



MATEMATICKO-FYZIKÁLNÍ FAKULTA

Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Filip Čižmář

Analýza real-time dat vozidel městské hromadné dopravy

Katedra softwarového inženýrství

Vedoucí bakalářské práce: doc. Mgr. Martin Nečaský, Ph.D.

Studijní program: Informatika

Studijní obor: SW a datové inženýrství

Praha 2020

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

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 dne
Podpis autora

Především děkuji svému vedoucímu práce Martinu Nečaskému, který mi pomohl najít zajímavé zaměření mé práce, umožnil přístup k otevřeným datům a pomohl při vypracování.

Stejně tak děkuji i pánu Vlasatému a Benediku Kotmelovi, kteří mi poskytli odbornou pomoc při získávání dat z datové platformy Golemio a inspiraci pro obsah mé práce.

Dále děkuji panu profesoru Jakubu Klímkovi za zapůjčení počítače za účelem získání testovacích dat.

Název práce: Analýza real-time dat vozidel městské hromadné dopravy

Autor: Filip Čižmář

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: doc. Mgr. Martin Nečaský, Ph.D., Department of Software Engineering

Abstrakt: Tato práce se zaměřuje na analýzu dostupných otevřených real-time dat z vozidel hromadné dopravy v Praze a okolí. Jejím cílem je poskytnout základní statistické informace a na základě historických dat zlepšit odhad zpoždění spoje na trase mezi dvěma referenčními body. Jako vedlejší produkt vytvoří aplikaci pro webové rozhraní, kde zobrazí aktuální polohy spojů do mapového podkladu a rozšiřující infomace o nich. Aplikace bude aktivně interagovat s uživatelem.

Klíčová slova: zpoždění MHD otevřená data veřejná doprava

Title: Analysis of real-time data of public transport vehicles

Author: Filip Čižmář

Department: Department of Software Engineering

Supervisor: doc. Mgr. Martin Nečaský, Ph.D., Katedra softwarového inženýrství

Abstract: Abstract. Tato práce se zaměřuje na analýzu dostupných otevřených real-time dat z vozidel hromadné dopravy v Praze a okolí. Jejím cílem je poskytnout základní statistické informace a na základě historických dat zlepšit odhad zpoždění spoje na trase mezi dvěma referenčními body. Jako vedlejší produkt vytvoří aplikaci pro webové rozhraní, kde zobrazí aktuální polohy spojů do mapového podkladu a rozšiřující infomace o nich. Aplikace bude aktivně interagovat s uživatelem.

Keywords: delay open data public transport

Obsah

Úvod	3
1 Analýza	5
1.1 Úvod	5
1.2 Popis problému odhadu zpoždění	5
1.2.1 Příklad nelineárního profilu trasy	6
1.2.2 Současná řešení	8
1.3 Analýza zdroje dat	8
1.3.1 Přístup k datům	8
1.3.2 Analýza statických dat	14
1.4 Analýza vizualizačních nástrojů	16
1.4.1 Mapové podklady	16
1.4.2 Současná řešení	16
2 Návrh řešení	19
2.1 Úvod	19
2.1.1 Funkční a kvalitativní požadavky	19
2.2 Zpracování vstupních dat	21
2.2.1 Databáze	21
2.2.2 Plnění databáze	24
2.3 Algoritmus odhadu zpoždění	25
2.3.1 Základní předpoklady	25
2.3.2 Analýza dat a návrh modelování	26
2.4 Vizualizace dat	33
2.4.1 Funkční požadavky	33
2.4.2 Kvalitativní požadavky	33
2.4.3 Návrh grafiky a UI	34
3 Implementace	35
3.1 Úvod	35
3.2 Zpracování dat	36
3.2.1 Konstrukce objektů vozidel	36
3.2.2 Odhad zpoždění	37
3.2.3 Kompletace dat a jejich uložení	37
3.3 Konstrukce modelů	39
3.3.1 Čtení dat	39
3.3.2 Příprava dat	41
3.3.3 Práce s modely	41
3.4 Vizualizace dat	42
3.4.1 Klientská část	42
3.4.2 Serverová část	45

4 Tesstování a evaluace	48
4.1 Testování softwarového řešení	48
4.1.1 Unit testy	48
4.1.2 Integrační testy	48
4.1.3 Výkonostní testy	48
4.2 Evaluace výsledků	50
4.2.1 Sestrojení modelů	50
4.2.2 Odhady zpoždění	50
5 Porovnání se stávajícím řešením	51
6 Navrhy na zlepšení	52
7 Statistiky	53
Závěr	54
Seznam použité literatury	55
Seznam obrázků	56
Seznam tabulek	57
Seznam použitých zkratek	58
A Přílohy	60
A.1 První příloha	60

Úvod

Městská hromadná doprava v Praze a Středočeském kraji je jeden z hlavním pilířů přepravy osob na tomto území. Jejím rozsahem a důležitostí se přímo dotýká každého z nás a její fungování do značné míry ovlivňuje naše konání v krátkém i dlouhém časovém horizontu.

Každého cestujícího v přepravě jistě někdy trápilo zpoždění svého spoje. To člověka přivádí k myšlenkám jestli by nebylo možné určit s jakou pravidelností, pokud s nějakou, takové zpoždění vznikají. A jestli by nemohl být informován za včasu o vzniklé anomálii a vzniklému zpoždění.

Cílem této práce je zpřesnit odhad zpoždění vozidel VHD, zejména autobusů, na trase mezi dvěma sousedícími zastávkami. Dále pak tyto výsledky vizualizovat v mapových podkladech.

Ve vymezené oblasti operuje spousta soukromých i městských dopravců. Ti kteří spadají do naší zájmové oblasti zastřešuje organizace ROPID, která objednává jednotlivé spoje. Pro naši práci je důležité, že organizace ROPID zadala jednotlivým dopravcům vysílat aktuální polohy jejich vozů. Tato data o polohách jsou přes zprostředkovatele zveřejňována na pražské datové platformě zvané Golemio¹, jež je ve správě společnosti Operátor ICT, a. s., která je vlastněná hlavním městem Praha. Takových spojů, o kterých máme všechna požadovaná data je v pracovní den vypraveno necelých deset tisíc.²

V době návrhu práce, kvůli právním komplikacím a složitost informačního systému³, nebyly k dispozici real-time data z majoritního dopravce na území Prahy Dopravního podniku Prahy. Avšak protože je práce zaměřená na odhad zpoždění spoje mezi dvěma sousedícími zastávkami na trase, má tedy větší význam odhadovat zpoždění mezi zastávkami, mezi kterýma je větší vzdálenost. A to jsou převážně spoje jednoucí mimo Prahu. Proto tato data z DPP nenabývají takové důležitosti, jako data od dopravců operujících mimo Prahu. Vzhledem k tomu, že zbylí dopravci využívají převážně autobusy k přepravě cestujících, bude práce vypracována pouze s ohledem na autobusovou dopravu.

V práci se tedy pokusíme využít dostupná otevřená real-time data k získání infomarcí o zpoždění spojů na trase a využít je k lepším odhadům zpoždění. Řešení ovšem není pouze založeno na real-time datach, ale využívá také statická data o jízdních řádech nebo zastávkách hromadné dopravy, jejichž zdrojem je přímo ROPID⁴, a také mapové podklady. Ty jsou potřeba zejména pro vizualizaci zastávek a jízdních řádů nebo vykreslní trasy spoje přímo do mapy. Avšak i tyto statická data jsou dostupná přímo z datové platformy pomocí stejného rozhraní jako data real-timová.

¹www.golemio.cz

²ze dne 20. 2. 2020 podle testovacích dat

³www.irozhlas.cz/zpravy-domov/data-o-poloze-vozidel-dpp-mhd-tramvaje-autobusy-praha-hrib-soud-informace_1904290600_kno

⁴pid.cz/o-systemu/opendata/

Protože disponujeme daty o aktuálních polohách vozidel MHD, která navíc rozšíříme o lepší odhady zpoždění. Nabízí se jejich využití tak, že budou vynezena do mapy a tím vznikne vizuálně přívětivé uživatelské prostředí pro prohlížení aktuálního stavu sítě vozidel. V práci tedy navrhujeme a implementujeme uživatelskou aplikaci, která vozidla zobrazí a bude komunikovat s uživatelem tak, že na žádost zobrazí více informací o daném spoji nebo vybrané zastávce.

1. Analýza

V této kapitole je detailně popsán problém a způsoby jeho navrženého a současného řešení. Dále zdroje dat se kterými budeme v této práci pracovat.

1.1 Úvod

Spoje které zajišťují hromadnou dopravu jezdí podle jízdních řádů, které definují jejich trasu. Trasa se udává sekvencí projíždících zastávek, časy příjezdu a odjezdu do, resp. z těchto zastávek a vzdáleností zastávek od výchozího bodu spoje. Tyto zastávky jsou zpravidla jediné referenční body u kterých je možno zjistit skutečné zpoždění, nebo předjetí (dále uvažováno jako zpoždění se zápornou hodnotou). Dále jsou součástí jízdních řádů také velice detailní nákresy tras každého spoje, formou lomené čáry definovanou posloupností souřadnic, kde každý bod je doplněn o jeho vzdálenost od výchozího bodu spoje.

Délka trasy mezi dvěma referenčními body nezřídká dosahuje i několika desítek kilometrů¹. Na těchto úsecích mohou vznikat mimořádné události, které se dají predikovat jen s těží. Nicméně ve většině případů je průběh jízdy ovlivněn pouze obvyklým provozem v dané denní době.

Detailní rozbor počtu průjezdů mezi zastávkami v daných vzdálenostech je vidět na grafu 1.1. Kde průjezdem se myslí každý jednotlivý průjezd vozidla mezi danou dvojicí zastávek v daný den. Data jsou platná pro spoje jedoucí v 20. 2. 2020.

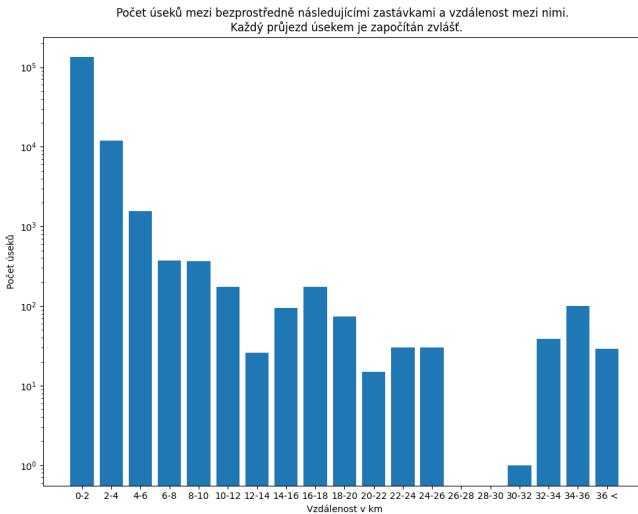
1.2 Popis problému odhadu zpoždění

Řešený problém se týká případu, kdy vozidlo projízdí mezi dvěma referenčními body a tato trasa má části, ve kterých vozidlo jede různou rychlostí. Např. vozidlo při vyjíždění z města jede mnohem pomaleji než při jízdě mezi městy. Takových úseků, na kterých se rychlosť jízdy liší může být na trase více a nedají se všechny jednoduše detektovat.

Tato Práce tedy modeluje profily jízd mezi referenčními body. A na základě toho zpřesnit odhad zpoždění. Tento odhad by měl být mnohem přesnější než současné odhady, které odpovídají tomu, že vozidlo jede konstantní rychlostí po celou dobu jízdy. Nebo je takové možné brát jako aktuální zpoždění spoje poslední změřené zpoždění při průjezdu nějakých referenčním bodem (zastávkou, nebo např. pro tramvaje se používají návěstidla).

Přidaná hodnota je tedy v tom, že Práce navrhne takové modely, které nebudou penezalizovat zvyšováním zpožděním za pomalou jízdu v úsecích, které se pomaleji projíždějí vždy. A také naopak zvýhodňovat snížením zpožděním za

¹Podle dat pro spoje jedoucí v 20. 2. 2020 je medián vzdálesnotí mezi zastávkama, mezi kterýma projede alespoň jeden spoj denně 943 m. Průjezdů mezi zastávkami ve vzdálenosti více než 10 kilometrů je 784, přičemž průjezdů mezi zastávkami ve vzdálenosti alespoň 2 km je přibližně 15000



Obrázek 1.1: Graf počtu úseků mezi následujícími zastávkami a vzdálenotí mezi nimi.

rychlou jízdu v úsecích, které se projízdějí rychle. Pokud bychom se tedy podívali na změny zpoždění na trase mezi dvěma referečními body, v ideálním případně by měli být nulové.

Pro řešení odhadu toho typu spoždění stačí navrhnout systém na odhat zpoždění v půběhu jízdy mezi referenčními body z historických dat jízd.

