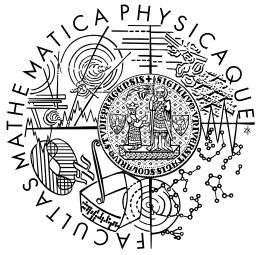


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

DIPLOMOVÁ PRÁCE

Tomáš Pokorný

Plánovač spojení ve městě

Katedra aplikované matematiky

Vedoucí diplomové práce:	Mgr. Martin Mareš, Ph.D.
Studijní program:	Informatika
Studijní obor:	Diskrétní modely a algoritmy

Praha 2017

Prohlašuji, že jsem tuto diplomovou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Praze dne 21. července 2017 Podpis autora

Na tomto místě bych rád poděkoval Mgr. Martinu Marešovi, Ph.D. za cenné rady a odbornou pomoc při konzultacích k této práci. Dále bych rád poděkoval svým kamarádům a speciálně Ing. Petru Pechovi za podporu a motivaci při psaní práce. Také bych rád poděkoval Anetě Šťastné za pomoc s finální úpravou práce.

Název práce: Plánovač spojení ve městě

Autor: Tomáš Pokorný

Katedra: Katedra aplikované matematiky

Vedoucí diplomové práce: Mgr. Martin Mareš, Ph.D., Katedra aplikované matematiky

Abstrakt: Cestování po městě je součástí každodenního života mnoha lidí. Zvolit správnou kombinaci pěší chůze a cestování hromadnou dopravou bývá náročné, obzvlášť v neznámých částech města. Zpracovali jsme veřejně dostupná data a vytvořili vyhledávač kombinovaných tras pěšky a hromadnou dopravou. Vyhledávač byl navržen s důrazem na možnost přizpůsobit hledané trasy preferencím uživatelů a lze jej použit i jako webovou aplikaci nebo sdílenou knihovnu.

Klíčová slova: vyhledávač spojení jízdní řád penalta

Title: Urban transport planner

Author: Tomáš Pokorný

Department: Department of Applied Mathematics

Supervisor: Mgr. Martin Mareš, Ph.D., Department of Applied Mathematics

Abstract: Travelling in the city is a part of everyday life for many people. It is sometimes difficult to choose the right combination of walking and public transport especially in unfamiliar parts of the city. We processed publicly available data and made a search engine for multimodal paths. The search engine was designed to be able to personalise results according to user needs and could be used as a web application or a shared library.

Keywords: multimodal search engine timetable penalty

Obsah

Úvod

Efektivní cestování ve městech bývalo vždy obtížné. S rostoucí velikostí města roste i složitost dopravní sítě v něm a umět se dostat na místo určení rychle vyžaduje dobrou místní znalost nebo kvalitní vyhledávač a ideálně obojí. Čím více se člověk cestující po městě vzdaluje od svých vyzkoušených tras, tím více se musí spolehnout na vyhledávače spojení. Ty ale mohou poskytnout cennou službu i pro trasy důvěrně známé, zvláště, pokud trasy mají několik alternativ lišících se podle aktuálního času a návazností po cestě. Od takového vyhledávače je ale očekáváno, že zvládne naplánovat pěší přestupy i na větší vzdálenosti, než jsou zastávkové stojany téhož jména a že ho bude možné nastavit dle zkušeností a preferencí jednotlivých uživatelů.

V současné době existuje několik veřejně dostupných vyhledávačů spojení po městech České republiky: IDOS, Mapy.cz a Google Maps. Žádný z nich bohužel zároveň neumožňuje nastavit podrobnější parametry hledané trasy a plánovat kvalitní pěší přestupy. Rozhodli jsme se proto zpracovat veřejně dostupná mapová data a data o jízdních řádech a vytvořit vyhledávač, který by obě zmíněné podmínky splňoval. S vyhledávačem také nutně souvisí vytvoření vhodného formátu dat pro vyhledávání spojení.

V první kapitole se zabýváme různými známými datovými reprezentacemi sítí hromadné dopravy. Ve druhé kapitole představujeme mapová data a data o jízdních řádech, ze kterých jsme vycházeli a také popisujeme formáty dat, které v práci používáme. Ve třetí kapitole popisujeme zpracování zdrojových dat a přípravu formátů pro vyhledávání spojení, návrhem vyhledávače se zabývá čtvrtá kapitola. Pátá kapitola je věnována popisu implementace přípravy dat a samotného vyhledávače, v šesté kapitole jsou podrobně rozepsány používané datové struktury. Sedmá kapitola je tvořena dokumentací pro uživatele, kteří by si chtěli vyhledávač vyzkoušet. V osmé kapitole popisujeme experimenty, kterými jsme ověřovali schopnosti vyhledávače v porovnání s konkurencí a zkoumali, jaký vliv na vyhledané trasy budou mít různé konfigurační parametry. Také jsme zkoumali rychlost vyhledávače a hledali místa, která by pomohla hledání urychlit. V závěru shrnujeme dosažené výsledky a navrhujeme možné další úpravy a rozšíření, kterými by bylo možné vyhledávač dále vylepšit.

V příloze A popisujeme obsah přiloženého archivu se zdrojovými soubory.

1. Reprezentace sítí MHD

Data o různých cestních sítích obvykle ukládáme ve formě grafu, ve kterém pak vyhledáváme jednotlivé trasy pomocí prohledávání grafu, na které existuje mnoho známých algoritmů. Pokud ale potřebujeme udržovat kromě dat o cestní síti data o jízdních řádech, stává se situace mnohem složitější, protože zatímco po cestách můžeme jít kdykoli, cestovat hromadnou dopravou můžeme pouze tehdy, když zrovna jede nějaký spoj. Pro reprezentaci sítí hromadné dopravy se vyvinuly různé způsoby, dále představíme nejčastější z nich.

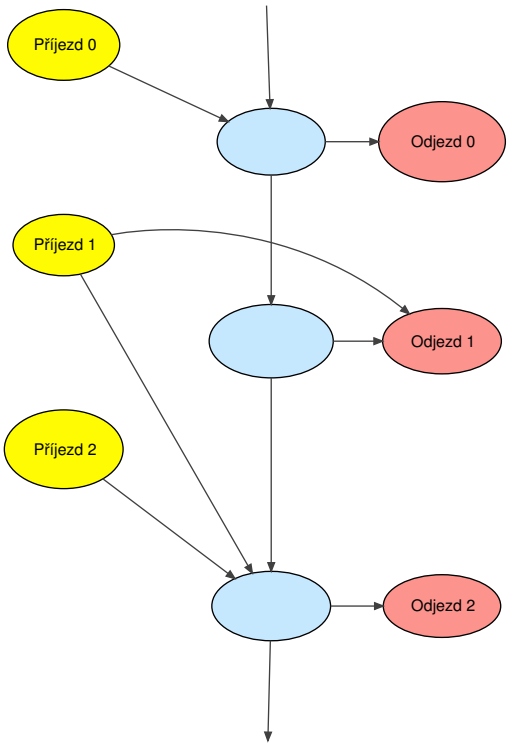
1.1 Time expanded modely

Time-expanded modely [1] jsou konstruovány tak, aby ceny hran byly konstantní a šlo na takovýto graf použít běžné vyhledávání v grafu. V těchto modelech jsou jako vrcholy dvojice (zastávka, čas odjezdu) a (zastávka, čas příjezdu) pro každý čas odjezdu a příjezdu z každé zastávky. Hrany pak jsou jednak „spojové“ spojující vždy dvojici odjezd-příjezd mezi dvěma zastávkami pomocí nějakého spoje, jednak „čekací“, které spojují jednotlivé časy v rámci jedné zastávky tak, jak jde čas. Spojení pak hledáme tak, že na množině vrcholů patřících k výchozí zastávce najdeme vrchol s nejbližším vyšším časem, než je náš odjezdový. Běžným průchodem do šířky podle času přes spojové a čekací hrany pak najdeme cestu do cílové zastávky. Nevýhodou této reprezentace je velikost grafu.

Tento základní model nerespektuje časy potřebné pro přestup, což může být zvlášť problematické u rozsáhlých stanic či zastávek s mnoha zastávkovými stojany. Tento problém se dá vyřešit rozdělením linie událostí u jedné zastávky na více. Vrcholy patřící k jedné zastávce rozdělíme na zastávkové, příjezdové a odjezdové. Odjezdové vrcholy odpovídají odjezdům spojů ze zastávky a příjezdové vrcholy příjezdům do zastávky. Zastávkové vrcholy odpovídají každý jednomu odjezdovému vrcholu. Zastávkové vrcholy jsou spojeny hranami do posloupnosti stejně jako v původním grafu. Z příjezdového vrcholu vedou hrany do nejbližšího zastávkového vrcholu, do kterého se stihne pěší přesun a do všech dřívějších odjezdových vrcholů, do kterých se stihne pěší přesun, viz obr. 1.1. Ze zastávkového vrcholu vede navíc hrana do odpovídajícího odjezdového vrcholu. Všechny přestupní hrany mohou být ohodnocené potřebným časem na přestup mezi linkami.

1.2 Time-dependent modely

Time-dependent modely [2] se snaží odstranit problém s velikostí grafu hromadné dopravy. Místo toho, abychom měli pro každý spoj zvláštní hranu, sdružíme spoje do linek, kde všechny spoje jedné linky mají stejnou posloupnost zastávek. Vrcholy tentokrát budou jen jeden pro každou zastávku. Hrany mezi zastávkami budou jedna pro každou linku, která danou dvojici zastávek spojuje. V takovémto grafu již nemůžeme vyhledávat pomocí běžného průchodu grafem, potřebujeme mít upravenou funkci, která vyhledá v jednotlivých časech odjezdu pro každou linku nejbližší od okamžiku příjezdu do zastávky. Tento přístup ne-



Obrázek 1.1: Příjezdové (žlutě), odjezdové (červeně) a zastávkové (modře) vrcholy. Všimněte si, že z příjezdu 1 lze stihnout odjezd 1, ale jen proto, že jsou blízko sebe, obecný spoj odjíždějící ze zastávky v čase odjezd 1 bychom nestihli.

vytváří grafy s velkým počtem vrcholů a hran, navíc je tato reprezentace snáze připojitelná do vyhledávacího grafu pro obyčejnou cestní síť.

Existuje také varianta, která má jeden staniční vrchol a pro každou linku projíždějící danou stanicí linkový vrchol. Tyto vrcholy jsou pak propojeny přestupovými hranami a umožňují realističtější modelování rozsáhlejších stanic a přestupů v rámci nich. Ani tento model však nevyužívá všech vlastností spojů MHD, jako je například to, že spoj má danou trasu a že když na něj někde nastoupíme, tak snadno můžeme do dalšího vyhledávání přidat všechny průjezdní stanice.

1.3 Contraction hierarchy

Contraction hierarchy [3] je způsob předzpracování dat tak, aby následné vyhledávání nemuselo procházet celý graf. U cestní sítě tomu odpovídá obvyklý postup při cestování – nejprve cestujeme po místních silnicích, abychom se dostali na silnice první třídy a dálnice, po těch pak dojedeme blízko cíle, kde opět přecházíme na místní silnice, abychom dosáhli cíle. U cestní sítě se příprava dat provádí obdobně – jsou nalezeny významné body, přes které přecházíme na silnice vyšších tříd a cesty na dlouhé vzdálenosti jsou pak rozděleny na hledání cesty do významných bodů a hledání cesty mezi nimi. Tato hierarchie může mít i několik

stupňů a cesty mezi významnými body jsou často předpočítány.

Obdobně se dá vytvořit podobná hierarchie i u hledání spojení MHD, kdy si udržuji zkratky mezi uzlovými zastávkami a nemusíme pak prohledávat jednotlivé mezilehlé zastávky, kde stejně nemůžeme na nic přestoupit. Problém této reprezentace pro nás je s přidáváním pěších tras, protože je potřeba mít všechny dopředu spočítané. Předvýpočet je stejně tak problematický i u cestní sítě, protože předpokládá pevné nastavení cestovních rychlostí a preferencí cest, na základě kterých jsou nalezeny uzlové body a vypočítány zkratky. Náš cíl je umožnit uživateli si zvolit tyto parametry při hledání dle svých potřeb, proto pro nás není tento model vhodný.

1.4 RAPTOR

RAPTOR (Round-Based Public Transport Routing) [4] je oproti předchozím modelům založen na zcela odlišných myšlenkách a při svém běhu nevyužívá algoritmy pro procházení grafu. RAPTOR se snaží využít co nejvíce vlastností, kterými se odlišuje síť veřejné dopravy od obyčejné cestní sítě. Také umožňuje snadno optimalizovat na počet přestupů.

Algoritmus pracuje po kolech. Každé kolo znamená nástup do dalšího dopravního prostředku, celkově je tedy počet kol o 1 větší než maximální počet přestupů. Algoritmus pracuje s linkami. Každá linka má několik spojů, což jsou konkrétní vozidla, která všechna projíždí stejnou posloupnost zastávek ve stejném směru. Každý spoj má uloženy časy odjezdu z jednotlivých zastávek, předpokládáme, že se dva spoje jedné linky na trase nepředjíždí. Pro každou zastávku si pro každé kolo pamatujeme, zda a kdy je dosažitelná pomocí kterého kola. Nedosažitelnost zastávky reprezentujeme pomocí nastavení času příjezdu v daném kole na ∞.

Na začátku jsou všechny zastávky ve všech kolech nedosažitelné. Nastavíme výchozí zastávce čas dosažení v prvním kole na zadaný čas odjezdu a spustíme algoritmus. Ten postupně prochází linky po jejich zastávkách a pokud narazí na dosažitelnou zastávku, tak najde nejbližší spoj linky, který z dané zastávky odjíždí po čase dosažitelnosti a tímto spojem se „vydá“ a průběžně upravuje časy dosažitelnosti na dalších zastávkách. Pokud po cestě nalezne další dosažitelnou zastávku, vybírá spoj z této zastávky z původního spoje a spoje navazujícího na původní čas dosažitelnosti ten, který odjíždí dříve. Takto pokračuje až do konce linky. Po průchodu všech linek se všechny časy dosažitelnosti zkopírují do dalšího kola a algoritmus se znovu spustí. Po stanoveném počtu kol algoritmus skončí a u jednotlivých zastávek je pro každé kolo (odpovídající počtu přestupů -1) uloženo, zda je s daným počtem přestupů dosažitelná a v jakém čase. K jednotlivým časům u zastávek je vhodné si uložit linku, která způsobila úpravu na daný čas, abychom byli schopni zrekonstruovat spojení využité k dosažení dané zastávky.

Tento algoritmus je snadno implementovatelný a při vhodně zvolených datových strukturách velmi rychlý a vhodně využívající keš.

1.4.1 Uložení dat

Pro rychlý výpočet a efektivní využití keše je potřeba mít vhodně uložená data. Data stejného typu jsou vždy uložená v poli za sebou. Pokud různé objekty mají každý mít seznam stejného typu, jsou všechna data od všech objektů držena

v jednom poli a každý objekt si drží počet svých prvků a odkaz na první z nich. Tímto má každý objekt vyhrazen svůj úsek a může s ním efektivně pracovat. Tento mechanismus nazveme „polním mechanismem“ a budeme na něj takto odkazovat ve zbytku kapitoly.

Základem pro prohledávání je pole linek. Každá linka má odkaz na své zastávky a na časy zastavení spojů v zastávkách. Obojí je implementováno polním mechanismem a rozdělení na zastávky a spoje zajišťuje snadnou možnost hledání spojů v daný čas. Časy zastavení spojů v zastávkách jsou uspořádány v poli za sebou podle zastávek a pak podle času výjezdu spoje z výchozí zastávky. Protože předpokládáme, že se spoje nepředjíždí a všechny spoje jedné linky mají stejný počet zastávek, na předchozí spoj snadno přejdeme skokem v poli zastavení o počet zastávek linky vzad, následující spoj najdeme obdobným skokem vpřed. Také je možné pro konkrétní zastávku a konkrétní čas použít jednoduše binární vyhledávání pro nalezení nejbližšího spoje odjíždějícího po konkrétním čase.

Pro vyhledávání z konkrétní zastávky máme obdobně implementovány datové struktury kolem zastávek. Zastávky jsou uloženy v poli za sebou, každá si drží seznam linek, které jí projíždějí, a seznam pěších přestupů z dané zastávky. Obojí je reprezentováno polním mechanismem.

2. Zdrojová data

Zdrojová data pro vyhledávač pochází ze dvou zdrojů. Prvním zdrojem jsou mapová data, která obsahují silnice, cesty, budovy a další mapové prvky. Druhým zdrojem jsou data o jízdních řádech, která obsahují linky, zastávky a spoje.

2.1 Mapová data

Mapová data jsou použita z projektu OpenStreetMap [5] (OSM). Tato data neobsahují informace o nadmořské výšce jednotlivých bodů; k doplnění nadmořské výšky jsou použita data z projektu NASA Shuttle Radar Topography Mission (SRTM), která jsou lineárně interpolována pro získání nadmořských výšek jednotlivých bodů v mapě.

O datech OSM a jejich problémech jsme již rozsáhle pojednávali v bakalářské práci [6], níže citujeme obecný úvod a popis základních primitiv.

2.1.1 Projekt OpenStreetMap

Projekt OpenStreetMap [5] vznikl v Anglii v roce 2004 a jeho prvotním cílem bylo vytvořit volně dostupná geografická data pro Velkou Británii. Iniciativa se postupně rozrostla do celého světa a dnes mapu pomáhá tvořit přes milion dobrovolníků. Česká republika je dnes poměrně kvalitně pokryta a zvláště velká města mají dostatečně detailní pokrytí i pro vyhledávání pěších tras.

2.1.2 Datová primitiva OSM

Projekt OpenStreetMap používá tři základní geografická primitiva: uzly, cesty a relace. Ke každému z těchto primitiv mohou být přiřazeny atributy, což jsou dvojice klíče a hodnoty. Každý typ primitiv má svou číselnou řadu, ze které dostává každý prvek unikátní číslo – id. U polohových dat se neukládá výška, výsledná mapa je pouze dvourozměrná. Nyní popíšeme jednotlivá primitiva:

Uzly jsou body s určenými souřadnicemi. Uzly mohou mít atributy, ty pak určují bodový mapový objekt, například rozcestník nebo závoru. Existují i uzly bez atributů sloužící pouze jako součást cest nebo relací.

Cesty jsou lomené čáry definované posloupností uzlů. Uzly se na cestě nemohou opakovat s jedinou výjimkou: první a poslední bod mohou být shodné, potom se jedná o uzavřenou cestu. Cesty jsou orientované, tzn. na pořadí uzlů záleží. Mohou existovat cesty bez atributů jako součásti relace, ale obvykle mají atributy určující, jaký objekt reálného světa popisují.

Pomocí cest popisujeme linie a plochy. Pokud má být cesta plochou, musí být uzavřená, ale ne každá uzavřená cesta je plocha. Tento problém rozebíráme níže. Jedna cesta také může reprezentovat více fyzických objektů (například silnici s tramvajovou tratí, park s oplocením).

Relace jsou posloupnosti uzlů a cest opatřené atributy. Každý prvek v relaci navíc může mít určenou roli. Relace může obsahovat jako prvek i relaci, ale tato situace není příliš dobře podporována programy pracujícími s daty OSM. Obvykle

jsou relacemi reprezentovány složitější objekty, které by se cestami a uzly popisovaly obtížně, nebo také „virtuální“ objekty jako například trasy linek MHD, cyklotrasy a územní hranice.

Pro nás důležitý typ relace je **multipolygon**, který se používá k reprezentaci složitějších ploch. Plocha reprezentovaná multipolygonem se může skládat z více nesouvisejících částí nebo obsahovat díry. Multipolygon obsahuje cesty s rolemi `INNER` resp. `OUTER` indikující, zda je cesta součástí vnější resp. vnitřní hranice plochy. Hranice multipolygonu se mohou skládat z více částí, ale vždy musí dohromady tvořit jednu nebo několik uzavřených částí tvořících obvody plochy resp. díry v ploše.

2.2 Výšková data

O výškových datech jsme také pojednávali v bakalářské práci, opět citujeme [6]: Abychom mohli správně odhadnout náročnost pěší trasy, musíme znát i informace o nadmořské výšce jednotlivých bodů. Stejně tak jako většina jiných projektů jsme použili data SRTM [7], což jsou volně dostupná výšková data pro celý svět.

Shuttle Radar Topography Mission byl projekt NASA, kdy při letu raketoplánu Endeavour v roce 2000 byla pomocí radarové interferometrie změřena velká část Země a zpracovaná data byla následně poskytnuta volně k dispozici.[1]

Výšková data byla měřena po třech úhlových vteřinách (v USA po jedné vteřině), tudíž v České republice jsou data v mřížce 90×60 m. Data jsou rozdělena do tabulek po jednom stupni, přičemž sousední tabulky se vždy jedním sloupcem nebo řádkem překrývají. Každá tabulka má tedy 1201 sloupců a 1201 řádků, kde řádky odpovídají zeměpisné šířce a sloupce zeměpisné délce.

Nadmořské výšky jsou kódovány celými 16bitovými čísly v big endian udávajícími nadmořskou výšku v metrech. Pokud se v některém místě nepodařilo výšku změřit, je udávána jako -32768. Tabulka je kódována jako binární soubor, v němž jsou jednotlivé řádky zapsány za sebou.

2.3 Jízdní řády

Data o jízdních řádech jsou očekávána ve formátu General Transit Feed Specification (GTFS) [8], který je dobře specifikovaný a široce používaný ve světě. Konkrétně pro testování byla použita data o pražské integrované dopravě od IPR Praha.[2] Tato data obsahují metro, tramvaje, autobusy a přívozy v Praze a okolí, bohužel neobsahují integrované vlakové spoje.

GTFS je distribuováno jako archiv ZIP obsahující jednotlivé tabulky ve formátu CSV. Význam jednotlivých tabulek je následující:

- *agency.txt* obsahuje informace o dopravcích na jednotlivých linkách. V našem vyhledávání není používán.
- *stops.txt* obsahuje informace o zastávkách. Zastávky jsou dvou typů. Prvním typem je zastávkový stojan, který reprezentuje fyzické místo, kde zastavuje

[1] `http://dds.cr.usgs.gov/srtm/version2_1/`

[2] `http://opendata.iprpraha.cz/DPP/JR/jrdata.zip`

nějaký spoj některé linky. Druhým typem je „stanice“, která udává oblast několika stojanů, obvykle stejného jména. Nemusí mít fyzickou reprezentaci a slouží pro zobrazování v mapě a svázání logicky blízkých stojanů. Protože stanice pro náš vyhledávač nenese důležitou informaci, využíváme pouze zastávkové stojany. Pro vyhledávání spojení využíváme následující položky:

  - `stop_id` – jednoznačný identifikátor zastávky
  - `stop_name` – název zastávky
  - `stop_lat, stop_lon` – poloha zastávky
  - `location_type` – informace, zda jde o stojan, nebo stanici

- *routes.txt* obsahuje informace o linkovém vedení. Pro vyhledávání spojení využíváme následující položky:

  - `route_id` – jednoznačný identifikátor linky
  - `route_short_name` – krátký název linky, v Praze označení linky
  - `route_type` – typ dopravního prostředku linky, například autobus, loď, metro

- *trips.txt* obsahuje informace o spojích. Tato tabulka slouží ke svázání spoje (cesty konkrétního dopravního prostředku po posloupnosti zastávek v daný čas), linky a množiny dní, kdy daný spoj jezdí. Pro vyhledávání využíváme následující položky:

  - `route_id` – identifikátor linky
  - `service_id` – identifikátor jízdních dní spoje
  - `trip_id` – jednoznačný identifikátor spoje

- *stop_times.txt* obsahuje informace o časech zastavení jednotlivých spojů v jednotlivých zastávkách. Pro vyhledávání spojení používáme následující položky:

  - `trip_id` – identifikátor spoje
  - `arrival_time` – čas příjezdu spoje do zastávky
  - `departure_time` – čas odjezdu spoje ze zastávky
  - `stop_id` – identifikátor zastávky
  - `stop_sequence` – pořadí zastávky na spoji. Je potřeba, aby bylo nezáporné, celočíselné a po trase spoje se zvyšovalo, zvyšování po 1 ani počítání od 0 není nutné.

- *calendar.txt* obsahuje informace o tom, které dny v týdnu jezdí které spoje. Pro vyhledávání spojení používáme následující položky:

  - `service_id` – identifikátor jízdních dní spoje
  - `monday ... sunday` – informace, zda spoj jede daný den.
  - `start_date` – první den platnosti jízdního řádu pro daný spoj

  - `end_date` – poslední den platnosti jízdního řádu pro daný spoj (tento den ještě v rámci platnosti).

