{ulaz}^{min} \in S$ označava jezgrene-ulazne-čvorove sa minimalnom udaljenosti od s u G-u, a $v_{izlaz}^{min} \in T$ predstavlja jezgrene-izlazne-čvorove sa minimalnom udaljenosti od t u $\overleftarrow{G}$-u.

Ako se prva faza prekine zbog drugog uvjeta upit se može zaustaviti i prikazati izračunati put P kao najkraći *s*-*t* put. U ovom slučaju najkraći put koristi samo čvorove komponente [21].

Neka se pretpostavi da u prvoj fazi nijedan put P u komponenti nije pronađen. U tom slučaju druga faza algoritma se instancira sa *many-to-many S*-*T* upita opuštajući bridove koji su sadržani u jezgri. Međutim, *forward* i *backward* upiti se ponovno pune tako da se inicijalni ključevi postavljaju na udaljenosti (od s i t) izračunate u prvoj fazi. Algoritam koji se koristi u drugoj fazi nije specificiran, zbog toga tehnika skupljanja daje modularan dizajn koji omogućuje kombinaciju sa proizvoljnim tehnikama ubrzanja primjenjenim u jezgri. Završni *s*-*t* put se tada kombinira određivanjem minimalnog *s*-*T*-*t* puta $P_{s,v,t}$ za svaki čvor $v \in T$ gdje se duljina računa kao [21]:

$$duljina(P_{s,v,t}) := daljina(s, v) + daljina(v, t). \tag{4.11}$$

Slika 4.4 prikazuje algoritam upita. Prva faza (a) provodi dvosmjenu pretragu dok svi jezgreno-ulazni i jezgreno-izlazni čvorovi nisu dohvaćeni. U drugoj fazi (b) *S*-*T* upit se obavlja na jezgrenom grafu. Najkraći put se tada kombinira uzimajući iz svih *s*-*T*-*t* puteva onaj koji ima najmanju duljinu [21].



Slika 4.4: Prikaz algoritma rutiranja temeljenog na jezgri.

Vremenska ovisnost. Adaptacija metode skupljanja u vremenski ovisnom grafu dolazi sa nedostatkom dugog vremena predprocesuiranja te zauzimanja memorijskog prostora. Prilikom predprocesuiranja potrebno je procesirati funkcije vremena putovanja umjesto konstantnih težina. Stoga, prilikom ubacivanja prečaca $e = (u,w)$ za dva brida $e_{ul} = (u,v)$ i $e_{izl} = (v,w)$, funkcija f_e se računa kao $f_e := f_{e_{ul}} \oplus f_{eizl}$. Ako brid $e' = (u,w)$ već postoji, u grafu se nova funkcija na e' računa kao $f_{e'} := \min\{f_e, f_{e'}\}$. Ovdje operacija povezivanja i operacija spajanja mogu povećati broj interpolacijskih točki i na taj način dovesti do velikog zauzimanja memorijskog prostora. Što se tiče potrošnje vremena, više nije dovoljno izračunati jednostavno stablo najkraćeg puta da bi se odredilo za čvor v koje bridove prema njegovim susjedima w obrisati. Umjesto toga potrebno je izračunati profilne upite. Brid $e = (v,w)$ se tada jedino može obrisati ako vrijedi $f_w < f_e$, što znači da je izračunati put prema w brži od brida koji se koristi direktno za sva doba dana [21].

Prilagodba algoritma upita vremenskoj ovisnosti zahtjeva određene modifikacije. Pošto prva faza upita koristi dvosmjernu pretragu dolazi do istog problema kao i sa čistom dvosmjernom pretragom: točno vrijeme dolaska na čvor t ne zna se unaprijed. Stoga se algoritam modificira na sljedeći način: *forward* pretraga provodi vremenski upit sa vremenom polaska τ sa čvora s, dok je *backward* pretraga vremenski neovisna na povratnoj donjoj granici grafa $\overleftarrow{G}$. Ali uvjet zaustavljanja razlikuje se od *time-independent* algoritma. Prva faza se stoga zaustavlja ako se ispuni jedan od uvjeta:

1. *Forward* i *backward* uvjeti moraju biti prazni.

2. Ako je S skup jezgrenih-ulaznih, a T je skup jezgreno-izlaznih čvorova, pretraga se zaustavlja ako vrijedi $S \cap T \neq \oslash$, odnosno ako je bar jedan jezgreni čvor otkriven sa obje pretrage. U tom slučaju je započet standardan jednosmjeran *s-t* vremenski upit sa vremenom polaska τ (na cijelom G-u), a najkraći put je prikazan.

Ako se prva faza zbog prvog uvjeta ispuni inicijalizira se jezgreni algoritam na isti način kao i u vremenski neovisnoj verziji- sa skupovima S i T. Međutim, *forward* pretrazi je dopušteno da izađe iz jezgre na jezgreno-izlaznim čvorovima T. Drugim riječima, čim *forward* pretraga smjesti čvor $v \in T$, algoritmu je također dozvoljeno da opusti bridove izvan jezgre. Na taj način je najkraći *s-t* put pronađen direktno *forward* pretragom preko svih izlaznih čvorova T [21].

Prilagodba tehnike skupljanja multimodalnom rutiranju u svrhu boljeg razumijevanja objašnjena je u segmentima:

- *Node-reduction* - kad se umetne prečac *e = (u,w)* za vrijeme zaobilaženja čvora v, bridovi *(u,v)* i *(v,w)* povezani su u jedan brid. Ako se ovo gleda sa perspektive automata, prelazak duž jednog brida mreže podrazumijeva točno jedan prijenos u automatu. Ako su dva brida povezana u jedan brid automat tijekom izvođenja upita jedino može obaviti jedan prijenos gdje bi se u grafu obavila dva. Stoga se za zaobilaženje čvorova ne može koristiti samo jedan automat za izvođenje upita. Međutim, taj problem se može riješiti restrikcijom prilikom skupljanja čvorova v tako da moraju zadovoljavati sljedeće uvjete: za sve dolazne bridove e_{ul} prema v i za sve izlazne bridove e_{izl} od v mora vrijediti da imaju iste oznake bridova. U tom slučaju samo se ujedinjuje prijenos koji ima istu oznaku (simbol u automatu).

- *Edge-reduction* - smanjenje bridova mora biti promjenjeno zbog istog razloga kao što smanjenje broja čvorova mora biti ograničeno. Ako je v čvor koji pripada određenoj mreži tipa T, tada algoritam stabla najkraćeg puta koje raste od v mora biti ograničeno na tu određenu mrežu T, što znači da je algoritmu najkraćeg puta dozvoljeno da samo smjesti čvorove označene sa T. U ovom slučaju je garantirano da se najkraći put od v do jednog od njegovih susjeda w sastoji samo od bridova sa istom oznakom kao

e. Tada je moguće odstraniti brid *e* iz grafa. Ako vrijedi da oznaka(v) $\neq$ oznaka(w), w nije nikad smješten algoritmom najkraćeg puta i tada *e* nije nikad odstranjen. Ako se izvodi sa ovom restrikcijom, smanjenje bridova ima točno suprotan efekt od smanjenja broja čvorova. Odstranjivanjem brida *e* od v prema w održava se udaljenost od v prema w točnom, ali povećava se broj sljedećih prijenosa označenih sa oznaka(*e*) u automatu s obzirom na segment od v do w na putu.

- Izvođenje upita - Što se tiče algoritma upita čvorovi se zamjenjuju sa produkcijskim čvorovima koji su izračunati implicitno. Odatle prva faza daje dva skupa S i T produkcijskih čvorova koji su ponovno ubačeni u *forward* i *backward* redove sa pogodnim udaljenostima. Za algoritam druge faze na jezgri može se odabrati *many-to-many* multimodalni algoritam rutiranja. Proširenja dvosmjernog rutiranja u multimodalnim mrežama također vrijede ovdje za prvu fazu izvršavanja upita [21].

Tehnika skupljanja je osnovni dio najefikasnijih tehnika ubrzanja na unimodalnim mrežama danas. Algoritam upita je kompleksniji nego kod drugih osnovnih primjera tehnika ubrzanja. Tehnika skupljanja se može implementirati sa vremenskom ovisnosti, ali sa penalom zauzimanja više memorije i vremena prilikom predprocesuiranja. Korištenje u multimodalnim mrežama je moguće, ali potrebna je pažnja pri odabiru čvorova i bridova za zaobilaženje i brisanje.

5 Prikupljanje podataka za multimodalno planiranje putovanja

Glavni cilj *Web* aplikacije za planiranje putovanja je da izlazna vrijednost, odnosno generirana putanja na *Google* karti bude što preciznija. Stoga je potrebno sakupiti točne i ažurne podatke vezane za planiranje putovanja i iste na pravilan način spremiti u bazu podataka aplikacije. Ti podaci mogu biti informacije o autoprijevozniku, lokacije i vremena dolaska na stanice na kojima se razvoze putnici u prometu (ako se radi o npr. autobusu ili tramvaju), opis ruta kojima se razvoze putnici i sl. Praktično rješenje rada obuhvaća više modova putovanja, stoga je potrebno prikupiti i unijeti podatke vezane za autobusni, brodski i zračni prijevoz. U bazi podataka *Web* aplikacije se prikupljeni podaci spremaju u GTFS (eng. *General Transit Feed Specification*) formatu. Također postoji više načina prikupljanja navedenih podataka, automatski, polu-automatski i ručno. Ti će načini uz GTFS u nastavku biti detaljnije objašnjeni.

5.1 GTFS

GTFS definira zajednički format za raspored javnog prijevoza koji je povezan sa geografskim informacijama. Osmišljen je od strane Bibiane McHugh, IT menadžerice u *Portland TriMetMAX* autoprijevozničkoj agenciji. Autoprijevoznici objavljuju karakteristike svojih usluga unutar skupa tekst (*.txt*) datoteka (*GTFS feed*-a). Svaka .txt datoteka sadrži informacije o autoprijevozniku (*agency*), stanicama (*stops*), vremenu dolaska na pojedinu stanicu (*stop_times*), rutama (*routes*), rasporedu vožnje (*schedules*), kalendaru(*calendar*), cijenama (*fares*) i frenkvencijama vožnje (*frequencies*). Te datoteke pomažu pri snabdjevanju tranzitnih informacija korisnicima na *Google maps Web* stranicama. Mnoge velike tranzitne agencije su napravile svoje GTFS podatke javno dostupnim, kao što su *TriMetMAX (Portland)*, *Barth (San Francisco)*, *DART (Dallas)* i tako dalje [18].

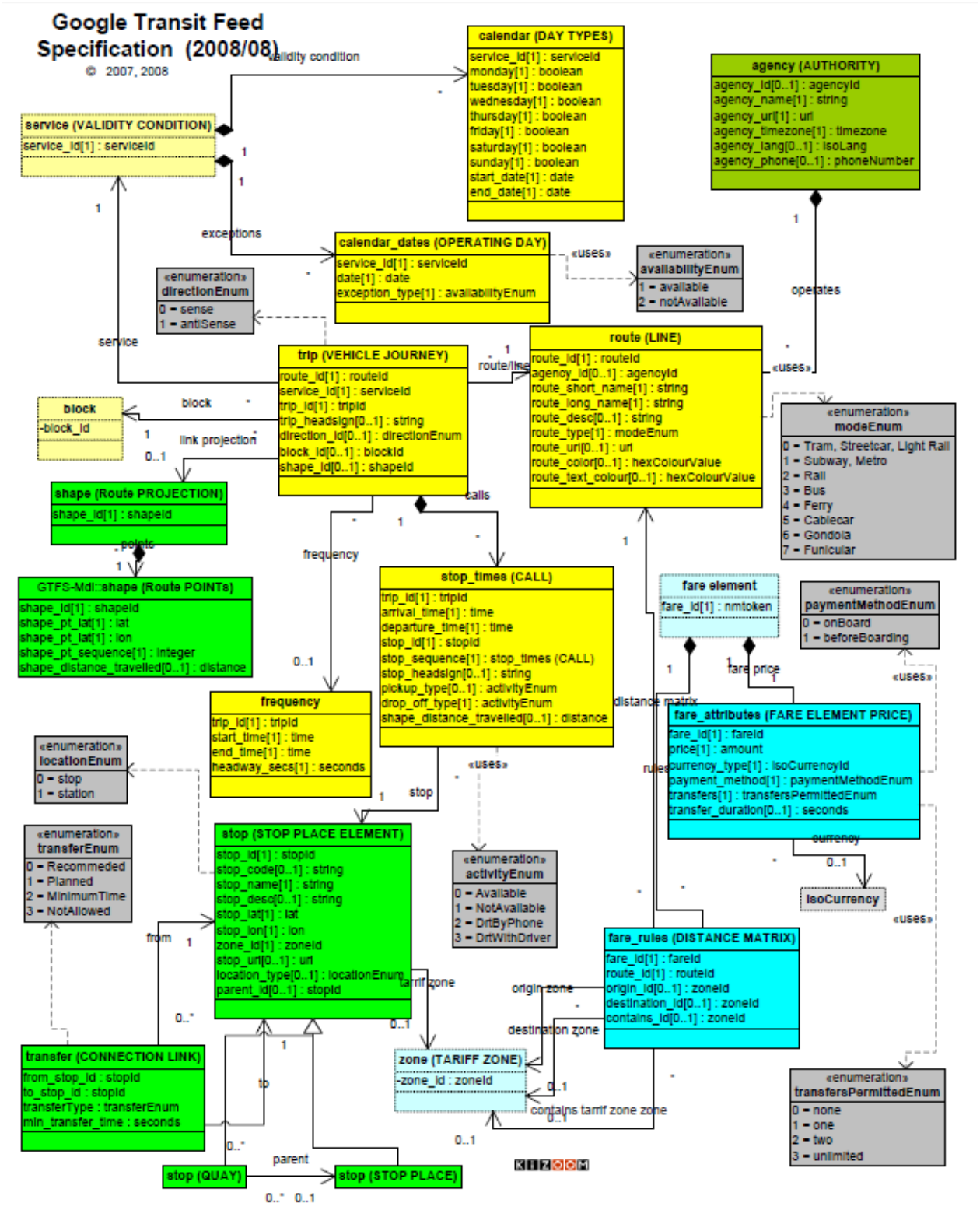
5.1.1 Nastanak i razvoj GTFS-a

Kako je GTFS postao zajednički jezik za tranzitne podatke je zanimljiva priča koja počinje u Portlandu u SAD-u. Nakon putovanja u inozemstvo ljeti 2005. godine, Bibiana McHugh je bila frustrirana jer nije mogla pristupiti tranzitnim informacijama na *Web* aplikacijama za prikaz geografskih karata kao što je *Mapquest*. Zbog toga nije mogla planirati tranzitno putovanje jednako lako kao i putovanje osobnim vozilom. Kad se vratila u SAD poslala je upit tvrtkama *MapQuest, Yahoo* i *Google* da uključe tranzitne podatke u svoje kartografske usluge, a tvrtka u kojoj je ona radila, *TriMetMAX*, bi bio partner u realizaciji tog projekta. Jedino je *Google* odgovorio na upit, a programski inženjer *Google*-a, Chris Harrelson, je implementirao tranzitne podatke *TriMetMAX*-a u *Google maps* uslugu te je tako nastao *Google Transit Trip Planner*. *TriMetMAX* je surađivao s *Google*-om u pripremi *TriMetMAX* skupa podataka u obliku koji će funkcionirati sa *Google Maps* uslugom. "Tranzitni podaci su iznimno složeni", rekla je Bibiana McHugh. "Sastoje se od vremenskog elementa i prostornog elementa i potrebna je relacijska baza podataka koja će omogućiti upravljanje tim podacima." Također je dodala, "Mnogo agencija ima taj strah da će ti podaci biti pogrešni ili se neće iskoristiti točno." *TriMetMAX* je bio vrlo proaktivan u upravljanju svojim tranzitnim podacima, zbog toga je *Google* naknadno detaljnije posložio tranzitne podatke sa *Google maps* uslugom. Na taj način je nastala GTFS specifikacija. *Google Transit Trip Planner* je pokrenut 7. prosinca 2005. godine, a većinu prve godine njegovog postojanja koristila ga je samo tvrtka *TriMetMAX*. Nakon uvida u beneficije dostupnosti svojih usluga na *Google* kartama, ovu specifikaciju su počele koristiti tranzitne agencije drugih američkih gradova kao što su *Eugene, Honolulu, Pittsburgh, Seattle i Tampa*. Trenutno *Google Maps* ima ugovor s više od 100 tranzitnih operatora u SAD-u, a više od 400 diljem svijeta [22].

5.1.2 Relacijska reprezentacija GTFS-a

U nastavku su objašnjene pojedine datoteke GTFS specifikacije (GTFS *feed*) i polja od kojih se sastoje. Te datoteke su međusobno spojene određenim relacijama koje su prikazane UML dijagramom na slici 5.1 [20]. Dijagram prikazuje originalnu GTFS specifikaciju osmišljenu od strane *Google*-a koja sadrži ukupno sve elemente koji se mogu pojaviti u nekom GTFS

feed-u. U praktičnom dijelu rada se ne koriste svi elementi nego samo oni koji su potrebni za potpunu funkcionalnost zamišljene *Web* aplikacije za izradu planova putovanja. Također je potrebno na temelju odabranih GTFS datoteka kreirati bazu podataka koju koristi *Web* aplikacija. Ta baza podataka sadržava podatke koji predstavljaju relacijski prikaz odabranih GTFS datoteka. Na taj način je svaka *.txt* datoteka sadržana u jednoj tablici baze, a zapisi u tablici odgovaraju onima iz GTFS datoteke.



Slika 5.1: GTFS UML dijagram.