Pro vyloučení všech pochybností je hodno uvést, že se naše Práce nesnaží předpovědět zpoždění, které spoj může nabrat vzhledem k dosavadnímu průběhu trasy. Tedy např. nijak nezohledněuje to, že spoj právě stojí v mimořádné koloně a dalo by se tedy předpokládat, že zpoždění bude rychle růst i v následujících minutách. Ale naopak Práce se snaží odhadnou zpoždění v danném bodě na trase vzhledem k obvyklému profilu jízdy. Tedy např. pokud by výše uvažovaná kolona byla pravidelná Práce ji zohlední ve statistických modelech.

1.2.1 Příklad nelineárního profilu trasy

Celé ilustrováno na příkladě jízd mezi dvěma zastávkama K letišti a ZLičín, kde je nelineární profil jízdy vidět velice dobře. Jedná se totiž o trasu přesně odpovídající popisu problému.

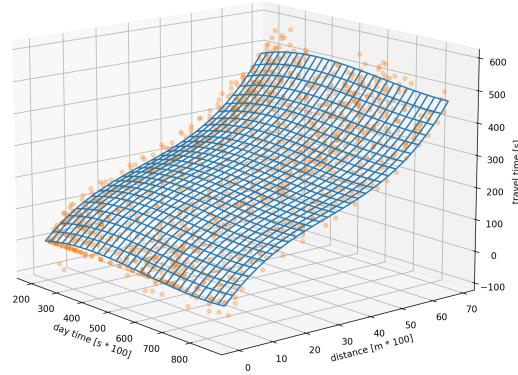
Popsané rozdíly v rychlosti a nelineární profil trasy je patrný na grafu 1.2. Za povšimnutí táké stojí viditelné spomalení průjezdů v ranní špičce, 7–9² hodina ráno.

Tento příklad dále podrobněji analyzován v kapitole 2.3.2.

Na obrázku 1.3 je pro bližší představu popsané trasy vidět trasa spoje vykreslená do mapy.

²časy jsou v UTC

K Letišti → Zličín



Obrázek 1.2: Modrá plocha značí vymodelovaný profil trasy. Oražové body jsou jednotlivé vzorky poloh vozidel. Data pro graf jsou ze dnů 20.–21. 2. 2020



Obrázek 1.3: Trasa mezi zastávkama K Letišti a Zličín. Zdroj: mapy.cz

Rozbor trasy

Celá tato trasa má necelých 7 km a její průjezd spojem VHD trvá 10 minut. Prvních 600 metrů je vedeno po obecní komunikaci, přes křižovatku a nájezd na Pražský okruh. Průměrná rychlosť vozidel byla 35 km/h^3 .

Dále trasa pokračuje přes Pražský okruh rovně až do vzdálenosti 4.9 km od zastávky K Letišti, kde začíná nájezd na ulici Na Radost. Dá se předpokládat, že vozidla se na komunikaci vyšší třídy pohybují rychleji což dokazuje, že na tomto úseku trasy se průměrná rychlosť vozidel zvýšila na 63 km/h.

Poslední úsek se tedy skládá z výjezdu z Pražského okruhu, průjezdu křižovatkou, jízdy po obecní komunikaci a vjezdu do stanice Zličín. Délka úseku je 2 km. Průměrná rychlosť za celou trasu se na tomto úseku snížila na 55 km/h.

1.2.2 Současná řešení

Algoritmus na odhad aktuálního zpoždění mezi dvěma referenčními body již existuje a je součástí Datové Platformy – Golemio, ze kterého se čerpají data pro tuto práci. (Detailní popis dat uveden v kapitole ??.)

Nicméně tento algoritmus nijak nezohledňuje variabilitu profilu trasy. Totiž v tomto řešení je nahlíženo na postup vozidla na trase jako na lineární funkci vůči času. Je ovšem zřejmé, že rychlosť vozidel není konstantní, neboli doba jízdy není linárně závislá na ujeté vzdálenosti.

Proto je potřeba tento odhad zpřesnit, což je cílem naší práce. K tomuto cíli jsme byli nasměrování v rámci schůzky s pracovníky společnosti OICT, kde bylo řečeno, že toto je problém současného řešení, který je potřeba vyřešit.

1.3 Analýza zdroje dat

V této kapitole je popsán zdroj real-timových dat o polohách vozidel využívané v této práci.

1.3.1 Přístup k datům

Vozidla vysílají data o své poleze při různých událostech. Zejména pak při brzdění, rozjezdu, vyhlášení zastávky, nebo jinak každých 20 sekund⁴.

Taková data pak přímo putují k provozovatelovi systému na monitorování vozidel, kterým je společnost Kapsch jakožto partner DPP. Ten však tato data zpracovává a posílá ke zveřejnění na platformě Golemio. Bohužel při tomto procesu zpracování se vytratí informace o události v jaké byly data pořízeny. Tedy informace o příjezdu nebo odjezdu ze zastávky jsou zjistitelné pouze z GPS souřadnic a následném odhadu pozice vozidla na trase dané linky.

³Počítáno podle vozidel, které poslaly polohu v 600m (resp. 4.9km, resp. 6.6km pro další údaje o rychlosti) vzdálenosti od zastávky. Počet záznamů o poloze vozidel se v různých vzdálenostech liší.

⁴Řečeno zaměstnancem OICT na schůzce 4. 5. 2019

Po té co jsou tyto data přeneseny do společnosti Operátor ICT by měla být zveřejněna, nicméně data ve výše popsané podobě jsou poměrně chudá, proto je k nim přidáno více atributů. Jedná se o dopočet poslední projeté zastávky, ujeté vzdálenosti od výchozí stanice, zpoždění v poslední zastávce.

Z pohledu této práce je nejzajímavější informace o vzdálenosti, kterou vozidlo urazilo od jeho výchozí zastávky. Dále jsou přidána data o jízdních rádech a zastávkách jejichž původcem je ROPID.

Real-time data o polohách, která jsou již neplatná (zastaralá) se neposílají (posílá vždy pouze neaktuální informace) a i z Datové platformy jsou data po pár minutách nenávratně smazány.

Dokumentace

Na úvod je nutné poznamenat, že datová platforma je stále ve vývoji a formát dat se může měnit. S tím mohou přicházet určité výpadky a problémy. K jednomu takovému výpadku došlo i při vývoji této práce, kdy po dobu 14 dnů plafomarma vůbec neodpovídala na dotazy nebo vracela prázdné datasety.

Současně s využívanou verzí API, je nasazená i pokročilejší API ve verzi 2, které obsahuje více informací a je přehledněji upravena. Nicméně při zahájení vývoje této práce nebyla verze 2 k dispozici, proto jsou využívána data pouze ze starší verze.

Oficiální uživatelská dokumentace datové platformy⁵ je poměrně zastaralá sama o sobě tak, že aktuální sada parametrů jí neodpovídá a neobsahuje žádné popisy nebo vysvětlení dat. Proto vysvětlení jednotlivých atributů se zakládá na intuitivním pochopení nebo vyplynulo z jednání se správci platformy. V následujících kapitolách bude popsán formát dat, tak jak přichází ze zdroje. Ten se může od oficiálně vystavené dokumentace lišit. A také budou popsány pouze atributy využívané v této práci nebo zajímavé pro její budoucí rozvoj.

Každá datová sada je exportována ve formátu GEOJSON pokud se jedná o geografická data, nebo jinak ve formátu JSON. Přistupuje se k nim přes jednotné webové rozhraní pomocí HTTP požadavku daného URL adresou a jeho hlavičkou.

Ačkoli se dokumentace tváří tak, že data jsou exportována ve formátech JSON nebo GEOJSON, většinou formát dat není přesně podle specifikace těchto formátů. Například může být uveden atribut `wheelchair_accessible`, který je typu `bool` a je nastaven na hodnotu `True`, nicméně podle specifikace se tyto hodnoty píší s malým písmenem⁶. Pro tuto práci to sice nepředstavuje komplikaci, protože tento atribut není potřeba, ale mohlo by se stát, že některé parsery JSONu vyhodnotí řetězec jako nevalidní a skončí chybou.

⁵Golemio: <https://golemioapi.docs.apiary.io>

⁶V průběhu tvorby této práce byla chyba opravena.

Celá datová platforma Golemio je pojatá jako Open Source projekt⁷. Tedy je možné její zdrojový ko'd vylepšit či opravit nebo také čtením ko'du detailně porozumět jak zde popisované zpracování dat funguje. Avšak takový rozbor zdrojového ko'du je mimo rozsah této práce.

Pozice vozidel

Ze zveřejněných dat na této platformě jsou nejdůležitější data pro tuto práci pozice vozidel. Jelikož se jedná o real-time data, data rychle zastarávají a je nutné je velmi často aktualizaovat.

Využívané atributy jsou:

- `coordinates` aktuální GPS souřadnice vozidla
- `origin_timestamp` čas zachycení pozice vozidla, v časovém pásmu UTC
- `gtfs_trip_id` unikátní identifikátor tripu pro spárování s jízdním řádem
- `gtfs_shape_dist_traveled` vzdálenost vozidla uražená od začátku tripu v metrech
- `delay_stop_departure` zpoždění zachycené při odjezdu z poslední projeté zastávky v sekundách

Příklad dat popisující aktuální polohu vozidla, na kterém je možno vidět strukturu dat i další atrubuty. Řada z nich je pro tuto práci zbytečná. Dále je možno si povšimnout atrubutu `all_positions`, který obsahuje všechny zaznamenané pozice daného vozidla na jeho aktuální trase, tento atribut je z důvodů objemu dat volitelný a pro tuto práci se nevyužívá.

```
"geometry":{  
    "coordinates":[14.91724,50.41881],  
    "type":"Point"  
},  
"properties":{  
    "trip":{  
        "cis_agency_name":"ČSAD Česká Lípa",  
        "cis_id":"260467",  
        "cis_number":3008,  
        "cis_order":2,  
        "cis_parent_route_name":"467",  
        "cis_real_agency_name":"ČSAD Česká Lípa",  
        "cis_short_name":null,  
        "cis_vehicle_registration_number":1073,  
        "gtfs_route_id":"L467",  
        "gtfs_route_short_name":"467",  
        "gtfs_trip_id":"467_252_200105",  
    }  
}
```

⁷Programátorská dokumentace je dostupná na <https://operator-ict.gitlab.io/golemio/documentation/>

```

    "id":"2020-02-23T18:50:00Z_260467_467_3008",
    "start_cis_stop_id":30107,
    "start_cis_stop_platform_code":"A",
    "start_time":"19:50:00",
    "start_timestamp":"2020-02-23T18:50:00.000Z",
    "vehicle_type":4,
    "wheelchair_accessible":true
},
"last_position":{
    "bearing":20,
    "cis_last_stop_id":21393,
    "cis_last_stop_sequence":28,
    "delay":261,
    "delay_stop_arrival":null,
    "delay_stop_departure":287,
    "gtfs_last_stop_id":"U3389Z1",
    "gtfs_last_stop_sequence":30,
    "gtfs_next_stop_id":"U2987Z30",
    "gtfs_next_stop_sequence":31,
    "gtfs_shape_dist_traveled":"64.1",
    "is_canceled":false,
    "lat":"50.41881",
    "lng":"14.91724",
    "origin_time":"21:29:37",
    "origin_timestamp":"2020-02-23T20:29:37.000Z",
    "speed":20,
    "tracking":2,
    "trips_id":"2020-02-23T18:50:00Z_260467_467_3008"
},
"all_positions":{
    "features":[],
    "type":"FeatureCollection"
},
"type":"Feature"
}

```

Jízdy

Dále jsou k dispozici data o každé jízdě. To je popis trasy vozidla, včetně zastávek a časů příjezdů a odjezdů do/z nich. Také může být vyžádáno k informacím o jízdě připojit celý shape trasy, tj. lomená čára kopírující celou trasu daného tripu po povrchu Země.

Míra unikátnosti identifikátorů těchto tripů je předmětem dohadů a zřejmě jsou pod správou plánovačů MHD, nicméně pro účely této práce může být předpokládáno, že každá jízda má vlastní jízdní řád, který se váže na čas a každá jízda jede nejvýše jednou za den.

- `trip_headsign` nápis na čele vozidla, typicky cílová stanice
- `route_id` číslo linky

- `trip_id` unikátní identifikátor tripu pro spárování s real-time daty, pravděpodobně odpovídá atributu `gtfs_trip_id`

Navíc s každým tripem může být vyžádáno zaslání seznamu zastávek, kterýma projízdí. Zde jsou k dispozici informace vázající se k danému průjezdu zastávkou. Každá uvená zastávka s sebou nese i kompletní informaci o sobě, tedy má stejnou informační hodnotu jako samostatný dotaz na jednotlivé zastávky z datové sady zastávek.