- *calendar_dates.txt* obshauje výjimky z pravidelnosti spojů, například státní svátky či jiná omezení. Obsahuje změny oproti souboru `calendar.txt`. Může být použit i samostatně bez souboru `calendar.txt`, pak obsahuje všechny jízdní dny daného spoje. Pro vyhledávání spojení používáme následující položky:

  - `service_id` – identifikátor jízdních dní spoje
  - `date` – datum změny
  - `exception_type` – typ změny (1 = jede navíc, 2 = nejede, ač by měl)

2.4 Používané formáty dat

2.4.1 Protocol Buffers

Protocol Buffers (PBF) [9] je způsob uložení strukturovaných dat. Byl navržen Googlem a je používán v případech, kdy potřebujeme přenést po síti zprávy s danou strukturou. Formát podporuje různé datové typy (integer různých přesností a znaménkovosti, čísla s plovoucí řádovou čárkou, řetězce znaků, ...) a strukturování zpráv. Komunikace probíhá ve fázích naplnění zprávy, její zabalení, odeslání, přijmutí a rozbalení. Při zabalení jsou data zkomprimovaná jednoduchým algoritmem, aby například malá čísla nezabírala zbytečně mnoho místa. Existují PBF verze 2 a verze 3. V naší práci používáme verzi 2, protože v době psaní bakalářské práce, kterou v naší práci rozšiřujeme, nebyly PBF verze 3 ještě k dispozici a verze 3 nepřináší novinky, které by motivovaly k přechodu.

Základní jednotkou PBF je zpráva, která obsahuje několik položek. Položkou může být základní datový typ nebo jiná zpráva, položky mohou být povinné, volitelné, nebo opakované. Každá položka má jméno, typ a číslo, které ho identifikuje v binární podobě zprávy. Zprávy se popisují pomocí vlastního jazyka, ze kterého se pak pomocí kompilátoru vytvoří vazby do jednotlivých programovacích jazyků. Podrobnou dokumentaci včetně tutoriálů lze najít na stránkách projektu. [9]

2.4.2 JavaScript Object Notation

JavaScript Object Notation [10] (JSON) je formát navržený pro předávání dat ve webových aplikacích. Díky své jednoduchosti a čitelné reprezentaci je hojně využíván nejen ve webových projektech, ale často i jako konfigurační jazyk. Vychází z Javascriptu, datové typy v něm jsou kromě primitivních typů slovník – neuspořádaný seznam dvojic klíč-hodnota a pole – uspořádaná posloupnost položek. Formát zpráv není dopředu dán a podpora v programovacích jazycích je přímo integrovaná nebo dodaná pomocí knihovny.

2.4.3 GeoJSON

GeoJSON [11] je, jak již název napovídá, nadstavba formátu JSON pro přenos geografických informací. Z hlediska syntaxe se jedná o korektní JSON, který má

ale předepsanou strukturu a názvy položek. Umožňuje ukládat body, linie, plochy, multipolygony a další objekty spolu s jejich atributy a je nativně podporován většinou JavaScriptových knihoven pro zobrazování mapových dat.

2.4.4 XML

XML [12] je značkovací formát používaný pro přenos strukturovaných dat mezi aplikacemi. Jedná se o textový formát, ve kterém jsou uložena strukturovaná data ve formě entit uspořádaných do stromové struktury. Každá entita má jméno a může mít parametry.

2.4.5 GPX

Formát GPX [13] je určen pro výměnu polohových dat mezi různými zařízeními s GPS, popřípadě mobilními a jinými aplikacemi. Data jsou uložena ve formátu XML, jsou dělena na cesty, trasy a významné body. Každá cesta, trasa i významný bod může mít uložené některé informace v předdefinovaných atributech, případně libovolné další informace uvnitř elementu `<extensions>`.

2.4.6 Unixový čas

Unixový čas je standardní způsob reprezentace data a času na Unixových systémech. Je obvykle ukládán v sekundách od půlnoci 1. 1. 1970 UTC.

2.5 Systémy zeměpisných souřadnic

Zeměpisné souřadnice jsou v rámci práce používány ve dvou souřadnicových systémech – UTM a WGS84. Při klasifikaci dat z OSM jsou jejich souřadnice převedeny do UTM a tak jsou dále uchovávány. Při přípravě výsledků vyhledávání jsou body na trasách převedeny zpět do WGS84 a takto jsou vráceny. Veškerá data předávaná knihovně či vrácená knihovnou jsou ve WGS84.

2.5.1 WGS84

World Geodetic System 1984 (WGS84) [14] je geodetický standard definující souřadnicový systém a referenční elipsoid pro popis polohy objektů na povrchu Země. Využívá polárních souřadnic, zeměpisná šířka se určuje od rovníku, zeměpisná délka od „IERS Reference Meridian“. Výhodou je pokrytí celého povrchu Země jedním systémem, vzhledem k použití elipsoidu a polárních souřadnic je obtížné počítat vzdálenosti mezi dvěma body.

2.5.2 UTM

Univerzální příčný Mercatorův systém souřadnic (UTM) [15] je způsob určování polohy na povrchu země založený na mřížkách. Povrch země je rozdělen na 60 zón a každá zóna je pak zobrazena pomocí příčného Mercatorova zobrazení. Souřadnice v rámci jedné zóny jsou pak udávány v metrech, lze proto snadno počítat vzdálenosti. Pozice bodu na Zemi se udává pomocí čísla zóny a pozice

uvnitř ní. Díky tomu, že UTM kvůli rozdělení na zóny zobrazuje vždy úzký pás Země, nevzniká velké zkreslení vlivem válcového zobrazení. Praha a většina dalších měst se také nenachází v polárních oblastech, kde je zkreslení velké, nebo se používá jiný systém souřadnic.

3. Předzpracování dat

3.1 Příprava mapových dat

Nejprve jsou stažena aktuální data pro Českou republiku, která jsou k dispozici s denními aktualizacemi.[1] Z těchto dat je na základě nastavení vyříznut obdélník, který pokrývá zpracovávanou oblast. Poté začneme data zpracovávat.

Nejprve jsou klasifikovány jednotlivé objekty. Klasifikují se uzly, hrany a multipolygony, jiné typy relací nejsou v současné době používány. Každý objekt je klasifikován nějakým typem podle toho, jaké má tagy. V rámci konfigurace lze přidělit nějakému tagu (buď samotnému klíči, nebo dvojici klíč-hodnota) určený typ s danou prioritou. Objekt bude mít typ daný pravidlem s nejvyšší prioritou, které jeho tagy splňují. Pokud objekt nesplňuje žádné pravidlo, získá speciální typ označující neklasifikovaný objekt.V dalším zpracování se již nehledí na původní tagy ale jen na typ, který daný objekt má.

Současně s klasifikací typu objektu se stejným způsobem určují hrany, které se nachází na mostech a uzly, které jsou v podzemí. Klasifikovaná data nahrajeme do databáze a další kroky provádíme jako databázové operace.

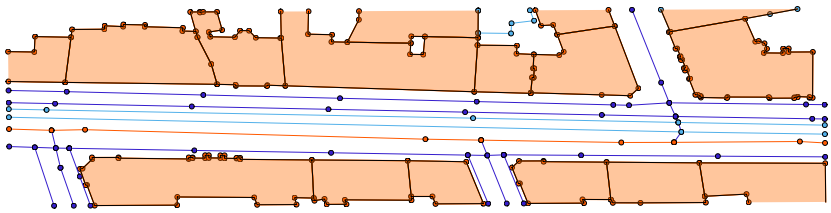
3.1.1 Dělení dlouhých linií

Protože chceme hledat co nejblíže od výchozího zadaného bodu, v případě dlouhých rovných ulic, které jsou reprezentovány pomocí lomených linií s dlouhými úseky by se nám mohlo stát, že by nejbližší bod od hledaného místa neležel na ulici, kde jsme, ale na nějaké sousední, od které nás dělí pás budov. Proto dlouhé úseky cest rozdělíme na menší podúseky (viz obr. 3.1), čímž tento problém eliminujeme. Rozdělení se nám také bude hodit pro vytváření zkratek, o kterých pojednáváme níže.

3.1.2 Překážky

Pro další zpracování potřebujeme znát nejen cestní síť, ale i překážky, přes které se nedá projít, jako jsou například domy, ploty a dálnice. V tomto místě se musíme vypořádat s multipolygony. Protože multipolygony, které klasifikujeme jako bariéry, jsou většinou budovy, které mají více částí nebo mají nádvoří, nebo jiné překážky, uvnitř kterých neočekáváme velkou plochu, bereme jako překážku

[1] `osm.kyblsoft.cz/archiv/`



Obrázek 3.1: Rozdělení dlouhých linií (všimněte si, že jsou děleny jen pochozí linie)

Obrázek 3.2: Vygenerované zkratky mezi cestami (zeleně)

jejich vnější obrys, případně obrysy. To nám sice způsobí, že nebudeme moci vytvářet na vnitřních prostranstvích zkratky, ale to nám nevadí, protože jednak mají vnitřní prostranství obvykle velmi malou plochu a nevede do nich mnoho pěších cest, jednak vnitřní prostranství jsou buď zmapována kompletně, nebo nedostatečně, tudíž by případné vytváření zkratek ve vnitřním prostranství vedlo k cestám, které by ve skutečnosti nebyly možné.

Výsledná množina překážek bude obsahovat vnější obrysy multipolygonů, další polygonové objekty a liniové objekty.

3.1.3 Body uvnitř objektů a body pod zemí

Abychom mohli generovat zkratky, které budou průchozí i ve skutečnosti, je potřeba určit, které body se nachází na volném prostranství na povrchu a které se nacházejí pod zemí. Podzemní vrcholy jsme určili už v rámci klasifikace. Vrcholy uvnitř objektů jsme ztotožnili s vrcholy, které se nachází uvnitř nějaké překážky, protože překážky jsou pro nás takové objekty, přes které se nedá pěšky projít napříč. Body, které se nacházejí na plášti překážky, jako vnitřní neuvažujeme, protože se z nich dá jít libovolným směrem, kde se nenachází překážka.

3.1.4 Zkratky

Pro doplnění chybějících vazeb v mapě používáme kromě cest v mapových datech již obsažených i automaticky generované zkratky. Zkratka spojuje dva pochozí body na volném prostranství, které jsou nedaleko od sebe a úsečka mezi nimi, reprezentující zkratku, neprotíná žádnou překážku. Dvojic bodů, které splňují zadané podmínky, je ale velké množství a přidání všech zkratek by neúměrně zvyšovalo velikost výsledných dat. Od určitého počtu zkratek z daného vrcholu přidávání dalších zkratek má jen malý vliv na délky hledaných cest, proto jsou ze všech možných zkratek náhodně vybrány jen některé a to tak, aby z každého vrcholu vycházelo průměrně omezené množství zkratek. Takto zachováme pozitivní vliv zkratek na hledané cesty, ale zbytečně nezvětšujeme vyhledávací graf.

3.2 Příprava jízdních řádů

Data z jízdních řádů dostáváme ve formátu GTFS [8] a musíme je upravit do formátu vhodného pro algoritmus RAPTOR [4]. V rámci převodu mezi formáty je potřeba respektovat specifické požadavky algoritmu RAPTOR a také je potřeba připravit si informace potřebné pro propojení mezi zastávkami v jízdním řádu a zastávkami na mapě.

3.2.1 Rozdělení linek

Formát GTFS rozlišuje linky a spoje. Linka obsahuje všechny společné údaje a jednotlivé spoje pak reprezentují jízdu vozidla dané linky mezi konkrétními stanicemi, různé spoje jedné linky mohou projíždět různé posloupnosti stanic (například spoje zatahující do vozovny, zkrácené vložené spoje, ...). Algoritmus RAPTOR ale vyžaduje, aby spoje konkrétní linky projížděly vždy stejnou posloupnost zastávek. Abychom toho dosáhli, v rámci předzpracování najdeme všechny různé posloupnosti zastávek projížděné jednou linkou a vytvoříme sublinky pro každou takovou posloupnost. Tyto sublinky zdědí společné údaje z původní linky a již splňují požadavky kladené algoritmem RAPTOR.

3.2.2 ID zastávek

Zastávky mají dle specifikace GTFS jako ID použit obecný string. Data pro pražskou MHD mají toto ID rozdělené na na část reprezentující zastávku a část reprezentující konkrétní zastávkové stojany. Pro další zpracování je vhodné zvolit číselný identifikátor, jednotlivé zastávky jsou očíslovány čísly od 0 do počet zastávek -1 a veškeré odkazy na konkrétní zastávky ve zpracovaných datech používají právě tato čísla. Původní identifikátor je u zastávky stále uložen kvůli následnému párování (viz níže), ale již se pro vazbu mezi daty nepoužívá.

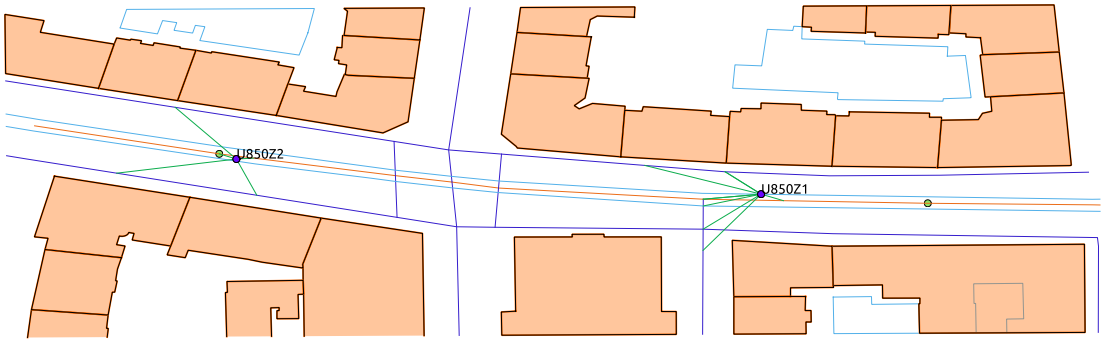
3.2.3 Platnost jízdního řádu

Ve formátu GTFS jsou linky, spoje a dny, ve kterých daný spoj jede, provázány pomocí tabulky trips. V předchozím odstavci jsme popsali rozdělení linek podle toho, kudy spoje jedou, nyní využijeme, že každý spoj v GTFS má právě jednu množinu dní, kdy jede a tuto informaci si uložíme i u jednotlivých spojů v nově vytvářeném formátu.

Místo dvojího způsobu záznamu, kdy daný spoj jede – pomocí výčtu dnů v týdnu a seznamu výjimek – si pro každou možnost, jak může nějaký spoj jet, uložíme bitmapu platnosti a datum začátku a konce platnosti současného jízdního řádu. Bitmapa začíná první den platnosti a i-tý bit udává, zda i-tý den od začátku platnosti daný spoj jede.

3.2.4 Podzemní stanice

Při párování zastávek a pozic na mapě budeme potřebovat zvlášť ošetřovat podzemní stanice, ze kterých se nelze vydat libovolným směrem, ale jen eskalátorovým tunelem. V rámci předzpracování označíme jako podzemní takové zastávky, ve kterých jezdí metro. V Praze takovýto předpoklad funguje správně, protože



Obrázek 3.3: Zkratky pro připojení tramvajové zastávky (zeleně)

všechny nadzemní stanice metra jsou zmapovány detailním způsobem, kdy zpracování dat funguje korektně, pro jiná města s jinou kvalitou zmapování by bylo potřeba stanovit odlišná kritéria.

3.3 Párování zdrojových dat

Abychom moli plánovat spojení využívající jak pěší chůzi, tak jízdu MHD, je potřeba data z obou zdrojů vhodně provázat. Máme k dispozici následující údaje:

1. OSM
    - jméno zastávky
    - pozici zastávky
    - ID zastávky (jen u některých)
2. GTFS
    - jméno zastávky
    - pozici zastávky
    - ID zastávky

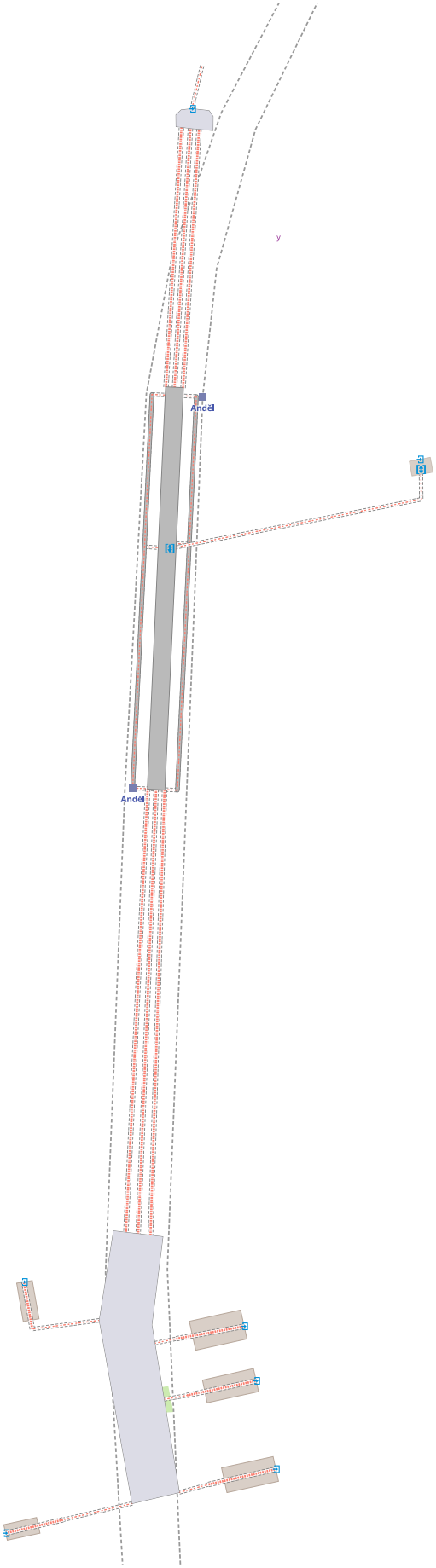
V ideální případe by bylo možné spárovat zastávky jednoduše dle ID, bohužel v OSM má ID jen několik zastávek, většinu zastávek je tedy potřeba spárovat jinak. Nabízelo by se párování podle pozic a jmen zastávek, ale bohužel zastávky v GTFS jsou výrazně posunuté oproti OSM i skutečnosti, navíc ne všechny zastávkové stojany jsou v OSM vyznačeny, zvláště tam, kde je několik zastávkových stojanů za sebou, například v autobusových terminálech. Pokoušet se párovat zastávky v GTFS pouze na zastávky v OSM by bylo velmi náročné s nejistým výsledkem. Využíváme proto toho, že v OSM máme zmapované nejen zastávky, ale i cesty a zastávkám v GTFS vytváříme speciální vrcholy dle jejich zeměpisné pozice v GTFS a pomocí zkratek je spojujeme s nejbližšími cestami (viz Obrázek 3.3. Bod reprezentující zastávku pak v mapových datech označíme jako zastávku.

Pro zastávky z GTFS, pro které máme v OSM odpovídající ID, použijeme polohu z OSM a zkratky k cestní síti hledáme z této polohy. Zkratky jsou i zde potřeba, protože dle pravidel OSM [5] se zastávka umisťuje na místo, kde

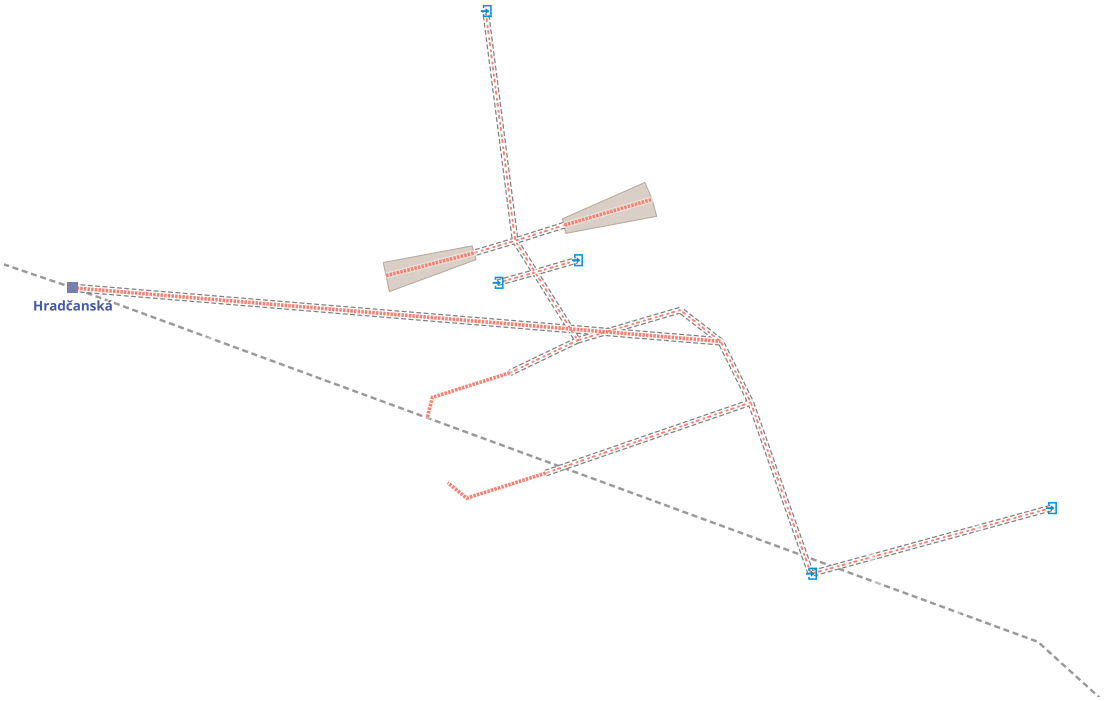
zastavuje vozidlo, což například u tramvají je bod na kolejích, které ale pro pěší plánování nepoužíváme, tudíž je potřeba najít vhodný blízký bod v cestní síti.

Zvláštní pozornost je potřeba věnovat u párování zastávek metra. V současné chvíli je metro v Praze zmapované dvěma způsoby. První způsob (viz obrázek 3.4) je novější a přesnější, jsou při něm zmapována nástupiště a eskalátorové tunely. Při tomto podrobném mapování jsou také přidána ID stanic, tudíž je možné stanice jednoduše spárovat a hledat cestu od hrany nástupiště. Častějším způsobem je ale starší způsob (viz obrázek 3.5), kdy je stanice metra pouze bod, od kterého vede eskalátorový tunel na povrch. Tento eskalátorový tunel je pouze virtuální spojka, neodpovídá reálné poloze podzemních tras. Takovéto stanice rovněž nemají přiřazená ID. U těchto stanic používáme polohu z GTFS a zkratky spojující zastávku s cestní sítí hledáme do blízkých míst, která jsou v podzemí, což vede k poměrně dobré aproximaci přístupu do metra. Jak bude postupovat mapování stanic metra, bude tento typ stanic postupně eliminován a dojde ke zpřesnění navigace při přestupech.

Vždy jsou preferovány zastávky a stanice zmapované přesněji, které mají ID, stačí tedy vylepšovat mapu a při dalším předzpracování dat se nově zmapované zastávky dostanou i do vyhledávače.



Obrázek 3.4: Detailně zmapovaná stanice metra Anděl



Obrázek 3.5: Stanice metra Hradčanská zmapovaná starším způsobem

4. Vyhledávání trasy

V připravených datech je možné opakovaně vyhledávat spojení. Ke hledání spojení se využívá Dijkstrův algoritmus a RAPTOR. Nejprve se naleznou nejbližší vrcholy grafu k zadaným výchozím a cílovým souřadnicím. Poté se začne Dijkstrovým algoritmem prohledávat graf pěších cest. Pokud při prohledávání narazíme na zastávku MHD, provedeme z této zastávky jedno kolo algoritmu RAPTOR, najdeme tedy všechny zastávky, kam se umíme bez přestupu dostat z dané zastávky. Všechny tyto zastávky přidáme do fronty Dijkstrova algoritmu s časem dosažení rovným času příjezdu do dané stanice.

4.1 Penalty

Samotná nejrychlejší cesta v síti pěších cest a MHD není vždy ta, kterou chceme při reálném cestování použít. Navíc vyhledávačů, které takovou cestu i pro pražskou síť hledají, existuje několik funkčních a široce používaných. Naším cílem bylo, aby si uživatel mohl parametry nalezené cesty určovat dle svých preferencí. Proto jsme se snažili navrhnout systém penalt tak, aby se v něm dalo vyjádřit co nejvíce běžných situací, kterým bychom se chtěli ve vyhledaném spojení vyhnout.

Systém penalt funguje tak, že za každou událost na trase si uživatel může zvolit nějakou penaltu, která říká, jak moc nerad by danou událost měl na vyhledané trase. Současně existuje daná mez přípustnosti trasy. Pokud penalta nějaké částečně vyhledané trasy překročí tuto mez, není již dále pokračováno ve vyhledávání dál z této trasy. Tímto zajišťujeme možnost mít jak „měkká“, tak „tvrdá“ kritéria pro hledanou trasu. Také je vhodné mít penalty konstantní a závislé na době trvání. Například za deště je pro nás čekání na zastávce bez přístřešku tím horší, čím déle čekáme, zatímco pokud nám vadí cesta eskalátorem na metro, tak když už ji jednou podnikneme, na délce jízdy metrem už nám nezáleží.