5.2 GTFS datoteke

Ranije je spomenut GTFS *feed*. Već je navedeno da se sastoji od skupa *.txt* datoteka koji su spremljeni u *.zip* datoteku, a svaka *.txt* datoteka modelira poseban aspekt tranzitnih informacija. Postoji poveći broj *.txt* datoteka koje čine GTFS *feed*, ali obvezne su sljedeće: *agency.txt, stops.txt, routes.txt, trips.txt, stop_times.txt i calendar.txt*. Svaka od navedenih datoteka se sastoji od više polja, a svako polje ima svoje značenje u pojedinom zapisu. U nastavku su opisana polja za pojedinu datoteku.

agency.txt:

- *agency_id* - neobvezno polje koje jedinstveno označava tranzitnu agenciju (prijevoznika). *GTFS feed* može sadržavati podatke od više od jednog prijevoznika i u tom slučaju je unos podataka u ovo polje obvezno, suprotno tome ako se radi samo o jednom prijevozniku.
- *agency_name* - obvezno polje koje sadrži potpuno ime prijevoznika. *Google maps* prikazuje ime prijevoznika.
- *agency_url* - obvezno polje koje sadrži *Web* adresu prijevoznika.
- *agency_timezone* - obvezno polje - sadrži vremensku zonu u kojoj se prijevoznik nalazi. Ako je više prijevoznika specificirano u GTFS *feed*-u, svaki mora imati istu vremensku zonu.
- *agency_lang* - neobvezno polje - sadrži šifru primarnog jezika koji koristi tranzitna agencija.
- *agency_phone* - neobvezno polje koje se sastoji od telefonskog broja prijevoznika.
- *agency_fare_url* - neobvezno polje koje specificira *Web* adresu internet stranice na kojoj putnik može kupiti kartu za putovanje tim prijevoznikom [7].

stops.txt:

- *stop_id* - obvezno polje - jedinstveno identificira stanicu javnog prijevoza. Više različitih ruta može koristiti istu stanicu.
- *stop_code* - neobvezno polje - sadrži kratki string ili broj koji jedinstveno identificira stanicu. Koristi se kod sustava pružanja informacija temeljenih na telefonskim uređajima.
- *stop_name* - obvezno polje - sadrži ime stanice. Poželjno je koristiti ime koje će razumjeti lokalno stanovništvo i turisti.
- *stop_desc* - neobvezno polje - sadrži opis stanice.
- *stop_lat* - obvezno polje koje sadrži *WGS84* geografsku širinu stanice.
- *stop_lon* - obvezno polje koje sadrži *WGS84* geografsku dužinu stanice.
- *zone_id* - neobvezno polje - definira platnu zonu za pojedinu stanicu.
- *stop_url* - neobvezno polje - sadrži URL *Web* stranice koja sadrži podatke o pojedinoj stanici.
- *location_type* - neobvezno polje - pruža informacije o vrsti stanice, je li to kolodvor ili neka manja postaja.
- *parent_station* - neobvezno polje - ako se neka stanica nalazi unutar neke veće postaje odnosno kolodvora, tada ovo polje identificira taj kolodvor kao roditeljsku stanicu.
- *stop_timezone* - neobvezno polje - sadrži vremensku zonu u kojoj se određena stanica nalazi.
- *wheelchair_boarding* - neobvezno polje koje identificira je li stanica omogućuje ulazak u vozila gradskog prijevoza putnicima s invalidskim kolicima [7].

routes.txt:

- *route_id* - obvezno polje - jedinstveno identificira pojedinu rutu.
- *agency_id* - neobvezno polje - definira prijevoznika za određenu rutu. Ovaj podatak se povlači iz *agency.txt* datoteke.
- *route_short_name* - obvezno polje - sadrži kratko ime rute.
- *route_long_name* - obvezno polje - sadrži dugo odnosno potpuno ime rute.
- *route_desc* - neobvezno polje - sadrži opis rute.
- *route_type* - obvezno polje - sadrži opis tipa prijevoza na određenoj ruti (tramvaj, metro, željeznica...).
- *route_url* - neobvezno polje - sadrži URL *Web* stranice koja opisuje rutu.
- *route_color* - neobvezno polje - u sustavima u kojima je omogućeno označavanje pojedinih ruta ovo polje dodjeljuje određenu boju ruti.
- *route_text_color* - neobvezno polje - dodjeljivanje određene boje tekstu koji opisuje pojedinu rutu [7].

trips.txt:

- *route_id* - obvezno polje - jedinstveno identificira pojedinu rutu.
- *service_id* - obvezno polje - jedinstveno identificira skup datuma u kojima je pojedina prijevozna usluga dostupna za jednu ili više ruta.
- *trip_id* - obvezno polje - jedinstveno identificira *trip* (put).
- *trip_headsign* - neobvezno polje - sadrži tekst koji se pojavljuje na znaku koji putnicima identificira destinaciju puta .
- *trip_short_name* - neobvezno polje - sadrži kratko ime puta koje se nalazi u rasporedima i na znakovima, a služi da bi ga se identificiralo putnicima.

- *direction_id* - neobvezno polje - sadrži binarnu vrijednost koja predstavlja smjer određenog puta, 0-put u jednom smjeru, 1-put u drugom, suprotnom smjeru.
- *block_id* - neobvezno polje - identificira blok kojem pojedini put pripada.
- *shape_id* - neobvezno polje - sadrži *Id* koji definira *shape* (oblik) puta. Ova vrijednost se referencira na *shape_id* polje u datoteci *shapes.txt*.
- *wheelchair_accessible* - neobvezno polje - sadrži tri moguće vrijednosti: 0-nema informacija o mogućnostima korištenja invalidskih kolica za put, 1-vozilo koje se koristi za put može uključivati najmanje jedna invalidska kolica, 2-vozilo na putu ne može uključivati niti jedna invalidska kolica.
- *bikes_allowed* - neobvezno polje - također sadrži tri moguće vrijednosti - 0-nema informacija o mogućnostima korištenja bicikla za put, 1-vozilo koje se koristi za put može uključivati najmanje jedan bicikl, 2-vozilo na putu ne može uključivati niti jedan bicikl [7].

stop_times.txt:

- *trip_id* - obvezno polje - identificira put. Ova vrijednost se referencira na *trips.txt* datoteku.
- *arrival_time* - obvezno polje koje specificira vrijeme dolaska na određenu stanici za određeni put na ruti.
- *departure_time* - obvezno polje koje specificira vrijeme polaska sa određene stanice za određeni put na ruti.
- *stop_id* - obvezno polje - jedinstveno identificira stanicu (stajalište).
- *stop_sequence* - obvezno polje - identificira redoslijed stajališta za određeni put.
- *stop_headsign* - neobvezno polje koje se sastoji od teksta koji se pojavljuje na oznaci koja identificira putnicima destinaciju puta.

- *pickup_type* - neobvezno polje - određuje je li putnik ukrcan na određenoj stanici kao dio normalnog rasporeda ili je ukrcan pod posebnim okolnostima na određenoj stanici (posebnim zahtjevom).
- *drop_off_type* - neobvezno polje - određuje je li putnik iskrcan na određenoj stanici kao dio normalnog rasporeda ili je iskrcan pod posebnim okolnostima na određenoj stanici (posebnim zahtjevom).
- *shape_dist_traveled* - neobvezno polje - predstavlja realnu udaljenost puta u kilometrima. Ova informacija omogućuje planeru puta da odredi koliki dio oblika (*shape*) određenog puta treba nacrtati prilikom prikaza puta na mapi.
- *timepoint* - neobvezno polje čija vrijednost može odrediti je li vrijeme određenog dolaska ili odlaska sa stajališta strogo poštovano od strane vozila ili su ta vremena okvirna [7].

calendar.txt:

- *service_id* - obvezno polje koje jedinstveno identificira skup datuma kad je autoprijevozna usluga dostupna za jednu ili više ruta.
- *monday* - obvezno polje koje sadrži binarnu vrijednost koja određuje je li autoprijevozna usluga dostupna svaki ponedjeljak, 1-usluga je dostupna svakim ponedjeljkom u određenom razdoblju, 0-usluga nije dostupna ponedjeljkom u određenom razdoblju.
- *tuesday* - obvezno polje koje sadrži binarnu vrijednost koja određuje je li autoprijevozna usluga dostupna svaki utorak, 1-usluga je dostupna svaki utorak u određenom razdoblju, 0-usluga nije dostupna utorkom u određenom razdoblju.
- *wednesday* - obvezno polje koje sadrži binarnu vrijednost koja određuje je li autoprijevozna usluga dostupna svaku srijedu, 1-usluga je dostupna svaku srijedu u određenom razdoblju, 0-usluga nije dostupna srijedom u određenom razdoblju.

- *thursday* - obvezno polje koje sadrži binarnu vrijednost koja određuje je li autoprijevozna usluga dostupna svaki četvrtak, 1-usluga je dostupna svaki četvrtak u određenom razdoblju, 0-usluga nije dostupna četvrtkom u određenom razdoblju.

- *friday* -obvezno polje koje sadrži binarnu vrijednost koja određuje je li autoprijevozna usluga dostupna svaki petak, 1-usluga je dostupna svaki petak u određenom razdoblju, 0-usluga nije dostupna petkom u određenom razdoblju.

- *saturday* - obvezno polje koje sadrži binarnu vrijednost koja određuje je li autoprijevozna usluga dostupna svaku subotu, 1-usluga je dostupna svaku subotu u određenom razdoblju, 0-usluga nije dostupna subotom u određenom razdoblju.

- *sunday* - obvezno polje koje sadrži binarnu vrijednost koja određuje je li autoprijevozna usluga dostupna svaku nedjelju, 1-usluga je dostupna svaku nedjelju u određenom razdoblju, 0-usluga nije dostupna nedjeljom u određenom razdoblju.

- *start_date* - obvezno polje koje određuje početni datum trajanja usluge prijevoza.

- *end_date* - obvezno polje čija vrijednost određuje završni datum trajanja usluge prijevoza. Završni datum je također uključen u razdoblje [7].

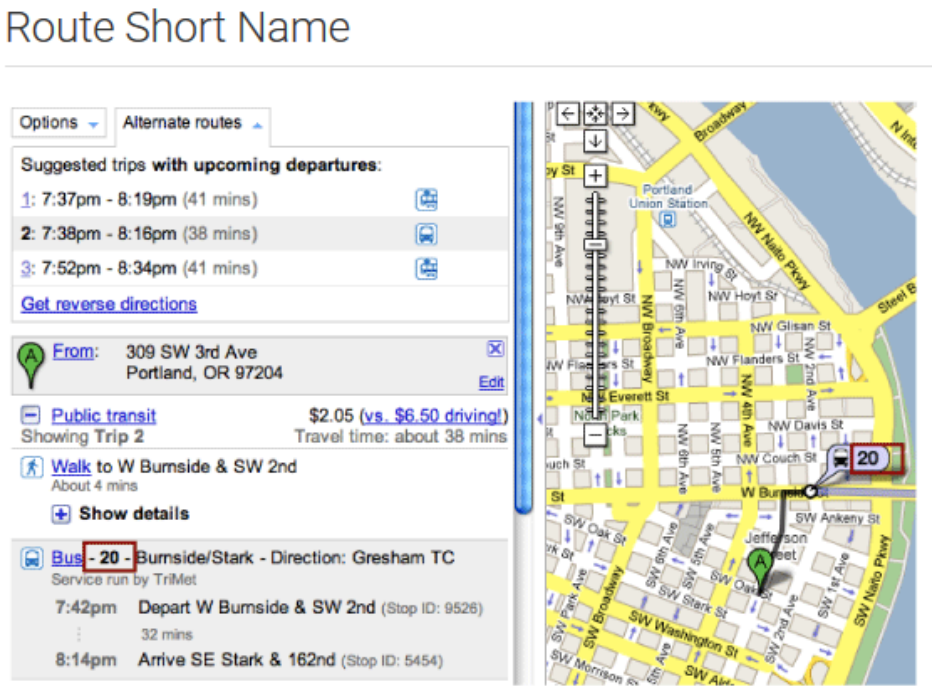
Bitno je naglasiti da pojedini prijevoznici uz navedene obvezne datoteke koriste i dodatne neobvezne ovisno o uslugama koje nude. Može se za primjer uzeti gradski prijevoznik Pariza, on za svoju GTFS specifikaciju koristi dodatne datoteke *calendar_dates.txt* i *transfers.txt*. *Calendar_dates.txt* im koristi da bi eksplicitno aktivirali ili onemogućili neke od usluga na određene datume, a *transfers.txt* za neka dodatna pravila pri kreiranju veza između ruta. Budimpeštanski gradski prijevoznik *Budapesti Közlekedési Központ* uz obvezne datoteke koristi i *shapes.txt* datoteku koja sadrži pravila za crtanje ruta na karti koje predstavljaju pravce kretanja prijevoznika. Portlandski *TriMet*, u kojem sve i počelo, uz navedene datoteke koristi i :

- *fare_attributes.txt* - informacije o cijenama za pojedine rute prijevoznika.

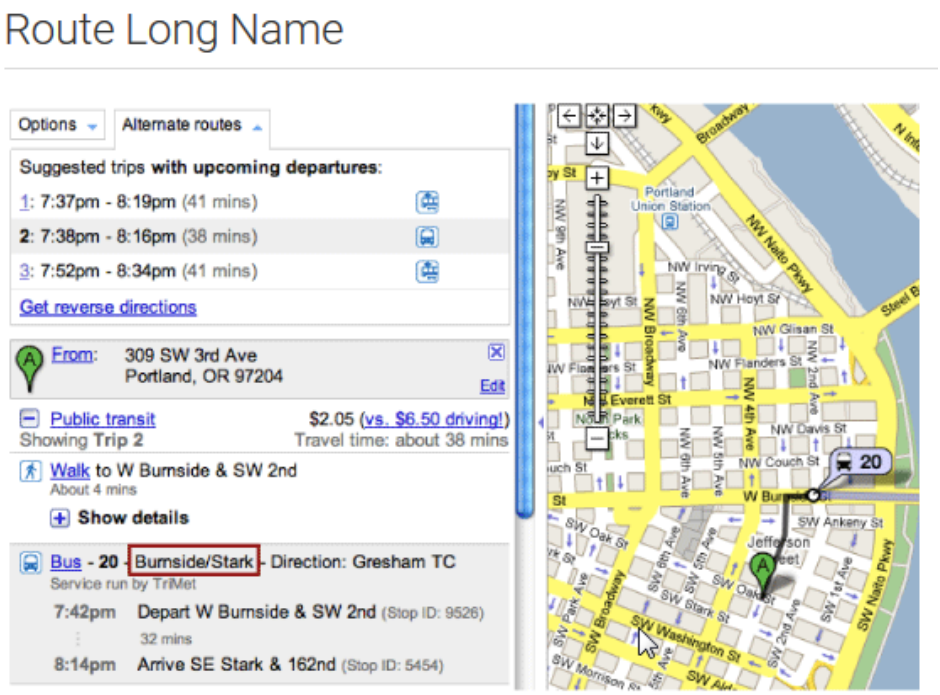
- *fare_rules.txt* - pravila za donošenje cijena pojedinih ruta prijevoznika.

- *feed_info.txt* - sadrži dodatne informacije o samoj GTFS specifikaciji, uključujući izdavača, verziju i informacije o roku upotrebe [7]

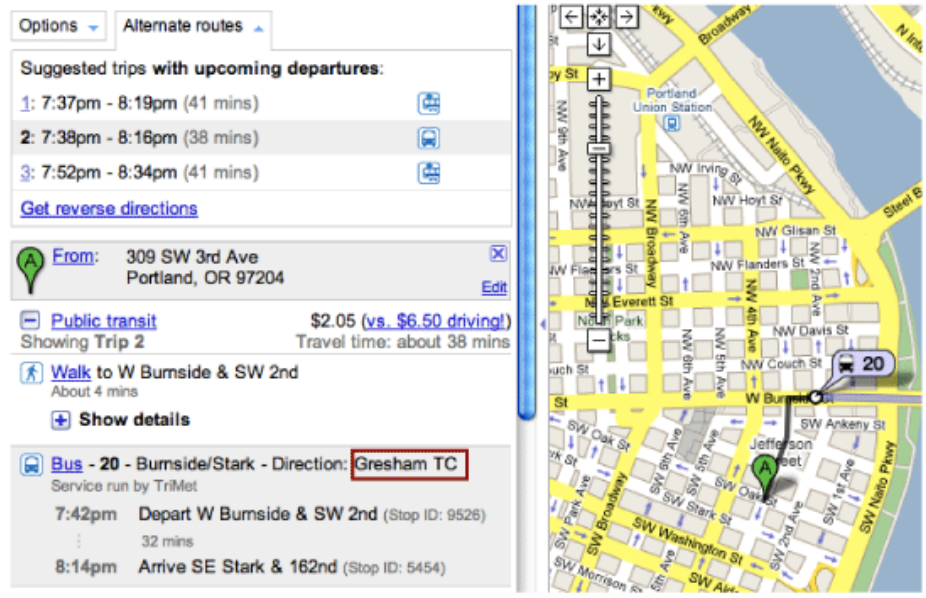
Ovi su podaci dostupni na stranicama *Google*-a, a navedeni prijevoznici su dobrovoljno pustili svoje GTFS podatke u javnost. Prijevoznik može po želji odlučiti želi li te podatke pustiti u javnost ili ih ostaviti u tajnosti u dogovoru sa *Google*-om. Ako se uzme za primjer zagrebački ZET, on je odlučio svoje GTFS specifikacije ne iznositi u javnost. Na slikama 5.2, 5.3, 5.4 i 5.5 vidljivi su primjeri kako se tranzitni podaci datoteke *routes.txt* prikazuju korisniku na *Google maps* sučelju [4]. Bitna polja na slikama su označena crvenim kvadratom.



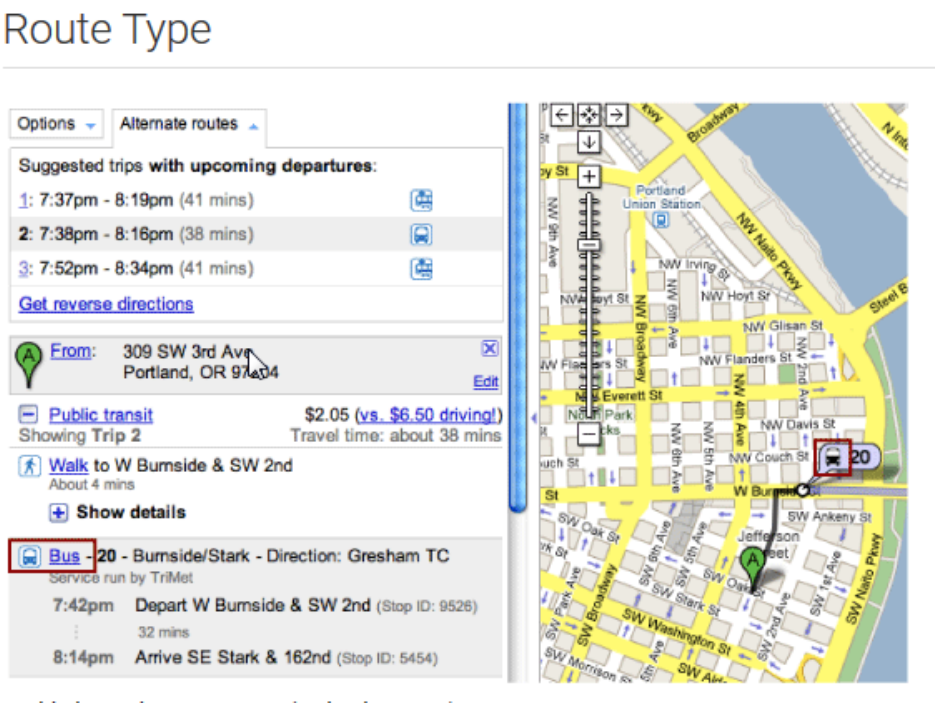
Slika 5.2: Prikaz polja *Route Short Name*.