Zastávky

- `arrival_time` čas příjezdu spoje do zastávky
- `departure_time` čas odjezdu spoje do zastávky
- `shape_dist_traveled` vzdálenost zastávky na trase od výchozího bodu daného tripu v metrech
- `stop_id` unikátní identifikátor zastávky
- `coordinates` GPS souřadnice zastávky, často `null`, je třeba využít atributy `stop_lat` a `stop_lon`
- `stop_name` název zastávky

Příklad dat popisujících jednu jízdu včetně zastávek. Seznam zastávek a body tras jsou zkráceny vzhledem k objemu dat.

```
"bikes_allowed":2,
"block_id":"",
"direction_id":1,
"exceptional":1,
"route_id":"L421",
"service_id":"1111100-1",
"shape_id":"L421V4",
"trip_headsign":"Kolín,Nádraží",
"trip_id":"421_225_191114",
"trip_operation_type":1,
"trip_short_name":"",
"wheelchair_accessible":2,
"stop_times": [
    {
        "arrival_time":"14:14:00",
        "arrival_time_seconds":null,
        "departure_time":"14:14:00",
        "departure_time_seconds":null,
        "drop_off_type":"0",
        "pickup_type":"0",
        "shape_dist_traveled":0,
        "stop_headsign":"",
        "stop_id":"U2033Z5",
        "stop_sequence":1,
        "timepoint":null,
    }
]
```

```

"trip_id":"421_225_191114",
"stop": {
    "geometry": {
        "coordinates": [
            null,
            null
        ],
        "type": "Point"
    },
    "properties": {
        "level_id": "",
        "location_type": 0,
        "parent_station": "",
        "platform_code": "5",
        "stop_code": null,
        "stop_desc": null,
        "stop_id": "U2033Z5",
        "stop_lat": 49.87486,
        "stop_lon": 14.9078,
        "stop_name": "S\u00e1zava,Aut.st.",
        "stop_timezone": null,
        "stop_url": "",
        "wheelchair_boarding": 0,
        "zone_id": "5"
    },
    "type": "Feature"
},
...
],
},
"shapes": [
{
    "geometry": {
        "coordinates": [
            14.90778,
            49.87494
        ],
        "type": "Point"
    },
    "properties": {
        "shape_dist_traveled": 0,
        "shape_id": "L421V4",
        "shape_pt_lat": 49.87494,
        "shape_pt_lon": 14.90778,
        "shape_pt_sequence": 1
    },
    "type": "Feature"
},
...
]

```

1.3.2 Analýza statických dat

Sběr dat probíhal ve dnech 20.–24. 2. 2020.

Data byla přebírána pouze z dat o každé jednotlivé jízdě a aktuálních poloh vozidel stahovaných každých 20 sekund. Tedy pokud určitou zastávkou žádný spoj neprojel nebo se uložení spoje z důvodu neúplných dat nezdařilo, může nějaká zastávka v databázi chybět. Stejně tak mohou chybět i nějaké spoje. Nejčastěji chybějící požadovaná data jsou informace o spoždění spoje v poslední zastávce, zcela nenalezený spoj ve zdroji dat a tedy chybějící jízdní řád. Nicméně jedná se zanedbatelné množství. Práce totiž nemůže počítat s poškozenými daty a není jejím úkolem držet všechny zastávky, ale pouze ty kterými nějaký spoj projíždí. To z důvodu přehlednosti zastávek při vizualizaci, kde ukazovat nepoužívané zastávky nemá smysl a také pro odhadu zpoždění nedává smysl počítat něco pro zastávku, která není obsluhovaná.

Zastávky

Do databáze bylo celkem vloženo 5820 nástupišť⁷, které naleží celkem 2961 zastávkám. Ale naprostá většina (74 %) zastávek jsou párová, tedy mají pouze 2 nástupiště. Jednosměrných zastávek je 18 %, zastávek se 3 nástupišti jsou 3 %. Nejvíce nástupišť mají stanice Slaný Aut. nádr. (14), Černý Most (12), Kladno Autobusové nádraží (11).

Jízdy

Celkem bylo nalezeno 12788 spojů, z nich naprostá většina vyjela opakově v následující den⁸.

Vstupní soubory

Pro testovací účely bylo celkem pořízeno 15 794 obrazů dopravní situace zá-znamenávající aktuální polohy vozidel.

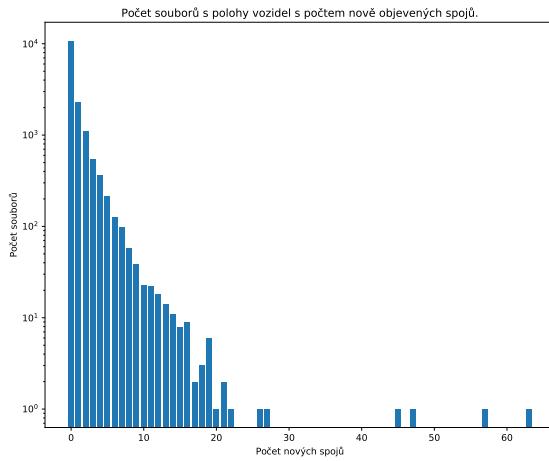
Pro stahování těchto souborů byl využit script napsaný v jazyce Bash, který po dobu 4 dnů periodicky každých 20 sekund stáhl aktuální obraz poloh vozidel. Tento stažený dokument pak uložil v komprimovaném formátu a označil časem stažení. Tento script běžel na počítači, který pro účely této práce zapůjčil k využití pan profesor Jakub Klímek, jemuž tímto děkuji.

```
#!/bin/sh

cur_file="current_json_file.json";

while :
do
    curl -s --header "Content-Type: application/json; charset=utf-8" \
--header "access token" \
```

⁸Ze všech vypravených jízd ve dnech 20. 2. 2020 (9334) a 21. 2. 2020 (9428) jich 9051 bylo označeno ve zdrojových datech stejným identifikátorem, tedy z našeho pohledu sdílely jízdní řád a cestu.



Obrázek 1.4: Počet souborů s počtem nově nalezených spojů v nich.

```
'https://api.golemio.cz/v1/vehiclepositions' > "$cur_file";

if [ 0 -ne $? ];
then
    today='date +%Y-%m-%dT%H.%M.%S';
    echo "Curl failed $today" >> downloader.log;
    sleep 19;
    continue;
fi

today='date +%Y-%m-%dT%H.%M.%S';
tar -czf "../raw_data-2/${today}.tar.gz" $cur_file;
echo "File ${today}.tar.gz saved." >> downloader.log
sleep 20

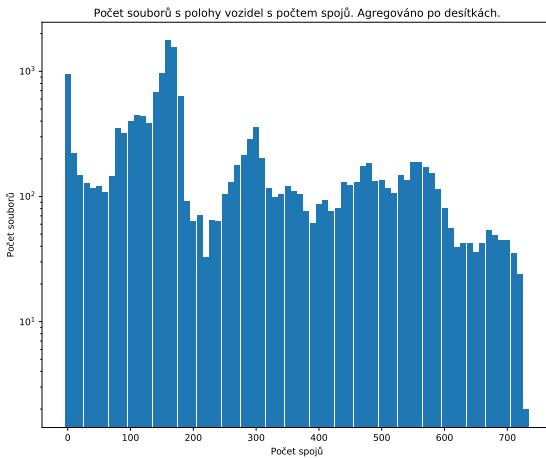
done
```

Z celkového počtu nalezených spojů vyplývá, že počet nově nalezených spojů v jednom obrazu je méně než jeden. Kompletní histogram počtu souborů poloh vozidel, které obsahují určitý počet nově objevených spojů je zobrazen na grafu 1.4.

Maximální počet vozidel obsažených v jednom souboru je méně než 800. Kompletní histogram počtu souborů s počtem vozidel celkem v jednom souboru je na grafu 1.5. Z tohoto grafu vyplývá, že velká většina vstupních souborů z celkového počtu 15793 obsahuje do 200 vozidel v každém souboru.

Avšak ne všechna vozidla v každém souboru jsou nová nebo dostala změněny oproti předešlému záznamu. To z důvodu, že je zdroj dat nastaven tak, aby vysílal polohu vozidla jako aktuální, i když už je zastarálá několik minut. V takovém případě se zpracovává vzorek volohy vozidla stále dokola.

Další rozbor dat na grafu 1.1 a v kapitole ??.



Obrázek 1.5: Počet souborů s počtem nově nalezených spojů v nich.

1.4 Analýza vizualizačních nástrojů

Jak bylo řečeno v úvodu, součástí práce je i vizualizace spočítaných dat.

To bude provedeno formou front endové aplikace, která zobrazuje mapu a do ní zanáší data o vozidlech VHD. Funkční požadavky této aplikace jsou inspirovány již existujícími řešeními tohoto problému.

1.4.1 Mapové podklady

Jak vyplývá z funkčních požadavků data budou zobrazována v mapě. Mapu si samozřejmě nebudeme kreslit sami, ale využijeme jedno z již existujících řešení, které umožňuje zobrazení mapy a do ní zanést vlastní data. Takové služby mohou být provozovatelem zpoplatněny, ale pro naše demonstrační účely, kdy budeme využívat tuto službu velmi málo, bývá od poplatků většinou upoštěno.

Jedním z těchto poskytovatelů je společnost Google, která má propracované mapové podklady a prostřednictvím služby Google Maps a poskytuje pro tuto práci požadovanou službu nazývanou Google My Maps⁹.

Další platformou je Mapbox¹⁰, který poskytuje s využitím dalších knihovem velmi podobné služby jako Google My Maps. Nicméně narozdíl od Googlu využívá jako mapový podklad OSM otevřená geografické data.

Protože smyslem práce je v co největší míře využít otevřená data je žádoucí využít právě službu Mapbox.

1.4.2 Současná řešení

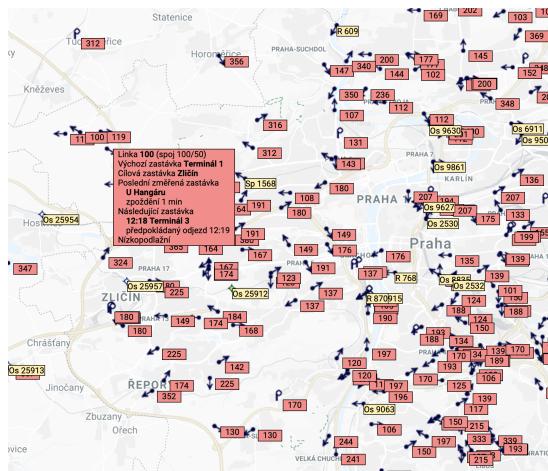
Vizualizaci vozidel VHD do mapy již nabízí několik portálů. Všechny jsou však poměrně strohé.

⁹<https://www.google.com/maps/about/mymaps/>

¹⁰<https://www.mapbox.com>



Obrázek 1.6: Mapa z golemio.cz.



Obrázek 1.7: Mapa z www.tram-bus.cz.

Golemio

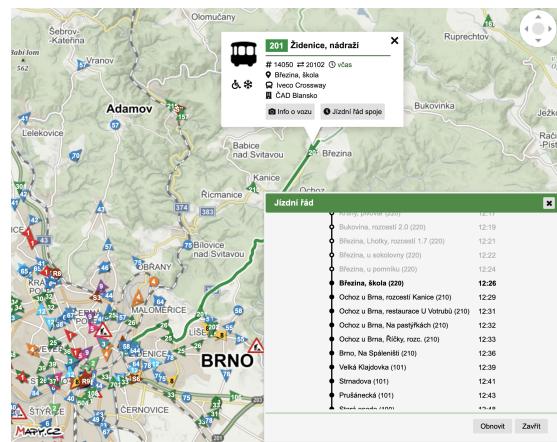
Takovou mapu zobrazuje i samotný provozovatel datové platformy. Nicméně nejsou zde vidět ani čísla linek zobrazených autobusů, natož pak nějaké další informace. Příklad vizualizace je uveden na obrázku 1.6.

Tram-bus

Dalším poskytovatelem je portál tram-bus, který si vede o něco lépe. Ukazuje směr jízdy vozidel, čísla linek a po kliknutí informace o zpoždění a nejbližší zastávky. Pozn.: na mapě již jsou vidět spoje DPP, protože v době psaní této práce již byly data veřejné. Příklad vizualizace je uveden na obrázku 1.7.

IDSJMK

Mimo Prahu je velice pěkně udělaná aplikace pro zobrazení vozidel IDSJMK (Integrovaný dopravní systém Jihomoravského kraje). Ten ihned po načtení stránky zobrazuje všechny dobravní prostředky, tedy tramvaje, autobusy a vlaky vše s čísly linek. Dále pak umožňuje po kliknutí na vybraný spoj zobrazit více informací včetně jízdního řádu. Příklad vizualizace je uveden na obrázku 1.8.