Důležité je také rozmyslet, jaké všechny události chceme mít možnost penalizovat:

- *typ cesty* – Toto je přímočará myšlenka implementovaná již v původním vyhledávání pěších tras. Můžeme jednak penalizovat chůzi po silnici, čímž dáme preferenci vyhledávání po chodnících, jednak můžeme penalizovat chůzi po nezpevněných pěšinách, například v zimním období či po dešti.

- *typ průchozího vrcholu* – Touto událostí je například čekání na semaforech, které jsou reprezentovány bodovým znakem na přechodu přes silnici. Tvrdým kritériem je například zavřená brána, která spojuje dvě volně přístupné oblasti.

Další události již souvisí s cestováním MHD a přestupy:

- *počet přestupů* – V našem případě používáme počet použitých vozidel, protože můžeme hledat i pouze pěší trasu. Obvyklé kritérium používané i běžnými vyhledávači.

- *čas na přestup* – V základní formě jednoduché tvrdé kritérium, které udává minimální čas strávený na zastávce mezi příchodem / příjezdem a odjezdem. Toto kritérium lze výrazněji rozšířit různými směry. Jednak je možné

uvažovat, jak jsme se na zastávku dostali a čím z ní odjíždíme (metro je přesné a spolehlivé, pokud jsme přišli pěšky, můžeme popoběhnout, pokud jedeme autobusem, radši si necháme větší rezervu), jednak je možné místo fixního času zavést funkci závislou na době čekání a tím například preferovat krátká, ale ne moc krátká čekání.

- *nástup* – Nejen čas na přestup je možné penalizovat, i samotný nástup do vozidla může být penalizován. Základní možností je fixní penalta za nástup, která je měkkou variantou omezení počtu přestupů. Dále je možné penalizovat určité dopravní prostředky či přestupní místa, například zastávky bez přístřešku za deště.

- *linka* – Kromě penalizací dopravních prostředků je také vhodné mít možnost penalizovat i linky. Pokud víme, že některá autobusová linka je vlivem dlouhé trasy přes město nespolehlivá, můžeme ji přiřadit velkou penaltu nebo ji rovnou vyřadit z vyhledávání.

Další možností rozšíření systému penalt je zapojení denní doby do výpočtu penalty. Ve večerních hodinách můžeme například penalizovat podchody a jiná potenciálně nebezpečná místa.

Všechny výše uvedené možnosti jsme při návrhu systému penalt zohlednili. Základní penalty je možné určit pomocí konfiguračního souboru, pokročilé je potřeba přidat do kódu sloužícího pro výpočet penalt. Více viz kapitola 5.

4.2 Průběh hledání

Při běhu algoritmu počítáme pro každou cestu její penaltu – čím větší má cesta penaltu, tím horší pro nás je. Samotný Dijkstrův algoritmus bere vrcholy podle času dosažení. Po nalezení nejrychlejší cesty se nezastaví, ale počítá dokud nezpracovává vrchol o nějaký koeficient horší, než je nejkratší cesta. Tento postup byl zvolen, protože při hledání cesty se chce uživatel dostat z výchozího do cílového bodu co nejrychleji, ale má nějaké preference na to, jak by měla cesta vypadat. Pokud vyhledáváme podle času, dostane uživatel jednak nejkratší cestu, jednak alternativní cesty, které jsou sice o něco delší, ale více splňují představy uživatele o optimální cestě.

Pokud bychom během vyhledávání udržovali všechny trojice (vrchol, čas příjezdu, penalta), byl by procházený stavový prostor obrovský a vyhledávání by bylo velmi náročné na paměť i čas. Pro redukci stavového prostoru uvažujeme uspořádání pro každý vrchol, kde spojení A je horší než spojení B, pokud A má pozdější čas dosažení vrcholu a vyšší penaltu než B. V takovém případě spojení A do fronty otevřených vrcholů vůbec nepřidáváme, případně ho zahodíme, když na něj ve frontě narazíme. Takovéto situaci říkáme, že *spojení B majorizuje spojení A*.

4.3 Redukce duplicitních tras

I přes uvažování pouze nemajorizovaných spojení získáme mnoho spojení, která se liší jen drobnými změnami v pěších částech trasy. Pro skutečnou cestu

podle nalezeného spojení nemají tyto drobné rozdíly význam, proto se snažíme takovéto duplicity ignorovat. Základním předpokladem, který učiníme, budiž to, že pokud se dvě spojení liší v použitém spoji MHD, tak chceme obě zobrazit. Dále se tedy zabýváme jen spojeními, které mají shodné části využívající MHD. Jako duplicitní spojení bychom rádi označili taková spojení, která vedou celou cestu blízko sebe.Takovéto pravidlo je ale časově náročné implementovat, proto místo něj použijeme zjednodušenou variantu.

Vezmeme pouze pěší části spojení a porovnáváme vždy odpovídající pěší části proti sobě. Procházíme hrany a hledáme, kde se liší. Zapamatujeme si index první takovéto hrany. Pak projdeme úsek od konce a opět hledáme první hranu, kde se liší. Tímto máme ohraničený úsek, kde se cesty liší. Podíváme se u obou cest na vrchol uprostřed tohoto úseku a pokud jsou tyto vrcholy příliš blízko sebe, označíme úsek jako duplicitní. Pokud mají dvě spojení všechny pěší úseky duplicitní, jedno ze spojení zahodíme. Idea je, že pokud se dvě cesty liší v nějakém úseku, tak uprostřed tohoto úseku budou od sebe nejdál. Toto pravidlo sice může eliminovat i cesty, které jsou odlišné a potkávají se zrovna uprostřed, ale to se nestává často a pravidlo lze implementovat v lineárním čase.

5. Implementace

5.1 Příprava mapových dat

Počáteční fáze přípravy dat je shodná s přípravou data v bakalářské práci – pomocí shellového skriptu je nalezeno poslední vydání OSM dat pro Českou republiku, tato data jsou stažena, rozbalena a pomocí programu osmconvert je z nich vyříznuta definovaná oblast. Současně jsou stažena i výšková data z projektu SRTM. V případě potřeby generování vyhledávacích dat pro jiné město nebo jinou zemi je potřeba tento skript upravit, pro jiné české město stačí přepsat hranice výřezu, jiný stát by potřeboval i vlastní zdroj dat OSM a upravit souřadnice pro stahování SRTM.

Další pokračování přípravy dat je sice sémanticky stejné, jako bylo v bakalářské práci, ale mechanizmus přípravy byl kompletně změněn. Data jsou nejprve klasifikována; pro tento účel byl vytvořen kód v jazyce C, který dle dodané konfigurace přiřadí typ hranám a nově i vrcholům. Jazyk C jsme zvolili z důvodu rychlosti a paměťové efektivity, velkou roli také hrála znalost jazyka autorem. Ve většině programů, které jsme napsali je také využívána knihovna LibUCW [16], která poskytuje velké množství algoritmů, datových struktur a obvyklých konstrukcí v C a je naprogramována s ohledem na maximální výkon.

Výsledek klasifikace je uložen do formátu PBF jako objekt Premap. Tento objekt je následně nahrán do databáze PostgreSQL s nadstavbou PostGIS pomocí kombinace shellového skriptu a programu v C, který čte Premap a vytváří z něj CSV. Zkoušeli jsme různé varianty vkládání do databáze, protože jde o velké množství dat a tento způsob se ukázal nejefektivnější z jednoduchých způsobů vkládání.

Pro další zpracování jsme využili databázi, protože příprava dat je velmi podobná práci s databází – zpracováváme velké množství nezávislých objektů či jejich dvojic, na což jsou databázové systémy vhodné a dobře připravené. Využití nadstavby PostGIS pak zjednodušuje geometrické operace s prvky, protože tyto operace má již připravené jako funkce. Samotný přepis do PostgreSQL byl vykonán v rámci studentského fakultního grantu, dokumentace k němu níže uvedená vznikla až v rámci diplomové práce. Také ty části přípravy dat, které operují se zastávkami, byly implementovány až v rámci diplomové práce.

Další zpracování provádíme v databázi pomocí databázových dotazů a tvorby nových pomocných tabulek. Oproti zpracování v C, které bylo uvedeno v bakalářské práci, jsme zpracováním v databázi získali možnost připravovat data i pro velká města na běžném počítači. Původní verze programu měla totiž všechna data v paměti, což pro Prahu znamenalo až desítky gigabajtů dat a přepsání programu tak, aby byl prostorově úspornější by bylo poměrně náročné, byla proto raději zvolena jiná platforma, která poskytuje několik výhod:

- Výměna paměti za čas – v PostgreSQL si snadno můžeme podle dostupné paměti stroje zvolit množství paměti, kterou databáze bude využívat. Na strojích s menší velikostí operační paměti se data budou připravovat déle, ale zvládnou se připravit. Stejně tak i výkonnější stroje mohou být využity na maximum a čas zpracování bude nižší. Všechna tato nastavení jsou pouze nastavení databázového systému, na kódu práce se nic nemění

- Abstrakce – použitím databázového systému nás odstíní od většiny implementačních detailů původní práce a zůstane nám kratší a výstižnější kód, který popisuje přípravu dat. Na druhou stranu s použitím databázového systému vzniká nutnost tvorby indexů, aby databáze pracovala efektivně. Některé jsou přímočaré, u jiných je potřeba zkoumat různá řešení, ale použitím nevhodného indexu neztratíme kvalitu dat, jen rychlost výpočtu.

- Snadná rozšiřitelnost – protože jsme se díky abstrakci oprostili od implementačních detailů, je mnohem jednodušší upravovat průchod přípravou dat, případně do něj přidávat další fáze.

5.1.1 Rozdělení dlouhých cest

Rozdělení příliš dlouhých úseků cest jsme implementovali pomocí funkce v jazyku plpgsql, který je integrován v databázovém systému. Funkce dostane id cesty ke zpracování a maximální délku segmentu jako parametry. Pak postupně prochází jednotlivé segmenty mezi body cesty a v případě, že narazí na segment delší, než je maximální délka segmentu, tak ho postupně od začátku dělí na úseky maximální povolené délky a přidává do tabulky uzlů uzly, které vznikly dělením. Všechny uzly (jak původní, tak nově vzniklé) funkce průběžně vrací v tom pořadí, v jakém se vyskytují na cestě. Z tabulky, která váže uzly k cestám, jsou pak odebrány staré vazby cest, které byly děleny, a jsou nahrazeny novými vazbami včetně vložených bodů. Tabulka po této operaci zůstává konzistentní a lze ji použít pro tvorbu geometrií cest. Dělí se jen pochozí cesty, dělit překážky a neklasifikované cesty nemá smysl, protože se u nich absence dlouhých úseků nevyužívá.

5.1.2 Tvorba geometrií

Abychom mohli využívat nadstavby PostGIS pro zodpovídání prostorových dotazů, musíme nejprve objektům přiřadit geometrie. V datech OSM máme určené pozice bodů a u složitějších objektů seznam bodů, ze kterých se objekty skládají a jejich pořadí. Z těchto informací sestavíme pomocí prostředků PostGISu geometrie jednotlivých objektů a k objektům si je uložíme. Jak jsme již zmínili dříve, používáme pouze uzly, cesty a z relací multipolygony. Z multipolygonů bereme pouze vnější okraj, což zařídíme snadno pomocí prostředků PostGISu, který přímo podporuje multipolygony a umí určit jejich vnější okraj.

5.1.3 Příprava překážek

Jako překážky bereme všechny objekty, které klasifikací získaly typ překážka. U uzavřených cest je potřeba rozhodnout, zda se jedná o obvod nebo i vnitřní oblast. V rámci klasifikace se u každé cesty na základě kritérií rozhodne, jestli by daná cesta měla být plochou a tato informace je využívána právě při přípravě překážek. Součástí tvorby překážek také musí být ošetření nesprávných geometrií, které vznikají jednak na okrajích mapy, kde některé cesty nejsou kompletní, protože obsahují body mimo výřez, jednak uvnitř mapy vlivem chybných editací od uživatelů, kde vznikají špatně uzavřené polygony, dva uzly na stejném místě a další chyby, které nesmí vést k selhání přípravy dat.

5.1.4 Tvorba zkratek

Tvorba zkratek je stejně jako v bakalářské práci i zde nejsložitější a nejnáročnější částí přípravy dat. Nejprve je nutné zvolit body, mezi kterými je možné zkratky vytvářet. Protože musí spojovat nějaké pochozí cesty, nejprve vybereme všechny vrcholy, které jsou na nějakých pochozích cestách. Zkratky nelze vytvářet uvnitř překážek ani v podzemí, tyto vrcholy posléze z výběru odebereme a získáme všechny kandidáty na začátky a konce zkratek.

Nyní bychom potřebovali najít všechny dvojice bodů, které jsou k sobě blíže než 30 m[1] a z nich vybrat některé, ze kterých se stanou zkratky. Takováto přímočará implementace je ale příliš pomalá, protože vrcholů je velké množství a funkce pro počítání vzdálenosti je výpočetně náročná, vede to tedy na dlouhý výpočetní čas.

Abychom tento problém obešli, využili jsme toho, že zkratky jsou lokální a není tedy třeba porovnávat dvojice bodů, které jsou od sebe příliš vzdáleny. Použili jsme techniku proložených čtvercových sítí, kde mapový výřez rozdělíme čtvercovou sítí na čtverce 100×100 metrů, čtverce očíslujeme a každému bodu přiřadíme číslo čtverce, ve kterém leží. Poté čtvercovou síť posuneme o 50 metrů v jednom směru, v druhém směru a úhlopříčné a akci pokaždé opakujeme. U každého bodu si nyní pamatujeme čísla čtyř čtverců, ve kterých bod leží. Pokud nyní vezmeme libovolné 2 body, které jsou k sobě blíže než 50 m, tak budou mít společný některý čtverec. Pro důkaz si představme první čtverec jako fixní. Pokud jsou oba body v něm, jsme hotovi. Pokud v něm oba nejsou, pak jeden leží uvnitř v nějakém kvadrantu a druhý vně ve stejném či sousedním kvadrantu. Pro každou dvojici kvadrantů můžeme vybrat čtverec z ostatních posunutí, který oba kvadranty pokrývá. Toto je možné pouze pro čtverce, jejichž hrana je dlouhá alespoň dvojnásobek maximální vzdálenosti dvojice bodů, pokud bychom tedy chtěli vytvářet zkratky delší než 50 metrů, museli bychom odpovídajícím způsobem zvětšit i čtverce.

Protože platí předchozí tvrzení, můžeme nyní vzít vždy jen ty dvojice bodů, které sdílí jeden ze čtverců. Takto jednak získáme méně dvojic, u kterých je potřeba testovat vzájemná vzdálenost, jednak se takovýto dotaz dá snadno zrychlit použitím indexů, protože se porovnávají dvě čísla na rovnost.

Když máme všechny blízké dvojice bodů, jako kandidáty na zkratky uvážíme všechny úsečky mezi blízkými dvojicemi, které neprotínají žádnou překážku. Z takovýchto dvojic pak vybíráme zkratky, které skutečně použijeme. Děláme tak pomocí grafu, kde hrany jsou zkratky a vrcholy konce zkratek. Každou zkratku pak přidáme s pravděpodobností $5/\delta$, kde δ je stupeň konce zkratky s vyšším stupněm v grafu. Tento výběr vede k situaci, že z každého vrcholu vede v průměru maximálně 5 zkratek, což je empiricky zjištěná konstanta na základě analýzy mapových dat a experimentů s generováním zkratek.

5.2 Příprava jízdních řádů

Jízdní řády jsou nejprve staženy pomocí shellového skriptu a nahrány do PostgreSQL databáze, jejíž schéma odpovídá formátu GTFS. Průchod přes databázi

[1] Konstanta byla zvolena empiricky z mapových dat, aby zkratky doplňovaly potřebné pěší vazby a zároveň nebyly zbytečně dlouhé.

jsme zvolili, poněvadž nám pak dává možnost ptát se pouze na část dat, aniž bychom museli držet celá data v paměti. Výhledově je také možné přesunout celé předzpracování do databáze, což by ušetřilo paměť nutnou pro předzpracování v současné implementaci.

Zpracování jízdních řádů dále pokračuje skriptem v jazyce Python, který nejprve převede zastávky a platnosti jízdních řádů do výstupního formátu PBF a následně rozdělí linky na sublinky, přiřadí k nim spoje a sestaví soubor `tt.bin` obsahující jízdní řády pro hledání spojení. Dále vytvoří `routes.bin` obsahující informace o linkách, v současné době není využíván a `stopslut.csv` obsahující údaje nutné pro propojení dat z GTFS s daty z OSM: id zastávky v GTFS, id zastávky v připravených datech, jméno zastávky, její souřadnice a informaci, jestli je zastávka podzemní. Pro rozdělování linek na sublinky je potřebné pro každý spoj zkoumat, jestli je součástí nějaké již známé sublinky, nebo tvoří sublinku novou, která ještě není známá. Pokud je sublinka již známá, pouze se k ní přidá další spoj, v opačném případě se vytvoří nová sublinka. Tuto logiku řeší třída `Subroute`. Zpracování jízdních řádů je poměrně náročné na paměť, při dalších úpravách kódu by bylo vhodné algoritmus přepsat efektivněji či využít toho, že jsou data již v databázi a přepsat přípravu data pomocí databázových funkcí.

Data jízdních řádů jsou již připravena pro použití ke hledání, dále zpracováváme pouze propojovací soubor `stopslut.csv`. Ten nejprve nahrajeme do databáze jako novou tabulku. Poté zjistíme, které zastávkové stojany mají ID v OSM a tuto informaci si u zastávek uložíme. Protože následně budeme opět hledat zkratky, uložíme si pro každou zastávku i údaj o tom, ve kterých dvou čtvercích leží. Následuje generování zkratek mezi zastávkami a pochozími cestami. Toto se děje zvlášť pro povrchové a zvlášť pro podzemní zastávky, povrchové spojujeme s body na povrchu, podzemní s body v podzemí. Z nalezených zkratek opět vyloučíme ty, které kříží nějakou překážku a nakonec přidáme vazby stanic metra, které jsou detailně zmapované v OSM s body zastavení v mapě. Na tuto vazbu používáme speciální typ hrany, abychom tyto detailní vazby odlišili od obecných zkratek, které nemusí vždy být možné.

Nyní již máme v databázi připravená všechna data potřebná pro tvorbu vyhledávacího grafu. Data z tabulek jsou exportována do CSV a pak je pomocí jednoduchého programu v C vytvořen výstupní soubor PBF s vyhledávacím grafem a údaji o vazbě na jízdní řád. V tomto kroku je také kontrolována dosažitelnost všech vrcholů grafu a je vybrána největší souvislá komponenta, která obsahuje alespoň polovinu vrcholů. Samotné vyhledávání pak probíhá vždy v souvislém grafu. Protože při změně jízdních řádů dochází ke změnám očíslování zastávek, je potřeba přegenerovat i tu část generování vyhledávacího grafu, která zajišťuje párování zastávek. Tato část není výpočetně náročná, je proto možné ji spouštět při každé změně jízdních řádů. Při aktualizaci mapových dat jízdní řády není třeba přegenerovávat.

5.3 Vyhledávání

Vyhledávací část jsme navrhli jako knihovnu, která umožňuje snadnou integraci do jiných projektů. Knihovna je napsána v jazyce C s využitím knihovny LibUCW. Spolu s knihovnou jsme napsali i jednoduchý demonstrační program, který umožňuje vyhledávání z příkazové řádky. K pohodlnému interaktivnímu

hledání jsme rozšířili webovou aplikaci, která umožňuje hledání spojení na mapě spolu s vizualizací jednotlivých nalezených tras a umožňuje jejich export do formátu GPX.

Pro běh Dijkstrova algoritmu potřebujeme udržovat rozpracované vrcholy v haldě. Protože vrcholy vložené do haldy budeme potřebovat i po jejich zpracování, abychom dokázali zrekonstruovat nalezené trasy, ukládáme si je stranou a do haldy vkládáme jen pointery na tyto struktury. Protože během hledání vzniká těchto struktur velké množství, z důvodu efektivity při alokaci a snadného uvolňování po nalezení tras je ukládáme do memory-poolu, který poskytuje knihovna LibUCW. Takto můžeme snadno alokovat velké množství položek a pak všechny naráz na konci uvolnit.

Protože při vyhledávání zkoumáme nejen čas příjezdu, ale i penaltu, může se v haldě vyskytnout jeden grafový vrchol vícekrát s různými časy a penaltami.

5.3.1 Majorizace vrcholů

Při vyhledávání trasy často při zpracovávání hrany narazíme na situaci, kdy právě zpracovávaná hrana zajistí do cílového vrcholu dřívější příchod s nižší penaltou než nějaká jiná hrana, kterou jsme již do haldy přidali dříve. Měření na datech pro Prahu ukazují, že 20 % přidaných položek do haldy je v průběhu algoritmu majorizováno. V případě, že aktuální hrana majorizuje nějakou položku v haldě, pak tuto položku z haldy smažeme a smažeme ji i ze seznamu u vrcholu, ke kterému náleží, protože se tato položka nebude vyskytovat v žádné nalezené trase.

5.3.2 Vyhledávání přes půlnoc

Vyhledávání v MHD ve večerních hodinách přidává další problém, který je potřeba vyřešit. Pokud vyhledáváme na konci dne, může se stát, že budeme cestovat přes půlnoc. Pak je třeba, v současném formátu jízdních řádů, hledat jednak v jízdních řádech pro končící den, kde jsou zaznamenány trasy těch spojů, které začínají před půlnocí, ale do cílové zastávky přijedou až po půlnoci, jednak v jízdních řádech pro den následující, kde jsou uvedeny spoje vyjíždějící až po půlnoci. V naší implementaci si rovnou připravíme jízdní řád pro současný a příští den a v případě, že zpracováváme vrchol, který má čas příjezdu až po půlnoci, použijeme oba jízdní řády, jinak pouze ten pro současný den.

5.3.3 Výpočet penalt

Aby bylo možné hledat co nejvhodnější trasy pro různé požadavky, implementovali jsme obecné funkce pro výpočet penalty pro jednotlivé události, které mohou na trase nastat. Pro základní parametrizaci spojení jsme implementovali několik různých druhů penalt, které si uživatel může zvolit v konfiguračním souboru. Jejich popis je uveden v kapitole 7.

Protože požadavky na preferenci různých typů tras mohou být velmi různorodé a je těžké je obecně parametrizovat, očekáváme, že si uživatel případně upraví funkce pro výpočet penalt podle svých potřeb. Funkce jsme proto vyčlenili do souboru `penalty.c` a popis jejich parametrů uvádíme níže. Funkce vždy

vrací hodnotu typu `double`, která reprezentuje penaltu spočítanou v dané funkci. Protože penalty také mohou být závislé na předchozí vyhledané trase, je ve struktuře `mmdijnode_t` připravena podstruktura `state`, do které je možné si ukládat stavové informace o dosud vyhledané části trasy. Tato struktura se vždy při vytváření nového záznamu zkopíruje z předka, tudíž se stavové informace propagují podél vyhledávané cesty. V současné době obsahuje pouze položku `vehicles`, která udává počet nástupů do vozidla po cestě a používá se při omezení počtu přestupů.

Funkce pro výpočet penalt jsou následující:

- ```
  calcChangePenalty(Graph__Graph * graph,
                    struct config_t conf,
                    struct ptedge_t * toEdge,
                    struct mmdijnode_t * point,
                    uint64_t time)
  ```

  slouží k výpočtu penalty za změnu typu hrany. Obvykle se využívá k penalizaci přestupů při cestě MHD. Argumenty funkce jsou následující:

  - `graph` odkazuje na vyhledávací graf
  - `conf` popisuje konfiguraci vyhledávání od uživatele
  - `toEdge` odkazuje na následující hranu, která bude MHD
  - `point` odkazuje jednak na aktuální bod, ve kterém se nacházíme, jednak tento bod obsahuje i informace o předchozí hraně.
  - `time` udává čas čekání v daném vrcholu

- ```
  calcPointPenalty(Graph__Graph * graph,
                   struct config_t conf,
                   Graph__Vertex *vert)
  ```

  slouží k výpočtu penalty v daném bodě. Lze ji použít například k penalizaci přecházení na semaforech. Argumenty funkce jsou následující:

  - `graph` odkazuje na vyhledávací graf
  - `conf` popisuje konfiguraci vyhledávání od uživatele
  - `vert` odkazuje na vrchol, ve kterém počítáme penaltu

- ```
  calcWalkPenalty(Graph__Graph * graph,
                  struct config_t conf,
                  Graph__Edge * edge)
  ```

  slouží k výpočtu penalty za pěší úsek. Argumenty funkce jsou následující:

  - `graph` odkazuje na vyhledávací graf
  - `conf` popisuje konfiguraci vyhledávání od uživatele
  - `edge` odkazuje na hranu, pro kterou počítáme penaltu

- ```
  calcTransportPenalty(Graph__Graph * graph,
                       struct config_t conf,
                       struct stop_route * r,
  ```

```
time_t arrival)
```

slouží k výpočtu penalty za úsek projetý MHD. Argumenty funkce jsou následující:

- `graph` odkazuje na vyhledávací graf
- `conf` popisuje konfiguraci vyhledávání od uživatele
- `r` odkazuje na použitou linku
- `arrival` udává čas příjezdu na cílovou zastávku

Maximální penaltu, po jejímž dosažení se již dále nepokračuje v rozšiřování nalezeného částečného spojení, jsme stanovili na 1 000 000. Tuto konstantu v kódu reprezentuje makro `PENALTY_INFINITY`.