Slika 5.3: Prikaz polja *Route Long Name*.



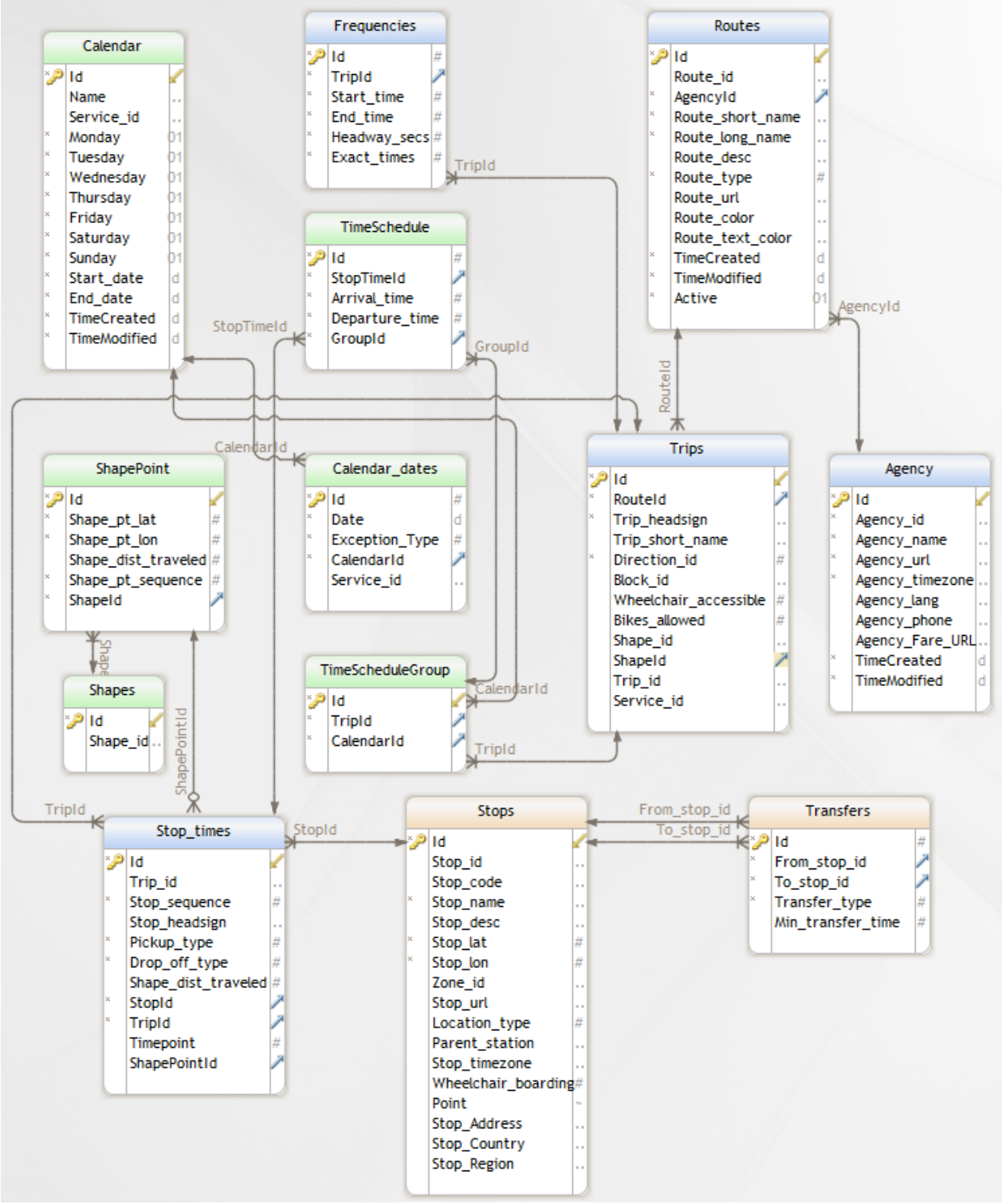
Slika 5.4: Prikaz polja *Trip Headsign*.



Slika 5.5: Prikaz polja *Route Type*.

5.3 Baza podataka planera putovanja

Na temelju GTFS datoteka koje su posebno odabrane za pravilno funkcioniranje planera putovanja, iz UML dijagrama na slici 5.1 je proizašla shema baze podataka koja se koristi za izradu praktičnog dijela diplomskog rada. Da bi se podaci što kvalitetnije prilagodili aplikaciji napravljene su određene preinake što se tiče strukture i relacija između pojedinih tablica baze podataka kako je i vidljivo na slici 5.6.



Slika 5.6: Shema baze podataka korištene za izradu planera putovanja.

Shema baze podataka na slici 5.6 sadrži neke tablice koje nisu ranije spomenute u kontekstu GTFS datoteka. To su tablice *TimeSchedule*, *TimeScheduleGroup* i *ShapePoint*. Tablica

TimeSchedule sadrži podatke o rasporedu putovanja, odnosno o vremenima polaska/dolaska sa/od pojedine stanice/luke. *TimeScheduleGroup* tablica je usko vezana uz *TimeSchedule* tablicu i koristi se za povezivanje pojedinih putovanja sa dostupnim kalendarima. *ShapePoint* tablica sadrži podatke vezane uz *Shapes* GTFS datoteku i kreirana je u svrhu prilagodbe baze podataka aplikaciji.

5.4 Prikupljanje tranzitnih podataka

Tranzitni podaci su svi podaci koji se nalaze u GTFS *feed*-u. Većina je tih podataka promjenjiva ovisno o učestalosti promjene rasporeda, ruta i vremena putovanja prijevoznika. Te podatke je potrebno na određene načine prikupljati i ažurirati.

5.4.1 Načini prikupljanja tranzitnih podataka

Postoje tri načina prikupljanja tranzitnih podataka, a to su:

- Ručno unošenje podataka u GTFS *feed*
- Poluautomatski - analiza *Web* stranice prijevoznika korištenjem određenih alata
- Automatski - integracija s prijevoznicima

Ručno unošenje podataka se obavlja u slučajevima kad prijevoznik nema gotove informacije o tranzitnim podacima. Stoga je potrebno ručno sakupljati te podatke. Ovaj način zahtjeva najviše vremena, a i nije najbolje rješenje kad je potrebno ažurirati GTFS *feed*. Najčešći podaci koje prijevoznik unaprijed ne posjeduje su: geografski podaci stanica, informacije o rutama, unaprijed definirano razdoblje trajanja pojedinih usluga prijevoza i točno vrijeme dolaska na pojedinu stanicu. Ovaj način prikupljanja GTFS podataka se koristi u praktičnom dijelu rada.

Poluautomatski način prikupljanja GTFS podataka se koristi kad prijevoznik posjeduje gotove GTFS podatke, ali raspored tih podataka nije strukturiran na prikladan način. Prijevoznik ih ne objavljuje na jednom mjestu na svojoj *Web* stranici nego mogu biti razbacani

na više lokacija stranice. Informacije koje sadrže ti podaci je potrebno prikupiti i rasporediti u GTFS *feed*, stoga se mogu koristiti alati koji mogu pri tome pomoći, takozvani *Internet bot*-ovi. To su softverske aplikacije koje obavljaju automatizirane zadatke putem Internet mreže. Općenito *Internet bot*-ovi obavljaju zadatke koji su istovremeno jednostavni i ponavljajući, i to puno većim brzinama nego što bi ih čovjek obavio. Najveća iskoristivost ovih alata je upravo u automatskom sakupljanju, analiziranju i pohranjivanju informacija sa *Web server*-a. Svaki *server* (poslužitelj) može sadržavati određenu *.txt* datoteku koja sadrži pravila automatskog prikupljanja podataka s njega koje neki *Internet bot* mora poštivati [11]. Na taj način može i prijevoznik na svojoj *Web* stranici implementirati pravila po kojima se jednom dnevno sakupljaju GTFS podaci te na taj način ubrzati postupak. Ukoliko se promijeni lokacija GTFS podataka koje *Internet bot* pronalazi na nekoj *Web* stranici također je potrebno promijeniti konfiguraciju tog alata da može nastaviti sa uspješnim traženjem tih podataka na pravom mjestu.

Kod automatskog načina prikupljanja podataka prijevoznik posjeduje GTFS podatke kreirane i pohranjene u vlastitom sustavu. Nabolji primjer bi bile zrakoplovne kompanije koje koriste vlastitu bazu podataka u kojima sadrže takve podatke, a njima je moguće lako pristupiti i prikupiti ih prilikom posjete na *Web* stranicu. Sve se češće integriraju različiti sustavi koji automatski razmjenjuju GTFS podatke te na taj način ostvaruju interoperabilnost. To je moguće postići između različitih tranzitnih agencija čija suradnja može povećati zajednički profit. Primjer za suradnju mogu biti aerodrom i lokalni autobusni prijevoznik koji prevozi putnike do aerodroma i sa njega. Najlakši i najbrži način prikupljanja GTFS podataka je upravo automatski jer prijevoznik pruža gotove i pripremljenje podatke spremne za integraciju sa drugim sustavima.

5.4.2 Validacija tranzitnih podataka

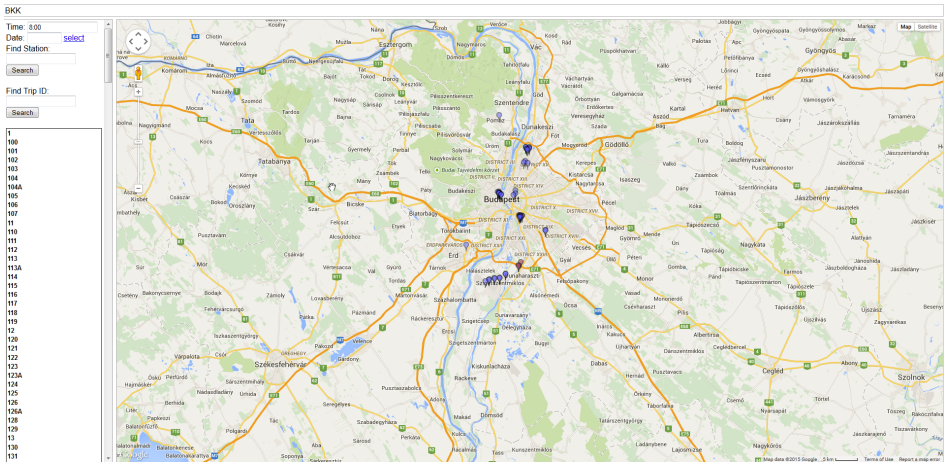
Ukoliko prijevoznik pristane da *Google* javno objavi njegove GTFS tranzitne podatke, postupak slanja tranzitnih podataka *Google*-u i konvergencija sa *Google maps* uslugom su besplatni. Vrlo je bitno da se prilikom popunjavanja GTFS datoteka popuni što više neobveznih polja jer se na taj način lakše ti podaci integriraju s *Google*-om. Prije no što

Google objavi GTFS podatke u javnost bitno je da su svi podaci što točniji, npr. geografske lokacije stanica moraju biti točne i ažurne, imena ruta i stanica moraju biti točna i moraju se podudarati sa informacijama u prometu. Rute integrirane u *Google* karte moraju biti točne. Nakon što se ti podaci objave njihov rok trajanja mora biti u podudarnosti sa trajanjem dostupnosti usluga koje prijevoznik nudi.

Obično je potrebno svoje GTFS podatke testirati prije slanja *Google*-u. Taj se postupak obavlja pomoću skupine alata: *Feed Validator* za provjeru pravilnosti formatiranja podataka, *Schedule Viewer* i *KML Writer* da bi se osiguralo da podaci imaju smisla te da točno reflektiraju usluge prijevoznika.

Feed Validator je alat koji provjerava sadrži li GTFS *feed* kakve greške. Rezultati analize se generiraju u HTML formatu. Pokretanje ovog alata omogućuje pronalazak grešaka koje mogu uzrokovati probleme pri rutiranju i prikazu tranzitnih podataka. *Feed Validator* je moguće pokrenuti nad direktorijem u kojem se nalaze GTFS podaci ili nad *.zip* datotekom koja sadrži *.txt* datoteke [6].

Schedule Viewer je alat koji se koristi za uvid u sadržaj GTFS podataka na karti. S ovim dijagnostičkim alatom se učitavaju GTFS podaci i kao izlazna vrijednost se dobije vizualni prikaz GTFS podataka (stanice, rute, generirani putovi) na karti. Na slici 5.7 vidljiv je primjer izlazne vrijednosti ovog alata ako se kao ulazne vrijednosti uzmu GTFS podaci gradskog prijevoznika Budimpešte [14].



Slika 5.7: Vizualni prikaz izlazne vrijednosti *Schedule Viewer* alata.

KML Writer je alat koji iscrtava stanice, rute, generirane putanje i oblike koji su definirani u GTFS *feed*-u. Ti podaci koje ovaj alat generira imaju ekstenziju *.kml* i mogu se pročitati u *Google Earth* aplikaciji. Na taj način se može utvrditi jesu li stanice i oblici locirani na pravom mjestu. Također omogućuje 3D vizualizaciju generiranih putanja koja može pomoći pri pronalasku određenih grešaka u *stop_times* datoteci [13].

5.5 Konzumiranje tranzitnih podataka

Validirani GTFS tranzitni podaci su uz još neke preinake spremni za slanje *Google*-u na nastavak njihovog procesuiranja. Prethodno je potrebno te podatke konvertirati u *google_transit.zip* datoteku. Osim ovog načina stavljanja tranzitnih podataka na korištenje postoji i drugi način izrade vlastitog planera koji nakon validacije ne uključuje slanje podataka *Google*-u. Oba postupka su u nastavku objašnjeni.

5.5.1 *Google Transit*

Nakon što su GTFS tranzitni podaci procesuirani kroz *Feed Validator*, *Schedule Viewer* i *KML Writer*, preimenuju se i konvertiraju u *google_transit.zip* datoteku. Potom slijedi otpremanje *google_transit.zip* datoteke na *Web* stranicu prijevoznika na način da se spremi u određeni direktorij čije se ime ne smije mijenjati, a *google_transit.zip* datoteka se mora jedina nalaziti u tom direktoriju. Na taj način se priprema datoteka za pristup i validaciju od strane *Google*-a. Slijedi kontaktiranje *Google*-a u vezi validacije i testiranja datoteke. Kad *Google* uspješno testira i prihvati *google_transit.zip* podatke proslijedit će te podatke prijevozniku koji mora potvrditi kvalitetu tih podataka testirajući rute na *Google* kartama. *Google_transit.zip* podaci su spremni za integraciju sa *Google* kartama tek kad prijevoznik potvrdi njihovu kvalitetu *Google*-u [1].

Ovaj postupak integracije s *Google*-om zahtjeva ažurnost tranzitnih podataka. Preporučljivo je svakih mjesec dana slati ažurirane podatke. Ova karakteristika predstavlja veliku prednost jer smanjuje rizik od pružanja krivih informacija putniku. Također je moguće da prijevoznik automatizira proces ažuriranja GTFS podataka na način da ih smjesti na

određeni *Web* poslužitelj s kojeg će *Google* regularno dohvaćati podatke.

5.5.2 Izrada vlastitog planera

Izrada vlastitog planera predstavlja stavljanje tranzitnih podataka na korištenje bez potrebe za integracijom sa *Google*-om. Nakon validacije, podaci se spremaju u bazu podataka koja predstavlja GTFS *feed*. Svaka tablica baze podataka predstavlja jednu *.txt* datoteku GTFS *feed*-a. Aplikacija za planiranje putovanja može primiti vrijednosti početne i ciljne lokacije te iz baze podataka izvući dvije geografske koordinate autobusne stanice, brodske ili zračne luke koje su najbliže unesenim lokacijama. Izvučeni podaci o koordinatama se kao izlazne vrijednosti mogu prikazati kao točke na *Google* karti. Korištenje baze podataka na ovaj način ne zahtjeva *upload* tranzitnih podataka *Google*-u.

Ovaj postupak je dobar ukoliko se tranzitni podaci prijevoznika ne ažuriraju često, sam prijevoznik ih može u bazi podataka proizvoljno mijenjati. Usprkos tome, ovakav način kojim se ne šalju podaci *Google*-u ne zahtjeva nikakvu ažurnost podataka od strane prijevoznika, što povećava rizik od pružanja krivih informacija korisnicima. Također se onemogućuje provjera kvalitete GTFS podataka od strana *Google*-a, dok navedeni alati za validaciju podataka često nisu dovoljni.

6 Postupak izrade lokalnog planera putovanja

Izrada praktičnog dijela diplomskog rada sastoji se od više faza. Prva faza predstavlja odabir tehnologija koje će se koristiti za izradu lokalnog planera putovanja. Druga faza predstavlja same korisničke zahtjeve koje izrađeno rješenje treba ispuniti. Na temelju korisničkih zahtjeva konstruira se arhitektura aplikacije i kontrola unosa podataka. Potom u sljedećoj fazi slijedi postupak generiranja grafa za algoritam koji će biti detaljno objašnjen. Kao izlaznu vrijednost aplikacija na grafičkom sučelju prikazuje rutu u obliku linije na *Google* karti koja povezuje prethodno unesenu početnu i ciljnu lokaciju. Bitno je razumjeti algoritam koji vrši izračun podataka kojima se prikazuje ruta na karti. Priprema podataka za algoritam i samo izvođenje algoritma predstavljaju zadnju fazu izrade praktičnog rješenja.