Obrázek 1.8: Mapa z mapa.idsjmk.cz.

Tato aplikace je po vizuální i funkční stránce dobrou inspirací pro tvorbu aplikace v této práci.

2. Návrh řešení

V této kapitole je popsán návrh technického řešení uvedených problémů.

2.1 Úvod

Běh celé aplikace bude rozdělen do dvou částí.

- Stahování a ukládání real-time dat o polohách vozidel do datového skladu, které budou doplněny o odhad zpoždění pro okamžité zveřejnění v uživatelské aplikaci.
- Modelování profilů jízd jednotlivých úseků. Tyto modely budou pak dále sloužit k odhadování zpoždění v budoucnu. Výpočet modelů bude prováděn jednou za delší časový úsek (nejlépe jednou za den).

Protože obě části jsou na sobě závislé v iniciálním běhu bude prováděna první část sběru dat bez odhadu zpoždění, nebo pomocí již existujícího triviálního lineárního odhadu.

Schéma návrhu celé aplikace a komunikační mapa jednotlivých komponent je ilusrrována na diagramu 2.1.

2.1.1 Funkční a kvalitativní požadavky

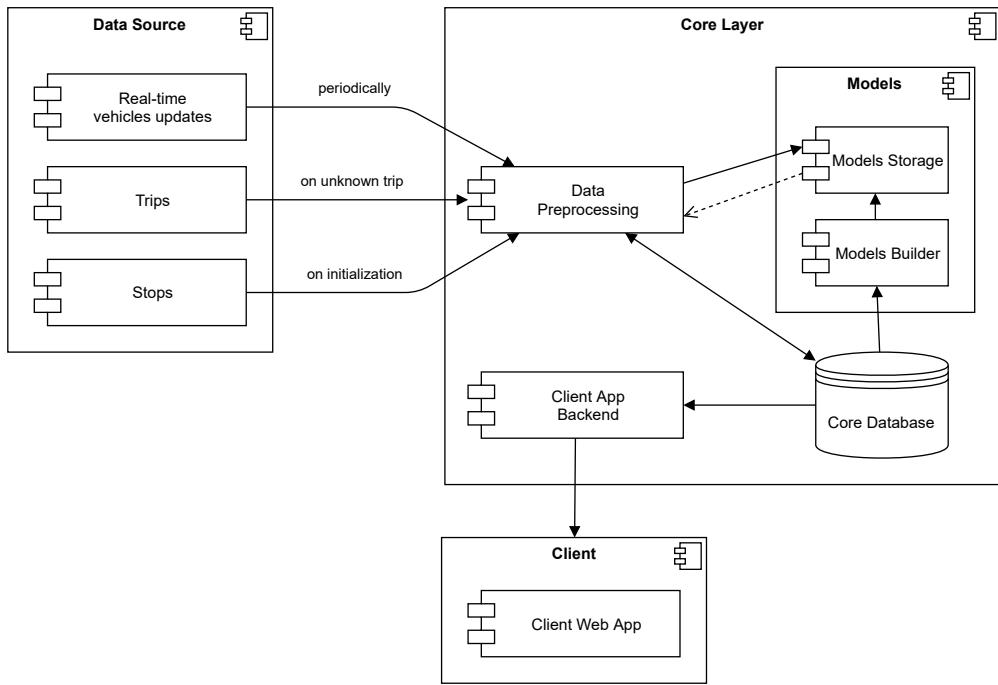
Nejprve specifikujme požadavky systému, na kterém se pak bude zakládat konkrétní návrh řešení jádra celé aplikace (bez backendu vizualizace).

Funkční požadavky

- Popsaný odhad změny zpoždění na trase mezi dvěma referenčními body je nutné počítat v co nejkratším čase tak, aby cestující byli dobře informováni o stavu jejich spoje a mohli tyto informace využít např. při dobíhání spoje. A proto je potřeba zpracovávat data okamžitě po jejich vydání, spočítat odhad zpoždění a vystavit tato data veřejně. Vzhledem k tomu, že tato data velmi rychle zastarávají je nutné provádět tento proces co možná nejrychleji¹.
- Data o polohách vozidel VHD v Datové platfomě jsou aktualizována nejpozději každých 20 sekund, více v kapitole 1.3. Tedy pro minimalizaci rychlosti zastarávání dat a získání všech existujících vzorků dat o polohách je nutné data stahovat alespon každých 20 sekund.
- Odhad zpoždění se bude provádět na základě historických dat z posledních vyšších jednotek dnů². Tím se sníží dopad mimořádné události na předpovědní model, která může na trase vzniknout. Zároveň by však neměla být

¹Průměrná doba jízdy spoje mezi zastávkami je cca 5 min. Rozložení počtu úseků mezi zastávkami k délce jízdy mezi nimi je závislé a podobné rozložení vůči vzdálenosti ilustrované na grafu1.1.

²Pro demonstrativní účely této práce jsou využívány historická data pouze ze 4 dnů (2 pracovní a 2 víkendové).



Obrázek 2.1: UML diagram návrhu aplikace

započítávána data starší několik týdnů, protože dopravní situace se mění v závislosti na ročním období nebo také pokud se na trase vyskytne delší omezení dopravy. Pak je požadováno, aby se takové omezení projevilo v modelu profilu jízdy co možná nejdříve. Navíc se bude rozlišovat mezi daty z pracovních dnů a nepracovních dnů, to protože samotné jízdní řády se mohou lišit³ a také se do velké míry liší hustota dopravy, která má velký vliv na profil jízdy. TODO do navrhy na zlepšení: Pro zlepšení výsledků by bylo lepsi respektovat svatky, kazdy den v tydnu zvlast atp.

- Zpracování historických dat bude probíhat vždy po delší době, nejlépe jednou za den. To umožní provádět náročnější výpočty, které by za normálního provozu neúměrně přetížily systém. Navíc vzhledem k povaze cíle práce ani není žádoucí zpracování historických dat provádět častěji než jednou denně, protože se nepokoušíme okamžitě reagovat na změnu dopravní situace, ale modelovat profil jízdy vždy až pro celý den.
- Uložená historická data budou strukturovaná tak, aby nad nimi mohly být prováděny statistické výpočty minimálně o frekvencích jízd spojů, vzdálostech tras a zpoždění spojů.

³Ve dnech pracovního volna se v některých případech liší doba jízdy mezi mezastávkama pro stejnou dvojici zastávek. A to je porušení základního předpokladu z kapitoly 2.3.1 Základní předpoklady.

Kvalitativní požadavky

- Řešení bude schopno při jedné aktualizaci zpracovat alespon⁴ 1000⁴ vzorků poloh vozidel, kde 10 % vzorků může být o dosut neznámých jízdách. V tomto případě je potřeba stáhnout jízdní řád konkrétní jízdy a její jízdní profil, což představuje navíc dotaz na Datovou platformu jakožto zdroje dat pro tuto práci.
- Vypočítané modely profilů jízd budou dávat odhad zpoždění lepší (až na vyjimku popsanou níže), než je lineární odhad. To znamená, že zpoždění vypočítaná pro každý přijatý vzorek polohy vozidla mezi dvěma referenčními body na trase bude mít menší rozptyl než lineární odhad zpoždění.
- V případě, že spočítaný odhad zpoždění vozidla by zastaral natolik rychle, že v okamžík zveřejnění by již nebyl platný, nedává smysl zpoždění odhadovat pokročilejší metodou.

2.2 Zpracování vstupních dat

Struktura uložení dat se zakládá na struktuře zdrojových dat popsaných v kapitole ?? Analýza zdroje dat.

Na datové platformě jsou real-time data o vozidlech dostupná do historie řádově jednotek minut, což je naprostě nedostatečné pro jakékoli pozdější využití v rámci této práce. Především pro počítání statistik a modelování profilů jízd nad daty je potřeba zřídit lokální databázi, která bude držet historická data tak, jak byla obdržena od zdroje. Navíc data jsou poskytována ve formátu JSON, který svou povahou není zrovna úsporný co se do velikosti souboru týče. Proto je vhodné zvolit ukládání dat v jiném formátu.

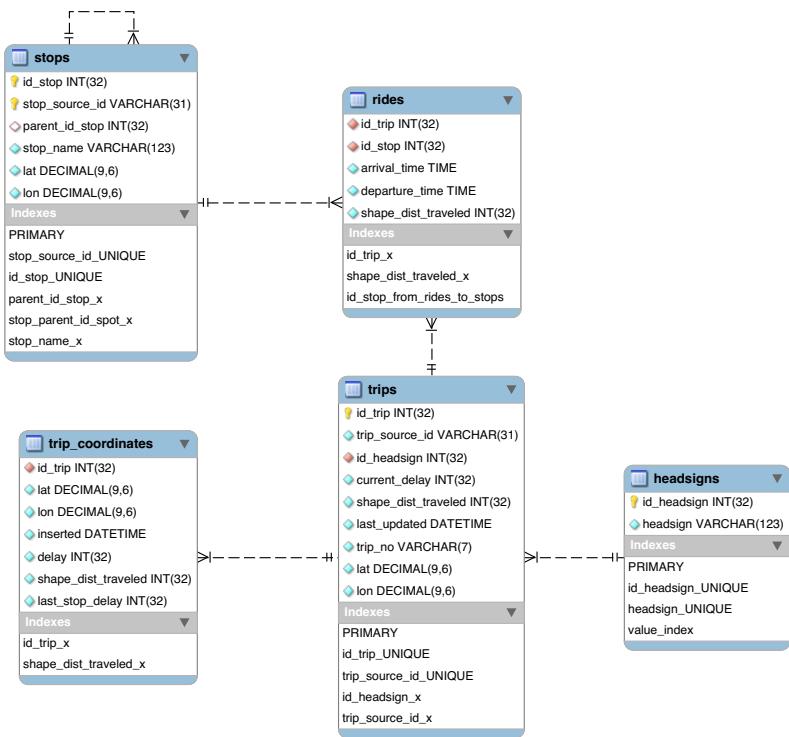
2.2.1 Databáze

Za tímto účelem tato práce využívá relační databázi obsluhovanou dotazovacím jazykem SQL. Struktura databáze je vyobrazena na EER diagramu 2.2⁵. Tato databáze se skládá z 5 tabulek. Jsou jimi:

- trips všechny objevené jízdy
 - id_trip unikátní identifikátor používaný v databázi
 - trip_source_id identifikátor tripu převzatý ze zdroje dat
 - id_headsign identifikátor nápisu pro daný trip
 - current_delay aktuální zpoždění tripu
 - shape_dist_traveled aktuální vzdálenost ujetá od výchozí stanice
 - last_updated čas poslední aktualizace, převzatý ze zdroje dat

⁴20. 2. 2020 mezi 7:00 a 7:10 bylo na trase přes 600 vozidel

⁵SQL dotazy na sestavení celé databaze jsou definovány v příloze database.sql. Pro testovací, debugovací a demonstrační účely slouží navíc i jiné databáze, které jsou strukturou totožné jako produkční databáze.



Obrázek 2.2: EER diagram databáze.

- **trip_no** číslo dané linky
- **headsigns** nápisu nad vozidlem, cílová stanice
 - **id_headsign** unikátní identifikátor nápisu
 - **headsign** text nápisu
- **trip_coordinates** všechna historická real-time data
 - **id_trip** identifikátor tripu, ke kterému se záznam váže
 - **lat** zeměpisná šířka polohy vozidla
 - **lon** zeměpisná délka polohy vozidla
 - **inserted** čas vložení záznamu
 - **delay** zpoždění zachycené v poslední projeté stanici před pořízením záznamu
 - **shape_dist_traveled** vzdálenost ujetá od výchozí stanice tripu
- **stops** všechny zastávky
 - **id_stop** unikátní identifikátor zastávky
 - **trip_source_id** identifikátor zastávky převzatý ze zdroje dat
 - **parent_id_stop** identifikátor rodičovské zastávky, pokud existuje
 - **stop_name** název zastávky
 - **lat** zeměpisná šířka polohy zastávky
 - **lon** zeměpisná délka polohy zastávky
- **rides** trasa každého tripu, seznam zastávek s časy odjezdů a příjezdů tvořící jízdní řád, relační tabulka mezi spoji a zastávkami, pořadí zastávek v jakém jsou spojem obsluženy je určeno časem příjezdu resp. odjezdu tak i atributem **shape_dist_traveled**
 - **id_trip** identifikátor tripu
 - **id_stop** identifikátor zastávky
 - **arrival_time** čas příjezdu tripu do zastávky
 - **departure_time** čas odjezdu tripu ze zastávky
 - **shape_dist_traveled** vzdálenost zastávky od výchozí zastávky tripu

Atributy se jménem *source_id jsou pravděpodobně unikátní identifikátor entity ve zdroji dat, nicméně z dokumentace zdroje to nevyplývá. Také je tento identifikátor ukládán jako textový řetězec, ačkoli je tvořen pouze číslicemi a podtržítky, není nikde zaručeno, že jej lze jednoduše převést na číselný ko'd. Takže pro lepší výkon databáze je použito automaticky generované id typu INT.