5.3.4 Konfigurační soubor

Pro konfiguraci vyhledávaní slouží soubor `config/speeds.yaml`, ve kterém jsou uvedeny jednak rychlosti pěší chůze, jednak se zde konfigurují připravené penalty. Detailní popis konfiguračních parametrů je uveden v kapitole 7, nyní se budeme zabývat implementací. Pokud nějaké rychlosti či penalty udávají hodnoty pro parametr nějakého výčtu, například typ objektu nebo typ dopravního prostředku, je připraveno pole pokrývající všechny možné hodnoty tohoto výčtu a jsou do něj dosazeny výchozí hodnoty, pro konstantní parametry 0, pro parametry sloužící jako koeficienty 1. Následně jsou tyto výchozí hodnoty přepsány hodnotami uvedenými v konfiguraci. Pro penalty za linky je vytvořeno pole struktur `line_config_t`, ve kterých je pak uložena dvojice linka→penalta.

5.3.5 Knihovna algoritmu RAPTOR

Knihovnu implementující algoritmus RAPTOR jsme navrhli jako oddělenou knihovnu, stejně tak skripty zajišťující přípravu dat z jízdních řádů jsou také oddělené, tudíž lze snadno implementovat vyhledávač využívající pouze algoritmus RAPTOR bez závislostí na zbytku vyhledávacího kódu a v rámci přípravných prací jsme takový vyhledávač implementovali. Tento vyhledávač ale nevyužívá žádné pěší přesuny, ani mezi zastávkami stejného jména, pro praktické vyhledávání není tedy příliš vhodný, sloužil pro testování správnosti úprav jízdních řádů.

Knihovnu jsme implementovali jako sdílenou knihovnu v jazyce C, tudíž je snadno navázatelná na zbytek vyhledávacího algoritmu. Knihovna obsahuje 3 hlavní funkce – `gen_tt_for_date`, `search_con` a `search_stop_cons`. Funkce `search_con` implementuje celý algoritmus RAPTOR, v naší práci se dále nevyužívá.

Funkce `gen_tt_for_date` zajišťuje přípravu vyhledávacích dat pro konkrétní den z celého jízdního řádu. K vyhledávání spojení by bylo možné použít přímo jízdní řád pro celé období, ale pak by při každém zkoumání možných spojů ze zastávky bylo potřeba ošetřovat případy, kdy daný spoj linky nejede či daná linka není konkrétní den vůbec v provozu. Pro co nejjednodušší zpracování konkrétní zastávky při vyhledávání jsme se rozhodli si pro vyhledávání připravit konkrétní jízdní řád na daný den, který obsahuje jen linky a spoje, které v daný den jedou.

V současnosti se jízdní řád generuje pro každý den použitý v konkrétním hledání, ale bylo by možné si kešovat jízdní řády pro nejpoužívanější dny, což by stále příliš nezhoršovalo nároky na paměť a ušetřil by se čas při hledání. Jízdní řád pro konkrétní den obsahuje všechny zastávky z celkového jízdního řádu, neobsluhované zastávky mají prázdnou množinu linek. Formát jízdního řádu pro konkrétní den je uveden v sekci 6.3.

Funkce `search_stop_conns` slouží k hledání zastávek dosažitelných bez přestupu z dané zastávky v daný čas. Funkce projde všechny linky, které daný den z dané zastávky jedou, najde první spoj, který odjíždí po čase příchodu na danou zastávku a všechny zastávky tohoto spoje uloží jako dosažitelné s časem příjezdu podle nalezeného spoje. Funkce vrátí seznam linek a pro každou dosažitelné zastávky s časem příjezdu. Přesný formát dat je uveden v sekci Formáty dat 6.3.

5.3.6 Webová aplikace

Webová aplikace je napsaná v Pythonu ve webovém frameworku Flask [17]. Python jsme zvolili z důvodu snadné interakce se sdílenými knihovnami pro C a rychlosti vývoje aplikací v něm. Flask pak je jednoduchým webovým frameworkem, který se osvědčil na jiných projektech. Webová aplikace má dvě části – backend, který zajišťuje hledání cesty a frontend, který předává informace o zvolených bodech backendu, zobrazuje mapu a výsledky vyhledávání.

Backend

Backend se kromě renderování hlavní stránky z připraveného templatu stará o hledání tras a jejich předávání frontendu. Když je ve frontendu zvolen výchozí a cílový bod, je vyvolán požadavek

`GET /search?flon=<flon>&flat=<flat>&tlon=<tlon>&tlat=<tlat>`,

kde `flon` a `flat` je zeměpisná délka a šířka výchozího bodu a `tlon` a `tlat` je zeměpisná délka a šířka cílového bodu. Souřadnice výchozího a cílového bodu jsou předány vyhledávací knihovně a počká se na vrácení výsledku hledání. Tento výsledek je ve formátu PBF, jak je specifikováno v sekci 6.3.5. Protože knihovna Leaflet vyžaduje pro zobrazení prostorových dat formát GeoJSON [11], je potřeba nalezené trasy převést do tohoto formátu. Při tomto převodu jsou současně spojeny všechny navazující hrany stejného typu (v případě MHD i stejného spoje) do jedné lomené linie a také jsou přidány časy příchodů na zastávku a odjezdu spoje MHD v případě nástupu do MHD a časy příjezdů na zastávku v případě výstupu z MHD. Všechna tato geografická data jsou pak zabalena jako FeatureCollection do formátu GeoJSON a spolu s informacemi o čase, pěší vzdálenosti a penaltě zabalena do pole všech tras, které je předáno frontendu ke zpracování. Přesný formát předávaných dat je popsán v sekci 6.3.5

Frontend

Frontend je napsán v JavaScriptu s použitím knihovny Leaflet pro operace s mapou. Jako mapový podklad je použita mapa od MapBoxu[2], přes kterou

[2] `https://www.mapbox.com/`

jsou vykreslovány jednotlivé nalezené trasy. Při nalezení trasy je také dynamicky vytvořena tabulka s informacemi o jednotlivých trasách. Klikáním na trasy je možné je zobrazovat a skrývat. Komunikace s backendem je řešena asynchronně pomocí callbacků.

6. Formáty dat

V rámci přípravy dat a vyhledávání tras jednak používáme interní struktury jednotlivých programovacích jazyků, jednak některé standardní formáty nezávislé na programovacím jazyku. Standardní formáty jsme představili v kapitole 2, v této kapitole se budeme věnovat již jen obsahu jednotlivých struktur.

6.1 Terminologie

V následujícím popisu struktur používáme u jednotlivých položek různá slovesa podle typu položky. Jedná se o následující:

- „udává“ značí, že položka je jednoduchého datového typu a je uložena přímo ve struktuře
- „odkazuje“ značí, že se jedná o odkaz na danou položku či jeho ekvivalent
- „popisuje“, případně „obsahuje popis“ značí, že položka je složený typ a je přímo uložena v popisované struktuře

6.2 Formáty používané při přípravě dat

6.2.1 Přípravný formát pro databázi

Program `parse` přečte data OSM, klasifikuje je a vytvoří 3 soubory reprezentující mapová data – (`nodes-stage1`, `ways-stage1` a `mp-stage1`), které udržují po řadě informace o uzlech, cestách a multipolygonech. Jednotlivé datové prvky jsou uloženy jako zprávy ve formátu PBF popsaném níže, soubor je tvořen vždy délkou zprávy a samotnou zprávou řazenými za sebe.

Zpráva pro uzel obsahuje tyto položky:

- `id` udává OSM id uzlu
- `lat` a `lon` udávají zeměpisné souřadnice uzlu v UTM
- `height` udává nadmořskou výšku uzlu v metrech
- `type` udává typ uzlu
- `inside` udává, zda je uzel uvnitř nějaké překážky
- `inTunnel` udává, zda je uzel v podzemí
- `onBridge` udává, zda je uzel na mostě
- `stop` udává, zda je uzel zastávka
- `ref` udává hodnotu tagu `ref` u uzlu, používá se pro párování zastávek MHD

Zpráva pro cestu obsahuje tyto položky:

- `id` udává OSM id cesty
- `refs` obsahuje posloupnost OSM id vrcholů, ze kterých se cesta skládá
- `area` udává, zda je daná cesta plocha
- `type` udává typ cesty (implicitně `WAY`)
- `bordertype` má udávat typ ohraničující cesty u nějaké oblasti, v současné době se nepoužívá
- `crossing` má udávat, jaký typ cesty zpracovávaná cesta kříží, v současné době se nepoužívá
- `bridge` udává, zda je cesta na mostě
- `tunnel` udává, zda je cesta v podzemí

Zpráva pro multipolygon obsahuje tyto položky:

- `id` udává OSM id multipolygonu
- `refs` udává, posloupnost cest, ze kterých se multipolygon skládá
- `roles` udává pro každou cestu z předchozí posloupnosti její roli (zda je vnější či vnitřní)
- `type` udává typ multipolygonu

6.2.2 Výstupní data z databáze

Po zpracování dat pomocí PostgreSQL a PostGISu jsou upravená data vyexportována do CSV. Vrcholy jsou vyexportovány do souboru `nodes.csv`, který obsahuje následující položky:

- `id` udává OSM id vrcholu
- `lat` a `lon` udávají zeměpisné souřadnice vrcholu v UTM
- `height` udává nadmořskou výšku vrcholu
- `type` udává typ vrcholu

Hrany jsou vyexportovány do souboru `ways.csv`, který obsahuje následující položky:

- `id` udává OSM id hrany
- `type` udává typ hrany
- `from` udává id výchozího vrcholu hrany
- `to` udává id cílového vrcholu hrany

Hrany jsou uvažovány jako obousměrné. Zkratky jsou vyexportovány do souboru `direct.csv`, který obsahuje následující položky:

- `id1` udává první vrchol zkratky
- `id2` udává druhý vrchol zkratky

Zastávky jsou vyexportovány do souboru `stops.csv`, který obsahuje následující položky:

- `id` udává OSM id zastávky
- `stop_id` udává id zastávky v GTFS
- `raptor_id` udává id zastávky v algoritmu RAPTOR
- `lat` a `lon` udávají zeměpisné souřadnice zastávky v UTM

Zkratky propojující zastávky se zbytkem grafu jsou vyexportovány do souboru `stops-direct.csv`, který obsahuje následující položky:

- `sid` udává id zastávky
- `nid` udává id vrcholu v grafu
- `objtype` udává typ zkratky

6.3 Formáty používané při vyhledávání tras

6.3.1 Formát jízdních řádů

Jízdní řády jsou po zpracování uloženy ve zprávě `Timetable`. Způsob provázání dat v jednotlivých položkách je popsán v kapitole 1.4.1. Zpráva má následující položky:

- `routes` obsahuje popis linek
- `stop_times` obsahuje časy zastavení spojů
- `trip_validity` obsahuje pro každý spoj index na platnost
- `validities` platnost – obsahuje popis, které dny nějaký spoj jede a které ne
- `route_stops` udává, která linka má jaké zastávky
- `stops` obsahuje popis zastávek
- `transfers` obsahuje popis pěších přestupů, v práci není používána
- `stop_routes` udává, které linky jedou přes jakou zastávku

Každá linka je reprezentována zprávou typu `Route` s následujícími položkami:

- `id` udává RAPTOR id linky
- `nstops` udává počet zastávek linky

- `ntrips` udává počet spojů linky
- `stopsidx` udává index do pole `route_stops`, kde začínají zastávky linky
- `tripsidx` udává index do pole `stop_times`, kde začínají časy spojů linky
- `servicesidx` udává index do pole `trip_validities`, kde začínají platnosti spojů linky
- `name` udává název linky
- `type` udává typ dopravního prostředku

Každá zastávka je reprezentována zprávou typu `Stop` s následujícími položkami:

- `id` udává RAPTOR id zastávky
- `nroutes` udává počet linek projíždějících zastávkou
- `ntransfers` udává počet pěších přestupů ze zastávky
- `routeidx` udává index do pole `stop_routes`, kde začínají linky dané zastávky
- `transferidx` udává index do pole `transfers`, kde začínají přestupy ze zastávky
- `name` udává jméno zastávky
- `underground` udává, zda je zastávka v podzemí

Každé zastavení spoje je reprezentováno zprávou typu `StopTime` s následujícími položkami:

- `arrival` udává čas příjezdu do zastávky
- `departure` udává čas odjezdu ze zastávky

Každá množina dní, kdy některý spoj jede, je reprezentována pomocí zprávy typu `Validity`, která má následující položky:

- `start` udává Unixový čas prvního dne platnosti
- `end` udává Unixový čas posledního dne platnosti
- `bitmap` udává bitmapu platnosti počínaje prvním dnem platnosti. Každý den odpovídá jednomu bitu.

Pokud bychom měli nějaké pěší přestupy, byly by reprezentovány zprávou typu `Transfer` s následujícími položkami:

- `from` udává počáteční zastávku
- `to` udává koncovou zastávku
- `time` udává čas potřebný pro přesun mezi zastávkami

6.3.2 Jízdní řád pro konkrétní den

Pro samotné vyhledávání si vždy nejprve připravíme jízdní řád pro tento konkrétní den. Jízdní řád je reprezentován strukturou `timetable` v C s následujícími položkami:

- `nroutes` udává počet linek
- `routes` obsahuje popis linek
- `st_times` obsahuje popis zastavení spojů v zastávkách
- `transfers` obsahuje popis pěších přestupů
- `nstops` udává počet zastávek
- `stops` obsahuje popis zastávek
- `rt_stops` obsahuje odkazy na zastávky pro linky
- `st_routes` obsahuje odkazy na linky pro zastávky

Každá linka je popsána strukturou `route` s následujícími položkami:

- `pbroute` odkazuje na linku v původním jízdním řádu
- `nstops` udává počet zastávek linky
- `ntrips` udává počet spojů linky
- `stops` odkazuje na začátek odkazů na zastávky dané linky
- `trips` odkazuje na začátek zastavení spojů linky v zastávkách
- `name` udává jméno linky

Každé zastavení spoje v zastávce je popsáno strukuturou `st_time`

- `arrival` udává čas příjezdu spoje do zastávky
- `departure` udává čas odjezdu spoje ze zastávky

Každá zastávka je popsána strukturou `stop` s následujícími položkami

- `pbstop` odkazuje na zastávku v původním jízdním řádu
- `nroutes` udává počet linek projíždějících zastávkou daný den
- `ntransfers` udává počet pěších přestupů z dané zastávky
- `routes` odkazuje na začátek odkazů na linky projíždějící zastávkou
- `transfers` odkazuje na začátek pěších přestupů pro danou zastávku
- `name` udává název zastávky

Každý pěší přestup mezi zastávkami je popsán strukturou `transfer`. Tato struktura není v současné době využívána. Má tyto položky:

- `from` odkazuje na výchozí zastávku
- `to` odkazuje na cílovou zastávku
- `time` udává čas v sekundách potřebný na přestup z výchozí do cílové zastávky

6.3.3 Vyhledávací graf

Vyhledávací graf je uložen v PBF ve zprávě `Graph`, která obsahuje tyto položky:

- `vertices` obsahuje popis vrcholů grafu
- `edges` obsahuje popis hran grafu
- `stops` obsahuje popis vazeb mezi zastávkami a vrcholy grafu

Každý vrchol je popsán pomocí zprávy `Vertex`, která obsahuje následující položky:

- `idx` udává číslo vrcholu
- `osmid` udává OSM id vrcholu
- `lat` a `lon` udávají zeměpisné souřadnice vrcholu v UTM
- `type` udává typ vrcholu
- `height` udává nadmořskou výšku vrcholu v metrech
- `stop_id` pokud je vrchol zastávka, pak udává GTFS id zastávky

Každá hrana je popsána pomocí zprávy `Edge`, která obsahuje následující položky:

- `idx` udává číslo hrany
- `osmid` udává OSM id hrany
- `vfrom` udává číslo výchozího vrcholu
- `vto` udává číslo cílového vrcholu
- `type` udává typ hrany
- `crossing` má udávat typ cesty, kterou hrana kříží, nepoužívá se
- `dist` udává délku hrany

Každá zastávka je popsána pomocí zprávy `Stop`, která obsahuje následující položky:

- `idx` udává číslo zastávkového vrcholu
- `osmid` udává OSM id zastávkového vrcholu
- `raptor_id` udává pořadí zastávky v algoritmu RAPTOR
- `stop_id` udává id zastávky v GTFS

6.3.4 Struktury používané při hledání trasy

Pro samotné hledání potřebujeme haldu, kterou budeme používat pro Dijkstrův algoritmus. Halda je implementována pomocí struktury `mmqueue_t`, která obsahuje následující položky:

- `pool` odkazuje na memory pool pro alokaci struktur pro zpracovávané vrcholy
- `graph` odkazuje na vyhledávací graf popsaný výše
- `vert` odkazuje na strukturu pro aktuální vrchol
- `vertlut` obsahuje pro každý vrchol grafu pole odkazů na jeho struktury vzniklé při vyhledávání
- `heap` obsahuje odkazy na struktury pro vrcholy, které jsou aktuálně v haldě
- `n_heap` udává počet prvků v haldě

Každý vrchol v haldě je popsán pomocí struktury `mmdijnode_t`, která obsahuje následující položky:

- `prev` odkazuje na předchozí vrchol na trase
- `osmvert` pokud je vrchol z OSM, odkazuje na vrchol ve vyhledávacím grafu
- `stop` pokud je vrchol zastávka, odkazuje na zastávku v jízdním řádu pro celou platnost
- `edge` popisuje hranu, která vede do zpracovávaného vrcholu
- `state` popisuje stavové informace o částečně nalezené trase, v současné době obsahuje pouze položku `vehicles` udávající počet použitých spojů dosud po trase
- `heapidx` pokud je položka v haldě, udává index položky v poli reprezentujícím haldu
- `reached` indikuje, zda již byl vrchol dosažen
- `completed` indikuje, zda již byl vrchol uzavřen
- `arrival` udává čas příjezdu do vrcholu
- `penalty` udává penaltu trasy až do současného vrcholu včetně

Protože přicházející hrana může být jak hranou z OSM, tak hranou MHD, je reprezentována pomocí struktury `edge_t`, která obsahuje následující položky:

- `edge_type` udává typ hrany (`EDGE_TYPE_WALK` nebo `EDGE_TYPE_PT`)
- union `osmedge` a `ptedge` odkazuje buď na konkrétní hranu ve vyhledávacím grafu, nebo na popis hrany MHD

Hrany pomocí MHD jsou popsány strukturou `ptedge_t`, která obsahuje následující položky:

- `departure` udává čas odjezdu z výchozí zastávky v sekundách od začátku dne jízdního řádu
- `route` odkazuje na linku v jízdním řádu

Pro reprezentaci nalezených spojení MHD z dané zastávky slouží struktura `stop_conns`, která má následující položky:

- `n_routes` udává počet linek jedoucích z dané zastávky
- `routes` popisuje linky jedoucí z dané zastávky

Každá linka je popsána pomocí struktury `stop_route`, která obsahuje následující položky

- `departure` udává čas odjezdu linky ze zastávky, v sekundách od počátku dne jízdního řádu
- `pbroute` odkazuje na linku v původním jízdním řádu
- `n_stops` udává počet zastávek na dané lince směrem na konečnou
- `stops` popisuje jednotlivá zastavení na lince

Jednotlivá zastavení spoje v zastávkách po cestě jsou popsána pomocí struktury `stop_arr`, která obsahuje následující položky:

- `to` odkazuje na zastávku, kde spoj stojí
- `arrival` udává čas příjezdu v sekundách od půlnoci dne, pro který byl vygenerován jízdní řád

6.3.5 Výsledky vyhledávání

Výsledky vyhledávání jsou z knihovny předávány jako zabalená zpráva PBF. Zpráva obsahuje pouze jednu opakovanou položku typu `Route`, která obsahuje všechny nalezené trasy. Každá trasa pak má následující položky:

- `time` udává čas příjezdu do cílového bodu
- `dist` udává celkovou délku pěších přechodů na trase
- `penalty` udává penaltu trasy
- `point` obsahuje jednotlivé body trasy

Jednotlivé body trasy jsou spolu s hranou, která do nich vede, popsány zprávou `Point` s následujícími položkami:

- `lat` a `lon` udávají zeměpisnou šířku a délku bodu v systému WGS-84
- `height` udává nadmořskou výšku bodu

- `departure` pokud je hrana MHD, pak udává čas odjezdu z místa nástupu
- `arrival` udává čas příchodu/příjezdu do bodu
- `osmvert` pokud je bod z OSM, pak obsahuje informace o vrcholu z OSM, je to zpráva typu `graph.Vertex`, která je popsána výše
- `stop` pokud je bod zastávka, pak obsahuje informace o vrcholu z GTFS
- `edgetype` udává, zda jde o pěší hranu, nebo přesun MHD
- `walkedge` pokud je hrana chůze, pak obsahuje informace o hraně z OSM, je to zpráva typu `graph.Edge`, která je popsána výše
- `ptedge` pokud je hrana MHD, pak obsahuje informace o hraně z GTFS.

Informace o zastávce jsou uloženy ve zprávě `Stop`, která má následující položky:

- `name` udává název zastávky
- `id` udává kód zastávky tak, jak je v GTFS

Informace o hraně využívající MHD jsou uloženy ve zprávě `PTEdge`, která obsahuje následující položky:

- `name` udává název linky
- `id` udává id linky tak, jak je v GTFS

Výsledek vyhledávání je interně reprezentován typem `search_result_t`, který je svou strukturou velmi podobný výše popsanému PBF.

Vyhledané trasy je možné uložit jako GPX [13]. V naší aplikaci používáme pouze trasy, každá nalezená vyhledaná trasa odpovídá jedné trase v GPX, tato trasa je rozdělena na pěší segmenty a segmenty MHD. Typ segmentu je uložen u každého segmentu v sekci `<extensions>` v tagu `<type>`. Každá trasa má ve své sekci `<extensions>` uložený čas příjezdu, pěší vzdálenost a penaltu postupně v tazích `<arrival>`, `<len>` a `<penalty>`.

6.4 Formáty používané ve webové aplikaci

6.4.1 Výsledky vyhledávání

Výsledky vyhledávání jsou předávány z backendu do frontendu webové aplikace (viz 5.3.6) pomocí formátu JSON [10]. Ten je tvořen polem vyhledaných tras, kde každá trasa je slovník s následujícími klíči:

- `time` udávající čas příjezdu do cíle
- `dist` udávající celkovou pěší vzdálenost na nalezené trase
- `penalty` udávající penaltu vyhledané trasy
- `geojson` obsahující grafickou reprezentaci nalezené trasy.

Grafická reprezentace nalezené trasy je reprezentována ve formátu GeoJSON [11] a obsahuje lomené linie jednotlivých typů popisující průběh cesty a bodové prvky pro nástup a výstup z prostředku MHD.

Liniové prvky jsou reprezentovány typem LineString, více sousedících hran stejného typu je vždy spojeno v jeden úsek typu LineString. Hrany reprezentující různé spoje MHD se do jednoho úseku nespojují. Úseky mají následující atributy:

- `type` udávající typ úseku (dle číslování objtype)
- `name` udávající jméno linky, atribut je přítomen pouze, pokud jde o úsek typu MHD

Bodové prvky jsou reprezentovány pomocí typu Point s následujícími atributy:

- `type` udávající typ bodu, zde vždy číslo, které reprezentuje objtype PUBLIC_TRANSPORT
- `name` udávající jméno zastávky
- `subtype` udávající operaci na zastávce (`departure` pro nástup, `arrival` pro výstup)
- `departure` čas odjezdu spoje ze zastávky
- `arrival` pro nástupní bod čas příchodu na zastávku, pro výstupní bod čas příjezdu do zastávky

7. Uživatelská dokumentace

7.1 Příprava dat

Pro přípravu dat je nejprve nutné nainstalovat potřebné externí knihovny (viz soubor `Install`) do systému nebo do adresáře `ext-lib`. Poté je nutné zkompilovat všechny zdrojové kódy zavoláním `make` v adresáři `compiled`. Dále je nutné vytvořit databáze pro přípravu dat. Pro mapová data je potřeba vytvořit databázi `osmawalk-prepare`, pro jízdní řády je potřeba vytvořit databázi `gtfs_praha`. Pak je již možné přistoupit k samotné přípravě dat. Pokud je vše správně nastavené, stačí spustit skript `prepare.sh` v kořenovém adresáři projektu a po několika hodinách budou data připravena. Pokud skript selže, dále popíšeme činnosti, které skript provádí, aby bylo možné jednotlivé fáze spouštět ručně a hledat, kde nastala chyba.