6.1 Korištene tehnologije i razvojna okruženja

HTML (eng. *Hyper Text Markup Language*) je prezentacijski jezik za izradu *Web* stranica. Hipertekst dokument stvara se pomoću HTML jezika kojim se oblikuje sadržaj *Web* stranice. Besplatan je i jednostavan za uporabu i lako se uči, što je jedan od razloga njegove opće prihvaćenosti i popularnosti. Temeljna zadaća HTML jezika jest uputiti *Web* pregledniku kako prikazati hipertext dokument. HTML nije programski jezik. Njime se ne može izvršiti nikakva zadaća, pa čak ni najjednostavnija operacija zbrajanja ili oduzimanja dvaju cijelih brojeva. On služi samo za opis hipertekstualnih dokumenata. HTML datoteke su zapravo obične tekstualne datoteke, ekstenzija im je *.HTML* ili *.htm*. Osnovni građevni element svake stranice su znakovi (tags) koji opisuju kako će se nešto prikazati u *Web* pregledniku. Poveznice unutar HTML dokumenata povezuju dokumente u uređenu hijerarhijsku strukturu i time određuju način na koji posjetitelj doživljava sadržaj stranica. HTML se razvija tako da je svaka nova verzija HTML-a razvijana na način da ostane čitljiva na svim *Web* preglednicima. Zadnja verzija HTML-a je HTML 5. Svaki HTML dokument

sastoji se od osnovnih građevnih blokova - HTML elemenata. Svaki HTML element sastoji se od para HTML oznaki (engl. *tag*). Dva osnovna elementa HTML dokumenta su *head* i *body*. Primjer oznaka kojima se označava *head* element: $< body >< /body >$ [8].

CSS (eng. *Cascading Style Sheets*) je stilski jezik koji se rabi za opis prezentacije dokumenta napisanog pomoću markup (HTML) jezika. Kako se *Web* razvijao prvotno su u HTML ubacivani elementi za definiciju prezentacije (npr. *tag* $< font >$), ali je dovoljno brzo uočena potreba za stilskim jezikom koji će HTML osloboditi potrebe prikazivanja sadržaja (što je prvenstvena namjena HTML-a) i njegovog oblikovanja (čemu danas služi CSS). Drugim riječima, stil definira kako prikazati HTML elemente. CSS-om se uređuje sam izgled i raspored stranice. CSS možemo pisati unutar same HTML stranice, na 3 načina [3]:

- Kao stilove u zaglavlju HTML dokumenta (tj. između $< style >$ i $< /style >$ elementa):

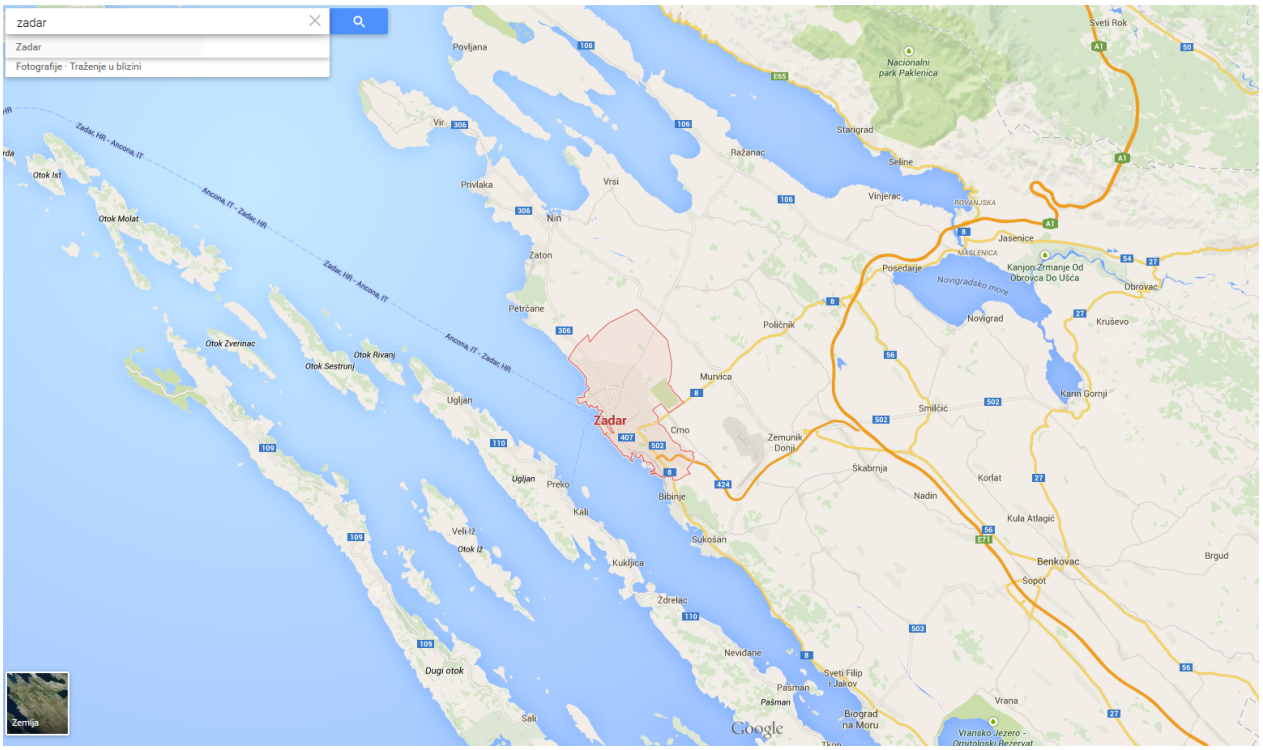
$$< styletype = "text/css" >$$
$$h1 \{color: blue\}$$
$$< /style >$$

- Unutar samih HTML *tag*-ova
- Ili ga možemo definirati u posebnom dokumentu, i rabiti pomoću poziva:

$$< link\ rel="stylesheet"\ href="xyz.css"\ type = "text/css" >$$

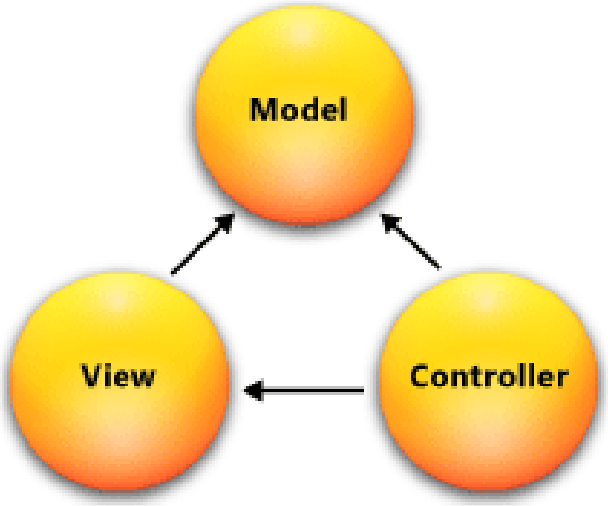
JavaScript je dinamični računalni programski jezik. Najčešće se koristi kao dio *Web* preglednika čija implementacija omogućuje skriptama na računalu korisnika (*client-side*) interakciju s korisnikom, kontrolu preglednika, asinkronu komunikaciju te mijenjanje sadržaja dokumenta koji će se prikazati. Također se koristi i u mrežnom programiranju sa poslužiteljske strane (*server-side*) u kreaciji *desktop* i mobilnih aplikacija. *JavaScript* je klasificiran kao skriptni jezik sa dinamičnim i prvoklasnim funkcijama. Podržava objektno-orijentirani, imperativni i funkcionalni stil programiranja. *JavaScript* se također koristi u sredinama koje nisu na *Web*-u, kao što su PDF i *desktop* dodaci. *JavaScript* je standardiziran u specifikaciji *ECMAScript* jezika [12].

***Google Maps* API (eng. *Application Programming Interface*)** predstavlja aplikacijsko programsko sučelje koje omogućuje tvrtka *Google*. *Google maps* pruža široku lepezu API-a koji se mogu ugraditi na vlastitu *Web* aplikaciju te svakodnevno koristiti. *Google* nudi usluge besplatnog spajanja na njihov servis te je na taj način moguće implementacijom programskog koda pozvati njihove mape i nadopuniti ih željenim podacima. Na slici 6.1 je prikaz grada Zadra na *Google maps* sučelju.



Slika 6.1: *Google maps* sučelje.

ASP.NET MVC (eng. *Model-View-Controller*) je arhitektonski uzorak koji dijeli aplikaciju u tri glavne komponente: *Model*, *View* (pogled) i *Controler*. ASP.NET MVC okvir pruža alternativu ASP.NET *Web* formama za stvaranje *Web* aplikacija. ASP.NET MVC okvir je lagan, vrlo testabilan prezentacijski okvir koji je (kao i kod *Web* aplikacija koje su izrađene na *Web* formama) integriran sa postojećim ASP.NET značajkama.



Slika 6.2: Arhitektura ASP.NET MVC-a.

MVC okvir uključuje sljedeće komponente:

- Modeli - objekti modela su dijelovi aplikacije koji implementiraju logiku za podatkovni dio *Web* aplikacije. Često objekti modela preuzimaju i pohranjuju stanje modela u bazu podataka. Na primjer, objekt nekog proizvoda bi mogao primiti informaciju iz baze podataka, obraditi je te ažurirane podatke upisati nazad u tablicu tog proizvoda u bazu podataka.

- Pogledi - komponente koje prikazuju korisničko sučelje aplikacije UI (eng. *User Interface*). Ovo grafičko sučelje je izrađeno od podataka modela. Primjer bi bio pogled za uređivanje tablice *proizvodi* koji prikazuje tekstualne okvire, *drop-down* popis i *check box*-ove koji se temelje na trenutnom stanju objekta *proizvod*.

- Kontroleri - to su komponente koje su zadužene za interakciju s korisnikom, rad sa modelom ili čak za odabir pogleda koji će prikazati korisničko sučelje. U MVC aplikacijama pogled prikazuje samo vizualne informacije dok kontroler upravlja i reagira na korisnikov unos i interakciju. Na primjer, kontroler može obraditi upit koji je podatkovnog tipa *string* te poslati te vrijednosti modelu koji pak može iskoristiti ove vrijednosti za obavljanje upita u bazi podataka.

MVC uzorci pomažu programeru pri razvoju aplikacije na način da razdvajaju različite aspekte aplikacije (ulazna logika, poslovna logika i UI logika), a pružaju opuštenu vezu

između tih elemenata. Uzorak specificira gdje bi svaka vrsta od navedenih logika trebala biti smještena u aplikaciji. UI logika spada u pogled, ulazna logika spada u kontroler, a poslovna logika spada u model. To odvajanje pomaže programeru pri upravljanju složenosti tijekom razvoja aplikacije jer omogućuje da se usredotoči na jedan aspekt implementacije istovremeno. Na primjer, moguće je usredotočiti se na pogled neovisno o poslovnoj logici. Mekana veza između tri glavne komponente MVC aplikacije također promiče i paralelni razvoj. Na primjer, jedan programer može raditi na pogledu, drugi programer može raditi na logici kontrolera, a treći programer se može usredotočiti na poslovnu logiku modela [2].

Microsoft SQL Server (MS SQL Server) je sustav za upravljanje relacijskim bazama podataka koji je razvijen od strane *Microsoft*-a. To je softverski proizvod (softverski poslužitelj) čija je primarna funkcija pohrana i dohvaćanje podataka u skladu sa zahtjevima drugih softverskih aplikacija koje mogu biti na istom računalu ili na drugim računalima povezanima na mrežu. Postoji više verzija Microsoft SQL Server sustava, a u radu je korištena Microsoft SQL Server Express 2012. To je besplatna inačica ovog sustava čija je razlika u odnosu na plaćenu verziju memorijska veličina baze podataka koju može pohraniti (maksimalno 10 GB).

Microsoft SQL Management Studio je softverska aplikacija razvijena od strane *Microsoft*-a koja se koristi za konfiguraciju, upravljanje i administraciju svih komponenti MS SQL Server-a. Aplikacija se sastoji od editora za pisanje SQL upita i grafičkih alata koji rade sa objektima MS SQL Server-a [15]. Upiti koji se editiraju i pokreću u MS SQL Management Studio-u su zapravo Transact-SQL (eng. *Transact-Structured Query Language*) naredbe kojima se upravlja podacima u bazama podataka MS SQL Server-a. Transact-SQL je *Microsoft*-ova ekstenzija za SQL (*Structured Query Language*), standardizirani računalni jezik koji je originalno razvijen od IBM-a u svrhu kreiranja upita te mijenjanje strukture i podataka u relacijskim bazama podataka. Transact-SQL je temelj za korištenje MS SQL Server-a. MS SQL Management Studio komunicira sa MS SQL Server-om na način da mu šalje Transact-SQL upite. U praktičnom dijelu diplomskog rada koristi se verzija MS SQL Management Studio 2012.

Microsoft Visual Studio je integrirano razvojno sučelje (eng. IDE - *Integrated Development Environment*) razvijeno od *Microsoft*-a. Koristi se za razvoj raznih aplikacija kao što su grafička korisnička sučelja, *Windows* forme i *Web* aplikacije. Pritom se za razvoj koristi čisti programski kod, a može se koristiti i upravljani (eng. *Managed*) kod za sve platforme upravljane od strane *Microsoft Windows*-a. Postoji više verzija MS Visual Studio-a: Visual Studio Express (besplatna verzija), Visual Studio Professional, Visual Studio Premium i Visual Studio Ultimate [19]. Za izradu praktičnog dijela rada koristi se Visual Studio Ultimate 2013.

6.2 Korisnički zahtjevi planera putovanja

Glavni zahtjev koji se postavlja pred planer putovanja jest točan izračun rute između dvije lokacije (adrese) na karti. Korisniku se prilikom učitavanja aplikacije prikazuje početno sučelje koje sadrži formu za tekstualni unos startne i ciljne lokacije. Prilikom ispunjavanja i potvrde podataka aplikacija učitava potvrđene podatke te otvara stranicu na kojoj se nalazi *Google* karta sa označenim prethodno odabranim lokacijama na karti. Označenu početnu i ciljnu lokaciju spaja linija (ruta) koja se generira na temelju pozadinskih algoritamskih izračuna. Algoritamski izračuni kalkuliraju najbrži put između dvije lokacije kombinirajući modove putovanja koji su potrebni za siguran dolazak od starta do cilja (zrakoplovna mreža, tranzitna cestovna mreža i pomorska mreža). Aplikacija kao uvjet za najbrže putovanje uzima u obzir vrijeme putovanja ili prostornu udaljenost od startne do ciljne lokacije.

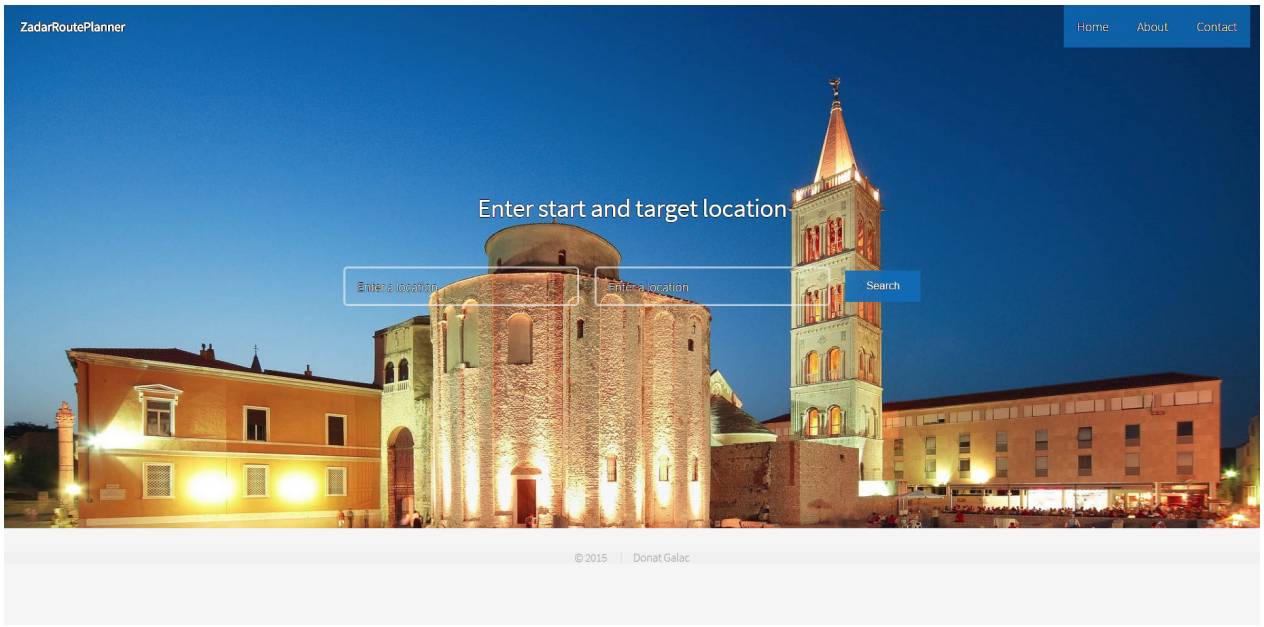
Korisnik mora imati mogućnost promjene lokacija na samoj stranici na kojoj se nalazi *Google* karta. Zato je uvedena dodatna forma na toj stranici koja omogućuje promjenu unesenih podataka za planiranje putovanja. Dodatna forma funkcionira na isti način kao i ona na početnoj stranici, jedina je razlika u tome što se nalazi na *Google* karti radi bolje praktičnosti.

Uz navedene stranice *Web* aplikacija sadrži dodatne dvije stranice, a to su:

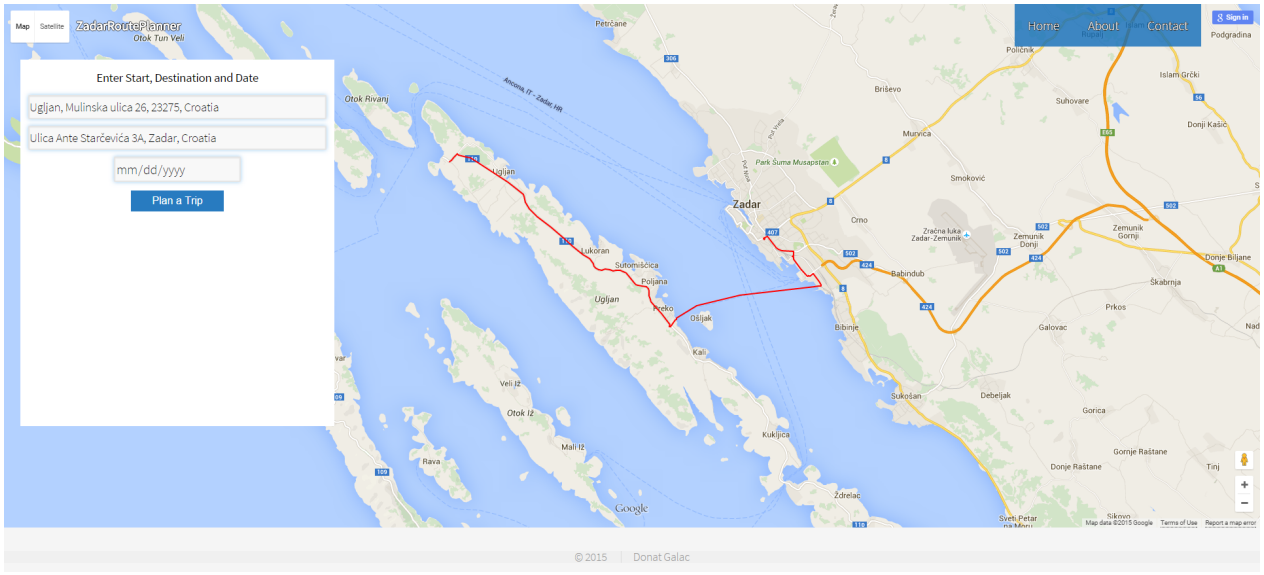
- Stranica kontakta koja sadrži kontaktne podatke administratora,

- Informacijska stranica koja sadrži informacijske podatke o aplikaciji.