Každá tabulka má několik indexů, které zlepšují výkon databáze při vkládání a hledání dat. Obzvláště pokud je atribut označen jako unikátní, kde se při každém vložení ověřuje unikátnost.

Databáza je nastavená tak, aby umožňovala získat všechny potřebné informace o vozidlech, ale hlavně přístup k historickým real-time datům a to separovaně pro dvojci referenčních bodů. K tomu se zejména využívá atribut `shape_dist_traveled`, který označuje vzdálenost na trase od výchozí zastávky a je součástí vstupních dat jízdních řádů i aktuálních poloh vozidel.

2.2.2 Plnění databáze

Tato databáze bude plněna skriptem, jehož bude naprogramován taky, aby vždy stál aktuální obraz dopravní situace a tato data uložil do dotabáze. Toto stahování z datové platformy probíhá podle následujícího algoritmu.

Algoritmus:

```
načti všechny dostupné zastávky
dokud skrip běží
    načti aktuální polohy vozidel
    pro každé nalezené vozidlo
        pokud jízda vozidla je známá
            aktualizuj data o jízdě
        jinak
            stáhní informace o jízdě
            zpracuj a vlož jízdu do databáze
```

Protože všechny infomace ukládané do databáze jsou důležité pro hlavní cíl této práce, tak pokud se vyskytne jízda, který neobsahuje některou z požadovaných infomarcí je pak automaticky zahozena. To je řešeno pomocí databázových transakcí tak, aby stav databáze byl vždy konzistentní. Transakce v obecném smyslu fungují tak, že můžeme měnit data v databázi (i více záznamů) a tyto změny se zapíší do samotné databáze až po potvrzení, že všechny změny byly provedeny správně, pokud během provádění změn nastane chyba, můžeme v jakékoli fázi provádění změn všechny dosud proveedené změny zahodit a vrátit se do původního stavu databáze před započetím transakce. Tedy pokud nejsou poskytnuta data ve formátu, který skript akceptuje, nebo nějaké povinné atributy chybí. Vložení celé jízdy nebude provedeno.

Nejčastěji chybějící atribut je zpoždění v poslední zastávce, toto je nutné vědět pro počítání zpoždění mezi referenčními body (zastávkami). Absence této informace může být způsobena tím, že vozidlo vůbec nevysílá data potřebná k jejím dopočtení, pak nemá smysl jej do databáze zahrnovat. Nebo vozidlo už vysílá, ale ještě nezahájilo jízdu, tedy nemá žádnou poslední projetou zastávku, v takovém případě budou data ignorována až do doby příchodu první relevantní informace.

Mimo popsanou databázi se do určeného adresáře ukládají trasy jednotlivých jízd, která jsou ve formátu GEOJSON jako lomená čára definována souřadnicemi. Navíc data o trasách jsou používána pouze pro vizualizaci a jsou přijímány vizualizačním nástrojem ve formátu GEOJSON, tedy tyto data není nutné vůbec transformovat a není nutné je držet v hlavní databázi.

Stejně tak i aktuální polohy vozidel jsou mimo databázi zapisovány do souboru, který je určen a formátován pro čtení webovou aplikací. Aktualizace tohoto souboru je provedena přednostně, ihned po načtení real-timových dat. Tím se zabrání nechtěnému čekání na aktualizaci celé databáze, která může trvat jednotky sekund.

2.3 Algoritmus odhadu zpoždění

Z toho jak je problém formulován vyplývá, že se má odhadovat zpoždění mezi dvěma refenčními body a jediné takové jsou zastávky na trase daného spoje. Proto cíl algoritmu může být formulován, jako vytvořit popis průběhu trasy mezi každou dvojcí zastávek, které alespoň jeden spoj obsluhuje a jsou bezprostředně sousedící ve sledu zastávek ve směru jízdy tohoto spoje. Nechť se všechny dvojce zastávek a spoje je obsluhující splňující předcházející předpoklad označují jako AB a S . Jedna dvojice zastávek pak bude (a, b) a spoje je obsluhující se značí jak S_{ab} .

Z definice problému chceme modelovat jízdu vždy mezi danou dvojcí zastávek. Ke každé dvojci zastávek (a, b) bude náležet jeden model, popisující průběh jízdy mezi nimi. Tyto modely budou vycházet z historických dat průjezdů mezi těmito zastávkami.

Příklad takové dvojce zastávek a a b je uveden na příkladu v kapitole 1.2.1.

2.3.1 Základní předpoklady

Hned na začátek je potřeba ustanovit základní předpoklady, ze kterých bude vycházet sestrojený algoritmus vytvářející modely profilů jízd.

Zastávky je potřeba rozlišovat na jednotlivá nástupiště. Toto výrazně nezvýší počet dvojic zastávek AB . Protože naprostá většina zastávek má pouze dvě nástupiště⁶. Pro každý směr jedno. Pokud má více nástupišť, pak tak bývá v případech, kdy ze zastávky odjíždí spoje do více směrů a tudíž pro každné nástupiště je jiná následující zastávka – počet dvojic (a, b) se nezvýší. Z toho plyne zjednodušení, kterého se dopouštíme v průběhu celé práce, především pak pro tento algoritmus ohadu zpoždění a to, že termíny zastávka a nástupiště splývají.

Všechny spoje S_{ab} bez ohledu na linku nebo dopravce jedou ze zastávky a do zastávky b po stejné trase a tedy vzdálenost je konstatní. – Předpokládá se, že žádý dopravce nevyužívá jinou komunikaci a pro všechny platí pravidla silničního provozu stejně.

Čas jízdy ze zastávky a do b závisí pouze na denní době a dne v týdnu. Navíc platí žádny z dopravců nedisponuje právem přednosti v jízdě před jiným dopravcem nebo výrazně výkonějším vozidlem. Dojezdové časy mohou být ovlivněny jen charakterem řidiče, avšak toto není zjistitelné z poskytnutých dat a zároveň se předpokládá, že charaktere řidičů jsou rovnoměrně rozloženy mezi všechny dopravce a linky. Podle jízdních řádů některé linky jedou ve stejnou denní dobu

⁶Rozepsáno v kapitole 1.3.2

rychleji než jiné, avšak skutečné doba jízdy je stejná. To že některý spoj zastávku projízdí a tím je rychlejší než jiný spoj není porušení tohoto předpokladu protože se jedná o dvě různé dvojce zastávek.

Během jízdy mohou nastat mimořádnosti, které porušují výše uvedené předpoklady, nicméně detekce mimořádností a jejich řešení je nad rámec této práce a jejich počet je zanedbatelný, proto na statistické modely nebudou mít vliv.

2.3.2 Analýza dat a návrh modelování

Na úvod uvedeme, že ve všech grafech a následně pro počítání modelů byly všechny zobrazené vzorky poloh vozidel v grafech zarovnány tak, aby jejich jízda ze zastávky a vždy začínala se spožděním 0 sekund. To podle nahlášeného zpoždění v zastávce a .

Z dat je však patrné, že ne vždy se zarovnání do nuly podařilo. Takové případy jsou pak způsobeny chybami v datech vysílaných z vozidel nebo špatně určeným zpožděním v poslední projeté stanici na straně dodavatele dat. Takové chyby však vznikají spíše vyjímečně a proto ovlivňují statistické výpočty jen málo. Pro eliminaci těchto chyb je pak v implementován modul, který se pokouší očividně chybné vzorky najít a smazat. To primárně z důvodů vizualizace vzorků v grafech, kde jeden vzorek zcela mimo škálu jiných vzorků rozhodí celý graf a graf se tak stává nepřehledným.

Lineární model

Odhad zpoždění vozidla na trase se v současné době provádí pomocí lineárního modelu. Tedy s předpokladem, že vozidlo jede konstantní rychlostí po celou dobu jízdy mezi dvojcí zastávek a a b .

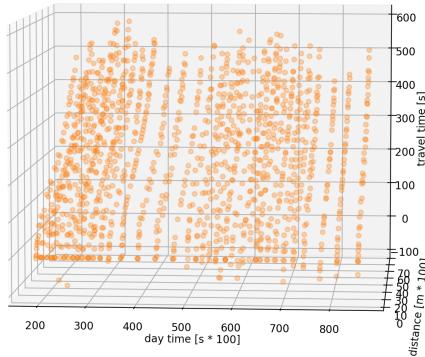
Ačkoli je snaha tento model nahradit lepším, v některých situacích může jeho použití i na dálé dávat smysl. Zejména pak v případech kdy není k dispozici dostatek dat a nebo je vzdálenost dvou zastávek natolik malá, že nemá smysl ani jakýkoliv odhad zpoždění dělat. (TODO do problemu: Lepší by bylo volit trasy s nejmenší dobou jízdy, ale čas jízdy je proměnlivý a težko se získá skutečná doba jízdy z dat. Navíc v praxi jsou vzdálenost a doba jízdy dostatečně závislé)

Polynomiální profil jízdy

Po analýze dat poloh vozidel a možných vlivů ovlivňujících profil jízdy je patrné, že v průběhu jednoho dne dochází na trase nejvíce k několika výkyvům rychlosti jízdy. Viditelné jako "vlny" v průběhu dne na grafu 2.3), v 8:30 hod⁷. jízda trvala téměř 10 minut, naopak ve večerních hodinách jízda trvá kolem 7 minut.

K tomuto dochází například v případech kdy spoj zastavuje ve městě a v následujících několika málo kilometrech jede pomaleji, poté zrychlý a dále opět vjede do města. Takový model se hodí spíše na delší trasy s plynulou jízdou.

⁷časy jsou uvedeny v UTC



Obrázek 2.3: Variabilita délky jízdy v průběhu dne

Stejně tak po analýze profilu jízdy vzávislosti na ujeté vzdálenosti je na grafu 2.4 vidět, že čas jízdy narůstá taky v jistých "vlnách".

Podle charakteru těchto dat nejlépe budou odpovídat modely získané pomocí lineární regrese, resp. polynomiální regrese.

Jako odhad zpoždění se pak vrací rozdíl skutečného počtu sekund na trase a predikce modelu. To celé se pak ještě přičítá k rozdílu predikce v modelu v čase a vzdálenosti příjezdu podle jízdního řádu a pravidelného příjezdu.

Polynomiální model se tedy hodí pro situace kdy je průběh trasy nějak ovlivněn vždy ve stejném úseku a má vliv na každý projíždějící spoj. Nebo se v průběhu dne pozvolna mění v závislosti na dopravním vytížení projížděných úseků.

Polynomiální model je pro představu vykreslen v grafu v úvodním příkladu v kapitole 1.2.1.

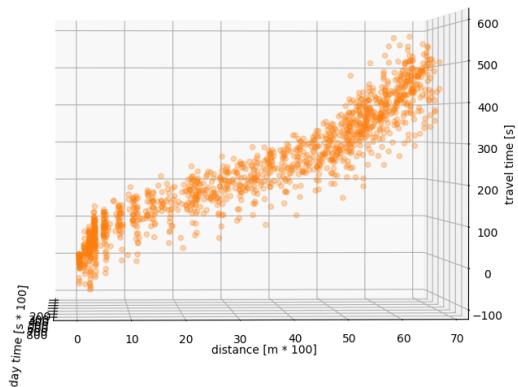
Nepravidelné profily jízdy

Výše popsaný příklad však ilustruje téměř ideální případ, kde je pravidelnost jízdy velmi dobře viditelná. Rozeberme si proto nyní i jiné druhy profilů jízd.

Na grafu 2.5 je patrné, že vzorky poloh vozidel profilu jízdy stále vytváří vlny. Ovšem už nejsou tak jednoznačně vidět jak v předchozím příkladě a především se v celém grafu objevuje pár vzorků, které zcela vybočují mimo největší zhluky vzorků. V těchto případech se, ale zřejmě jedná o případy vozidel, které potkala nějaká anomálie při výjezdu ze stanice a proto nabraly zpoždění hned na začátku.