1. `osm/prepare.sh` slouží ke stažení dat OSM, SRTM, spojení jednotlivých dílů dat SRTM a tvorbu výřezu z dat OSM. Výstupem skriptu je soubor `osm/praha.osm` s výřezem z dat OSM a soubor `heights.bin` se spojenými daty SRTM pro daný výřez.

2. `compiled/parse` slouží ke klasifikaci dat OSM, určení typů objektů, určení vrcholů na mostech a pod zemí a přiřazení nadmořské výšky k uzlům. Výstupem jsou soubory `data/nodes-stage1`, `data/ways-stage1` a `data/mp-stage1` s jednotlivými objekty. Formát uložení je popsán v sekci 6.2. Nastavení klasifikátoru je v souboru `config/waytypes.yaml` pro cesty a `config/nodetypes.yaml` pro body. Formát konfiguračních souborů je popsán v bakalářské práci [6].

3. `todb.sh` slouží k nahrání klasifikovaných dat do databáze.

4. `postgis/process.sh` řídí úpravy dat v databázi, uvnitř obsahuje volání jednotlivých kroků popsaných v kapitole 5. Výstupem skriptu jsou soubory `data/....csv` obsahující vrcholy a hrany dat připravených pro vyhledávání.

5. `compiled/csvtograph` vytváří vyhledávací graf a vybírá v něm největší souvislou komponentu, kterou uloží pro použití vyhledávačem jako soubor `data/postgis-graph.pbf`.

Pro přípravu jízdních řádů stačí v adresáři `ext-lib/mmpf/raptor` spustit skript `update-gtfs.sh`. Tento skript stáhne aktuální jízdní řády, nahraje je do databáze a spustí skript `prepare.py`, který připraví soubor `tt.bin` s jízdními řády.

Při přípravě dat je potřeba vždy nejprve aktualizovat jízdní řády a pak teprve aktualizovat mapu. Při přípravě mapových dat dochází i k párování zastávek MHD a je proto potřeba již mít vygenerované soubory sloužící ke spojení obou datových souborů.

7.2 Konfigurační soubor

Pro konfiguraci vyhledávání slouží soubor `config/speeds.yaml`. Tento soubor má formát YAML a skládá se z několika úrovní asociativních polí. Kořenové pole obsahuje jako klíče jednotlivé kategorie, které jsou nastavovány a jako hodnoty buď přímo nastavení, nebo další mapování uchovávající nastavení. Kategorie nastavení jsou následující:

- `speeds` udává absolutní rychlosti pěších přesunů. Hodnotou je mapování typ cesty → rychlost v km/h.
- `ratios` udává rychlosti pěších přesunů jako násobek rychlosti pohybu po cestě typu `WAY`. Hodnotou je stejné mapování jako u `speeds` V případě, že je definována rychlost obojím způsobem, má přednost absolutní rychlost před poměrnou.
- `penalties` udává penaltu za jednotlivé druhy cest. Hodnotou je mapování typ cesty → penalta. Penalta se násobí počtem sekund strávených procházením hrany.
- `heights` udává koeficient délkového prodloužení v závislosti na změně nadmořské výšky. Hodnotou je mapování s klíči `upscale` a `downscale`. Pokud je cesta do kopce, je prodloužena o `<rozdíl výšky>`×`upscale` metrů, pokud je z kopce, o `<rozdíl výšky>`×`downscale`. Cesta může být i zkrácena, pokud by celková délka byla záporná, je uvažována jako nulová.
- `pt-time-penalties` udává penalty za typ dopravního prostředku násobené dobou strávenou v dopravním prostředku v sekundách. Hodnotou je mapování typ prostředku → penalta, kde typ prostředku je brán z GTFS a psán malými písmeny, např. `tram`, `ferry` nebo `bus`.
- `pt-fixes-penalties` udává fixní penalty za typ dopravního prostředku. Počítá se pro každou cestu daným typem prostředku jednou nezávisle na délce cesty. Hodnotou je stejné mapování jako u `pt-time-penalties`.
- `line-penalties` udává penaltu za použití dané linky. Penalta se počítá jednou za každé použití dané linky. Hodnotou je mapování jméno linky → penalta.
- `max-vehicles` udává maximální počet spojů MHD použitých po cestě. Hodnotou je číslo udávající tento počet.
- `geton-penalty` udává penaltu za nástup do spoje MHD. Je to fixní penalta za každý nástup. Hodnotou je číslo udávají penaltu.
- `min-wait` udává minimální čas od příchodu / příjezdu na zastávku do odjezdu spoje. Hodnotou je čas v sekundách.

U kategorií, které udávají penaltu, lze místo čísla napsat `inf`, což znamená nekonečnou penaltu, tedy pokud nějaké částečná trasa dostane takovouto penaltu, tak už se dále nepokračuje ve vyhledávání zbytku trasy.

7.3 Konzolová aplikace

Konzolová aplikace slouží převážně pro testování, zda vyhledávací knihovna pracuje stabilně, ale dá se použít i pro jednoduché hledání. Aplikaci spustíme v adresáři `compiled` příkazem `./search <flat> <flon> <tlat> <tlon> [time]`. `flat` a `flon` určují zeměpisnou šířku a délku výchozího bodu, `tlat` a `tlon` souřadnice cílového bodu a `time` unixový timestamp času, od kterého hledat. Pokud není čas uveden, hledá se od okamžiku spuštění. Aplikace najde trasy a na standardní výstup vypíše ke každé trase použité spoje MHD. Navíc v aktuálním adresáři vytvoří soubor `track.gpx`, který obsahuje jednotlivé nalezené trasy.

7.4 Webová aplikace

Webovou aplikaci je možné vyzkoušet na `http://mhd.bezva.org` nebo si ji spustit lokálně pomocí příkazu `python wsgi.py` v adresáři `webapp`. V případě, že knihovna `libraptor.so` není umístěná tam, kde systém hledá sdílené knihovny, je potřeba ji do této cesty buď přesunout, nebo zavolat příkaz `export LD_LIBRARY_PATH=<cesta k libraptor.so>`. Po spuštění aplikace je možné si otevřít prohlížeč na stránce, která se vypíše během spouštění aplikace. I v případě, že spouštíme aplikaci lokálně, je potřeba mít k dispozici připojení k internetu, protože mapové podklady a javascriptové knihovny se načítají ze serverů třetích stran.

Po otevření webové aplikace je zobrazené hlavní okno s mapou. Interakce s mapou, kromě základního posouvání a zvětšování, probíhá kliknutím na bod. Prvním kliknutím je zvolen výchozí bod, ze kterého hledat, druhým kliknutím je zvolen cílový bod do kterého se hledá trasa. Volbou cílového bodu je odeslán požadavek na vyhledání trasy. Po vyhledání jsou v levém sloupci zobrazená shrnutí jednotlivých nalezených tras v pořadí podle času příjezdu do cíle a první trasa je zobrazena v mapě. Zobrazení nebo skrytí jednotlivých tras se provede kliknutím na její souhrn v levém sloupci. Jednotlivé části trasy jsou obarveny podle typů hran, po kterých vedou a dále jsou zvýrazněna místa přestupu na MHD. Detailní informace o MHD lze zobrazit kliknutím na ikonu u přestupní zastávky nebo na hranu spoje MHD. Trasy lze exportovat do GPX kliknutím na tlačítko „Stáhnout jako GPX“ v hlavičce stránky.

8. Výsledky

Pro zhodnocení výsledků našeho vyhledávače jsme provedli dva druhy testů. V rámci prvního jsme porovnávali výsledky našeho vyhledávače s ostatními veřejně dostupnými vyhledávači spojení. V druhém testu jsme na pevně dané trase a času zkoumali, jaký vliv na nalezené trasy má různé nastavení penalt a rychlostí chůze. Aplikaci jsme také profilovali, abychom zjistili, v jakých částech se tráví nejvíce času a měly by se optimalizovat jako první.

8.1 Porovnání s jinými vyhledávači

Výsledky našeho vyhledávače jsme porovnávali s veřejně dostupnými vyhledávači spojení po Praze: IDOS[1], Mapy.cz[2] a Google Maps[3].

IDOS je nejstarší a nejznámější vyhledávač spojení. Jako jediný z vyhledávačů nepodporuje hledání pěších tras, vyhledává pouze v jízdních řádech, přestupy mezi zastávkami řeší pomocí tabulky, která obsahuje dvojice zastávek a čas přesunu mezi nimi. Jako vstup je možné zadat adresu, ale vyhledávač nalezne pouze nejbližší zastávky podle vzdušné vzdálenosti a hledá z nich / do nich. Na druhou stranu umožňuje široké možnosti nastavení vyhledávání spojů – volbu minimálních časů na přestup, typů dopravních prostředků, počtu přestupů, ...

Mapy.cz a Google Maps jsou původně mapové služby, které do svých vyhledávačů přidaly možnost vyhledávání kromě pěších cest i spojení MHD. Protože se jedná primárně o mapové služby, vyhledávače neumožňují žádné (Mapy.cz) nebo jen velmi omezené (Google Maps) přizpůsobení.

Při porovnávání vyhledaných spojení jsme všude nechali výchozí nastavení a náš vyhledávač jsme nastavili tak, jak si myslíme, že jsou nastaveny ostatní vyhledávače, aby byly výsledky porovnatelné. Konkrétní nastavení je výchozí nastavení `config/speeds.yaml` v příloze. Přestupy nebyly nijak penalizovány, na zastávce bylo nutné mít aspoň 30 s čas mezi příchodem / příjezdem a odjezdem.

Testovali jsme následující trasy, které známe i z reálného provozu. Jednotlivé trasy zkoumaly různé aspekty vyhledávačů:

- *Kolej 17. listopadu – Přírodovědecká fakulta UK, Albertov*
  Tato cesta je zaměřena na obtížnou dopravní situaci okolo kolejí 17. listopadu. Spojení na Nádraží Holešovice je nejspolehlivější pěšky, autobus 201 většinou nejede dle jízdního řádu. V případě cesty tramvají č. 17 je možné jednak využít od kolejí autobus 112, ovšem s výstupem na Povltavské, nikoli na Trojské, případně dojít přímo na Trojskou pěšky. Všechny tyto možnosti dávají smysl při hledání cesty na Albertov. Na durhém konci také není situace přímočará, při cestě od metra je dobrou alternativou jít pěšky z Vyšehradu.

- *Kolej 17. listopadu – Čistírna odpadních vod, Bubeneč*
  Tato cesta zkoumá schopnosti pěší navigace, protože jednou z možností je jít pěšky přes Stromovku. Také je zde vidět rozdíl mezi naším vyhledávačem

[1] `http://idos.cz`
[2] `http://mapy.cz`
[3] `http://maps.google.com`

a Google Maps na jedné straně a IDOSem a Mapy.cz na straně druhé, které využívají znalosti jízdních řádů integrovaných vlaků a umí je na této cestě využít.

- *Kovanecká – Gymnázium Omská*
  Tato cesta opět zkoumá schopnost využívat delší pěší přesuny a také dávat přednost časově kratším cestám s přestupy.
- *FEL ČVUT, Dejvice – Hlávkova kolej*
  Tato cesta je zaměřena na hledání aktuálně nejrychlejší alternativy z několika rovnocenných. Dobré alternativy nevyžadují dlouhé pěší přesuny.

8.1.1 Kolej 17. listopadu – Albertov

Náš vyhledávač našel následující spojení:

- Spojení 0:

Zastávka	Příjezd	Odjezd	Linka
Kuchyňka		13:25	201
Nádraží Holešovice	13:28	13:32	C
I. P. Pavlova	13:40		

  Očekávaný příchod je 13:56.

- Spojení 2:

Zastávka	Příjezd	Odjezd	Linka
Kuchyňka		13:25	201
Nádraží Holešovice	13:28	13:32	C
Florenc	13:36	13:41	B
Karlovo náměstí	13:46		
Palackého náměstí		13:52	18
Albertov	13:56		

  Očekávaný příchod je 14:00.

- Spojení 3:

Zastávka	Příjezd	Odjezd	Linka
Kuchyňka		13:25	201
Nádraží Holešovice	13:28	13:31	6
Masarykovo nádraží	13:43	13:44	14
Albertov	13:59		

  Očekávaný příchod je 14:03.

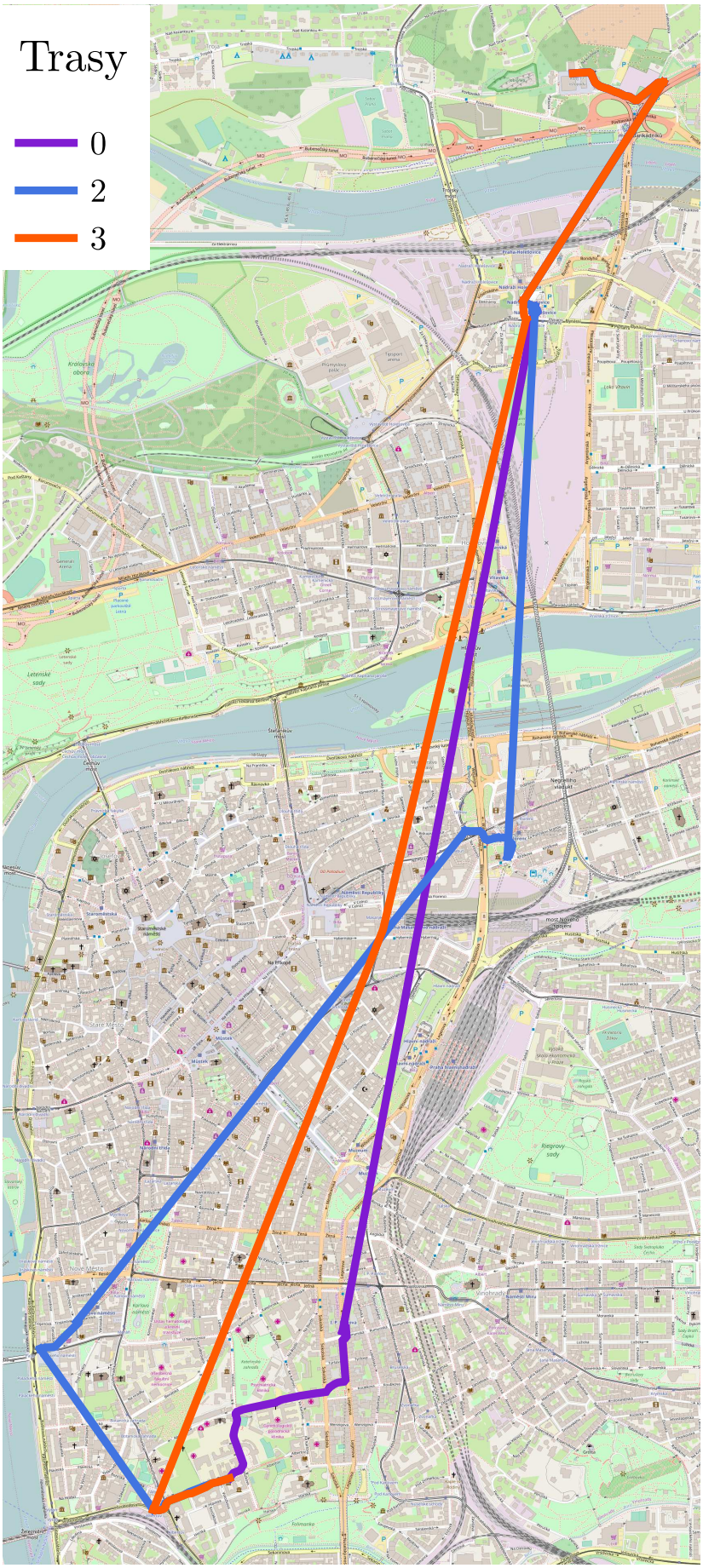
Vyhledávač IDOS našel následující spojení:

Zastávka	Příjezd	Odjezd	Linka
Kuchyňka		13:23	201
Bulovka	13:26	13:28	24
Albertov	14:04		

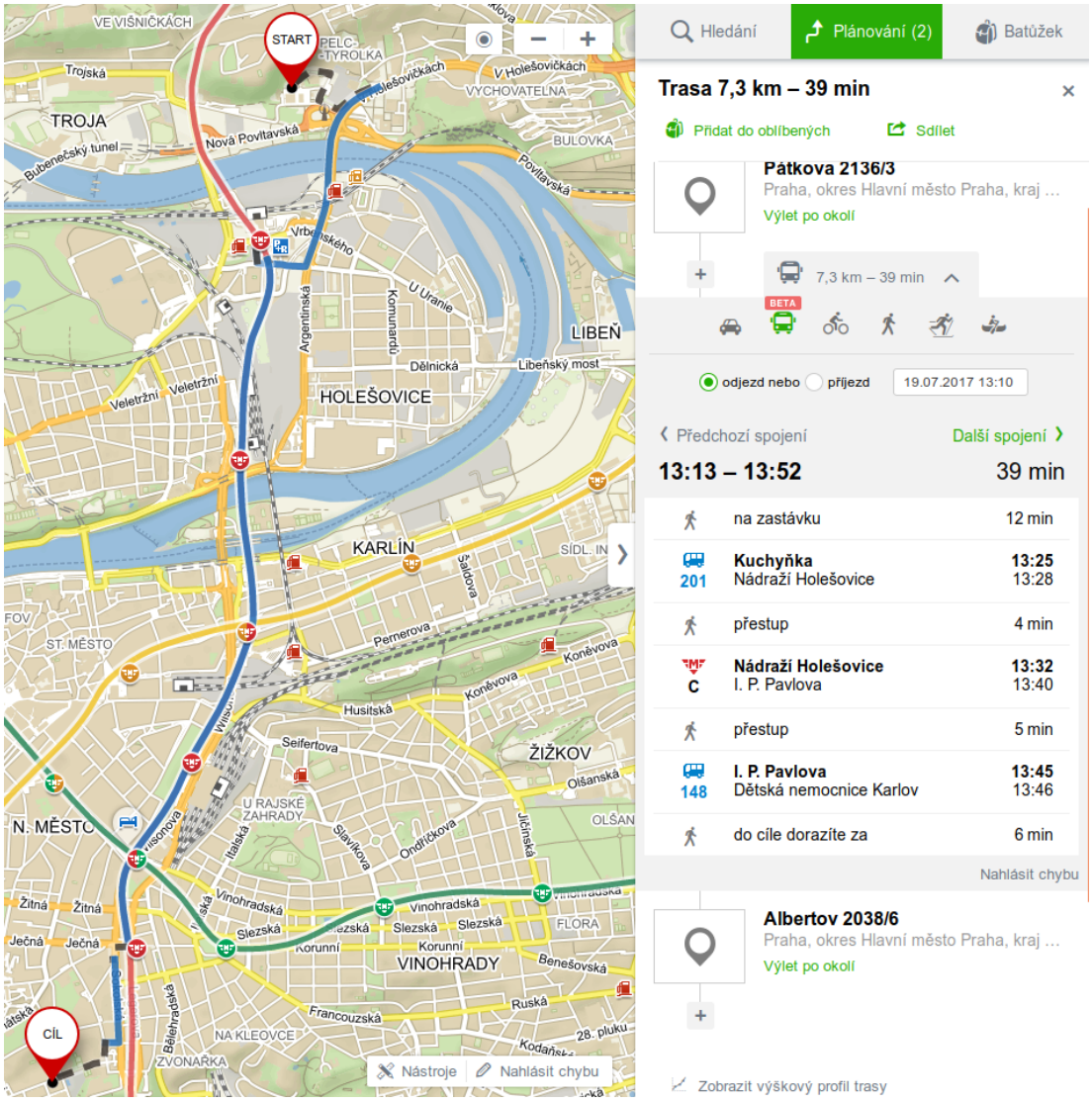
Zastávka	Příjezd	Odjezd	Linka
Kuchyňka		13:25	201
Nádraží Holešovice	13:28	13:32	C
Florenc	13:36	13:41	B
Karlovo náměstí	13:47		
Palackého náměstí		13:52	18
Albertov	13:56		

V tomto spojení se ukázala výhoda vyhledávačů, které umí delší pěší přesuny. Vlivem výluky není možné jet přímo z Karlova náměstí na Albertov a nejkratší cesta se ukazuje být přes I. P. Pavlova. Náš vyhledávač našel tuto trasu ve stejné podobě jako Google Maps, Mapy.cz ještě využily autobusu 148 k přiblížení se. Odhadované časy se liší jen o rychlost chůze. IDOS kvůli absenci mapových podkladů nalezl jen trasy vedoucí na Albertov, které náš vyhledávač nalezl také, i když s mírně jiným průběhem.

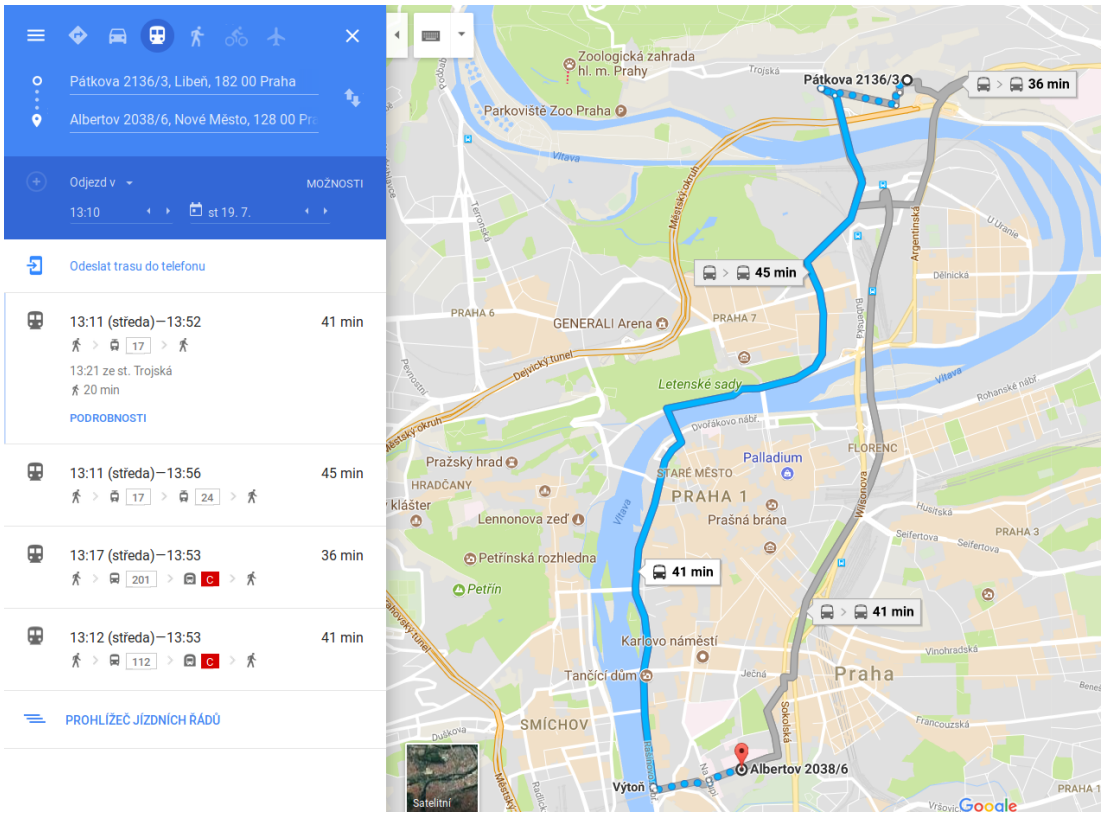
Zastávka	Příjezd	Odjezd	Linka
Pelc Tyrolka		13:26	112
Trojská	13:29	13:33	17
Výtoň	13:54	13:57	14
Albertov	13:59		



Obrázek 8.1: Spojení Kolej 17. listopadu – Albertov podle našeho vyhledávače



Obrázek 8.2: Spojení Kolej 17. listopadu – Albertov podle Mapy.cz



Obrázek 8.3: Spojení Kolej 17. listopadu – Albertov podle Google Maps

8.1.2 Kolej 17. listopadu – Čistírna odpadních vod

Spojení 0 nevyžaduje žádný přesun pomocí MHD, celá trasa je vedena pěšky. Očekávaný příchod je 14:08.

- Spojení 1:

Zastávka	Příjezd	Odjezd	Linka
Kuchyňka		13:25	201
Nádraží Holešovice	13:28	13:31	6
Strossmayerovo náměstí	13:36	13:39	26
Hradčanská	13:47	14:00	131
Goeteho	14:03		

  Očekávaný příchod je 14:08.

- Spojení 4:

Zastávka	Příjezd	Odjezd	Linka
Kuchyňka		13:25	201
Nádraží Holešovice	13:28	13:31	6
Strossmayerovo náměstí	13:36	13:39	26
Vítězné náměstí	13:50		
Dejvická		13:55	160
Nádraží Podbaba	13:59		

  Očekávaný příchod je 14:09.