Slika 6.3 prikazuje izgled početne stranice lokalnog planera putovanja, a slika 6.4 prikazuje izgled stranice na kojoj se nalazi karta sa primjerom rute.



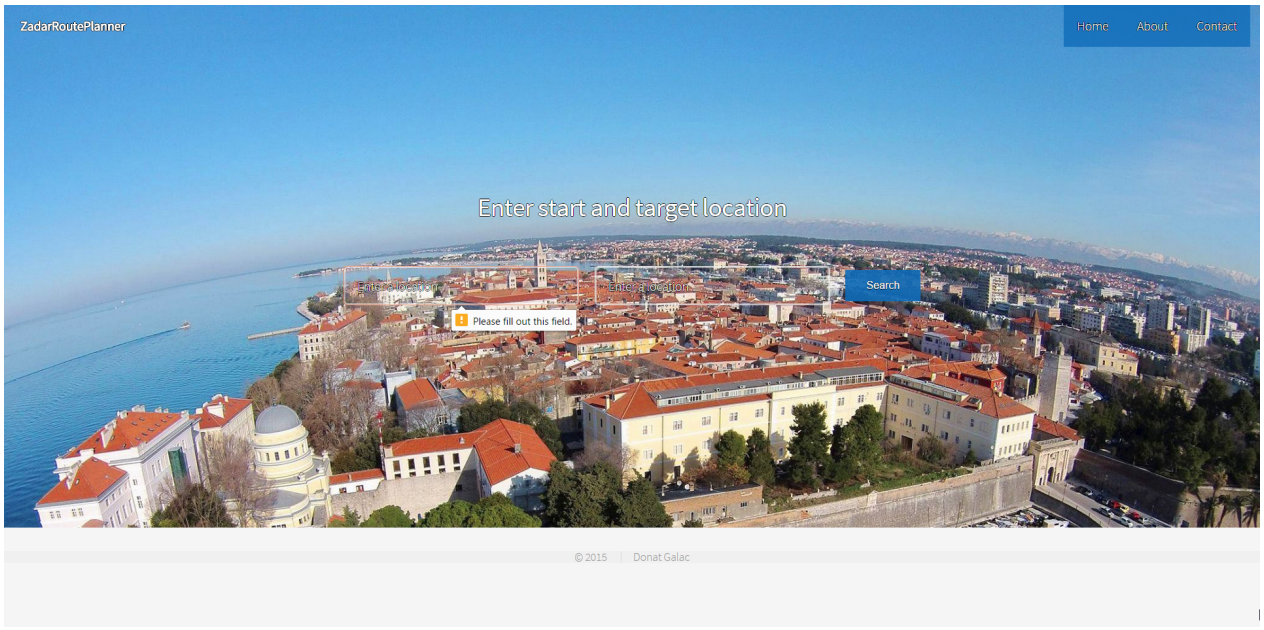
Slika 6.3: Početna stranica planera putovanja.



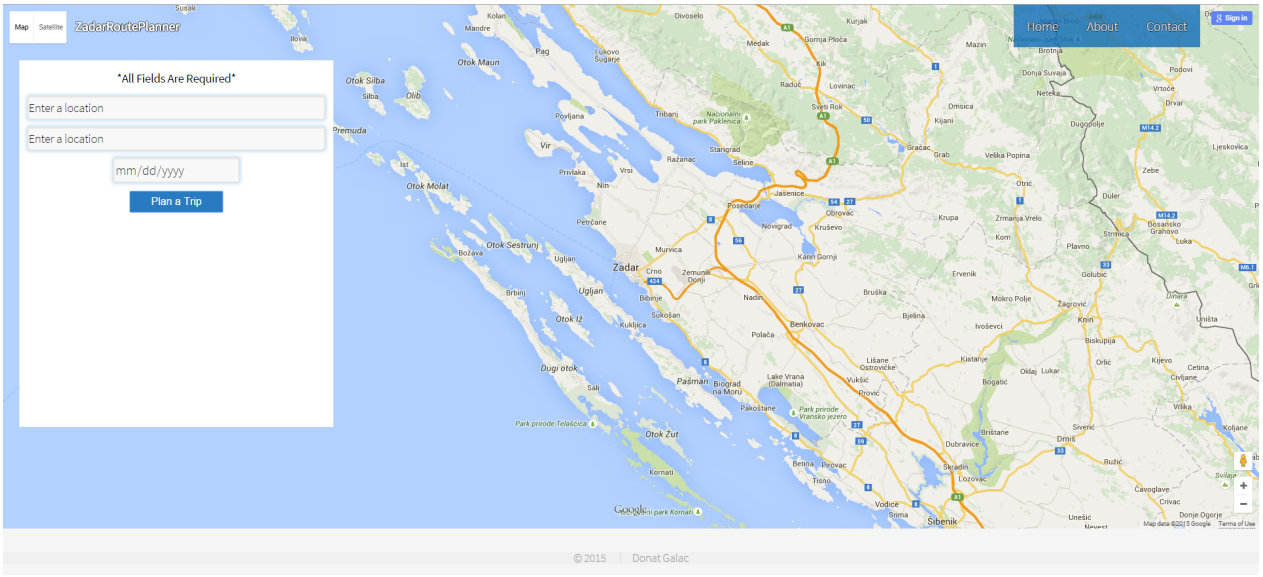
Slika 6.4: Stranica sa *Google* kartom planera putovanja.

6.3 Kontrola unosa podataka u planer putovanja

Korisniku je onemogućeno učitavanje stranice na kojoj se nalazi karta bez da upiše i startnu i ciljnu lokaciju. Na taj način se spriječava mogućnost da se ne prikažu obje lokacije na naknadno učitanoj karti. Obavijest o tome da je potreban unos podataka u oba polja prikazan je na slici 6.5. Također se na stranici karte obavlja provjera upisanih podataka gdje se korisniku, ukoliko nije upisao sve tražene podatke, na vrhu forme za unos podataka pojavljuje poruka da je unos svih podataka obvezan. Slika 6.6 prikazuje takav slučaj.



Slika 6.5: Primjer kontrole unosa i potvrde podataka na početnoj stranici planera putovanja.



Slika 6.6: Primjer kontrole unosa i potvrde podataka na stranici karte planera putovanja.

Izgled koda koji omogućuje provjeru unosa podataka u formu na *Map* stranici prikazan je na slici 6.7. Ukoliko prilikom klika na dugme *Plan a Trip* nisu sva polja ispunjena pokreće se metoda *event.preventDefault()* koja zaustavlja slanje podataka putem forme.

```
//provjera jesu li upisana sva polja u menu na Map stranici - ova funkcija se pokreće na otvaranje Map stranice
function checkForm() {
    var pr1 = document.getElementById("PocetnaAdresa");
    var pr2 = document.getElementById("CiljnaAdresa");
    var pr3 = document.getElementById("dateText");

    //u slučaju ako nisu zaustavi izvršavanje Calculate forme
    $('#Calculate').on('submit', function (event) {
        if (pr1.value == "" || pr2.value == "") {
            event.preventDefault();
            document.getElementById("naslov").innerHTML = "*All Fields Are Required*";
            return false;
        }
    });
}
```

Slika 6.7: Kod za kontrolu unosa i potvrde podataka na stranici karte planera putovanja.

6.4 Generiranje grafa za algoritam

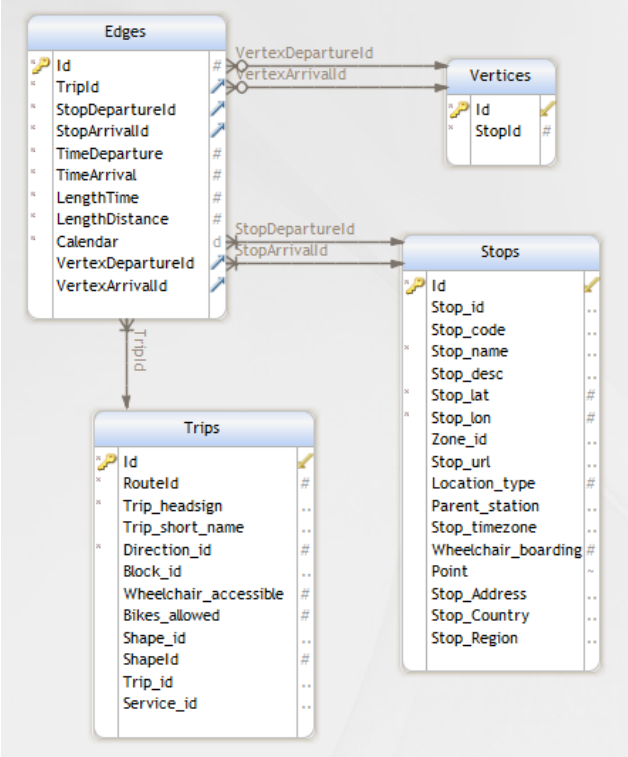
U prijašnjem poglavlju predstavljena je baza podataka koju koristi planer putovanja. Za pravilno funkcioniranje aplikacije prethodno je bilo potrebno upisati podatke u tu bazu podataka. Također je potrebno dodati još dvije tablice, *Edges* i *Vertices*. Podaci tih dviju tablica predstavljat će generirani graf koji se sastoji od skupa vrhova i bridova koji ih

povezuju. Graf služi kao priprema za pravilno izvođenje algoritma opisanog u nastavku. Podaci za graf upisuju se pozivanjem funkcije i pokretanjem procedura također opisanih u nastavku.

6.4.1 Tablice *Edges* i *Vertices*

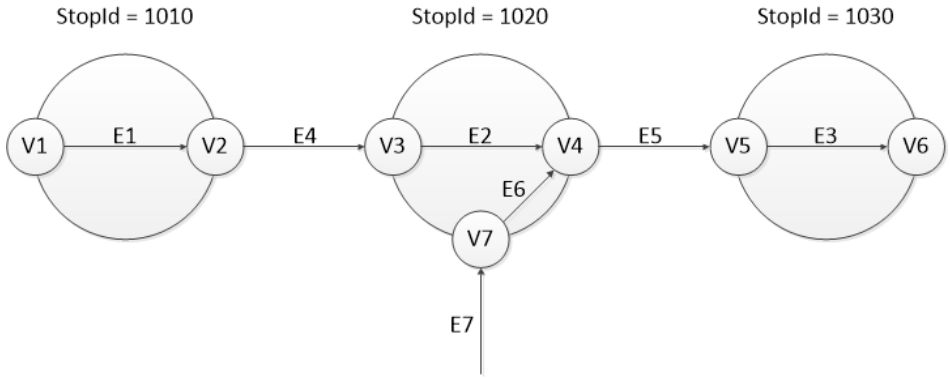
Slika 6.8 prikazuje *Edges* i *Vertices* tablice, njihov međusobni odnos te odnos *Edges* prema tablicama *Trips* i *Stops*. Polja *StopDepartureId* i *StopArrivalId* tablice *Edges* su strani ključevi koji su vezani za primarni ključ *Id* tablice *Stops*, a polje *TripId* je strani ključ koji je povezan sa primarnim ključem *Id* tablice *Trips*. Polja *VertexDepartureId* i *VertexArrivalId* su strani ključevi povezani sa primarnim ključem *Id* tablice *Vertices*.

Podaci *Edges* tablice modeliraju sve moguće bridove koji postoje u bazi podataka, odnosno sve veze između svih mogućih vrhova. Postoje tri vrste bridova, bridovi koji povezuju vrhove različitih stanica, bridovi koji povezuju vrhove iste stanice (njen početni i ciljni vrh) i već spomenuti bridovi prijenosa (*transfer edges*) koji povezuju vrhove različitih putovanja unutar iste stanice. Ti bridovi predstavljaju presjedanje putnika u drugo vozilo na stanici. Svaki zapis tablice predstavlja jedan brid koji je jedinstveno označen sa poljem *Id*. Polje *TimeDeparture* predstavlja vrijeme polaska vozila sa vrha *StopDepartureId*, dok je polje *TimeArrival* vrijeme dolaska vozila na stanicu *StopArrivalId*. Polje *LengthTime* predstavlja razliku između vrijednosti *TimeArrival* i *TimeDeparture* polja (to je zapravo vrijeme putovanja od polaznog do dolaznog vrha). Polje *LengthDistance* predstavlja prostornu udaljenost između *VertexDepartureId* i *VertexArrivalId* vrha, dok polje *Calendar* predstavlja datum kad je vožnja na tom bridu aktivna.
S druge strane, podaci tablice *Vertices* modeliraju sve moguće vrhove koji postoje u bazi podataka. Postoje dvije vrste vrhova, početni i ciljni (polja *VertexDepartureId* i *VertexArrivalId* tablice *Edges*), s tim da oni također mogu biti početni i ciljni vrhovi iste stanice. Svaki zapis tablice predstavlja jedan vrh koji je jedinstveno označen sa poljem *Id*. Polje *StopId* predstavlja *Id* stanice kojoj pripada vrh zapisa.



Slika 6.8: Tablice *Edges*, *Vertices*, međusobni relacijski odnosi i odnosi prema drugim tablicama.

Vrste bridova i vrhova detaljnije su prikazane u modelu na slici 6.9. Bridovi *E1*, *E2*, *E3* i *E6* povezuju vrhove *V1* i *V2*, *V3* i *V4*, *V5* i *V6* te *V7* i *V4*. To su bridovi koji povezuju početne i ciljne vrhove istih stanica, s tim da je brid *E6* prijenosni brid koji predstavlja presjedanje putnika. Bridovi *E4* i *E5* povezuju vrhove *V2* i *V3* te *V4* i *V5*. To su bridovi koji povezuju početne i ciljne vrhove različitih stanica. Brid *E7* povezuje vrh *V7* sa nekim vrhom prethodne stanice koja pripada drugom putovanju. Polja *StopId* ovdje jedinstveno označavaju stanice. Ako se za primjer uzme stanica 1010, ona se sastoji od vrhova *V1* i *V2* i brida *E1* koji ih povezuje.



Slika 6.9: Prikaz dijela rute usmjerenim težinskim grafom

6.4.2 Funkcija i procedure za upis podataka u tablice *Edges* i *Vertices*

Postoje četiri procedure čijim se izvršavanjem zapisuju podaci u *Edges* i *Vertices* tablice. Za sve procedure je predviđeno da se u bazi podataka planera putovanja izvršavaju periodično nakon određenog vremena. Ukupno vrijeme trajanja izvršavanja procedura ovisi o količini podataka u tablicama *Stop_times*, *TimeSchedule*, *TimeScheduleGroup* i *Calendar_dates* jer se izvršavanjem procedura podaci iz tih tablica upisuju u tablice *Edges* i *Vertices*. S obzirom na količinu podataka planera putovanja ukupno vrijeme izvršavanja procedura je 13 sekundi.

Slika 6.10 prikazuje proceduru *Step1* koja se izvršava prva. Procedura prvo briše vrijednosti tablice *Vertices* i pomoćne tablice *ResolvedEdges*. Tablica *ResolvedEdges* ima sva polja ista kao i tablica *Edges* uz dodan strani ključ *CalendarId* koji se referencira na tablicu *Calendars*. Procedura također kao početne vrijednosti *Id* polja obje tablice upisuje vrijednost 0. Potom u tablicu *ResolvedEdges* upisuje sve bridove koji povezuju vrhove istih stanica. Ti su bridovi jedinstveno označeni poljem *Id*. Procedura nakon toga iterira kroz prethodno kreirane zapise bridova te u polja *VertexDepartureId* i *VertexArrivalId* upisuje početni i ciljni vrh svakog brida. Kroz istu iteraciju procedura istodobno pristupa tablici *Vertices* te za svako od upisanih polja *VertexDepartureId* i *VertexArrivalId* tablice *ResolvedEdges* kreira poseban zapis u tablici *Vertices*. Na taj način svaki od ta dva polja svakog zapisa predstavlja jedan vrh koji je jedinstveno označen sa poljem *Id*. Na kraju svog izvođenja procedura poziva proceduru *Step2*.

```
CREATE PROC [dbo].[Step1]
AS
TRUNCATE TABLE [dbo].[ResolvedEdges]
TRUNCATE TABLE [dbo].[Vertices]

INSERT INTO [dbo].[ResolvedEdges](TripId, StopDepartureId, StopArrivalId,
 TimeArrival, TimeDeparture, LengthTime, LengthDistance, CalendarId, [Sequence], [Distance])
SELECT
    tsg.Id 'TripId',
    st.StopId 'StopDepartureId',
    st.StopId 'StopArrivalId',
    ts.Arrival_time 'TimeArrival',
    ts.Departure_time 'TimeDeparture',
    (ts.Departure_time - ts.Arrival_time) 'LengthTime',
    0 'LengthDistance',
    tsg.CalendarId 'CalendarId',
    st.Stop_sequence 'Sequence',
    st.Shape_dist_traveled
FROM [gtfs].[TimeSchedule] ts
INNER JOIN [gtfs].[TimeScheduleGroup] tsg ON ts.GroupId = tsg.Id
INNER JOIN [gtfs].[Stop_times] st ON st.Id = ts.StopTimeId


DECLARE cur CURSOR FOR
SELECT StopDepartureId, StopArrivalId, VertexDepartureId, VertexArrivalId FROM [dbo].[ResolvedEdges]
FOR UPDATE OF VertexDepartureId, VertexArrivalId
OPEN cur
    DECLARE @StopDepartureId int, @StopArrivalId int, @VertexDepartureId int, @VertexArrivalId int
    FETCH NEXT FROM cur INTO @StopDepartureId, @StopArrivalId, @VertexDepartureId, @VertexArrivalId

    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT INTO [dbo].[Vertices] ([StopId]) VALUES(@StopDepartureId)
        DECLARE @VertexDepartureId2 int = SCOPE_IDENTITY()
        INSERT INTO [dbo].[Vertices] ([StopId]) VALUES(@StopArrivalId)
        DECLARE @VertexArrivalId2 int = SCOPE_IDENTITY()

        UPDATE [dbo].[ResolvedEdges] SET
            VertexDepartureId = @VertexDepartureId2,
            VertexArrivalId = @VertexArrivalId2
        WHERE CURRENT OF cur

        FETCH NEXT FROM cur INTO @StopDepartureId, @StopArrivalId, @VertexDepartureId, @VertexArrivalId
    END
CLOSE cur
DEALLOCATE cur

EXEC  [dbo].[Step2]
GO
```

Slika 6.10: Procedura *Step1*.