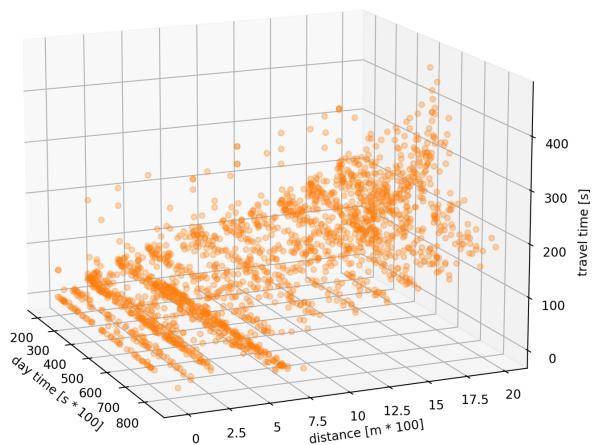
Pro velmi krátké trasy se zobrazené vzorky dat mohou jevit jako zcela nepravidelné. Příklad uveden na grafu ???. To je způsobeno tím, že doba jízdy trasy

K Letiště → Zličín

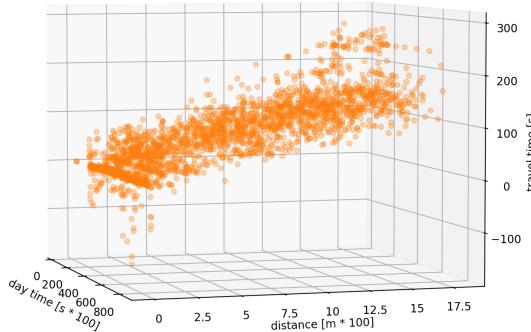


Obrázek 2.4: Variabilita času jízdy v závislosti na ujeté vzdálenosti

Černý Most → Chvaly



Obrázek 2.5: Úsek s nepravidelnostmi



Obrázek 2.6: Úsek s nepravidelnostmi 2

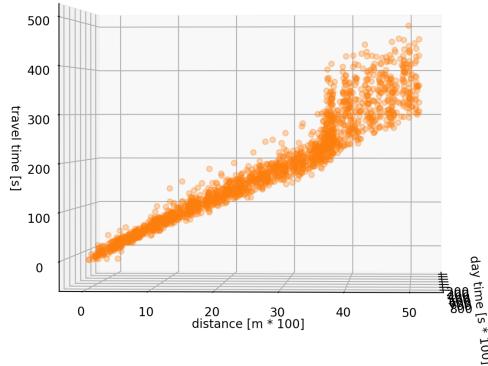
je natolik krátká, že jeden spoj trasu projede za minutu, ale druhý sopj, který se zdrží o zanedbatelný čas (z pohledu problému řešeného v této práci) přijede do následující stanice až za dvoujnásobou dobu. Dále je vysoký rozptyl vzorků způsoben nepřesnostma při měření polohy vozidel a dalších možných problémů. Jelikož se ale jedná o velmi krátké trasy nemá smysl jejich specifika vůbec řešit, protože spočítaná data by zastarala ještě před zveřejněním.

Model konkávním obalem

Na dalším grafu 2.7 je zobrazen příklad, kdy na trase exituje bod, který určité procento projíždějících spojů zdrží o netriviální dobu. Něco takového nastane, pokud spoje projíždí světelnou křížovatkou nebo místem kde se náhodně tvoří kolona vozidel. Zde dochází ke skové změně průběhu bodové funkce. Spojité modely, jakým je polynomální model, by s okolím tohoto kritického místa mely problém. Pro případy, kdy je na trase jen jeden takový bod by použití polynomiálních modelů vyhovovalo, byť by v bodě skoku odhad nebyl úplně přesný, ale předpokládejme, že nalezená polynomiální funkce by tento skok zohlednila. Ale teroreicky je potřeba algoritmus, který umí pracovat s více kritickými body na trase.

Nevyhovující průběh trasy se dvěma kritickými body x a y na trase odpovídá následně popsané modelové situaci jízdy vozidel mezi dvěma zastávkami. Uvažume, že se většina projíždějících spojů zdrží pouze v prvním kritickém bodě x o c sekund, nebo pouze v druhém kritickém bodě y o c sekund, nebo se lehce zdrží v obou kritických bodech o $c_y + c_x = c$ sekund⁸. S takovým zdržením je počítáno v jízdním řádu a tedy vozidla, která projedou první kritický bod bez zdržení jedou na čas stejně tak, jako vozidla v něm zdržená. O snížení nebo zvýšení případ-

⁸ c_x značí nabrané zpoždění v bodě x , pro y analogicky



Obrázek 2.7: Úsek s nepravidelnostmi

ného zpoždění spoje je možno rozhodnout až po projetí druhého bodu. Zatímco polynomiální model by svými odhady jen uváděl uživatele v omyl.

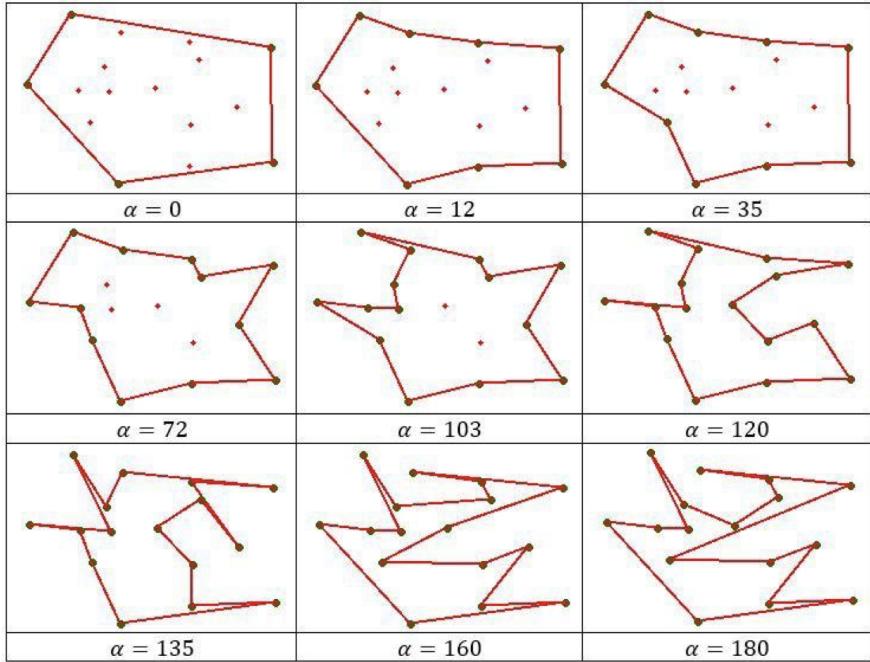
Jinými slovy na grafu času jízdý a vzdálenosti od vyjetí ze zastávky vzniká jakýsi podprostor v němž se zpoždění nemění. Pro ohraničení tohoto podprostoru je potřeba sestrojit konkávní obal všech vzorků všech spojů, které přijely do následující zastávky včas.

Nejprve k samotnému konkávnímu obalu je potřeba říct, že na množině bodů není definován jednoznačně, jak je vidět na obrázku 2.8⁹. Pro úshely této práce je zapotřebí spočítat obal ve třídimenzionálním prostoru, což je velmi komplikovaný úkol, a není ani snadné nalézt knihovny, které by konkávní obal ve 3D spočítaly. a proto je potřeba přijít s zjednodušením úlohy. Tedy počítat obal pouze pro dvoudimenzionální prostor. Toho se nedá dosáhnou jinak než diskretizací úlohy a počítání obalu pro každou hodinu zvlášt̄, tedy ze všech bodů, které byly zaznamenány v průběhu jedné hodiny.

Tím může dojít k větší granularitě obalu, než by bylo vhodné, nicméně předpokládá se, že hodina je dostatečně dlouhý časový interval na to, aby zde byly zachyceny všechny druhy průběhu jízdy a zároveň je to dostatečně krátký interval na nezkreslování denních výkyvů v čase jízdy.

Dále se jako netriviální ukazuje detekce spojů, které přijely včas a tedy všechny jejich body mají být předány k výpočtu obalu. Nabízí se použít data o všech spojí, které přijely do cílové zastávky s co nejmenším zpožděním, ale je nutné

⁹Zdroj: Alpha-Concave Hull, a Generalization of Convex Hull, Saeed Asaeedi, Farzad Didehvar, and Ali Mohades, Department of Mathematics and Computer Science, Amirkabir University of Technology, <https://arxiv.org/pdf/1309.7829.pdf>



Obrázek 2.8: Nejednoznačnost konkávního obalu

mít na paměti, že příjezdy podle jízdního řádu nemusí vůbec odpovídat realitě. Proto se zdá být nejlepší použít data od spojů, které přijely ve stejnou dobu jako je průměr všech příjezdů do cílové zastávky. Toho se docílí tak, že se poslední vzorky podle vzdálenosti všech spojů použijí pro odhad času příjezdu. Dále se pro každou hodinu použije se určité procento nejbližších spojů k tomuto odhadu.

Z předchozího popisu řešení vyplývá ovšem, že pro výpočet obalu jsou použity spoje, které ani zdaleka nemusely přijet včas jak je požadováno, ale předpokládá se, že se nepříliš vzdalují od průměrného času příjezdu. To že střední zpoždění pro celý obal není nulové se vyřeší spočtením odchylky průměrného příjezdu od příjezdu podle jízdního řádu a následně přičtení této konstanty k odhadnutnému zpoždění. Každopádně to, že rozptyl příjezdů spojů zahrnutých ve výpočtu obalu může být netriviální, vyžaduje nahlížet na tento obal jako na lineární prostor pohybu zpoždění. Tedy že odhad zpoždění pro bod nacházející se v obalu je lineárně závislý na vzdálenosti od hranice obalu, avšak protože je známo časové rozpětí příjezdu spojů použitých pro výpočet obalu, je možné tuto vzdálenost snadno přenést na skutečné pozdění.

Naštěstí pro nás se v průběhu analýzy dat o polohách spojů nepodařilo najít jediný případ dvojce zastávek, mezi kterýma by došlo k popsané situaci – výskytu dvou kritických bodů. Nebo tyto body jsou natolik nevýrazné, že by popsané řešení pomocí konkávního obalu nepřineslo žádné zlepšení odhadu zpoždění, ba naopak vzhledem k implementační náročnosti a množstvím chyb vznikajícím při tak algoritmicky náročných úkolech by přesnost odhadu zpoždění zhoršilo. Možných vysvětlení proč tato situace nenastává se nabízí více. Zejména vlivem složitosti dopravní sítě a závislostí v ní je možné, že pokud se vozidlo zdrží v jedné koloně vozidel a na jeho trase se ještě jeden kritický bod, pak je velmi pravděpodobné že se zdrží i v něm, protože hustota dopravy je ve stejném čase stejná na celé trase

vozidla. Tedy mohou nastat dvě situace: 1. vozidlo projede oba kritické body bez zdržení v časech s mírnou úrovní dopravy, 2. vozidlo se združí v obou kritických bodech stejně v časech s vysokou úrovní dopravy. Tyto situace jsou pokryty popisem profilu jízdy polynomiálním modelem. Další vysvětlení je, že popisované segmenty trasy (mezi zastávkami) jsou příliš krátké na to, aby se zde vyskytly 2 kritické body. Jiné vysvětlení může být, jistý druh práva přednosti v jízdě pro spoje VHD, čímž se myslí ovládání světelných křižovatek ve prospěch těchto vozidel, čímž se eliminuje dopad na zpoždění průjezdu křižovatkou.

TODO obrazek modelu

Návrh algoritmu pracující s konkávním obalem je následující. Nejprve ukažme konstrukce konkávního obalu pro dvojici zastávek a a b :

```
poslední_vzorky = vyber všechny vzorky poloh vozidel
    těsně před dojezdem do stanice $b$ od všech spojů;
odhad_příjezdu = odhadni čas příjezdu v průběhu celého
    dne podle bodů v poslní_vzorky, např.: pomocí poly regrese;
spoje_včas = prázdné pole spojů;

pro každou hodinu h:
    spoje_včas += vyber spoje, které přijely nejblíže
        odhadu v hodině h;

konkávní_obal = prázdné pole

pro každou hodinu h:
    vzorky_poloh = vyber všechny body zaznamenané
        v hodině h a náležící kterémukoli spoji v spoje_včas;
    konkávní obal += spočítej konkávní obal z vzorky_poloh;

Vrací: konkávní_obal;
```

Dále odhad zpoždění z konkávního obalu. Pro funkci `vzdálenost` bodu od hranice obalu se vždy myslí vzdálenost po kolmici na osu ujeté vzdálenosti a osu času dne.

Vstup: bod v prostoru vzdálenosti, průběhu dne
a času na trase (vzorek polohy vozidla)

```
pokud je bod v konkávní_obal:
    velikost_okna_příjezdu = rozdíl horní hranice obalu od spodní
        v zastávce příjezdu;
    spodek_okna = spodní hranice okna v čase
        příjezdu do zastávky;
    pomér = vzdálenost bodu od spodní hranice obalu
        ku vzdálenosti bodu od horní hranice obalu;
    odhad_příjezdu = velikost_okna * pomér + spodek_okna;
jinak:
    pokud je bod pod obalem:
        odhad_příjezdu = spodek_okna - vzdálenost bodu od obalu;
    jinak:
        vrch_okna = horní hranice obalu
```

v čase příjezdu do zastávky;
odhad_příjezdu = vrch_okna + vzdálenost bodu od obalu;

Vrací: odhad_příjezdu - pravidelný příjezd;

Modely neuronových sítí

Najdeme také funkci popisující průběh trasy pomocí neuronové sítě.