- Spojení 6:

Zastávka	Příjezd	Odjezd	Linka
Kuchyňka		13:25	201
Nádraží Holešovice	13:28	13:31	6
Dlouhá třída	13:38	13:42	8
Nádraží Podbaba	14:02		

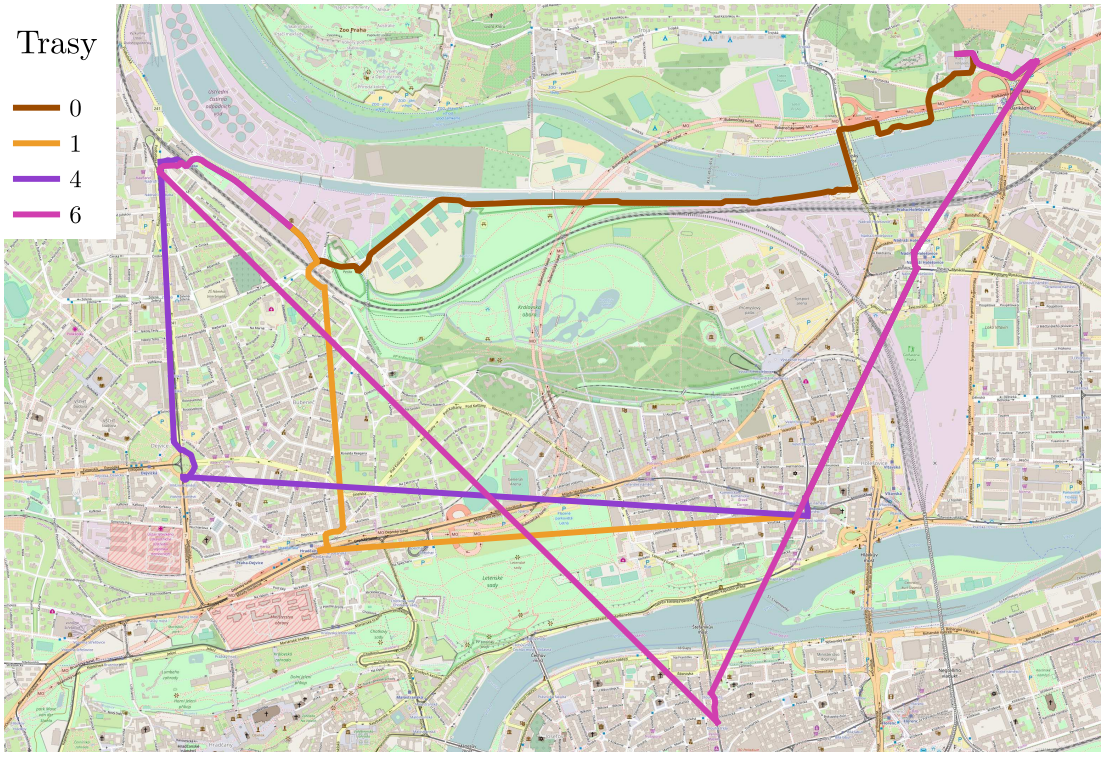
  Očekávaný příchod je 14:12.

Vyhledávač IDOS našel následující spojení:

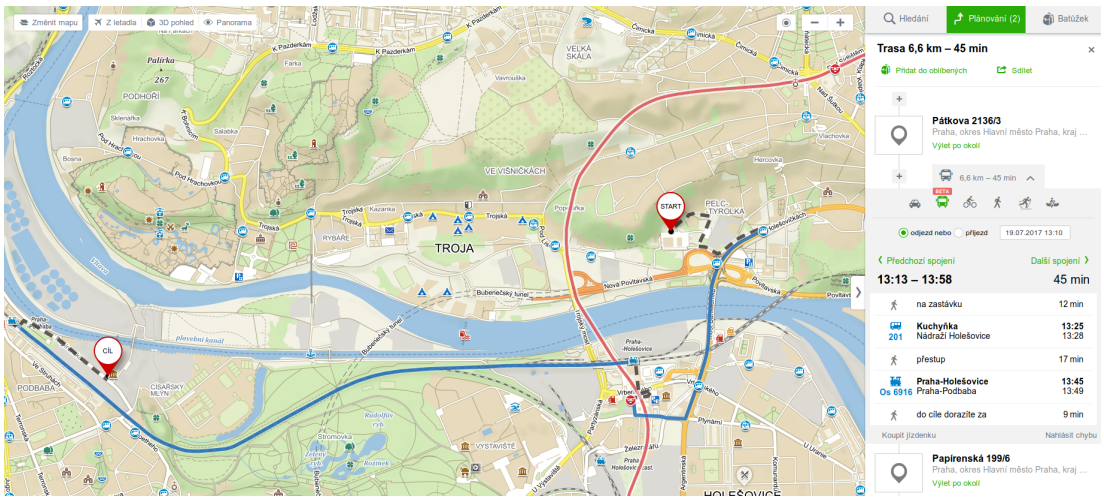
Zastávka	Příjezd	Odjezd	Linka
Kuchyňka		13:25	201
Nádraží Holešovice	13:28	13:45	Os 10440
Praha-Podbaba	13:49		

Zastávka	Příjezd	Odjezd	Linka
Pelc Tyrolka		13:26	112
Trojská	13:29	13:33	17
Nádraží Holešovice	13:35	13:45	Os 10440
Praha-Podbaba	13:49		

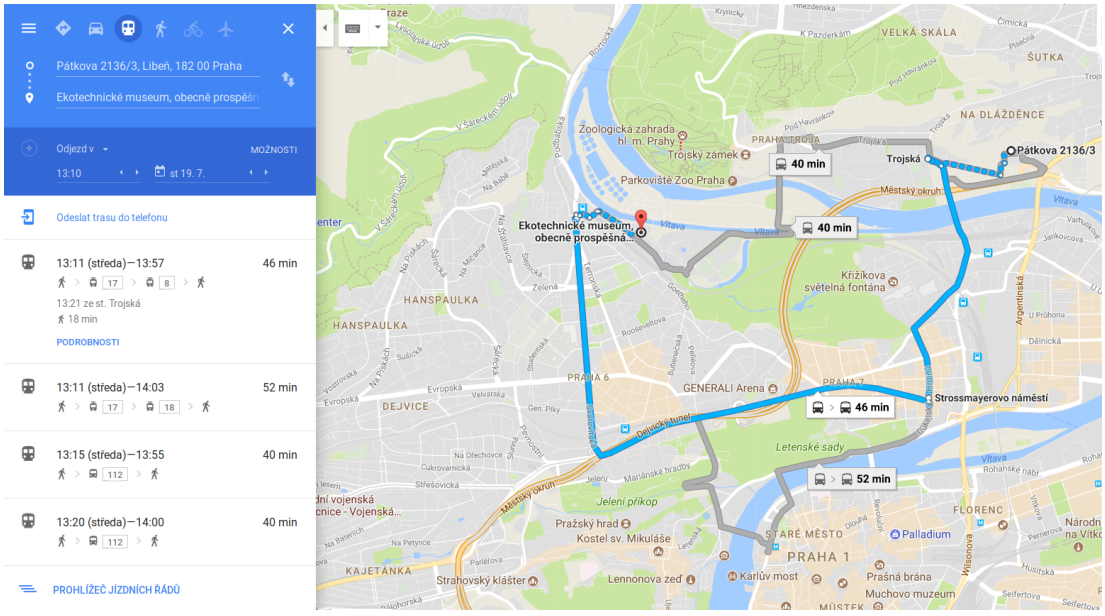
Na výsledcích tohoto spojení se významně projevila integrace železničních jízdních řádů do vyhledávačů. IDOS i Mapy.cz jízdní řády integrované mají, proto využily vlak mezi stanicemi Praha-Holešovice a Praha-Podbaba. Google Maps ani náš vyhledávač železniční jízdní řády integrované nemají, nalezly proto cesty oklikou a s delšími pěšími přesuny. Náš vyhledávač nalezl obě varianty cesty bez vlaků, a to přímo pěšky a oklikou přes Letnou, pro přímou cestu při tomto hledání nevyužil autobus 112 a proto je očekávaný čas horší než u Google Maps. Jak se také ukáže u dalších spojení, odhad rychlosti chůze byl při našem nastavení o něco konzervativnější než u Google Maps, vzálenější pěší přesuny tudíž vycházejí o něco delší.



Obrázek 8.4: Spojení Kolej 17. listopadu – Čistírna odpadních vod podle našeho vyhledávače



Obrázek 8.5: Spojení Kolej 17. listopadu – Čistírna odpadních vod podle Mapy.cz



Obrázek 8.6: Spojení Kolej 17. listopadu – Čistírna odpadních vod podle Google Maps

8.1.3 Kovanecká – Gymnázium Omská

- Spojení 0:

Zastávka	Příjezd	Odjezd	Linka
Divadlo Gong		13:21	16
Orionka	13:39	13:41	136
Bělocerkevská	13:47		

  Očekávaný příchod je 13:55.

- Spojení 1:

Zastávka	Příjezd	Odjezd	Linka
Divadlo Gong		13:21	16
Mezi hřbitovy	13:32	13:36	26
Nad Primaskou	13:42	13:43	7
Slavia	13:49	13:52	22
Kubánské náměstí	13:53		

  Očekávaný příchod je 13:55.

- Spojení 3:

Zastávka	Příjezd	Odjezd	Linka
Balabenka		13:19	25
Palmovka	13:21	13:23	24
Karlovo náměstí	13:45	13:46	22
Kubánské náměstí	14:05		

  Očekávaný příchod je 14:07.

- Spojení 5:

Zastávka	Příjezd	Odjezd	Linka
Balabenka		13:19	25
Palmovka	13:21	13:23	24
Kubánské náměstí	14:06		

  Očekávaný příchod je 14:08.

- Spojení 7:

Zastávka	Příjezd	Odjezd	Linka
Ocelářská		13:19	6
Kubánské náměstí	14:11		

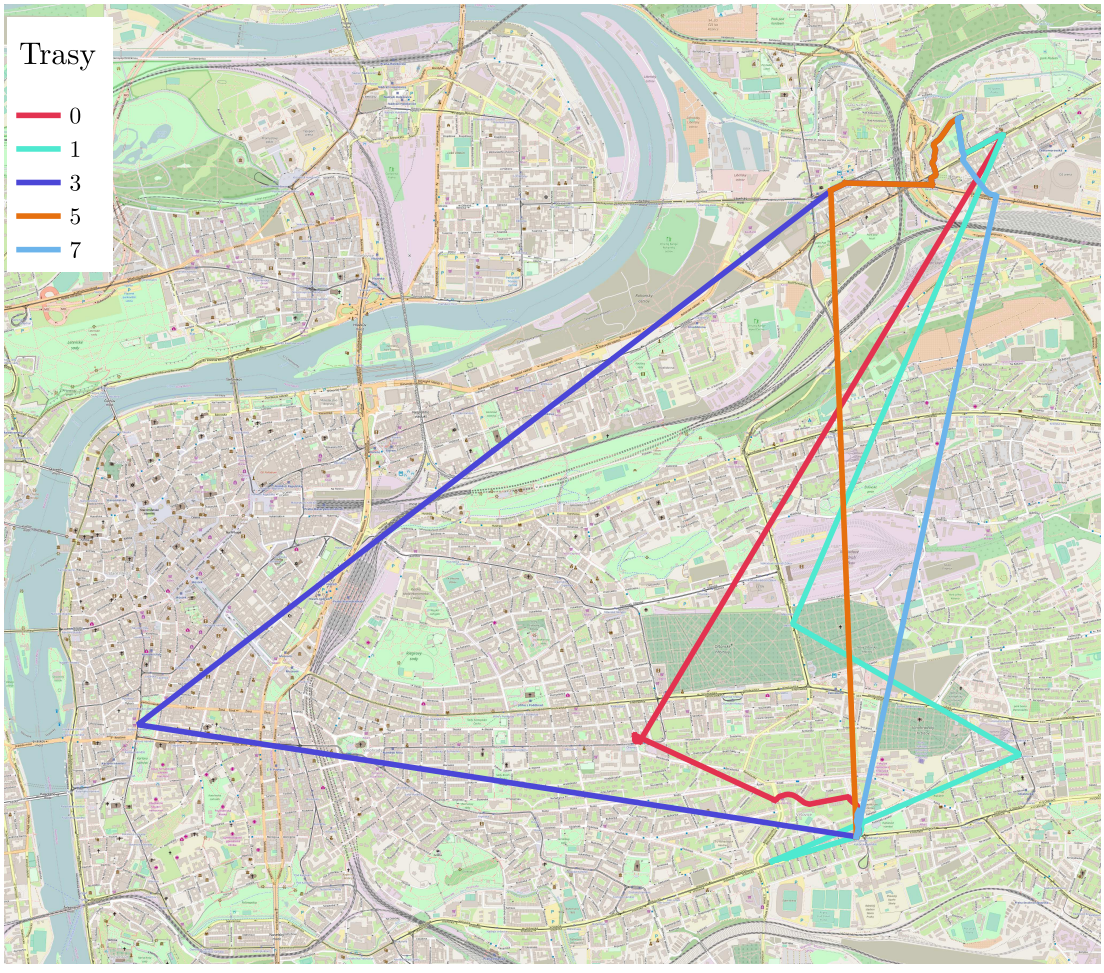
  Očekávaný příchod je 14:13.

Vyhledávač IDOS našel tato spojení:

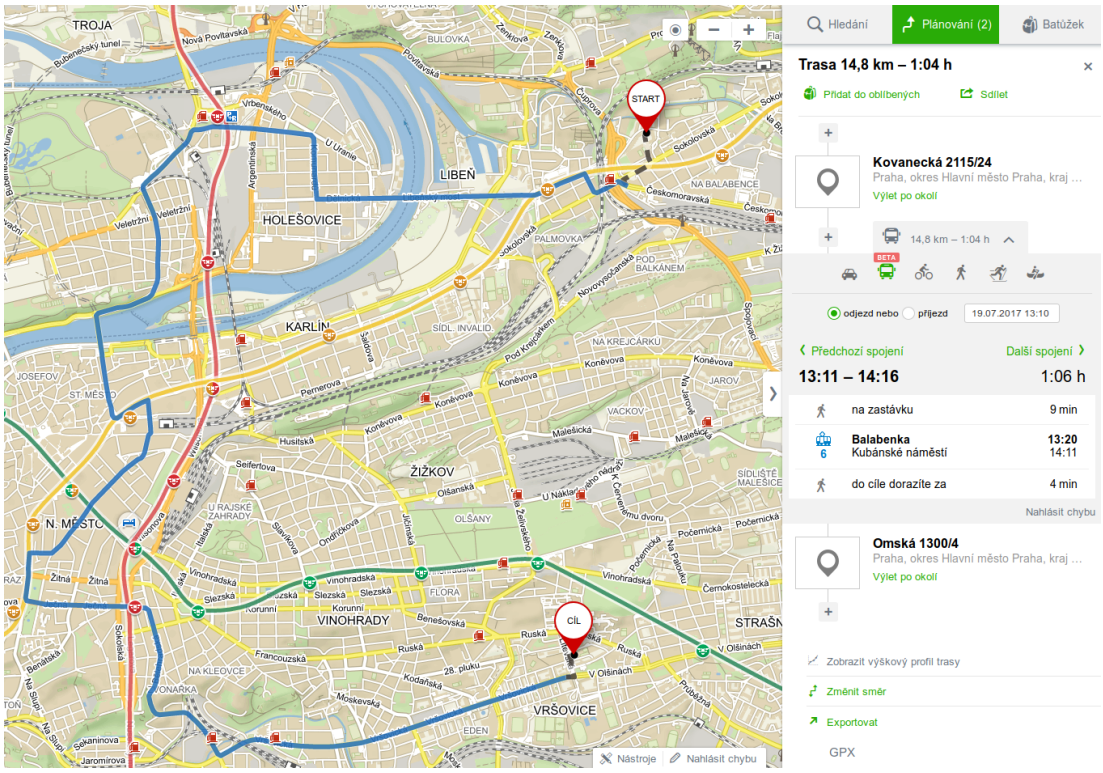
Zastávka	Příjezd	Odjezd	Linka
Balabenka		13:20	6
Kubánské náměstí	14:11		

Zastávka	Příjezd	Odjezd	Linka
Divadlo Gong		13:21	16
Želivského	13:35	13:41	150
Slavia	13:46	13:52	22
Kubánské náměstí	13:53		

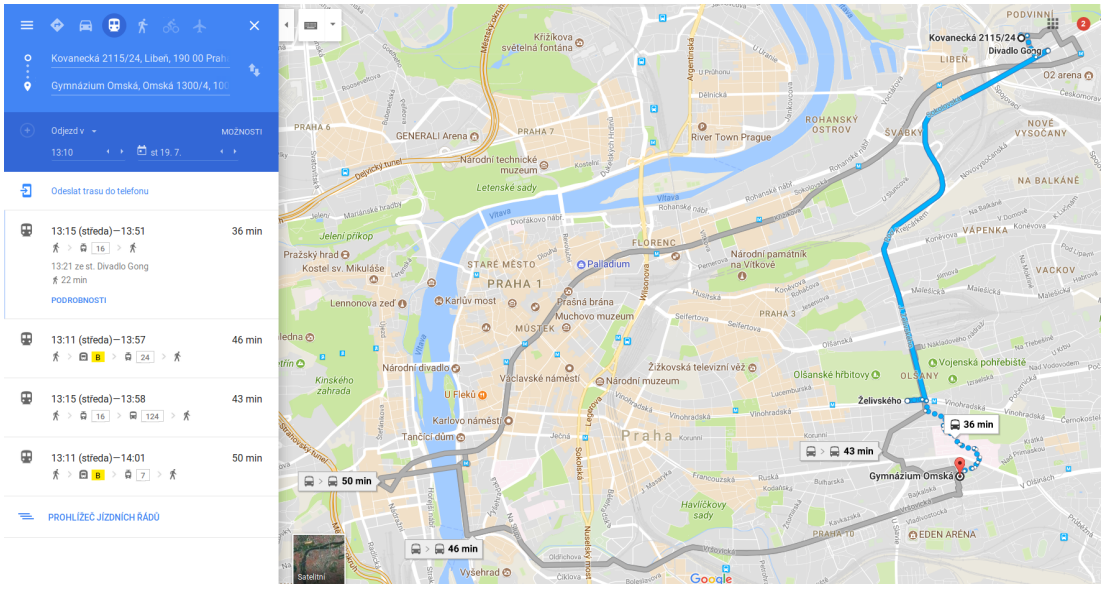
V tomto spojení se opět projevila výhoda vyhledávačů, které mají mapové podklady, protože v dané relaci je dle zkušeností výhodné nesnažit se dojet co nejblíže, ale jít pěšky už ze Želivského a projít skrz Vinohradskou nemocnici. Mapy.cz zde daly přednost jízdě bez přestupů, ale bohužel zvolily špatnou tramvaj, protože linka č. 6 objíždí celé město a cesta jí proto trvá neúměrně dlouho. IDOS hledal spojení na Kubánské náměstí, což se pro vhodnou volbu přestupů ukázalo jako poměrně dobrá volba. Google Maps našly cestu skrz Vinohradskou nemocnici, náš vyhledávač místo toho zvolil cestu přes Orionku která byla jen o něco delší. Alternativní spojení byly zajímavé jak u Google Maps, kde spojení vedlo dokonce až přes Anděl, tak i u našeho vyhledávače, který přejel zastávku Kubánské náměstí, aby se pak do ní vrátil z druhé strany. Zkoumali jsme, jestli to nemůže být zapříčiněno chybou v datech, ale připojení zastávky k cestní síti se jevilo v pořádku.



Obrázek 8.7: Spojení Kovanecká – Gymnázium Omská podle našeho vyhledávače



Obrázek 8.8: Spojení Kovanecká – Gymnázium Omská podle Mapy.cz



Obrázek 8.9: Spojení Kovanecká – Gymnázium Omská podle Google Maps

8.1.4 FEL ČVUT – Hlávkova kolej

- Spojení 0:

Zastávka	Příjezd	Odjezd	Linka
Dejvická		13:16	A
Malostranská	13:19	13:23	2
Moráň	13:34		

  Očekávaný příchod je 13:38.

- Spojení 1:

Zastávka	Příjezd	Odjezd	Linka
Lotyšská		13:18	18
Moráň	13:38		

  Očekávaný příchod je 13:42.

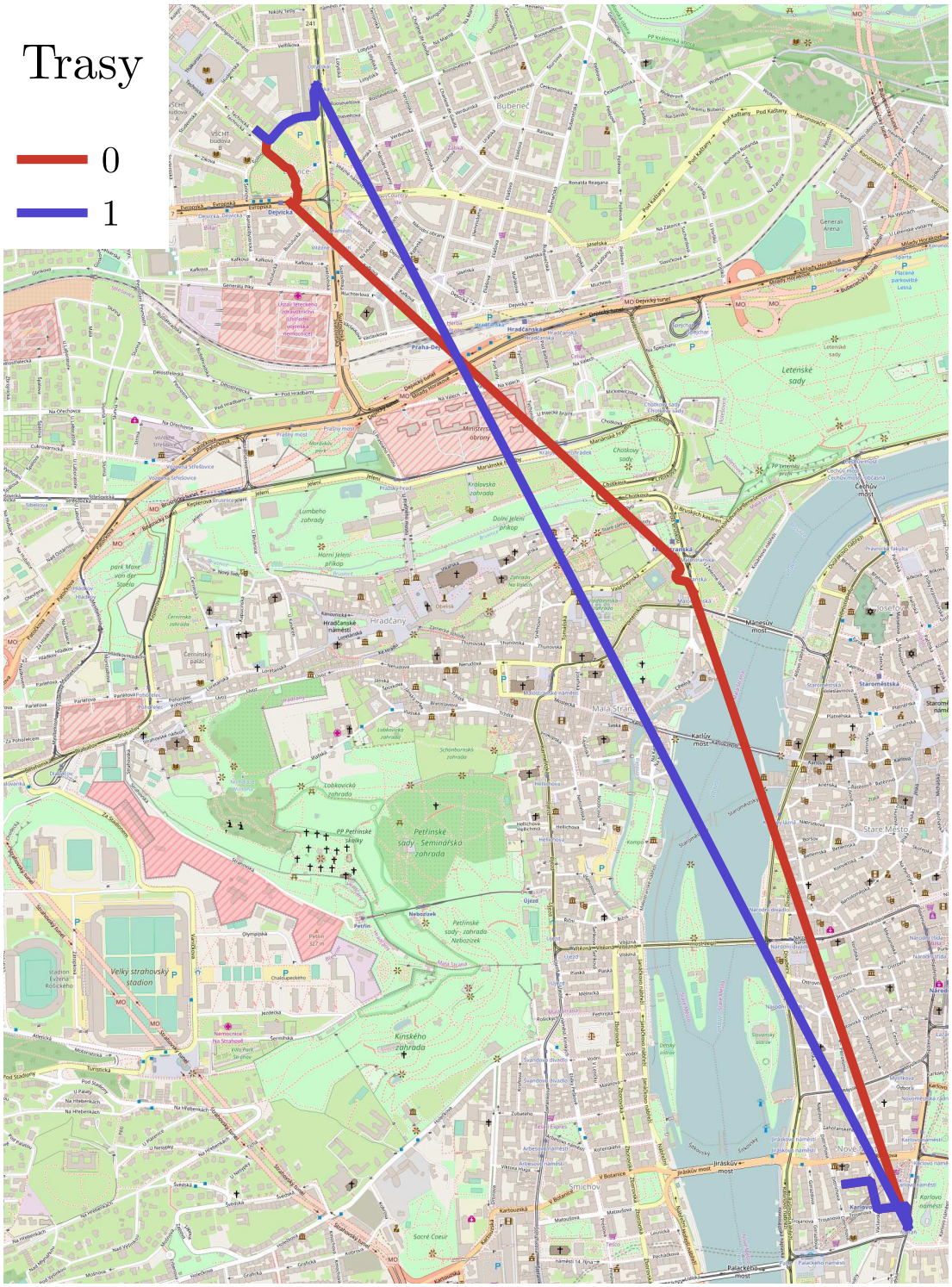
Vyhledávač IDOS našel tato spojení:

Zastávka	Příjezd	Odjezd	Linka
Dejvická		13:16	A
Můstek	13:22	13:29	B
Karlovo náměstí	13:32		