Nakon procedure *Step1* izvršava se procedura pod nazivom *Step2* koja je prikazana na slici 6.11. Ona se nadovezuje na prijašnju proceduru na način da iteracijom kroz prijašnje unesene podatke tablice *ResolvedEdges* provjerava za svaka dva uzastopna zapisa je li pripadaju istom putu (vrijednost polja *TripId*). U slučaju da pripadaju istom putu procedura će dodati novi zapis u tablicu. Na taj način se ovom procedurom dodaju zapisi bridova koji povezuju vrhove različitih stanica. Na kraju izvođenja poziva se sljedeća procedura, *Step3*.

```
1  CREATE PROC [dbo].[Step2]
2  AS
3  DECLARE cur SCROLL CURSOR FOR
4      SELECT TripId, StopDepartureId, StopArrivalId, TimeArrival, TimeDeparture, LengthTime, LengthDistance,
5          CalendarId, Distance, VertexDepartureId, VertexArrivalId
6      FROM [dbo].[ResolvedEdges] ORDER BY TripId ASC, Sequence ASC
7  OPEN cur
8      DECLARE @TripId int, @StopDepartureId int, @StopArrivalId int, @TimeArrival int, @TimeDeparture int,
9          @LengthTime int, @LengthDistance int, @CalendarId int, @Distance float, @VertexDepartureId int, @VertexArrivalId int
10     DECLARE @TripId2 int, @StopDepartureId2 int, @StopArrivalId2 int, @TimeArrival2 int, @TimeDeparture2 int,
11         @LengthTime2 int, @LengthDistance2 int, @CalendarId2 int, @TimeDeparture3 int, @Distance2 float,
12         @VertexDepartureId2 int, @VertexArrivalId2 int
13
14     FETCH NEXT FROM cur INTO @TripId, @StopDepartureId, @StopArrivalId, @TimeArrival, @TimeDeparture,
15         @LengthTime, @LengthDistance, @CalendarId, @Distance, @VertexDepartureId, @VertexArrivalId
16     WHILE @@FETCH_STATUS = 0
17     BEGIN
18         SET @TimeDeparture3 = @TimeDeparture
19         SET @TripId2 = @TripId
20         SET @CalendarId2 = @CalendarId
21         SET @StopDepartureId2 = @StopDepartureId
22         SET @TimeDeparture2 = @TimeDeparture
23         SET @Distance2 = @Distance
24         SET @VertexDepartureId2 = @VertexArrivalId
25
26         FETCH NEXT FROM cur INTO @TripId, @StopDepartureId, @StopArrivalId, @TimeArrival, @TimeDeparture, @LengthTime,
27             @LengthDistance, @CalendarId, @Distance, @VertexDepartureId, @VertexArrivalId
28         IF @TripId2 = @TripId
29         BEGIN
30             SET @LengthTime2 = (@TimeArrival - @TimeDeparture3)
31             SET @StopArrivalId2 = @StopArrivalId
32             SET @TimeArrival2 = @TimeArrival
33             SET @LengthDistance2 = CAST(((@Distance - @Distance2) * 1000) AS INT)
34             SET @VertexArrivalId2 = @VertexDepartureId
35
36             INSERT INTO ResolvedEdges(TripId, StopDepartureId, StopArrivalId, TimeArrival, TimeDeparture, LengthTime,
37             LengthDistance, CalendarId, Sequence, Distance, VertexDepartureId, VertexArrivalId)
38             VALUES(@TripId2, @StopDepartureId2, @StopArrivalId2, @TimeArrival2, @TimeDeparture2, @LengthTime2,
39             @LengthDistance2, @CalendarId2, 0, 0, @VertexDepartureId2, @VertexArrivalId2)
40         END
41     END
42 CLOSE cur
43 DEALLOCATE cur
44
45 DELETE FROM dbo.ResolvedEdges WHERE Id = (SELECT MAX(Id) FROM dbo.ResolvedEdges)
46 EXEC Step3
47 GO
```

Slika 6.11: Procedura *Step2*.

Struktura *Step3* procedure prikazana je na slikama 6.12 i 6.13. Ona prvo briše sve zapise u tablici *Edges* i vraća vrijednosti *Id* primarnih ključeva na 0. Potom pristupa tablici *ResolvedEdges* u kojoj se vrši iteracija kroz svaki njen zapis. Pristupom na svaki zapis vrši se referenciranje na tablicu *Calendars* iz koje se učitava početni i završni datum razdoblja u kojem je putovanje (*trip*) kojemu pripada trenutni brid aktivno. Potom se izvodi petlja u kojoj se za svaki dan između početnog i završnog datuma prvo provjerava je li postoji iznimka vezana za putovanje trenutnog brida (linija 19). Iznimka može predstavljati:

- Neaktivnost usluge na taj dan.
- Usluga je aktivna iznimno na taj dan.

Ukoliko je usluga aktivna na taj dan, u tablicu *Edges* se dodaje trenutni zapis tablice *ResolvedEdges*. Ukoliko ne postoji nikakva iznimka vezana za putovanje trenutnog brida, petlja provjerava je li dan u tjednu trenutnog datuma isti kao i kod datuma trenutno

gledanog brida (trenutni zapis tablice *ResolvedEdges*). Ukoliko je dan u tjednu isti, u tablicu *Edges* se dodaje trenutni zapis tablice *ResolvedEdges* (linije 30-73). Ovaj postupak se izvodi sve dok se ne završi iteracija kroz sve zapise tablice *ResolvedEdges*. Na kraju se poziva procedura *Step4*.

```
1  CREATE PROCEDURE [dbo].[Step3]
2  AS
3  BEGIN
4  TRUNCATE TABLE Edges
5  DECLARE cur SCROLL CURSOR FOR
6      SELECT TripId, StopDepartureId, StopArrivalId, TimeArrival, TimeDeparture, LengthTime, LengthDistance,
7      CalendarId, VertexDepartureId, VertexArrivalId
8      FROM [dbo].[ResolvedEdges]
9  OPEN cur
10     DECLARE @TripId int, @StopDepartureId int, @StopArrivalId int, @TimeArrival int, @TimeDeparture int,
11     @LengthTime int, @LengthDistance int, @CalendarId int, @VertexDepartureId int, @VertexArrivalId int
12     FETCH NEXT FROM cur INTO @TripId, @StopDepartureId, @StopArrivalId, @TimeArrival, @TimeDeparture,
13     @LengthTime, @LengthDistance, @CalendarId, @VertexDepartureId, @VertexArrivalId
14     WHILE @@FETCH_STATUS = 0
15     BEGIN
16         declare @dateStart datetime = (SELECT [Start_date] FROM [gtfs].[Calendar] WHERE Id = @CalendarId)
17         declare @dateEnd datetime = (SELECT [End_date] FROM [gtfs].[Calendar] WHERE Id = @CalendarId)
18         WHILE (@dateStart <= @dateEnd) BEGIN
19             IF EXISTS (SELECT * FROM [gtfs].[Calendar_dates] WHERE [CalendarId] = @CalendarId AND
20             [gtfs].[Calendar_dates].Date = @dateStart) BEGIN
21                 IF 1 = (SELECT Exception_Type FROM [gtfs].[Calendar_dates] WHERE [CalendarId] = @CalendarId AND
22                 [gtfs].[Calendar_dates].Date = @dateStart) BEGIN
23                     INSERT INTO [dbo].[Edges] (TripId, StopDepartureId, StopArrivalId, TimeDeparture, TimeArrival,
24                     LengthTime, LengthDistance, Calendar, VertexDepartureId, VertexArrivalId)
25                     VALUES(@TripId, @StopDepartureId, @StopArrivalId, @TimeDeparture, @TimeArrival, @LengthTime,
26                     @LengthDistance, @dateStart, @VertexDepartureId, @VertexArrivalId)
27                 END
28             END
29             ELSE
30             BEGIN
31                 DECLARE @DAY INT = DATEPART (DW, @dateStart)
32                 IF @DAY = 1 AND 1 = (SELECT Sunday FROM [gtfs].[Calendar] WHERE Id = @CalendarId) BEGIN
33                     INSERT INTO [dbo].[Edges] (TripId, StopDepartureId, StopArrivalId, TimeDeparture, TimeArrival,
34                     LengthTime, LengthDistance, Calendar, VertexDepartureId, VertexArrivalId)
35                     VALUES(@TripId, @StopDepartureId, @StopArrivalId, @TimeDeparture, @TimeArrival, @LengthTime,
36                     @LengthDistance, @dateStart, @VertexDepartureId, @VertexArrivalId)
37                 END
38                 ELSE IF @DAY = 2 AND 1 = (SELECT Monday FROM [gtfs].[Calendar] WHERE Id = @CalendarId) BEGIN
39                     INSERT INTO [dbo].[Edges] (TripId, StopDepartureId, StopArrivalId, TimeDeparture, TimeArrival,
40                     LengthTime, LengthDistance, Calendar, VertexDepartureId, VertexArrivalId)
41                     VALUES(@TripId, @StopDepartureId, @StopArrivalId, @TimeDeparture, @TimeArrival, @LengthTime,
42                     @LengthDistance, @dateStart, @VertexDepartureId, @VertexArrivalId)
43                 END
44                 ELSE IF @DAY = 3 AND 1 = (SELECT Tuesday FROM [gtfs].[Calendar] WHERE Id = @CalendarId) BEGIN
45                     INSERT INTO [dbo].[Edges] (TripId, StopDepartureId, StopArrivalId, TimeDeparture, TimeArrival,
46                     LengthTime, LengthDistance, Calendar, VertexDepartureId, VertexArrivalId)
47                     VALUES(@TripId, @StopDepartureId, @StopArrivalId, @TimeDeparture, @TimeArrival, @LengthTime,
48                     @LengthDistance, @dateStart, @VertexDepartureId, @VertexArrivalId)
49                 END
```

Slika 6.12: Procedura *Step3* - prvi dio.

```
50                  ELSE IF @DAY = 4 AND 1 = (SELECT Wednesday FROM [gtfs].[Calendar] WHERE Id = @CalendarId) BEGIN
51                      INSERT INTO [dbo].[Edges] (TripId, StopDepartureId, StopArrivalId, TimeDeparture, TimeArrival,
52                      LengthTime, LengthDistance, Calendar, VertexDepartureId, VertexArrivalId)
53                      VALUES(@TripId, @StopDepartureId, @StopArrivalId, @TimeDeparture, @TimeArrival, @LengthTime,
54                      @LengthDistance, @dateStart, @VertexDepartureId, @VertexArrivalId)
55                  END
56                  ELSE IF @DAY = 5 AND 1 = (SELECT Thursday FROM [gtfs].[Calendar] WHERE Id = @CalendarId) BEGIN
57                      INSERT INTO [dbo].[Edges] (TripId, StopDepartureId, StopArrivalId, TimeDeparture, TimeArrival,
58                      LengthTime, LengthDistance, Calendar, VertexDepartureId, VertexArrivalId)
59                      VALUES(@TripId, @StopDepartureId, @StopArrivalId, @TimeDeparture, @TimeArrival, @LengthTime,
60                      @LengthDistance, @dateStart, @VertexDepartureId, @VertexArrivalId)
61                  END
62                  ELSE IF @DAY = 6 AND 1 = (SELECT Friday FROM [gtfs].[Calendar] WHERE Id = @CalendarId) BEGIN
63                      INSERT INTO [dbo].[Edges] (TripId, StopDepartureId, StopArrivalId, TimeDeparture, TimeArrival,
64                      LengthTime, LengthDistance, Calendar, VertexDepartureId, VertexArrivalId)
65                      VALUES(@TripId, @StopDepartureId, @StopArrivalId, @TimeDeparture, @TimeArrival, @LengthTime,
66                      @LengthDistance, @dateStart, @VertexDepartureId, @VertexArrivalId)
67                  END
68                  ELSE IF @DAY = 7 AND 1 = (SELECT Saturday FROM [gtfs].[Calendar] WHERE Id = @CalendarId) BEGIN
69                      INSERT INTO [dbo].[Edges] (TripId, StopDepartureId, StopArrivalId, TimeDeparture, TimeArrival,
70                      LengthTime, LengthDistance, Calendar, VertexDepartureId, VertexArrivalId)
71                      VALUES(@TripId, @StopDepartureId, @StopArrivalId, @TimeDeparture, @TimeArrival, @LengthTime,
72                      @LengthDistance, @dateStart, @VertexDepartureId, @VertexArrivalId)
73                  END
74              END
75              SET @dateStart = DATEADD(day, 1, @dateStart)
76          END
77          FETCH NEXT FROM cur INTO @TripId, @StopDepartureId, @StopArrivalId, @TimeArrival, @TimeDeparture, @LengthTime,
78          @LengthDistance, @CalendarId, @VertexDepartureId, @VertexArrivalId
79      END
80  CLOSE cur
81  DEALLOCATE cur
82  END
83  EXEC [Step4]
84  GO
```

Slika 6.13: Procedura *Step3* - drugi dio.

U *Step4* proceduri (slika 6.14) vrši se iteracija kroz zapise *Edges* tablice. Pristupa se svakom zapisu brida koji povezuje dva vrha iste stanice. *Id* vrijednost stanice se zajedno sa datumom upisuje u posebne varijable (linija 10). Te se varijable koriste kao ulazne prilikom pozivanja funkcije *GetTransferEdges*. Rezultat izvršavanja funkcije dodaje se u tablicu *Edges* u obliku novog zapisa (linije 13-18). Ovaj postupak se izvodi sve do kraja izvršavanja iteracije kroz tablicu *Edges*.

```
1   CREATE PROC [dbo].[Step4]
2   AS
3   DECLARE cur SCROLL CURSOR FOR
4       SELECT StopDepartureId 'StopId', Calendar FROM [dbo].[Edges]
5       UNION
6       SELECT StopArrivalId 'StopId', Calendar FROM [dbo].[Edges]
7   OPEN cur
8       DECLARE @StopId int, @Calendar date
9
10      FETCH NEXT FROM cur INTO @StopId, @Calendar
11      WHILE @@FETCH_STATUS = 0
12      BEGIN
13          INSERT INTO dbo.Edges (TripId, StopDepartureId, StopArrivalId,
14          TimeDeparture, TimeArrival, LengthTime, LengthDistance, Calendar,
15          VertexDepartureId, VertexArrivalId )
16          SELECT TripId, StopDepartureId, StopArrivalId, TimeDeparture,
17          TimeArrival, LengthTime, 0, Calendar, VertexDepartureId, VertexArrivalId
18          FROM dbo.GetTransferEdges(@StopId, @Calendar)
19
20          FETCH NEXT FROM cur INTO @StopId, @Calendar
21      END
22  CLOSE cur
23  DEALLOCATE cur
24  GO
```

Slika 6.14: Procedura *Step4*.

Slika 6.15 prikazuje strukturu *GetTransferEdges* funkcije. Primjena ove funkcije je izračunati vrijednosti koje će predstavljati brid prijenosa u obliku novog zapisa u tablici *Edges*. Vrijednosti koje ova funkcija vraća predstavljaju sva polja tablice *Edges* (linije 10-18). U funkciji je prilikom vraćanja izračunate vrijednosti uveden uvjet da putnik može čekati na presjedanje najviše 1800 sekundi, odnosno 30 minuta (vrijeme prijenosa). Polje *LengthTime* tablice *Edges* predstavlja vrijeme prijenosa, stoga se unose samo zapisi čija je vrijednost tog polja manja ili jednaka 1800.

```
CREATE FUNCTION [dbo].[GetTransferEdges]
(
    @StopId int,
    @Calendar date
)
RETURNS TABLE
AS
RETURN
SELECT
    re1.TripId 'TripId',
    re2.StopDepartureId 'StopDepartureId',
    re2.StopDepartureId 'StopArrivalId',
    re1.VertexArrivalId 'VertexDepartureId',
    re2.VertexDepartureId 'VertexArrivalId',
    re1.TimeArrival 'TimeDeparture',
    re2.TimeDeparture 'TimeArrival',
    (re2.TimeDeparture - re1.TimeArrival) 'LengthTime',
    re1.Calendar 'Calendar'
FROM [dbo].[Edges] re1
INNER JOIN [dbo].[Edges] re2 ON re1.StopArrivalId = re2.StopDepartureId
WHERE ((re2.StopArrivalId = @StopId OR re2.StopDepartureId = @StopId) AND
re2.StopArrivalId <> re2.StopDepartureId) AND
((re1.StopArrivalId = @StopId OR re1.StopDepartureId = @StopId) AND
re1.StopArrivalId <> re1.StopDepartureId) AND
(re1.TripId <> re2.TripId) AND
re1.Calendar = re2.Calendar AND
re1.Calendar = @Calendar AND
((re2.TimeDeparture - re1.TimeArrival) <= 1800 AND
(re2.TimeDeparture - re1.TimeArrival) >= 0)
GO
```

Slika 6.15: Funkcija *GetTransferEdges*.

Izvršavanjem opisanih procedura i funkcije u tablicu *Edges* se upisuju svi postojeći bridovi, a u tablicu *Vertices* se upisuju svi postojeći vrhovi u bazi podataka. Ovakav pristup je odabran jer se radi o velikoj količini podataka bez obzira što planer putovanja obuhvaća samo područje grada Zadra. Na ovaj način se prilikom izvršavanja algoritma stvara mogućnost pristupanja svim potrebnim podacima iz samo dvije tablice (*Edges* i *Vertices*).