Ačkoli by řešení popsané výše mělo vyřešit problém řešený v této práci, bude zajímé pozorovat rozdíly odhadů zpoždění vozidel spočítaných popsaným polynomiálním model a odhadů spočítaných nauronovou sítí. Možná se také ukáže, že nejlepší řešení je právě kombinace těchto různých přístupů.

Vstupní data pro učení neuronové sítě budou vzorky poloh vozidel a spoždění zaznamenané v následující stanici.

2.4 Vizualizace dat

Aplikace vizualizace dat bude postavena z částí server a client. Tedy serverová strana se postará o přístup k otevřeným datům z databáze na základě požadavků klienta.

2.4.1 Funkční požadavky

Funkční požadavky vyplývají z rozboru existujících řešení v kapitole 1.4.2. Konkrétně z jejich uživatelské přívětivosti a užitečnosti zobrazovaných informací.

- Aplikace vykreslí interaktivní mapu Prahy, Středočeského kraje a širšího okolí obsluhovaného PIDem do webového rozhraní, kterou bude možné posouvat či zoomovat.
- V této mapě budou zobrazeny vozidla na aktuálních pozicích a budou se automaticky posouvat po mapě, tak jak se pohybují ve skutečnosti.
- Po kliknutí na vozidlo se zobrazí jeho celá trasa včetně zastávek a časů průjezdů a jeho dopočítaného zpoždění.
- Po kliknutí na zastávku se zobrazí seznam spojů, které budou projíždět vybranou zastávkou a jejich trasy se vykreslí do mapy.

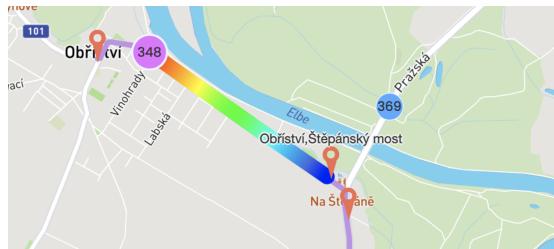
2.4.2 Kvalitativní požadavky

Front-end aplikace musí být schopný zobrazit v mapě řádově tisíce vozidel. Nebo je v případě velké hustoty vozidel na malém prostoru skrýt tak, aby nedošlo k matení uživatele.

Servovavá část včetně databáze bude schopná obsloužit jednotky dotazů za sekundu a to pouze pro demonstrační účely. Účelem práce tedy není budování rozbustního backe-endového řešení schoného produkčního nasazení.



Obrázek 2.9: Mapbox Mapa, použitý styl mapy: streets-v11



Obrázek 2.10: Design zobrazení elementů v mapě, linka je 348 je vybrána

2.4.3 Návrh grafiky a UI

Z výše popsaných funkčních požadavků budou vycházet grafické návrh uživatelského rozhraní. Předem je potřeba zdůraznit, že návrh do velkého detailu grafických a estetických vlastností této aplikace není předmětem této práce.

Externí mapové podklady mohou být zobrazeny v několika barevných variacích, umožn—ují zobrazení ortofoto a také zobrazení z důrazem na různé mapové vrstvy. Pro nás je nejzádoucí zobrazit vrstvu ulic a cest a dále pak určité orientační body jako jsou budovy, vodní plochy, lesy atp. Z palety barev je pak dobrou volbou neutrální béžová barva. Příklad takové mapy je na obrázku ??.

Zobrazení jednotlivých spojů bude pomocí bodů v mapě a to konkrétně barevným kruhem s číslem linky, vybrané vozidla se rozliší jinou barvou a velikostí kruhu. Návrh reprezentace vozidel v mapě je zobrazen na obrázku ??.

Pro zvolené vozidlo se vykreslí celá jeho trasa včetně všech zastávek. Trasa je reprezentována lomenou čarou a zastávky jako špendlíky v mapě, po přejetí symbolu zastávky se zobrazí i její název. Za zvoleným vozidlem je zobrazena i historie jeho jízdy za uplynulých několik minut, to pomocí barevné lomené čáry, tvořící ocas zvoleného vozidla. Rovněž k povšimnutí na obrázku ??.

Další infomace o spoji, nebo zastávce se zobrazí v tabulce, která z části překryje mapu. Tato tabulka bude obsahovat informace o zvoleném vozidle, konkrétně konečnou stanicí, jeho zpozdění a celý jízdní řád. Respektivně infomace o zvolené zastávce a to název zastávky a všechny spoje, které zvolenou zastávkou budou projíždět včetně jejich pravidelného odjezdu a aktuálního zpozdění. Po kliknutí na zastávku se tedy zobrazí její odjedová tabule.

3. Implementace

V této kapitole je detailně popsána implementace a volba technologií použití k implementaci navrženého díla.

3.1 Úvod

Nejdůležitější částí celého systému je modul výpočtu a používání pravděpodobnostních modelů odhadu zpoždění vozidel. To vytváří požadavek na využití technologií, které poskytují prostředí pro pohodlnou tvorbu těchto modelů. V současnosti jsou nejpokročilejší nástroje pro takový účel součástí balíčkové sady jakyka Python3, konkrétně se jedná o knihovnu scikit-learn¹ a další nástroje pro páci s velkými daty jako je knihovna NumPy². Tyto knihovny implemntují dobře známé algoritmy umělé inteligence, včetně optimalizací a pomocných funkcí zjednodušujících hledání nejlepšího modelu. Navíc jsou tyto knihovny naimplementovány s ohledem na vysokou výkonost³ a využití grafických karet⁴. Proto nedává smysl jejich služeb nevyužít. Využití jazyka Python3 je tedy pro jádro náší práce jasnou volbou.

Pro samotné zpracování dat žádné speciální požadavky nevyvstávají a je možné využít i jiné ověřené backendové programovací jazyky a technologie. Nicméně pro zachování jednoty vývoje není důvod měnit prostředí a využijeme též jazyk Python3. Může být namítnuto, že programy psané v jazyce Python3 nejsou výkonnostně příliš dobré, nicméně v našem případě se nebudou provádět žádné složité výpočty, ale pouze stahování dat z internetu a jejich transformace. Byť se jedná o poměrně velké objemy dat jakákoli operace s nimi je stále řádově rychlejší než stahování z internetu.

Databázi jsem již vybrali MySQL⁵ implementaci. To zejména z důvodů, že se jedná o open source projekt a tato implementace je velmi často vyžívaná v celé řadě jiných projektů s velkou komunitou. Konkrétně pro Python3 existuje knihovna MySQL Connector⁶ přes kterou je možné SQL databázi pohodlně obsluhovat.

Celý systém je ovládaný přes hlavní skript v souboru `download_and_process.py`. Tento skript slouží jak ke spuštění produkčního běhu aplikace tak i k údržbě dat, správným nastavení parametrů se vybere zdroj dat, kde je navýběr mezi demonstračními daty, vývojovými daty nebo real-time dady. Dále je možné spuštěním skriptu vytvořit modelu profilů jízd a táké odstranit historická data z databáze podle jejich data vložení.

¹<https://scikit-learn.org/stable/>

²<https://numpy.org>

³<https://scikit-learn.org/stable/developers/performance.html>

⁴Záleží na konkrétním hardwaru

⁵<https://www.mysql.com>

⁶<https://dev.mysql.com/doc/connector-python/en/>

Veškerý software bude naimplementován s ohledem na paradigmá PID. Tedy logické celky budeme dělit do tříd, jejichž instance budou reprezentovat vždy danou entitu. Každá třída bude implementovat sadu metod, odpovídající logice věci.

3.2 Zpracování dat

Základní myšlenka zpracování dat pocházejících ze zdroje dat popsáném v kapitole ?? je taková, že data se budou periodicky stahovat a ukládat do SQL databáze, tento postup je popsán v kapitole 2.2.

Jako součást projektu naimplementujeme pro přehlednost pomocné třídy celkově usnadňující využití zdrojů a technologií pro náš specifický účel. Dále pak z důvodu oddělení technických záležitostí, jako je např.: stahování dat ze sítě, tak aby nezasahovali do ko'du implementující logiku systému. Stejně tak se tímto eliminuje výskyt paternů v celém ko'du. Konkrétně se tím myslí komunikace s databází, komunikace se zdrojem dat, komunikace se souborovým systémem. Tyto třídy navíc definují důležité konstanty, jakými jsou např.: jména využívaných souborů nebo souborových adresářů, URL zdroje dat atp.

Hlavní smyčka ve které se stahují a zpracovávají data volá následující funkce.

```
# stažení aktuálních poloh vozidel
all_vehicle_positions.get_all_vehicle_positions_json()

# konstrukce interní reprezentace vozidel
all_vehicle_positions.construct_all_trips(database_connection)

# odhadnutí spoždění všech vozidel
estimate_delays(all_vehicle_positions, models)

# kompletace dat a uložení do databáze
asyncio.run(process_async_vehicles(all_vehicle_positions, database_connection, args))
```

3.2.1 Konstrukce objektů vozidel

Funkce `construct_all_trips` vytvoří z každého nalezeného vozidla ve vstupním JSON souboru instanci třídy `Trip`, která je interní reprezentací těchto vozidel.

Avšak ke konstrukci instancí vozidel je potřeba získat data z databáze o jízdách řádech. To potože součástí vstupního souboru z externího zdroje není informace o poslední projeté a další následující zastávce. Tuto informaci potřebujeme pro odhad zpoždění provádějící se v dalším kroku⁷. Proto se na začátku konstrukce instancí třídy `Trip` čtou data z tabulky `rides` pouze pro aktuálně zpracovávané jízdy, a dále se pak sadou funkcí hledá poslední projedoucí zastávka na trase každého spoje.

⁷Ve verzi 2 datového formátu souboru poloh vozidel je již tato informace zahrnuta

3.2.2 Odhad zpoždění

Tato funkce odhadu zpoždění musí být volána ještě před uložením do databáze, aby se do databáze vložily data včetně odhadu zpoždění. To ovšem zapříčiní, že pokud je vozidlo dosud nenalezeno neznáme ani jeho jízdní řád a tedy nemůžeme pro něj odhadnou zpoždění. To ovšem nehraje velkou roli, protože se tak stane pro každé vozidlo ihned po vyjetí z výchozí stanice, nebo ještě pře vyjetím, v těchto případech nemá počítání odhadu zpoždění velký význam. V další iteraci již jízdní řády spoje budou známé, tedy chybějící zpoždění doplníme velmi rychle.

Funkce odhad zpoždění využívá zkonstruovaných modelů profilů jízd, pokud takový model zatím není vytvořen použije se zpoždění v poslední projeté zastávce. Modely jsou uloženy v souborovém systému a pro každou dvojici zastávek je model uložen zvlášt^ˇ.

Pro rychlejší běh aplikace jsou však po prvním načtení modely drženy v paměti počítače v proměnné typu mapa, to nám pak umožní rychlé vyhledávání podle dvojice identifikátorů zastávek. Implementace funkce je následující, tento ko'd je volán pro každé vozidlo zvlášt^ˇ.

```
# najde model podle dvojce zastávek a dnů v týdnu
model = models.get(
    str(vehicle.last_stop or '') + "_" +
    str(vehicle.next_stop or '') +
    ("_bss" if lib.is_business_day(vehicle.last_updated) else "_hol"),
    Two_stops_model.Linear_model(vehicle.stop_dist_diff))

# vybere data potřebná k výpočtu odhadu zpoždění
tuple_for_predict = vehicle.get_tuple_for_predict()

# odhadne zpoždění
# jinak se při vkládání do databáze použije zpoždění z poslední zastávky
if tuple_for_predict is not None:
    vehicle.cur_delay = model.predict(*tuple_for_predict)
```

3.2.3 Kompletace dat a jejich uložení

Kompletace dat především obnáší stažení dalších dat, jako jsou jízdní řády v případě, že jízda doposud nebyla nalezena. Takových jízd může být v jedné iteraci běžící aplikace i několik desítek, při spuštění systému jsou všechny jízdy nenalezeny a tedy musí být staženy dodatečná data i pro několik stovek jízd. Aby se data o každé jízdě nestahovala sériově metoda zpracování vozidel je implementována asynchroně, resp. stahování dat je asynchronní. Díky tomu se začnou stahovat data o více jízdách v jeden okamžik. Byť čekání na stažení dat o jedné jízdě je při dobrém internetovém spojení otázkou několika desítek milisekund, tak v případech stahování dat o stovkách jízd sériově by jenom stahování dat prodloužilo běh jedné iterace o jednotky až nízké desítky sekund.