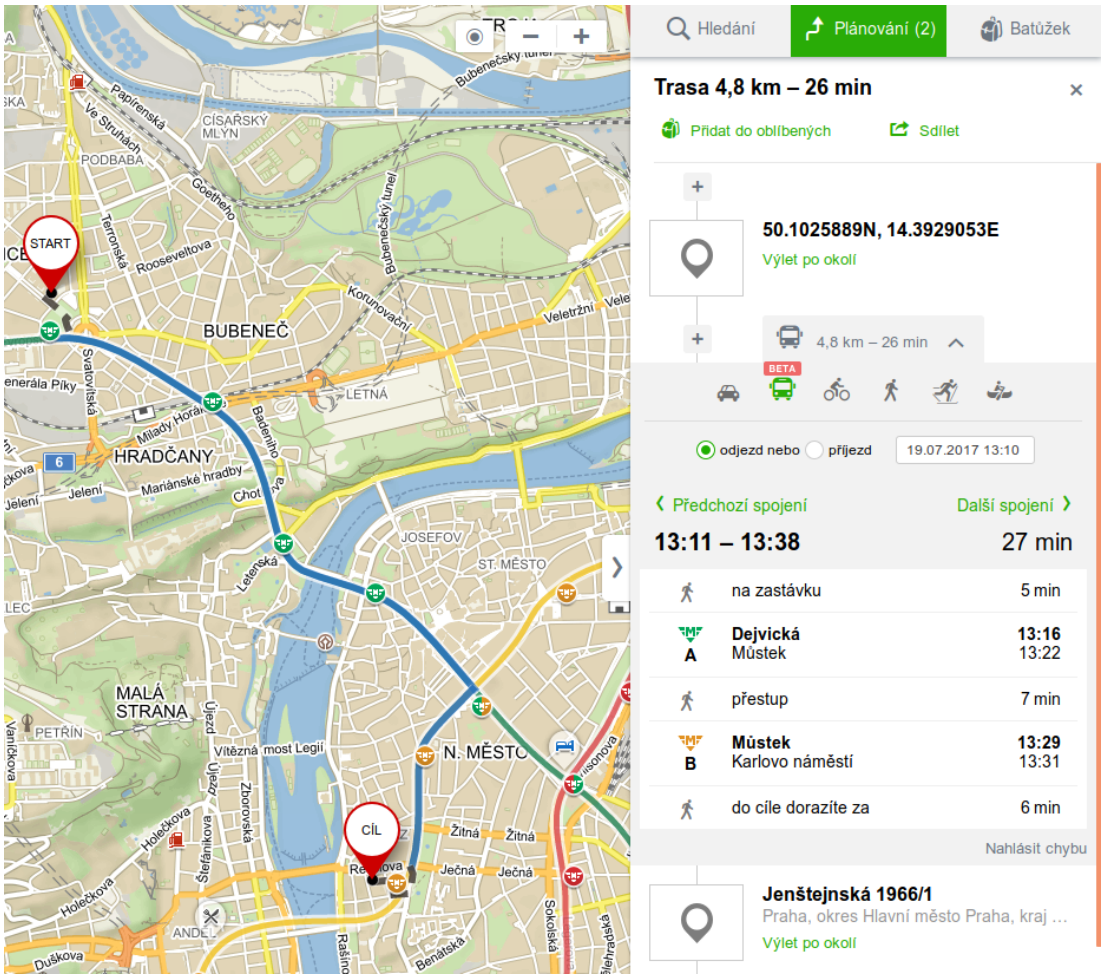
Zastávka	Příjezd	Odjezd	Linka
Vítězné náměstí		13:19	18
Karlovo náměstí	13:37		

Zastávka	Příjezd	Odjezd	Linka
Dejvická		13:21	A
Staroměstská	13:25	13:30	18
Karlovo náměstí	13:37		

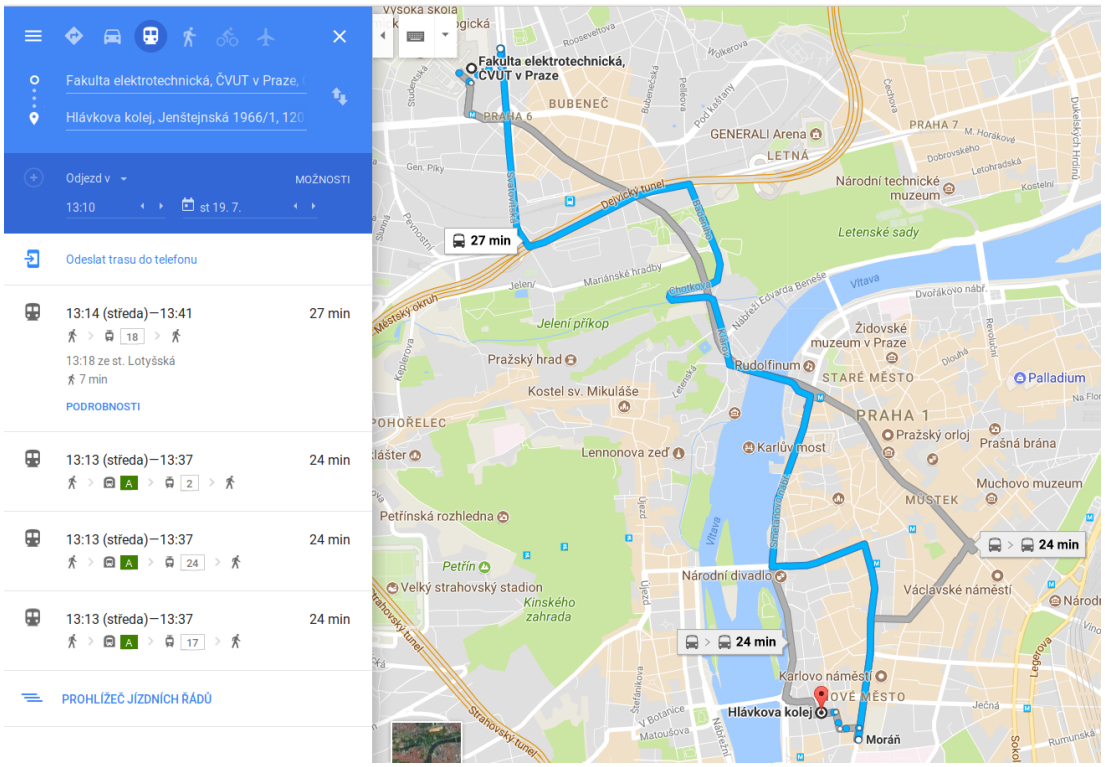
Toto spojení reprezentovalo situaci, kdy samotná trasa je jednoduchá, ale je množství různých alternativ v koncových a výchozích stanicích. Náš vyhledávač našel spojení identická s Google Maps. Mapy.cz a IDOS nalezly mírně odlišná spojení využívající i linku metra B, ale časově odpovídající našemu nalezenému spojení.



Obrázek 8.10: Spojení FEL ČVUT – Hlávkova kolej podle našeho vyhledávače



Obrázek 8.11: Spojení FEL ČVUT – Hlávkova kolej podle Mapy.cz



Obrázek 8.12: Spojení FEL ČVUT – Hlávkova kolej podle Google Maps

8.2 Porovnání různých nastavení

Pro porovnání různých nastavení našeho vyhledávače jsme zvolili trasu Kolej 17. listopadu – Albertov, protože tato trasa poskytuje několik různých alternativ spojení, tudíž jsme předpokládali, že se tyto alternativy projeví v nalezených trasách. Všechna spojení byla hledána od stejného času, pracovní den v brzkém odpoledni, tudíž by měl být eliminován vliv návazností, které fungují jen v určité minuty, protože podmínky jsou pro všechna hledání stejné.

Porovnávali jsme tato nastavení:

- *Standardní*
  Základní nastavení vyhledávače, měl by se chovat obdobně jako jiné webové vyhledávače

- *Bez autobusu 201*
  Linka 201 je ve směru z kolejí na Nádraží Holešovice tak nespolehlivá, že je lepší s ní vůbec nepočítat. Toto nastavení ji pomocí penalty `inf` vyřazuje z hledání.

- *Penalizace autobusů*
  Test na penalizaci typu dopravního prostředku, měl by se snažit vyhýbat spojení autobusem a hledat jiné alternativy.

- *Jen jeden spoj*
  Test na hledání, kterým spojem jet, pokud nechceme nikde přestupovat.

- *Pouze pěšky*
  Ačkoli je vyhledávač stavěn na kombinované hledání pěších přesunů a cest MHD, stále by měl umět najít i pouze pěší cestu.

- *Na kole*
  Tento test předpokládá jízdu na skládacím kole nebo koloběžce, tedy něčem, co lze vzít s sebou do MHD. Testuje nestandardní vyhledávací parametry.

- *Penalizace pěších přesunů*
  Opak předchozího testu, snažíme se minimalizovat pěší přesuny a co nejvíce se pohybovat pomocí MHD.

8.2.1 Standardní

Vyhledávání dle základního nastavení, které se nachází v repozitáři v souboru `config/speeds.yaml`. U ostatních testů bude uveden pouze rozdíl oproti tomuto nastavení.

Trasa nalezená vyhledávačem se standardním nastavením je již popsána výše na obr. 8.1, nebudeme ji zde znovu rozebírat.

8.2.2 Bez autobusu 201

Oproti standardnímu nastavení je pouze penalizace linky 201 nastavena na `inf`, tato linka by tedy k hledání vůbec neměla být použita.

Vyhledávač nalezl mnoho různých tras. Trasy s číslem menším než 12 mají příjezd do 14:05, zbylé pak do 14:15. Zajímavé jsou pro nás trasy 0 a 1, protože zbylé rychlé trasy jsou jen variacemi těchto tras. Trasa 0 odpovídá trase nalezené standardním vylhedáváním, ale na Nádraží Holešovice využívá pěší přesun. Trasa 1 poněkud netradičně využívá přestup na linku 14 na Florenci s pěším přesunem na Těšnov. Ve výsledcích jsou i linky, které využívají možnosti dojet na Vyšehrad a pak sejít do nuselského údolí z druhé strany.

8.2.3 Penalizace autobusů

Oproti standardnímu nastavení mají autobusy fixní penaltu nastavenou na 100 a za každou sekundu v autobuse dostanou penaltu dalších 10.

Vyhledávač opět nalezl několik alternativ, které jsou velmi podobné předchozímu vyhledávání. Je zde navíc varianta využívající dlouhý přesun tramvají č. 17 z Trojské na Palackého náměstí a tam přestoupit do tramvaje na Albertov, která je i podle zkušeností rychlá a spolehlivá. Všechny cesty na Trojskou a Nádraží Holešovice v této variantě jsou pěšky z důvodu penalizace autobusů.

8.2.4 Jen jeden spoj

Oproti standardnímu nastavení je počet nástupů do vozidla omezen na 1.

Nalezené spojení je vedeno tramvají č. 17 z Trojské na Výtoň, odkud je to na Albertov nejblíže. Cesta na Nádraží Holešovice a pak pěšky z I. P. Pavlova byla pravděpodobně delší a proto bylo toto spojení majorizováno.

8.2.5 Pouze pěšky

Oproti standardnímu nastavení je počet nástupů do vozidla omezen na 0.

Vyhledávač úspěšně nalezl pěší spojení až na Albertov, které by bylo zvládnutelné, i když by to oproti cestě MHD trvalo výrazně déle. Při tomto hledání byl vyhledávač výrazně pomalejší než pro kombinovaná spojení, což bylo pravděpodobně způsobeno jednak mnohem větším stavovým prostorem, který bylo potřeba projít, protože cesta trvá výrazně déle, jednak nutností zahazovat všechna nalezená spojení pomocí MHD. Tomuto případu bychom mohli výrazně pomoci tím, že bychom přepnuli na pouze pěší vyhledávač, ale neočekáváme, že by byly často hledány pouze pěší trasy, proto jsme se rozhodli kód dále nerozšiřovat.

Na začátku trasy nalezl vylhedávač dvě alternativní trasy, jednu vedoucí horními Holešovicemi přes kopec a druhou vedoucí po okraji dolních Holešovic podél Argentinské. Ačkoli obě trasy jsou jistě zvládnutelné pěšky, ani jednu bychom si pravděpodobně pro průchod holešovicemi nevybrali. První varianta obsahuje zbytečné stoupání do kopce a pak klesání zpět k Vltavě, což by se dalo omezit nastavením délkového prodloužení za kopce, problém druhé cesty, dlouhou procházku podél čtyřproudé silnice bychom v současném stavu odstranit nedokázali. Trasa je vedena po chodnících, tudíž penalty za silnice se neuplatňují, bylo by potřeba při přípravě dat zjišťovat pro každou cestu objekty v jejím bezprostředním okolí a podle toho jí přidávat atributy, na které by se pak dal brát ohled. Toto by však bylo výpočetně náročné a je to mimo rozsah této práce.

8.2.6 Na kole

Oproti standardnímu nastavení byly přenastaveny rychlosti pohybu na rychlosti mezi 10 a 25 km/h (mimo schodů, kde byly nastaveny 2km/h). Rovněž byly penalizovány cesty do kopce (10 m délky za metr převýšení) a zvýhodňovány cesty z kopce (-10 m délky za metr převýšení). Navíc byla nastavena penalta za nástup do vozidla na 100.

Toto vyhledávání byl spíše experiment, jak se bude vyhledávač chovat, pokud nastavíme rychlosti výrazně výš než je obvyklá rychlost chůze. Pro vyhledávání cyklistických tras není ve standardní konfiguraci vyhledávač vhodný, protože klasifikuje objekty s ohledem na pěší chůzi. Pro hledání cyklistických tras by bylo potřeba výrazně pozměnit klasifikaci objektů, zařadit například typ povrchu silnic a pravděpodobně zahodit zkratky, protože na kole nebývají potřeba a nemusí být tak snadno realizovatelné. V neposlední řadě existuje mnoho kvalitních cyklistických vyhledávačů, které dávají výrazně kvalitnější výsledky s podobnou či lepší mírou přizpůsobení, například BRouter[4].

Vyhledané trasy odpovídají očekáváním s ohledem na klasifikaci objektů. Všechny vyhledané trasy využívají metro z Nádraží Holešovice, liší se výstupní stanicí, kterou je buď Hlavní Nádraží, I. P. Pavlova nebo Vyšehrad. Všechny trasy by pravděpodobně byly průjezdné, ale překvapivě se na všech trasách nachází schody, na kterých byla nastavena rychlost na 2 km/h a tudíž se je vyplatí objet. Pro pohodlnější cesty by bylo vhodné přidat za schody i penaltu. Ověřenou nejrychlejší cestu z Vyšehradu, a to klesání ulicí Čiklovou a pak prokličkování ulicí Na Slupi vyhledávač nenašel, protože v současné klasifikaci by bylo těžké popsat vhodně rychlostní profily jednotlivých druhů komunikací a hlavně povrchů. Dlažba na trasách vedených mimo Vyšehrad činí tyto trasy výrazně pomalejší a velmi nepohodlné.

8.2.7 Penalizace pěších přesunů

Pěší přesun po jakémkoli typu cesty je penalizován 100 za každou sekundu strávenou pěším přesunem.

Při tomto hledání je dobře vidět princip majorizovaných tras. Je nalezena nejrychlejší trasa, ale protože obsahuje dlouhé pěší přesuny, má vysokou penaltu. Proto se projeví i cesty časově výrazně delší, které ale využívají více přesunů pomocí MHD a tedy mají nižší penaltu a nejsou majorizovány. Celkem vyhledávač nalezl čtyři základní cesty a pak další, které se však od těchto základních liší jen drobnými detaily. Nejkratší nalezená cesta má penaltu přibližně 86 000, trasa 30 má penaltu nejnižší a to přibližně 12 300.

8.3 Profilování

Jak jsme ověřili v předchozí části, náš vyhledávač dává výsledky srovnatelné s ostatními dostupnými vyhledávači spojení a má široké možnosti konfigurace. Rychlost vyhledávání spojení je však nízká, středně dlouhá spojení přes střed města hledá v jednotkách sekund, při specifických konfiguracích (například pouze pěší vyhledávání) jsou ale časy výrazně delší, dosahují i desítek sekund. Ačkoliv

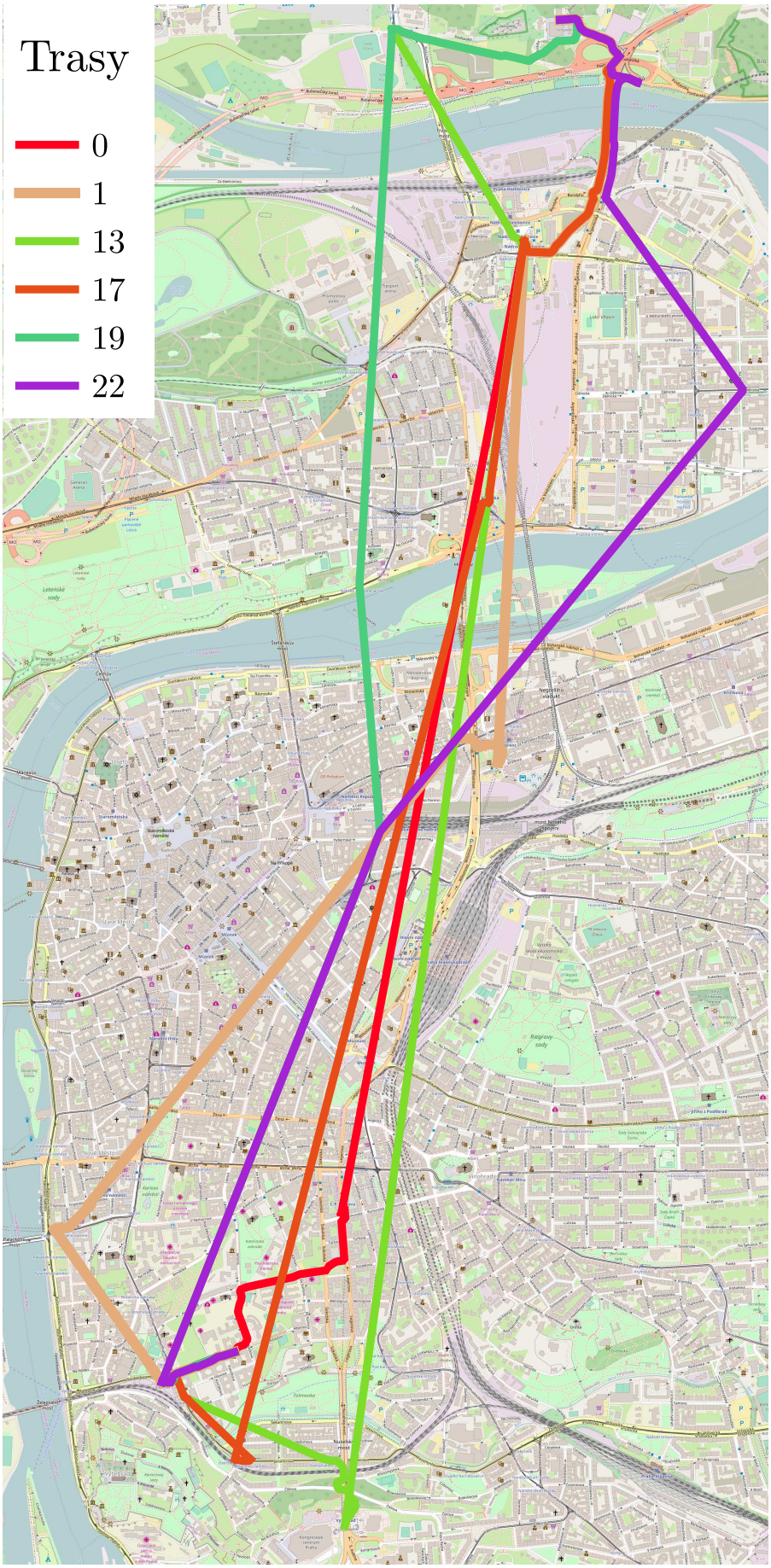
[4] `http://brouter.de`

rychlost nebyla našim hlavním cílem, pro praktické nasazení je nezbytné, aby vyhledávač pracoval co nejrychleji.

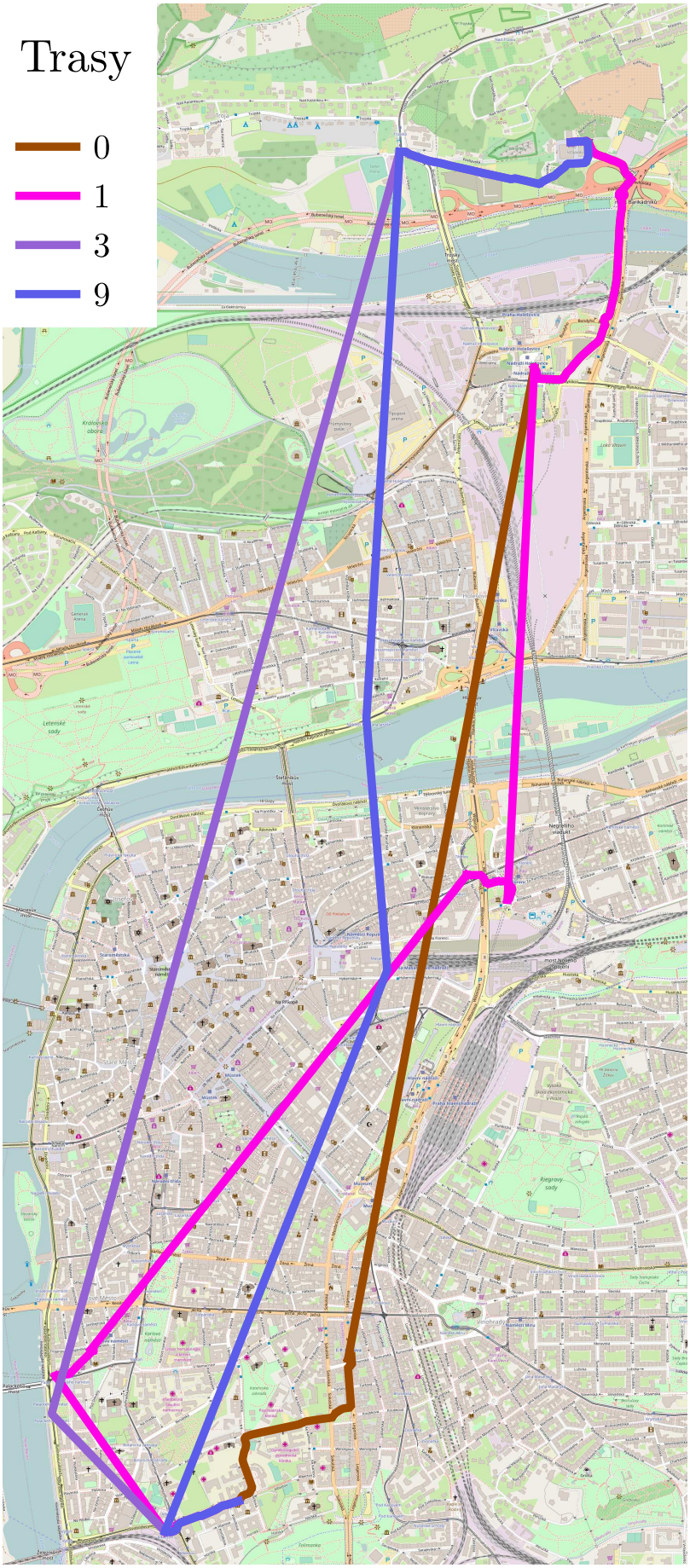
Abychom zjistili, kde se při hledání tráví největší čas, profilovali jsme vyhledávač pomocí aplikace `perf`.[5] Abychom eliminovali vliv načítání mapy a jízdního řádu, což se při běžném vylhedávání děje jen jednou při spuštění webové aplikace, vytvořili jsme testovací scénář pro konzolovou aplikaci. Ta si nejprve načetla mapová data a data jízdních řádů a potom postupně hledala různá spojení mezi předem zvolenými body. Nasbíraná data jsme pak analyzovali a na jejich základě provedli některé optimalizace, například vyhazování majorizovaných položek z haldy a seznamu vrcholů.

Z naměřených dat vyplývá, že místo, kde by se mělo optimalizovat nejdříve, je cyklus porovnávající, zda právě přidávaná položka do haldy není majorizována či naopak majorizuje nějakou jinou položku, která už v haldě je. Ve standardním nastavení se v tomto cyklu stráví okolo 25 % času, při hledání pouze pěších tras je to až 50 %. Dalším místem v pořadí je načítání pěší hrany při zpracovávání vrcholu a zjišťování, který její konec máme uvažovat. Zde si myslíme, že dochází ke zpomalení převážně z důvodu přístupu na náhodné místo v paměti, protože hran je mnoho a nejsou nijak setříděné. Překvapivě velký čas je zabrán počítáním penalty bodu. I když se jedná o funkci obsahující pouze 2 podmínky, tráví se zde 6 % celkového času, což nám přišlo zvláštní, ale nemáme pro to jasný důvod. Posledním významným časem je hledání minima v haldě, což je ale logaritmická operace a proto nás 7 % stráveného času příliš nepřekvapí.