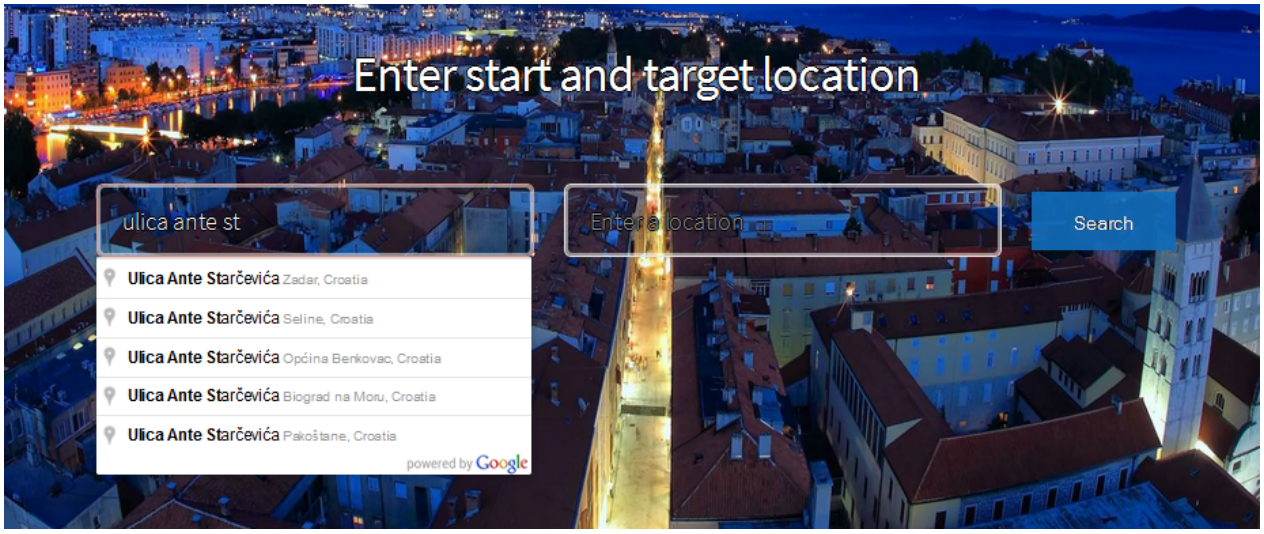
6.5 Priprema podataka za algoritam

Prilikom početka unosa početnih i ciljnih lokacija implementirani kod omogućuje automatsku ponudu adresa za svaku od lokacija (dodana je mogućnost da se prvotno nude adrese iz šireg područja grada Zadra). Programski kod koji omogućuje ovakvo ponašanje aplikacije prikazan je na slici 6.16. U varijablu *defaultBounds* upisuju se geografska širina i

dužina gornje i donje zemljopisne granice koje *Google* servis uzima kao primarne podatke za automatsku ponudu adresa. Time se uvodi spomenuta mogućnost da se prvotno nude adrese iz šireg područja grada Zadra. Uz određene zemljopisne granice, uvedena je restrikcija da automatska ponuda adresa obuhvaća samo hrvatska mjesta (*componentRestrictions: { country: 'hr' }*).

```
24      //kreiranje granica za autocomplete adresa, gornja granica predstavlja SW koordinate, a donja NE koordinate
25      var defaultBounds = new google.maps.LatLngBounds(
26          new google.maps.LatLng(43.80678315, 14.41680908),
27          new google.maps.LatLng(44.55916342, 15.87799072));
28
29      // Unos početne adrese na Home stranici, parametri koji se unose su id elementa stranice, tip autocomplete-a je adresa,
30      //prethodno kreirane koordinantne granice i restrikcija za autocomplete da moraju biti HR mjesta
31      autocompletePoc = new google.maps.places.Autocomplete(
32      (document.getElementById('autocompletePoc')),
33      {
34          types: ['geocode'],
35          bounds: defaultBounds,
36          componentRestrictions: { country: 'hr' }
37      });
38
39      //Unos Ciljne adrese na Home stranici
40      autocompleteCilj = new google.maps.places.Autocomplete(
41      (document.getElementById('autocompleteCilj')),
42      {
43          types: ['geocode'],
44          bounds: defaultBounds,
45          componentRestrictions: { country: 'hr' }
46      });
```

Slika 6.16: Programski kod za automatsku ponudu adresa.



Slika 6.17: Prikaz automatske ponude adresa početne i ciljne stanice.

Nakon odabira, početna i ciljna adresa se prvotno konvertiraju u geografske vrijednosti (geografska širina i dužina). Slika 6.18 prikazuje programske metode kojima se učitavaju

vrijednosti adresa početne i ciljne adrese te se stavljaju u varijable *autocompletePoc* i *autocompleteCilj*. Tada se dvije adrese dijele na geografske širine i dužine (varijable *temp1* i *temp2*).

```
49      //Konvertiranje upisanih adresa u geografsku širinu i dužinu
50      google.maps.event.addListener(autocompletePoc, 'place_changed', function () {
51          var place = autocompletePoc.getPlace();
52          if (!place.geometry) {
53              return;
54          }
55
56          var temp1 = place.geometry.location.lat();
57          var temp2 = place.geometry.location.lng();
58
59          console.log(temp1);
60          console.log(temp2);
61
62          $("#PocetnaLat").val(new Intl.NumberFormat(uiCulture).format(temp1));
63          $("#PocetnaLon").val(new Intl.NumberFormat(uiCulture).format(temp2));
64
65      });
66      google.maps.event.addListener(autocompleteCilj, 'place_changed', function () {
67          var place = autocompleteCilj.getPlace();
68          if (!place.geometry) {
69              return;
70          }
71
72          var temp1 = place.geometry.location.lat();
73          var temp2 = place.geometry.location.lng();
74
75          console.log(temp1);
76          console.log(temp2);
77
78          $("#CiljnaLat").val(new Intl.NumberFormat(uiCulture).format(temp1));
79          $("#CiljnaLon").val(new Intl.NumberFormat(uiCulture).format(temp2));
80
81      });
```

Slika 6.18: Kod za učitavanje adrese početne i ciljne lokacije.

Nakon što se geografske vrijednosti početne i ciljne lokacije izračunaju, potrebno ih je konvertirati u odgovarajući *DbGeography* format. Vrijednosti u tom obliku se upisuju u varijable *coordPoc* i *coordCilj*, kao što je vidljivo na slici 6.19. Taj oblik podatka služi kao priprema za pronalazak dvije početne i dvije ciljne stanice u bazi podataka koje se upisuju u varijable *pocetakStanice* i *ciljStanice* u obliku liste. Traže se dvije početne i dvije ciljne stanice iz razloga što se na području Zadra direktno povezuju najviše dva moda putovanja,

stoga se smatra da je to dovoljno.

```
27              //DOHVAĆANJE STARTNIH I CILJNIH STOPOVA
28              // konvertiranje upisane pocetne sirine i duzine u DbGeography format
29              var coordPoc = Geography.CreatePoint(PocetnaLon, PocetnaLat);
30
31              // konvertiranje upisane ciljne sirine i duzine u DbGeography format
32              var coordCilj = Geography.CreatePoint(CiljnaLon, CiljnaLat);
33
34              // dohvaćanje iz baze podataka 2 najbližih početnih stanica unutar radijusa od odabrane početne lokacije
35              var pocetakStanice = db.Stops
36                  .Where(x => x.Point.Distance(coordPoc) < radius)
37                  .Select(x => new { x.Id, Distance = x.Point.Distance(coordPoc).Value }).OrderBy(x => x.Distance).Take(2).ToList();
38
39              // dohvaćanje iz baze podataka 2 najbližih ciljnih stanica unutar radijusa od odabrane ciljne lokacije
40              var ciljStanice = db.Stops
41                  .Where(x => x.Point.Distance(coordCilj) < radius)
42                  .Select(x => new { x.Id, Distance = x.Point.Distance(coordCilj).Value }).OrderBy(x => x.Distance).Take(2).ToList();
```

Slika 6.19: Kod za pronalazak dvije najbliže stanice u bazi podataka u odnosu na unesenu početnu i ciljnu lokaciju.

Kod na slici 6.19 poziva klasu *Geography* koja sadrži tri metode. *CreatePoint* metoda konvertira vrijednosti geografske širine i dužine u *DbGeography* format. Ostale dvije metode (*getLatitude* i *getLongitude*) izvlače geografsku širinu i dužinu iz *DbGeography* vrijednosti. Klasa *Geography* prikazana je na slici 6.20.

```
10  public class Geography
11  {
        2 references
12      public static DbGeography CreatePoint(double longitude, double latitude)
13      {
14          var cultEn = CultureInfo.CreateSpecificCulture("en-GB");
15          return DbGeography.PointFromText(String.Format("POINT({0} {1})", longitude.ToString(cultEn), latitude.ToString(cultEn)), 4326);
16      }
17
        0 references
18      public static double? getLatitude(DbGeography point)
19      {
20          return point.Latitude;
21      }
22
        0 references
23      public static double? getLongitude(DbGeography point)
24      {
25          return point.Longitude;
26      }
27  }
```

Slika 6.20: *Geography* klasa.

Nakon dohvaćanja najbližih stanica iz baze podataka, potrebno je izračunati ukupno vrijeme koje je prošlo od ponoći do unesenog željenog vremena planiranja putovanja. Vrijeme se mjeri u sekundama, a izvršava se pozivanjem metode *GetSecondsForDate* kao što je vidljivo na slici 6.21. Izračunata vrijednost upisuju se u varijablu *seconds*. Potom je potrebno iz baze podataka učitati generirani graf, odnosno zapise tablica *Edges* i *Vertices*. To se obavlja na način da se pozove metoda *GenerateTimeIndependentGraph* sa ulaznim vrijednostima *datetime* (trenutni datum i vrijeme) i *seconds*. Izračunato vrijeme je bitno

jer omogućuje da se iz baze podataka dohvaćaju samo vožnje koje su dostupne nakon vremena odabira planiranja putovanja, odnosno koje nisu istekle.

```
45              //RAČUNANJE BROJA SEKUNDI OD PONOĆI
46              var seconds = TimeConverter.GetSecondsForDate(datetime);
47
48              //DOHVAĆANJE GENERIRANOG GRAFA IZ BAZE PODATAKA
49              var graph = GenerateTimeIndependentGraph(datetime, seconds);
```

Slika 6.21: Kod za računanje vremena od ponoči te dohvaćanje generiranog grafa.

Na slici 6.22 prikazana je metoda *GetSecondsForDate* koja kao ulaznu vrijednost prima željeno vrijeme, a kao izlaznu vrijednost vraća isteklo vrijeme u sekundama od ponoći do unesenog vremena.

```
23          public static int GetSecondsForDate(DateTime dateTime)
24          {
25              return dateTime.Hour * 3600 + dateTime.Minute * 60 + dateTime.Second;
26          }
```

Slika 6.22: *GetSecondsForDate* metoda.

Metoda *GenerateTimeIndependentGraph* prikazana na slici 6.23 kao ulazne vrijednosti prima željeno vrijeme planiranja putovanja te isteklo vrijeme od ponoći mjereno u sekundama. Metoda prvotno generira varijablu *graph* koja je tipa *TimeIndependentGraph*. Ovom inicijalizacijom se varijabla *graph* definira kao skup vrhova i bridova koji će sadržavati podatke tablica *Vertices* i *Edges* iz baze podataka. Potom se u varijablu *date* upisuje datum željenog vremena planiranja putovanja, a u listu *Edges* upisuju se svi zapisi iz tablice *Edges* koji se odnose na željeni datum planiranja putovanja te čije vrijeme vožnje još nije isteklo (vrijednost *TimeDeparture* je veća od isteklog vremena od ponoći). Nakon toga se dohvaćaju sve *VertexDepartureId* i *VertexArrivalId* vrijednosti prethodno upisanih zapisa u listu *Edges*, pronalaze se zapisi tablice *Vertices* čija *Id* polja imaju te vrijednosti te se ti se zapisi upisuju u listu *Vertices*. Ove dvije liste se dodjeljuju varijabli *graph* koja predstavlja generirani graf i kao takvu je metoda vraća kao izlaznu vrijednost.

```
139         private TimeIndependentGraph GenerateTimeIndependentGraph(DateTime datetime, int seconds)
140                 {
141                     TimeIndependentGraph graph;
142                     var date = datetime.Date;
143 
144                     var edges = db.Edges.Where(x => x.Calendar == date && x.TimeDeparture > seconds).ToList();
145 
146                     var verticesDep = edges.Select(x => x.VertexDepartureId).ToList();
147                     var verticesArr = edges.Select(x => x.VertexArrivalId).ToList();
148                     verticesDep.AddRange(verticesArr);
149 
150                     graph = new TimeIndependentGraph
151                     {
152                         Vertices = db.Vertices.Where(x => verticesDep.Contains(x.Id)).ToList(),
153                         Edges = edges
154                     };
155 
156                     return graph;
157                 }
158 
```

Slika 6.23: *GenerateTimeIndependentGraph* metoda.

Slika 6.24 prikazuje kod za pronalazak jednog startnog vrha i *Id* vrijednosti svih ciljnih vrhova iz skupine prethodno dobivenih stanica iz baze podataka. Proces pronalaska startnog vrha počinje tako da se u listu *edgesPoc* upisuju zapisi bridova varijable *graph* u kojima je vrijednost *StopDepartureId* ista kao jedna od vrijednosti *Id* prethodno kreirane liste *pocetakStanice*. Potom se obavlja iteracija po svakom bridu liste *edgesPoc*. Za svaki brid računa se trajanje šetnje od unesene početne lokacije do lokacije polazne stanice brida (*StopDepartureId*). Trajanje šetnje upisuje se u varijablu *distance* i oduzima se od vrijednosti vremena polaska (*TimeDeparture*) polaznog vrha na bridu. Potom se vrši provjera je li vrijeme polaska veće od proteklog vremena od ponoći. Ukoliko jest, šetnja se množi s koeficijentom 1,5 jer se pretpostavlja da je putniku hodanje do stanice od manjeg interesa. Na kraju se u varijablu *temp* upisuje zbroj prethodno izračunate razlike između vremena polaska i šetnje te šetnje pomnožene s koeficijentom. Ovdje varijabla *temp* predstavlja relativnu udaljenost od prethodno unesene početne lokacije do polaznog vrha trenutnog brida. Ukoliko je relativna udaljenost manja od relativne udaljenosti do polaznog vrha prethodnog brida, uzima se polazni vrh tog brida kao početni vrh za izvođenje algoritma i upisuje se u varijablu *start*. U slučaju ako nema valjanog polaznog vrha vraća se prazna lista rješenja kao rezultat.

Proces pronalaska ciljnih vrijednosti započinje upisom *Id* vrijednosti od prethodno pronađene dvije stanice u varijablu *ciljStaniceIds*. Potom se vrši upit za *Id* vrijednostima

svih dolaznih vrhova (*VertexArrivalId*) koji pripadaju pojedinim bridovima grafa *graph* čije su vrijednosti *StopArrivalId* iste kao vrijednosti varijable *ciljStaniceIds*. Lista dobivenih *Id* vrijednosti dolaznih vrhova upisuje se u varijablu *verticesCiljeviIds*. Samo pronalazak ovih vrijednosti je dovoljan jer postoji više vrhova koji pripadaju istoj ciljnoj stanici, a ukoliko algoritam pronađe samo jedan takav vrh on je s time uspješno završio potragu.

```
51              //PRONALAZAK JEDNOG STARTNOG VRHA IZ SKUPINE PRETHODNO DOBIVENIH POČETNIH STANICA
52              Vertices start = null;
53              var startStaniceIds = pocetakStanice.Select(x => x.Id);
54              var startStanicaId = startStaniceIds.FirstOrDefault();
55              var relativeDistancePoc = double.MaxValue;
56              var edgesPoc = graph.Edges.Where(x => startStaniceIds.Contains(x.StopDepartureId)).ToList();
57              foreach (var edge in edgesPoc)
58              {
59                  var distance = pocetakStanice.First(x => x.Id == edge.StopDepartureId).Distance;
60                  var timeDep = edge.TimeDeparture - distance; //varijabla distance predstavlja šetnju
61                  if (timeDep < seconds) continue;
62                  var temp = (distance * 1.5) + timeDep;
63
64                  if (!(temp < relativeDistancePoc)) continue;
65                  relativeDistancePoc = temp;
66                  start = edge.VertexDeparture;
67              }
68              //ukoliko nema rješenja vrati praznu listu
69              if (start == null)
70                  return new List<Edges>();
71
72              //PRONALAZAK Id VRIJEDNOSTI SVIH CILJNIH VRHOVA PRETHODNO DOBIVENIH CILJNIH STANICA
73              var ciljStaniceIds = ciljStanice.Select(x => x.Id);
74              var verticesCiljeviIds = graph.Edges.Where(x => ciljStaniceIds.Contains(x.StopArrivalId)).
75                                       Select(x => x.VertexArrivalId).Distinct().ToList();
76
```

Slika 6.24: Postupak pronalaska početnog i ciljnih vrhova iz skupine početnih i ciljnih stanica.

Izvršavanjem gore opisanih kodova planer putovanja ima pripremljene sve potrebne podatke za izvođenje algoritma za planiranje putovanja. Poznati su početni i ciljni vrhovi, a u varijablu *graph* iz tablica *Edges* i *Vertices* upisana je lista bridova i vrhova koji se odnose na traženi datum i čija je vrijednost vremena polaska veća od vrijednosti traženog vremena u sekundama.

6.6 Izvođenje algoritma

Algoritam koji je odabran za planiranje putovanja u *Web* aplikaciji je vremenski neovisan *Dijsktra* algoritam. U svrhu boljeg razumjevanja u nastavku izvođenje *Dijsktra* algoritma se

može podijeliti na 4 koraka:

1. Odrediti početni vrh u grafu i dodjeliti mu vrijednost težine jednaku 0.
2. Odrediti ciljni vrh u grafu i dodjeliti mu zajedno s ostalim vrhovima u grafu vrijednost težine jednaku maksimalnoj vrijednosti.
3. Iterirati kroz graf i odrediti svaki sljedeći vrh koji je najbliži prethodnom (čija je vrijednost težine najmanja) sve do ciljnog vrha.
4. Iterirati kroz graf u svrhu pronalaska bridova koji povezuje prethodno određene susjedne vrhove od ciljnog do početnog vrha i odrediti slijed bridova čiji je zbroj težina najmanji.

Slika 6.25 prikazuje *Dijkstra* algoritam planera putovanja. Prvotno se u varijablu *redCekanja* upisuje lista svih vrhova grafa *graph*. Potom se vrhu *start* dodjeljuje vrijednost težine *Weight* jednaku 0 (svi ostali vrhovi u grafu imaju inicijalno postavljene vrijednosti težina jednake maksimalnoj vrijednosti *double.MaxValue*). U varijablu *završetakAlgoritma* upisuje se vrijednost *false*, a varijabla se koristi kao provjera je li algoritam završio izvođenje. Vrijednost varijable *vertexGoal* koristi se kao polazni vrh pri izvođenju četvrtog koraka algoritma. S ovim dijelom koda obavljen je prvi korak izvođenja algoritma.

While petljom na liniji 84 slike 6.25 započinje se izvođenje algoritma koji će se izvoditi dok se *redCekanja* ne isprazni ili dok u *završetakAlgoritma* ne bude upisana vrijednost *true*. U varijablu *u* se upisuje sljedeći vrh koji ima najmanju težinu (linija 86). To se obavlja pozivom metode *IzvuciNajboljeg*. Tada se vrši još jedna iteracija kroz sve odlazne bridove koji pripadaju tom vrhu (linija 87, gledaju se odlazni bridovi jer se izvođenje algoritma obavlja prema ciljnom vrhu). U toj se iteraciji za svaki brid provjerava je li *Id* vrijednost dolaznog vrha tog brida jednaka *Id* vrijednosti vrha *u*. Ako je jednaka, tada se u varijablu *v* upisuju vrijednosti polaznog vrha trenutno gledanog brida (*VertexDeparture*), a ukoliko nije, u varijablu *v* se upisuju vrijednosti dolaznog vrha trenutno gledanog brida (*VertexArrival*). Potom se vrši još jedna provjera je li težina vrha *v* veća od zbroja težine vrha *u* i vremena putovanja (*LengthTime*) trenutno gledanog brida (linija 91). Ako je veća, vrijednost težine vrha *v* se mijenja u taj zbroj i kao vrijednost pokazatelja prethodnog vrha (*Previous*) stavlja

se vrijednost vrha *u*. Na kraju se vrši provjera je li postoji vrijednost u prethodno kreiranoj listi *verticesCiljeviIds* koja je jednaka vrijednosti *Id* vrha *v* (linija 97). Ako ne postoji jednaka vrijednost, preskače se na sljedeći brid u iteraciji. Ako postoji jednaka vrijednost, vrijednosti vrha *v* se upisuju u varijablu *vertexGoal*, završava se iteracija po odlaznim bridovima, u varijablu *završetakAlgoritma* se upisuje *true* te se ponovno poziva metoda *IzvuciNajboljeg* ukoliko lista *redCekanja* još uvijek sadrži vrhove. S ovim dijelom koda obavljeni su drugi i treći korak algoritma.

U nastavku izvođenja koda deklarira se varijabla *putBridova* koja predstavlja listu bridova koji povezuju susjedne vrhove najmanjih težina (linija 103). Potom se u varijablu *trenutniVertex* upisuju vrijednosti vrha *vertexGoal*. Nakon toga se započinje izvršavanje *do-while* petlje u kojoj se izvršava četvrti korak algoritma (linija 105). Svakom iteracijom petlje u pomoćnu varijablu *edge* se upisuje vrijednost brida koji spaja trenutni vrh (*trenutniVertex*) sa njegovim prethodnim vrhom na putu (*trenutniVertex.Previous*). Potom se vrijednost pomoćne varijable *edge* dodaje u listu *putBridova* (linija 110). Izvođenje *do-while* petlje se izvršava sve dok *StopId* vrijednost trenutno gledanog vrha nije jednaka kao jedna od *Id* vrijednosti prethodno pronađenih početnih stanica (*pocetakStanice*). Završetkom izvođenja ovog dijela koda obavljen je zadnji korak algoritma.

Metoda *IzvuciNajboljeg* koja računa podatke za treći korak algoritma prikazana je na slici 6.26. Metoda kao ulaznu vrijednost prima listu vrhova grafa *graph* (varijabla *red*), a kao rješenje vraća jedan zapis vrha. U metodi se potom vrši iteracija po listi vrhova grafa i pronalazi se onaj zapis vrha čija je vrijednost težine *Weight* najmanja. Pritom se zapis tog vrha vraća kao rješenje metode te se uklanja iz liste vrhova grafa. Na taj način će kod sljedećeg pozivanja ove metode ulazna vrijednost *red* sadržavati jedan zapis manje.

```
77          //IZVOĐENJE DIJSKTRA ALGORITMA
78          //1. korak
79          var redCekanja = graph.Vertices.ToList();
80          start.Weight = 0;
81          var završetakAlgoritma = false;
82          Vertices vertexGoal = null;
83          //2. i 3. korak
84          while (redCekanja.Count > 0 && !završetakAlgoritma)
85          {
86              var u = IzvuciNajboljeg(redCekanja);
87              foreach (var edge in u.DepartureEdges)
88              {
89                  var v = edge.VertexArrival.Id == u.Id ? edge.VertexDeparture : edge.VertexArrival;
90
91                  if (v.Weight > u.Weight + edge.LengthTime)
92                  {
93                      v.Weight = u.Weight + edge.LengthTime;
94                      v.Previous = u;
95                  }
96
97                  if (!verticesCiljeviIds.Contains(v.Id)) continue;
98                  vertexGoal = v;
99                  završetakAlgoritma = true;
100             }
101         }
102         //4. korak
103         var putBridova = new List<Edges>();
104         Vertices trenutniVertex = vertexGoal;
105         do
106         {
107             if (trenutniVertex == null) break;
108
109             var edge = trenutniVertex.ArrivalEdges.First(x => x.VertexDeparture.Id == trenutniVertex.Previous.Id);
110             putBridova.Insert(0, edge);
111             trenutniVertex = trenutniVertex.Previous;
112         } while (trenutniVertex.StopId != startStanicaId);
```

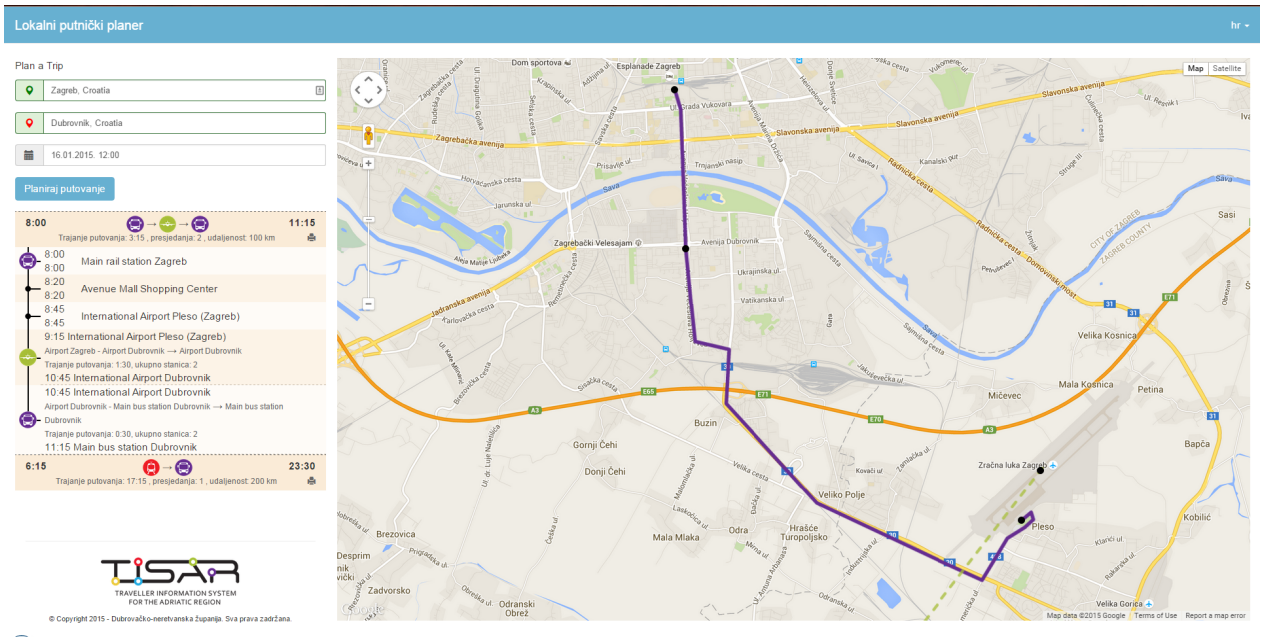
Slika 6.25: Kod za izvršavanje *Dijkstra* algoritma.

```
156         private Vertices IzvuciNajboljeg(List<Vertices> red)
157         {
158             double min = double.MaxValue;
159             Vertices najbolji = null;
160             foreach (var v in red)
161             {
162                 if (v.Weight < min)
163                 {
164                     min = v.Weight;
165                     najbolji = v;
166                 }
167             }
168             red.Remove(najbolji);
169             return najbolji;
170         }
```

Slika 6.26: Metoda *IzvuciNajboljeg*.

Podaci dobivenih zapisa bridova u varijabli *putBridova* dovoljni su za prikaz puta u obliku linije (*Polyline*) na *Google* karti. Daljnjim izvođenjem koda na temelju izračunatih vrijednosti prikazuje se rješenje na stranici karte. Na slici 6.27 je prikazan ogledni primjer kompletne izlazne vrijednosti koju bi planer putovanja daljnom implementacijom koda

trebao prikazivati.



Slika 6.27: Ogledni primjer konačnog rješenja planera putovanja.

7 Zaključak

S obzirom na geografsko područje može se zaključiti da planer putovanja izrađen u praktičnom dijelu rada ima lokalne karakteristike iz razloga što obuhvaća područje grada Zadra i okolice. S obzirom da obuhvaća tri vrste prometnih mreža (mreža javnog cestovnog prijevoza, pomorska mreža i zrakoplovna mreža) planer putovanja je također multimodalan.

Zbog potreba planera odabran je model koji ima vremenski neovisne karakteristike. To znači da su težine bridova, odnosno vremena putovanja, konstantna. Vremenska neovisnost je manje realna i izračun putovanja daje manju točnost u odnosu na vremenski ovisne modele, ali je implementacija daleko jednostavnija i brža. Prikupljanje tranzitnih podataka za planer putovanja u praktičnom dijelu rada obavljen je ručno korištenjem GTFS editora (`http://www.dubrovnik-travel-planner.com`). Iako je to daleko dugotrajniji postupak od automatskog prikupljanja podataka to je jedini mogući način, jer za ostale postupke trebaju se stvoriti tehnički preduvjeti od strane prijevoznika. Algoritam koji je odabran za implementaciju u planer putovanja je *Dijsktra* algoritam. Za dodatnim tehnikama ubrzanja nema potrebe jer se radi o prototipu.

Kako bi napravljen prototip za potrebe ovog diplomskog rada mogao biti u potpunosti funkcionalan potrebno je doraditi još nekoliko stvari. Prva je prikupljanje potpunih podataka o linijama i rasporedima njihovih vožnji. Nakon toga bi bilo potrebno sam algoritam ubrzati nekom od opisanih tehnika ubrzanja. Pri postupku generiranja grafa također ima prostora za napredak jer se sada koriste kursori i generira se potpuni graf bez obzira je li bilo promjena ili ne na pojedinoj liniji ili njenom važećem kalendaru. Također je moguće pri samoj tehničkoj implementaciji ubrzati postupak. Kod dohvaćanja dijela težinskog grafa koji se predaje algoritmu moguće je ubrzati upit indeksiranjem tablice *Edges* te je još moguće spremiti u radnu memoriju dio grafa odnosno rješenja. Zaključno treba detaljnije prezentirati krajnje rezultate planera krajnjem korisniku. Prototip u ovom radu predstavlja kostur sa svim elementima planera koji je još potrebno dodatno nadograditi.

Literatura

[1] *Add your transit data to google maps*. http://maps.google.com/help/maps/mapcontent/transit/participate.html, pristupljeno 1.4.2015.

[2] *Asp.net mvc overview*. https://msdn.microsoft.com/en-us/library/dd381412%28v=vs.108%29.aspx, pristupljeno 23.3.2015.

[3] *Cascading style sheets - wikipedia, the free encyclopedia*. http://hr.wikipedia.org/wiki/CSS, pristupljeno 22.3.2015.

[4] *Displaying transit data to users*. https://developers.google.com/transit/gtfs/examples/display-to-users, pristupljeno 30.3.2015.

[5] *Extended gtfs route types*. https://support.google.com/transitpartners/answer/3520902?hl=en, pristupljeno 21.3.2015.

[6] *Feed validator tool*. https://github.com/google/transitfeed/wiki/FeedValidator, pristupljeno 1.4.2015.

[7] *General transit feed specification reference*. https://developers.google.com/transit/gtfs/reference#fare_attributes_fields, pristupljeno 30.3.2015.

[8] *Hypertext markup language - wikipedia, the free encyclopedia*. http://hr.wikipedia.org/wiki/HTML, pristupljeno 22.3.2015.

[9] *Intermodal journey planner - wikipedia, the free encyclopedia*. http://en.wikipedia.org/wiki/Journey_planner, pristupljeno 21.3.2015.

[10] *Intermodal passenger transport in europe*. http://www.fgm.at/linkforum/docs/214/link_intermodality_brochure.pdf, pristupljeno 21.3.2015.

[11] *Internet bot - wikipedia, the free encyclopedia*. http://en.wikipedia.org/wiki/Internet_bot, pristupljeno 7.4.2015.

[12] *Javascript - wikipedia, the free encyclopedia*. http://en.wikipedia.org/wiki/JavaScript, pristupljeno 22.3.2015.

[13] *Kmlwriter tool*. https://github.com/google/transitfeed/wiki/KMLWriter, pristupljeno 1.4.2015.

[14] *Scheduleviewer tool*. https://github.com/google/transitfeed/wiki/ScheduleViewer, pristupljeno 1.4.2015.

[15] *Sql server management studio - wikipedia, the free encyclopedia*. https://en.wikipedia.org/wiki/SQL_Server_Management_Studio, pristupljeno 23.3.2015.

[16] *Multimodal trip planner system final evaluation report*, 2011. Dostupno na stranici: http://www.fta.dot.gov/documents/MMTPS_Final_Evaluation_05-24-2011(1).pdf.

[17] A. DICKSON, *Introduction to Graph Theory*, 2006. Dostupno na stranici: http://www.math.utah.edu/mathcircle/notes/MC_Graph_Theory.pdf.

[18] S. GANDAVARAPU, *Using google transit feed specification in travel modeling*, 2012. Dostupno na stranici: http://onlinepubs.trb.org/onlinepubs/conferences/2012/4thITM/Papers-R/0117-000113.pdf.

[19] H.-P. HALVORSEN, *Introduction to Visual Studio and C#*, Telemark University College, Porsgrunn, Norway, 2014.

[20] P. M. NICK KIZOOM, *A Transmodel based XML schema for the Google Transit Feed Specification - With a GTFS / Transmodel comparison*, Kizoom, 2008.

[21] T. PAYOR, *Multi-Modal Route Planning*, Universität Karlsruhe (TH), Germany, 2009. Dostupno na stranici: http://i11www.iti.uni-karlsruhe.de/extra/publications/p-mmrp-09.pdf.

[22] M. ROTH, *How google and portland's trimet set the standard for open transit data*. Dostupno na stranici: http://sf.streetsblog.org/2010/01/05/how-google-and-portlands-trimet-set-the-standard-for-open-transit-data/.

Popis kratica

GTFS	*General Transit Feed Specification*	Specifikacija za općenitu tranzitnu pohranu
GPS	*Global Positioning System*	Globalni pozicijski sustav
FIFO	*First In First Out*	Prvi došao prvi poslužen
UTC	*Coordinated Universal Time*	Koordinirano svjetsko vrijeme
EU	*European Union*	Europska Unija
SAD	*United States of America*	Sjedinjene Američke Države
RH	*Republic of Croatia*	Republika Hrvatska
REGL-CSPP	*Label constrained shortest path problem restricted to regular languages*	Problem najkraćeg puta sa uvjetnim oznakama ograničen na regularne jezike
PQ	*Priority queue*	Prioritetni red
ALT	*A* with landmarks algorithm*	Algoritam A* sa simbolima
UML	*Unified Modeling Language*	Ujedinjeni jezik za modeliranje
WGS	*World Geodetic System*	Svjetski geodetski sustav
URL	*Uniform Resource Locator*	Usklađeni lokator sadržaja
HTML	*HyperText Markup Language*	Jezik za označavanje hiperteksta
CSS	*Cascading Style Sheets*	Stranice za kaskadne stilove
API	*Application Programming Interface*	Aplikacijsko sučelje
ASP	*Active Server Pages*	Aktivne poslužiteljske stranice
MVC	*Model-View-Controller*	Model-pogled-upravljač
UI	*User Interface*	Korisničko sučelje
SQL	*Structured Query Language*	Jezik za strukturirane upite
IDE	*Integrated Development Environment*	Integrirano razvojno okruženje

Popis slika

Popis tablica

Popis algoritama

Metapodaci

Naslov rada: Izrada multimodalnog planera putovanja za područje grada Zadra

Naslov rada na engleskom jeziku:

Development of a Multi-modal Journey Planner for the Zadar's Area

Autor: Donat Galac

Mentor: prof. dr. sc. Tonči Carić

Neposredni voditelj: Mario Buntić, mag. ing. traff.

Povjerenstvo za obranu:

- izv. prof. dr. sc. Štefica Mrvelj (predsjednik)
- prof. dr. sc. Tonči Carić (mentor)
- doc. dr. sc. Edouard Ivanjko (član)
- prof. dr. sc. Hrvoje Gold (zamjena)

Ustanova koja je dodjelila akademski stupanj: Fakultet prometnih znanosti Sveučilišta u Zagrebu

Zavod: Zavod za informacijsko komunikacijski promet

Vrsta studija: sveučilišni

Naziv studijskog programa: Promet

Stupanj: diplomski

Akademski naziv: mag. ing. traff.

Datum obrane završnog rada: 25.9.2015.

Izjava o akademskoj čestitosti i suglasnosti

Izjavljujem i svojim potpisom potvrđujem da je DIPLOMSKI RAD isključivo rezultat mog vlastitog rada koji se temelji na mojim istraživanjima i oslanja se na objavljenu literaturu, a što pokazuju korištene bilješke i bibliografija. Izjavljujem da nijedan dio rada nije napisan na nedozvoljen način, odnosno da je prepisan iz necitiranog rada, te da nijedan dio rada ne krši bilo čija autorska prava. Izjavljujem, također, da 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 DIPLOMSKOG rada pod naslovom IZRADA MULTIMODALNOG PLANERA PUTOVANJA ZA PODRUČJE GRADA ZADRA, na internetskim stranicama i repozitoriju Fakulteta prometnih znanosti, Digitalnom akademskom repozitoriju (DAR) pri Nacionalnoj i sveučilišnoj knjižnici u Zagrebu.

U Zagrebu,
17.9.2015.

Student/ica:
Donat Galac, 0246026066