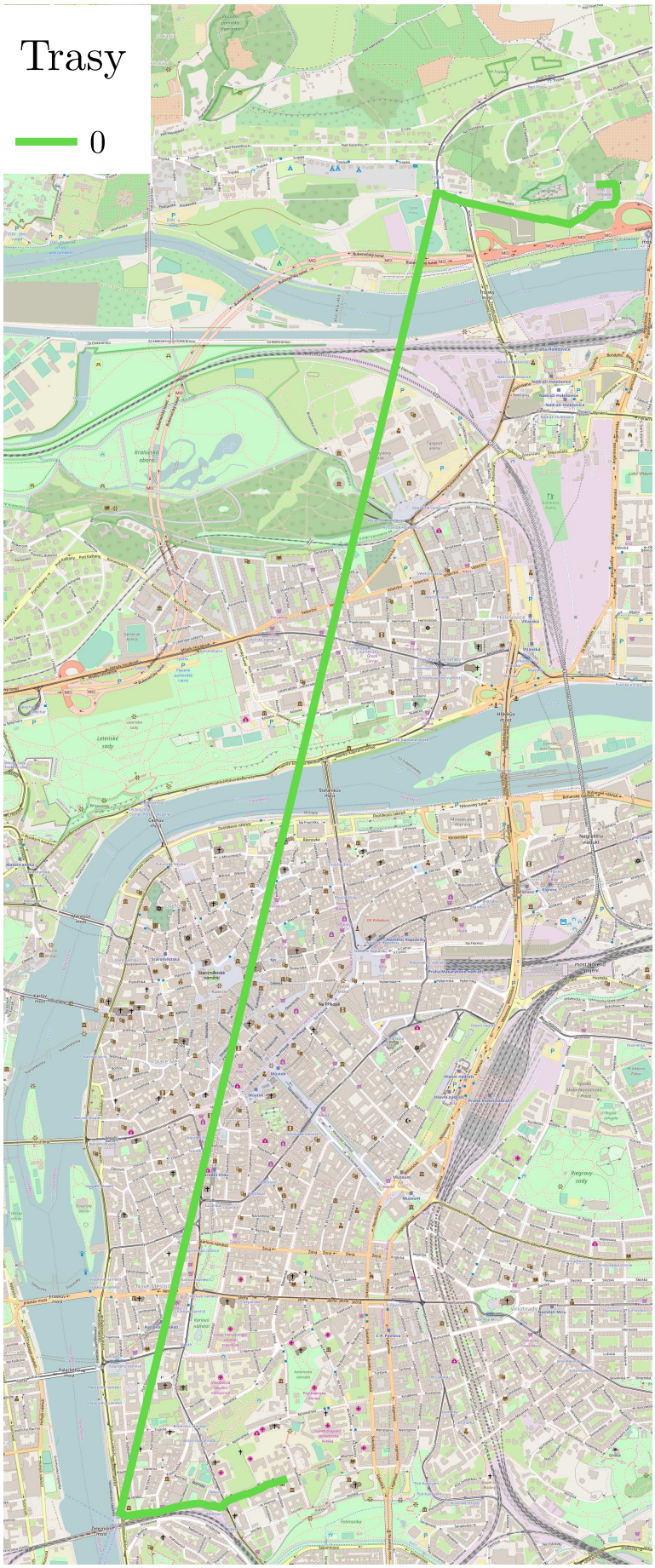
[5] `https://perf.wiki.kernel.org/index.php/Main_Page`



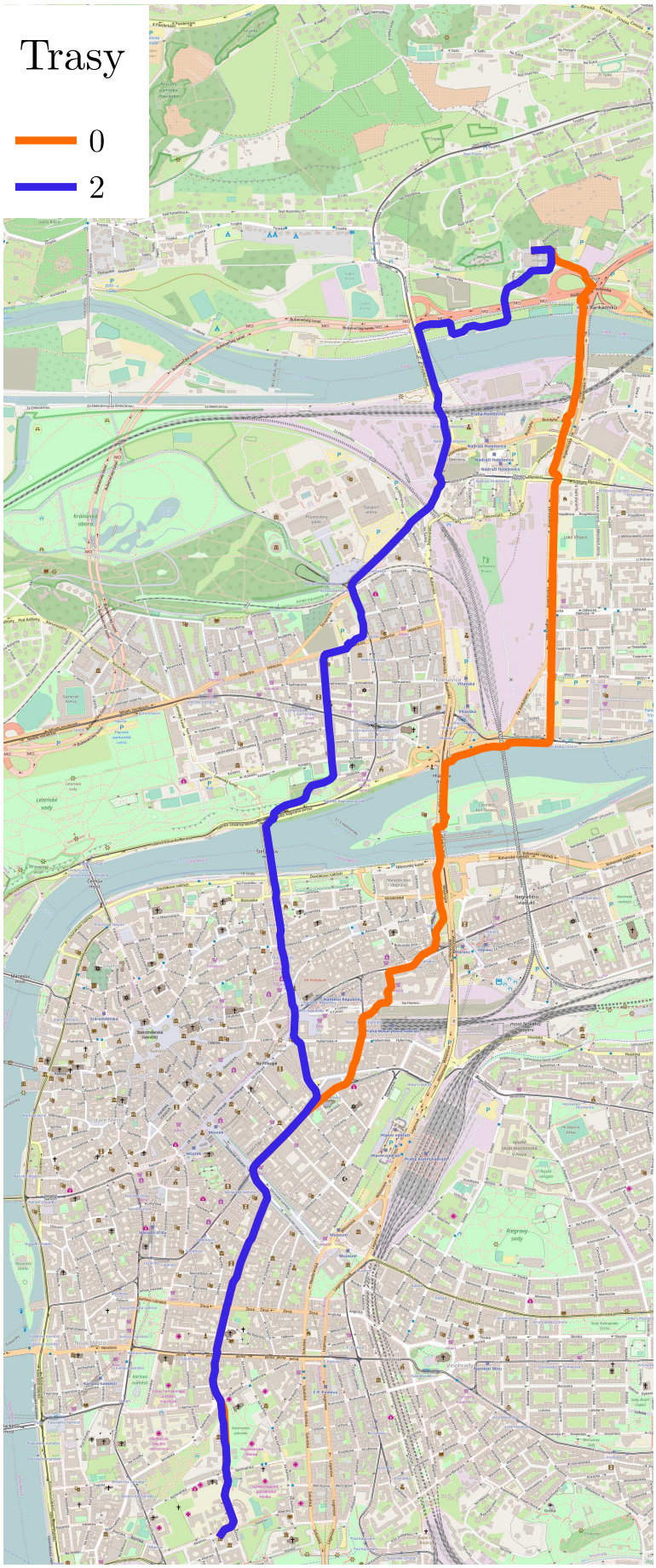
Obrázek 8.13: Spojení nevyužívající autobus 201



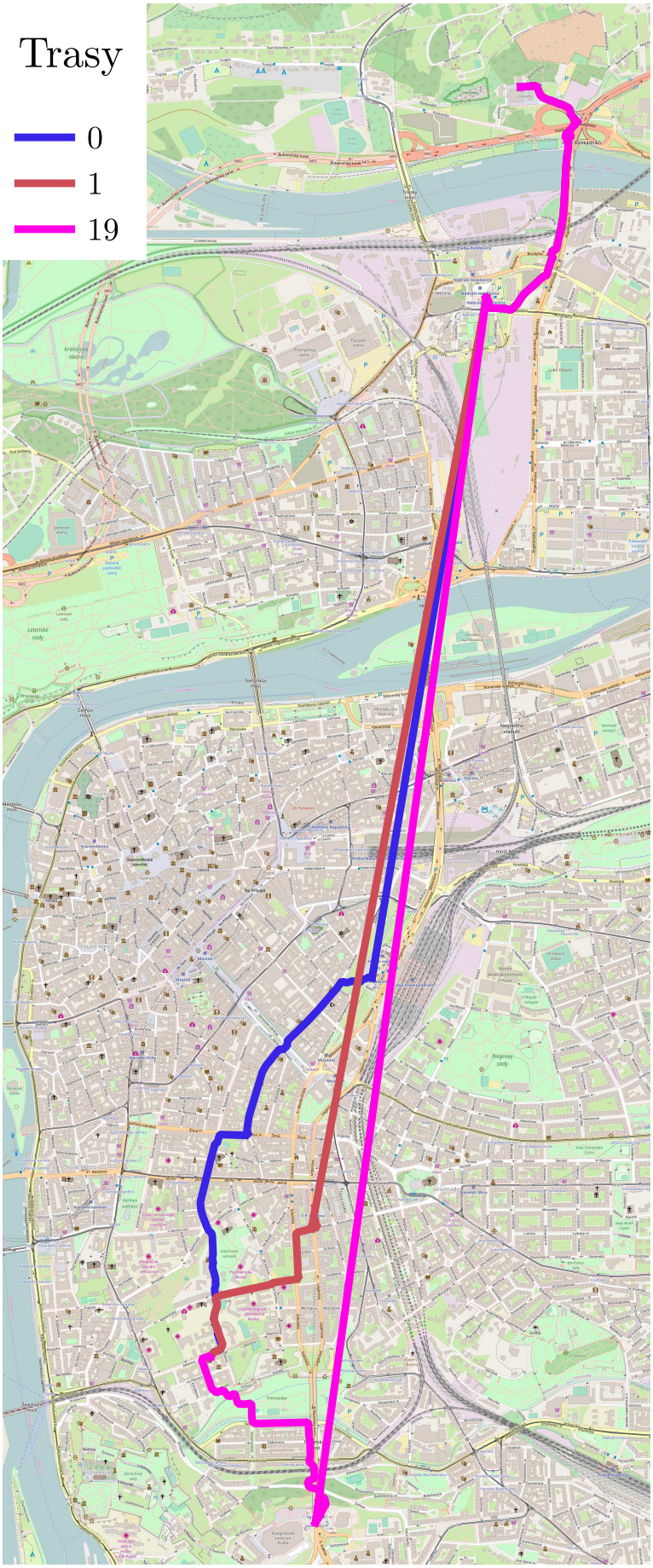
Obrázek 8.14: Spojení penalizující autobusy



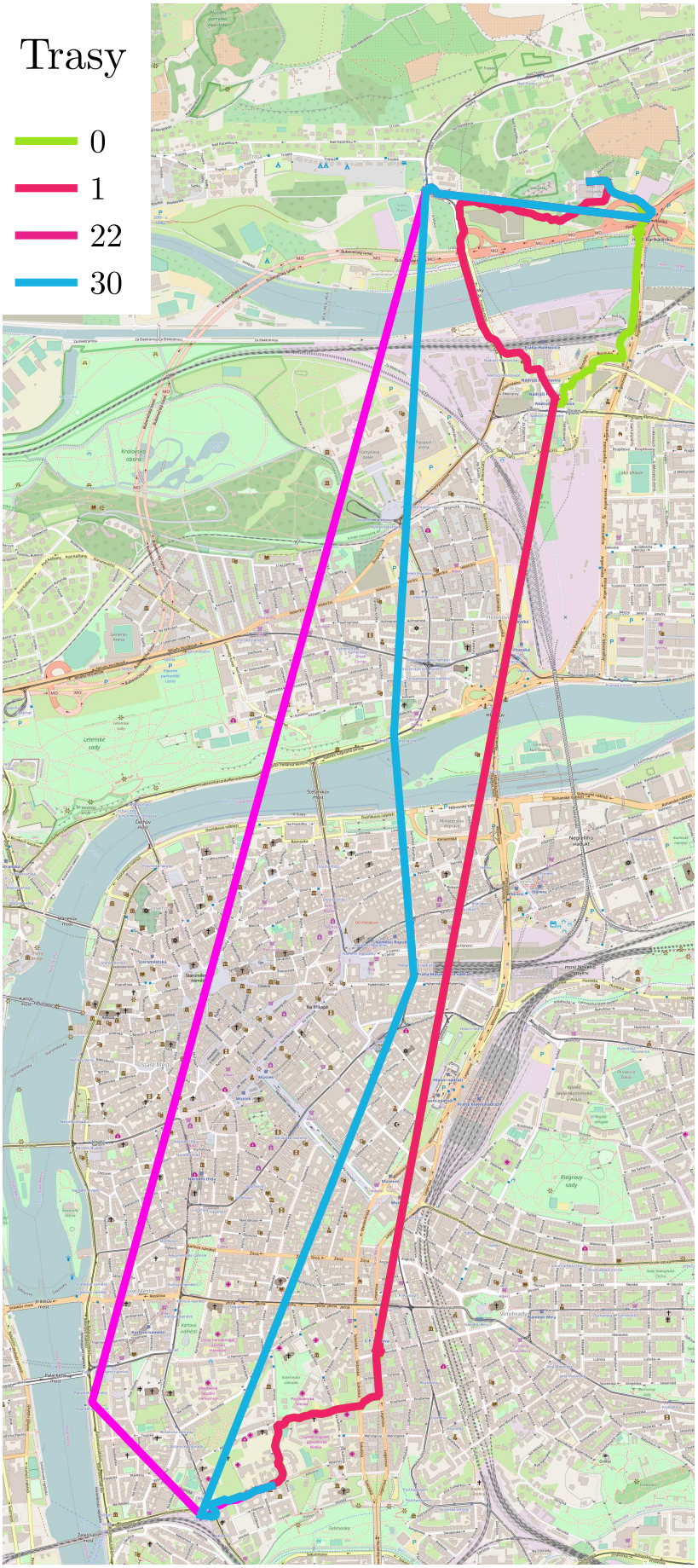
Obrázek 8.15: Spojení využívající nejvýše 1 spoj MHD



Obrázek 8.16: Spojení pouze pěšky



Obrázek 8.17: Spojení na kole



Obrázek 8.18: Spojení penalizující pohyb pěšky

9. Závěr

9.1 Zhodnocení

Úlohou práce bylo zpracovat veřejně dostupná mapová data a data o jízdních řádech a vytvořit z nich formát vhodný pro vyhledávání spojení kombinujících pěší přesuny a přesuny hromadnou dopravou. Součástí práce měl být také vyhledávač, který bude umožňovat nad připravenými daty vyhledávat spojení a bude možné ho parametrizovat.

Cíl práce se nám podařilo splnit, pro práci s mapovými daty jsme využili dřívější bakalářskou práci, kterou jsme dále rozšířili a datový formát připravili na propojení s daty z jízdních řádů. Pro ukládání dat z jízdních řádů jsme navrhli formát vycházející z potřeb algoritmu RAPTOR pro hledání v sítích hromadné dopravy. Připravili jsme skripty pro automatický převod dat jízdních řádů do našeho formátu a implementovali algoritmus RAPTOR jako sdílenou knihovnu, kterou je možné použít v jiných projektech nezávisle na zbytku naší práce.

Provázali jsme data z jízdních řádů s mapovými daty a vypořádali jsme se s různou kvalitou zmapování zastávek veřejné dopravy v mapě. Nad výslednými daty jsme pak implementovali vyhledávač spojení, který je možné používat jako sdílenou knihovnu nebo konzolovou či webovou aplikaci. Navrhli jsme systém hodnocení nalezených tras pomocí penalt a umožnili jsme uživatelům snadno nastavovat různé penalty pomocí konfiguračního souboru.

9.2 Výsledky

Spojení nalezená naším vyhledávačem jsou plně srovnatelná se spojeními nalezenými současnými vyhledávači spojení a na rozdíl od nich můžeme zároveň využívat možnosti plánovat kvalitní pěší přesuny a nastavovat si parametry vyhledávače dle svých požadavků. Problémem zůstává nízká rychlost vyhledávání spojení, víme však, na která místa se zaměřit v dalších optimalizacích. Návrh aplikace a převedení přípravy dat do PostgreSQL se osvědčilo a umožnilo snadnou kontrolu výstupních dat pomocí geografického informačního systému. Rozdělení přípravy dat do modulů a možnost živého náhledu na zpracovávaná data výrazně zrychlilo a usnadnilo vyhledávání a odstraňování chyb.

9.3 Náměty pro další rozvoj

Aplikace je v současné době plně použitelná pro hledání spojení lokálním uživatelem. Pro použití jako serverová aplikace by kromě zrychlení vyhledávání bylo nutné i připravit pro uživatele možnost nahrát si vlastní vyhledávací profily a na straně knihovny se naučit sdílet neměnná mapová data a jízdní řády mezi více vlákny pro úsporu paměti.

Zajímavou možností by také bylo upravit vyhledávač do podoby webové aplikace, optimálně i s offline daty, a to buď jako samostatnou aplikaci, nebo jako plugin do některé ze současných mapových aplikací, například OsmAnd.

9.3.1 Příprava dat

Příprava mapových dat je již odladěný proces, který je rychlý a nenáročný na operační paměť, případné změny by tudíž mířily k dalšímu vylepšování připravovaných dat. Jednou z možností, jak již bylo naznačeno v kapitole 8, by bylo zkoumat okolí cest a mít u hran v grafu uložen typ okolí – jestli se nacházíme uvnitř zástavby, v parku či u velké silnice. Mohly by tak být penalizovány trasy podle „krásy“.

Přípravu dat jízdních řádů by bylo vhodné přepsat buď jako databázovou aplikaci, když svá data stejně bere z databáze, nebo jako samostatnou aplikaci v kompilovaném jazyce, která by byla paměťově úspornější. Do výstupního formátu jízdních řádů by bylo vhodné přidat různé atributy, které se obvykle u spojů vyskytují: nízkopodlažnost vozidla, název konečné a další informace, které by uživatelům usnadnily cestování.

9.3.2 Zrychlení vyhledávání

Pro jiné než lokální použití na počítači je nutné zrychlit vyhledávač spojení. Zrychlovat vyhledávání je v současné době možné jak úpravou kódu na efektivnější, tak změnami v samotném vyhledávacím algoritmu. Jako první se nabízí třídit položky u jednotlivých vrcholů, aby bylo možné majorizace hledat binárním vyhledáváním místo průchodu celým polem. Z principu majorizace totiž u položek setříděných podle času vzestupně klesá penalta, protože v opačném případě by položka s větším časem a penaltou byla majorizována. Tento přístup by přinesl složitost do přidávání nových položek, ale zrychlil by rozhodování, zda položku budeme vůbec přidávat.

Další možnost zrychlení je přímo ve vyhledávacím algoritmu, kdy můžeme zkoušet různé jiné typy hald či se pomocí vhodné heuristiky snažit hledat více směrem k cíli, jak je uvedeno v [18].

9.3.3 Vylepšení vyhledávání

Mimo zrychlení jsme během práce narazili na různé možnosti, jak zlepšit kvalitu vyhledávání. Koncept majorizovaných spojení, jak jsme ho navrhli, dává poměrně dobré výsledky, ale může se snadno stát, že dojde k majorizaci některých spojení, které bychom chtěli zachovat. Typická situace je několik možných začátků trasy, které všechny vedou na stejný spoj. U nástupního vrcholu do tohoto spoje se pak sejde několik částečných tras a z principu penalt bude dále pokračovat jen jeden z nich a to ten s nejmenší penaltou a ostatní se zahodí, protože mají všechny stejný čas odjezdu. Možným řešením je nemajorizovat jinou částečnou trasu, která se liší použitými spoji, bylo by však potřeba zjistit, zda pak vyhledaných spojení nebude příliš mnoho.

Obdobným problémem je preference častějších spojení, kdy bývá výhodné jít na zastávku, odkud jezdí spoje často a případný ujetý nebo zpožděný spoj nám nezkazí celý zbytek cesty. K tomuto problému je možné přistupovat dvěma způsoby, jednak lze pro každou vyhledanou trasu zkoumat alternativy v případě ujetí jednotlivých spojů po cestě, jednak lze frekvenci spojení nějakým způsobem zanést do systému penalt a preferovat tak frekventovanější trasy před ostatními.

Systém penalt lze rozšiřovat mnoha dalšími způsoby, například zavést směrové penalty. Mezi takové by mohla patřit penalta za chůzi do schodů, ale směrové penalty dávají smysl u spojů MHD, kdy poblíž konečné mohou být spoje směrem na konečnou často zpožděné, zatímco spoje z konečné bývají spolehlivé a jedou včas. Dalším vylepšením penaltového systému by byla konfigurace pomocí vhodného skriptovacího jazyka na místo konfiguračního souboru, kdy by bylo možné si výpočet penalt přizpůsobit přesně podle svých potřeb a nebylo by pak potřeba znovu překládat celý program. Tato úprava by ale mohla mít negativní dopad na rychlost vyhledávání.

Kombinované vyhledávání pěších tras s přesuny hromadnou dopravou má ještě jedno specifikum a tím je čas nutného odchodu z výchozího místa. Pokud jdeme několik stovek metrů na autobus, který jezdí jednou za půl hodiny, pak není třeba vycházet hned po nalezení spoje, ale stačí nám vyjít například až za 10 minut a do cíle dorazíme ve stejný čas. Tato situace je řešitelná poměrně snadno posunutím pěší části trasy před prvním přejezdem MHD tak, aby pěší trasa na spoj MHD akorát navazovala. Problém je ale obecnější, pokud autobus s půlhodinovým intervalem používáme až na konci cesty, pak nestačí jen posunout první pěší část, ale mohli bychom jet i pozdější tramvají a tudíž by bylo potřeba i změnit některé spoje na trase či celou část trasy. Problém se dá (neefektivně) řešit postupným hledáním v pozdějších časech, pokud na cestě narazíme na spoj s velkým intervalem, ale pro efektivnější řešení by bylo potřeba nastudovat potřebnou literaturu, provést měření a pravděpodobně by se jednalo o velký zásah do vyhledávacího algoritmu.

Literatura

[1] Pyrga, Evangelia a kol. *Efficient models for timetable information in public transportation*. J. Exp. Algorithmics. 12:2.4:1–2.4:39. 2008-06.

[2] Brodal, Gerth Stølting, Riko, Jacob. *Time-dependent networks as models to achieve fast exact time-table queries*. Electronic Notes in Theoretical Computer Science. 92:3 – 15. 2004.

[3] Geisberger, Robert. *Contraction of timetable networks with realistic transfers*. CoRR. abs/0908.1528. 2009.

[4] Delling, Daniel, Pajor, Thomas, Werneck, Renato F. *Round-based public transit routing.* Transportation Science. 49(3):591–604. 2015.

[5] přispěvatelé OSM. *OpenStreetMap* [online] 2017 [cit. 2017-07-15]. Dostupné z `http://www.openstreetmap.org/about`

[6] Pokorný, Tomáš. *Hledání pěších cest v mapě*. Praha, 2014. Bakalářská práce. Matematicko-fyzikální fakulta, Katerda aplikované matematiky.

[7] National Aeronautics and Space Administration. *The Shuttle Radar Topography Mission* [online]. 2009-06-17 [cit. 2014-05-04]. Dostupné z: `http://www2.jpl.nasa.gov/srtm/`.

[8] Google. *General Transit Feed Specification* [online]. 2017 [cit. 2017-07-15]. Dostupné z: `https://developers.google.com/transit/gtfs/reference/`.

[9] Google. *Protocol Buffers* [online]. 2012-04-02 [cit. 2014-05-05]. Dostupné z: `https://developers.google.com/protocol-buffers/`.

[10] ECMA. *The JSON data interchange format* [online]. Technical Report Standard ECMA-404 1st Edition. 2013-10 [cit. 2017-07-15]. Dostupné z: `http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf`

[11] Butler, H. Daly, M. a kol. *The GeoJSON format* [online]. RFC 7946. 2016-08 [cit. 2017-07-15]. Dostupné z: `https://tools.ietf.org/html/rfc7946`

[12] Bray, Tim. Paoli, Jean a kol. *Extensible Markup Language (XML) 1.0 (Fifth Edition)* [online]. W3C. 2008 [cit. 2017-07-15]. Dostupné z: `http://www.w3.org/TR/REC-xml/`

[13] *GPX: the GPS Exchange Format* [online]. 2017 [cit. 2017-07-15]. Dostupné z: `http://www.topografix.com/gpx.asp`

[14] National Imagery and Mapping Agency. *World geodetic system 1984* [online]. 3. vydání, 1. dodatek. 2000-01-03 [cit. 2017-07-15]. Dostupné z: `http://earth-info.nga.mil/GandG/publications/tr8350.2/wgs84fin.pdf`.

[15] Hager JW, Behensky JF, Drew BW. *The universal grids: Universal Transverse Mercator (UTM) and Universal Polar Stereographic (UPS)* [online]. Tech. Rep. TM 8358.2, Defense Mapping Agency. 1989, [cit. 2017-07-15].
Dostupné z: `http://earth-info.nga.mil/GandG/publications/tm8358.2/TM8358_2.pdf`.

[16] Mareš, Martin a kol. *LibUCW* [online]. 2016 [cit. 2017-07-15].
Dostupné z: `https://www.ucw.cz/libucw/`

[17] *Flask Documentation* [online]. 2017. [cit. 2017-07-15].
Dostupné z: `http://flask.pocoo.org/docs`

[18] Mareš, Martin. *Krajinou grafových algoritmů*. [online] ITI Series. 2007 [cit. 2017-07-15].
Dostupné z: `http://mj.ucw.cz/vyuka/ga/`

Seznam použitých zkratek

GTFS	General Transit Feed Specification
SRTM	Shuttle Radar Topography Mission
MHD	Městská hromadná doprava
GPX	GPS exchange format
XML	Extensible Markup Language
PBF	Protocol Buffer
OSM	OpenStreetMap

Příloha A

V této příloze popíšeme adresářovou strukturu přiloženého archivu.

- `compiled/` – zdrojové kódy programů v C pro přípravu dat a vyhledávání tras
- `config/` – soubory s nastavením pro klasifikaci a vyhledávání
- `data/` – adresář obsahující generované datové soubory
- `ext-lib/` – adresář pro externí knihovny, obsahuje implementaci RAPTOR
- `osm/` – soubory a programy pro stažení a předpřípravu OSM a SRTM dat
- `postgis/` – skripty pro zpracovávání dat v databázi
- `text/mgr` – zdrojové kódy tohoto textu
- `webapp/` – Webová verze vyhledávače tras