Jak tedy vyplývá z textu výše tato funkce dělí běh na dvě části podle toho jestli je jízda vozidla nalezena nebo nenalezena. V případě nalezené jízdy v databázi se jen aktualizuje záznam v tabulce `trips`, mezi aktualizovaná data patří aktuální

zpoždění, zpoždění v poslení pojeté zastávce, ujetá vzdálenost, souřadnice vozidla a časová známka aktualizace dat ve zdroji dat. Zároveň s aktualizací dat v tabulce `trips` se provádí i vložení nového záznamu do tabulky `trip_coordinates`, která slouží jako datový sklad všech zaznamenaných poloh vozidel. Celá logika aktualizace je řešena v SQL funkci.

```

CREATE DEFINER='root'@'localhost' FUNCTION
    'update_trip_and_insert_coordinates_if_changed'(
        trip_source_id_to_insert VARCHAR(31),
        current_delay_to_insert INT(32),
        last_stop_delay_to_insert INT(32),
        shape_dist_traveled_to_insert INT(32),
        lat_to_insert DECIMAL(9,6),
        lon_to_insert DECIMAL(9,6),
        last_updated_to_insert DATETIME) RETURNS int(1)
        DETERMINISTIC
BEGIN
    SELECT last_updated, id_trip
    INTO @last_updated, @id_trip
    FROM trips
    WHERE trips.trip_source_id = trip_source_id_to_insert
    LIMIT 1;

    IF @last_updated <> last_updated_to_insert THEN
        INSERT INTO trip_coordinates (
            id_trip,
            lat,
            lon,
            inserted,
            delay,
            shape_dist_traveled,
            last_stop_delay)
        VALUES (
            @id_trip,
            lat_to_insert,
            lon_to_insert,
            last_updated_to_insert,
            current_delay_to_insert,
            shape_dist_traveled_to_insert,
            last_stop_delay_to_insert);

        UPDATE trips
        SET trips.last_updated = last_updated_to_insert,
            trips.current_delay = current_delay_to_insert,
            trips.shape_dist_traveled = shape_dist_traveled_to_insert,
            trips.lat = lat_to_insert,
            trips.lon = lon_to_insert
        WHERE trips.id_trip = @id_trip;
        RETURN 1;
    ELSE
        RETURN 0;
    END IF;
END;

```

```
END IF;  
END
```

3.3 Konstrukce modelů

Pro spočítání polynomiálního modelu se využívá knihovna sklearn konkrétně algoritmus zvaný Rigde, který sám o sobě hledá linární závislosti, nicméně vstupní hodnoty jsou mezi sebou náležitě pronásobeny tak, aby simulovali polynomiální funkci. Toho se dosáhne pomocí funkce PolynomialFeatures. Optimální stupeň polynomu se zjistí spočítáním modelu pro každý stupeň v rozumných mezích a nakonec se zvolí ten s nejmenší chybou. To se v jazyce Python3 za pomocí knihovny sklearn provede následujícím ko'dem. Omezení stupn̄ů polynomiální regrese vyplývá ze skušenosti, kdy s nejmenší chybou jsou modely stupn̄ 5 nebo 6.

```
for degree in [3, 4, 5, 6, 7, 8, 9, 10]:  
    model = make_pipeline(PolynomialFeatures(degree), Ridge(), verbose=0)  
    model.fit(X_train, y_train)  
    pred = model.predict(X_test)  
    error = mean_squared_error(y_test, pred)  
    if error < best_error:  
        best_degree = degree  
        best_error = error  
  
self.model = make_pipeline(PolynomialFeatures(best_degree), Ridge())  
self.model.fit(input_data, output_data)
```

Samotné nalezení správného modelu se nyní může zdát jednoduché, nicméně nejsložitější prací pro jakoukoli úlohy z oblasti umělé inteligence a strojového učení je příprava dat a v naší práci tomu není jinak. At̄ už se jedná o samotná zpracování dat popsané výše v kapitole 3.2, tak také je potřeba tyto zpracovaná data dále trasnformovat z formátu v jakém jsou uloženy v databázi do formátu jaký je vhodný pro počítání lineární regrese. Dále je pak potřeba data vyčistit od zcela nesmyslných vzorků poloh vozidel, kterých je vstupních datech spousta a mohly by negativně ovlivnit správnost odhadů nalezených modeů.

3.3.1 Čtení dat

Pro zkonztruování modelů popisujících profily jízd mezi vsemi dvojcemi zastávek je nejprve potřeba zjistit všechny dvojce zastávek, mezi kterými jede alespoň jeden spoj. To se dá zjistit pomocí jízdních řádů, které reprezentujeme v tabulce `rides` v naší databázi popsané v kapitole 2.2.1

Dále pokud máme všechny dvojce zastávek je potřeba získat všechny označené polohy vozidel mezi nimi pro každou dvojici zastávek zvlášt̄. To je možné realizovat pomocí následujícího SQL dotazu.

```
SELECT schedule.id_trip,  
       schedule.id_stop,
```

```

    schedule.lead_stop,
    departure_time,
    schedule.lead_stop_departure_time,
    (schedule.lead_stop_shape_dist_traveled - schedule.shape_dist_traveled)
        AS diff_shape_trav,
    trip_coordinates.inserted,
    (trip_coordinates.shape_dist_traveled - schedule.shape_dist_traveled)
        AS shifted_shape_trav,
    trip_coordinates.delay
FROM (
    SELECT id_trip, id_stop, shape_dist_traveled, departure_time,
        LEAD(id_stop, 1) OVER (PARTITION BY
            id_trip ORDER BY shape_dist_traveled) lead_stop,
        LEAD(shape_dist_traveled, 1) OVER (PARTITION BY
            id_trip ORDER BY shape_dist_traveled) lead_stop_shape_dist_traveled,
        LEAD(departure_time, 1) OVER (PARTITION BY
            id_trip ORDER BY shape_dist_traveled) lead_stop_departure_time
    FROM rides) AS schedule
JOIN trip_coordinates
ON trip_coordinates.id_trip = schedule.id_trip AND
    schedule.lead_stop_shape_dist_traveled -
        schedule.shape_dist_traveled > 1500 AND
    trip_coordinates.shape_dist_traveled
        BETWEEN schedule.shape_dist_traveled AND
    schedule.lead_stop_shape_dist_traveled
ORDER BY id_stop, lead_stop, shifted_shape_trav

```

Tento SQL dotaz nejprve získá všechny dvojce po sobě jdoucích zastávek z jízdních řádů v tabulce `rides`. Protože v tabulce je jízda spoje uložená jako sekvence zastávek, kde každé náleží čas příjezdu resp. odjezdu a její vzdálenost na trase spoje od výchozí stanice spoje (atribut `shape_dist_traveled`). Tedy dvojce zastávek po sobě následující se získají tak, že se seřadí všechny zastávky pro každý spoj podle atributu `shape_dist_traveled`. Následující zastávka pak je ta ležící na následujícím řádku v seřazené tabulce, tento řádek se přečte pomocí funkce `LEAD`.

K těmto dvojcím zastávek dále získáme všechny vzorky poloh vozidel. To tak, že vezme všechny vzorky pro daný spoj, které leží mezi vybranou dvojcí zastávek. V tomto dotazu zároveň vyloučíme zastávky, které jsou k sobě blíže než nebo vzdálené přesně 1500 m (v implementaci je pak vzdálenost určena parametricky), zdůvodnění této vzdálenosti je výše v návrhu modelů. Pro produkční nasazení je ještě potřeba omezit čtené vzorky podle času vytvoření, tedy např. nevyužívat vzorky starší několika dní, jak je popsáno v analýze problému. Nicméně toto omezení by vycházelo z reálného provozu ze skušeností jak rychle se vývíjí dopravní síť nebo jak často se mění jízdní řády.

Takto jak je SQL dotaz napsán, je jeho provedení velmi časově náročné. To nám, ale nemusí vadit protože dotaz bude volán pouze před výpočtem modelů, což je samo o sobě mnohem časově náročnější operace a navíc tato operace bude spouštěna tak, aby nepřetěžovala kapacitu stroje. Pokud by se ukázalo, že dotaz vybírá z databáze příliš velké množství dat, s kterými se poté těžce manipuluje v

paměti počítače, je možné doplnit stránkování výběru dvojic zastávek klíčovým slovem s parametry `LIMIT offset, limit`.

3.3.2 Příprava dat

Přečtená data z databáze se dále třídí podle dne v týdnu, ve kterém byla zaznamenána a pak pro každou sadu dat je vytvořena instance třídy `Two_stops_model`. Do této třídy se pak ukládají přečtená data.

Dále následuje čištění dat od chyb a jejich odstranění. Čištění funguje tak, že pokud nějaký vzorek dat je významně mimo kluster všech ostatních, tak není odstraněn pouze tento jeden vzorek, ale rovnou všechny vzorky daného spoje. To proto, že s vysokou pravděpodobností jsou ovlivněny chybou i ostatní vzorky, ale nesplňují poměrně volné kritéria na odstranění. Hledání chyb pak probíhá tak, že se spočítá poměr čas jízdy ku vzdálenosti všech vozorků a za chybné se označí ty příliš vzdálené od průměru. Popsaný algoritmus je implementován takto.

```
trips_to_remove = set()
trip_times_to_remove = dict()
coor_times = self.norm_data.get_coor_times()
norm_shapes = np.divide(self.norm_data.get_shapes(), 100)

rate = np.divide(coor_times, norm_shapes, where=norm_shapes!=0,) != np.array(None)

high_variance = np.where((
    abs(rate - rate.mean()) > rate.std() * Two_stops_model.REDUCE_VARIANCE_RATE
).astype(int) == 1)[0]

# for all indicated indices get trip ids
# and create dictionary of trip ids of all corrupted samples
for hv in high_variance:
    trip_id = self.norm_data.get_ids_trip()[hv]
    trips_to_remove.add(trip_id)

    if trip_id in trip_times_to_remove:
        trip_times_to_remove[trip_id].append(self.norm_data.get_timestamps()[hv])

    else:
        trip_times_to_remove[trip_id] = [self.norm_data.get_timestamps()[hv]]

self.norm_data.remove_items_by_id_trip(trips_to_remove, trip_times_to_remove)
```

Po vykonání této procedury jsou data připravena jako vstupní data pro výpočet modelu profilu jízdy podle algoritmu uvedeném výše.

3.3.3 Práce s modely

Pro vložení odhadnutého zpoždění do databáze je potřeba využít předvypočítané modely pro jeho odhad.

Tento odhad se počítá pro každé vozidlo zvlášť na základě vstupních dat o aktuální poloze vozidla obohacené o další informace. Tento vstupní vektor se konstruuje ve třídě `Trip` následovně.

```

self.shape_traveled - self.last_stop_shape_dist_trav,
lib.time_to_sec(self.last_updated),
self.departure_time.seconds,
self.arrival_time.seconds

```

První položka je aktuální vzdálenost vozidla od poslední projeté zastávky, dále je uveden čas zaznamenání polohy vozidla, třetí položkou je čas pravidelného odjezdu z poslední projeté zastávky a dále příjezdu do následující zastávky. Pro poslední dvě položky je nutné přečíst jízdní řád pro daný spoj z databáze.

Po sestrojení vstupního vektoru je vložen jako vstupní parametr funkce pro předpověď odhadu zpoždění, kterou má každá instance modelu. Podle typu modelu se pro odhad zpoždění využije lineární nebo polynomiální model. V obou případech je potřeba pouze ošetřit případ, kdy vozidlo jede přes půlnoc a rozdíl tedy rozdíl času příjezdu a odjezdu by vyšel záporně.

V případě, že se využívá polynomiální model pro odhad zpoždění, je navíc ošetřena situace, kdy čas zaznamenání polohy vozidla je mimo rozsah modelu, v případě, že tomu tak je, je použita nejbližší hraniční hodnota. K něčemu takovému by docházet nemělo nebo případné posunutí času nebude mít velký vliv, protože nejpozdější, resp. nejdřívější čas průjezdu mezi dvěma zastávkami se v realitě často nemění vůbec nebo nijak výrazně. Ošetření této situace se však míjí účinkem a to, protože se nejdřívější, resp. nejpozdější průjezd počítá vždy od půlnoci⁸. Pro eliminaci chyby odhadu zpoždění polynomiálním modelem z důvodu, že model profiluje příjezd do následující zastávky v jiném čase než je pravidelný čas příjezdu, je navíc ještě zjištěn skutečný čas na jízdy tím, že se zjistí odhadovaná hodnota v čase a vzdálenost následující zastávky. Tato hodnota se pak přičte k odhadnutému zpoždění.

3.4 Vizualizace dat

Data budou zobrazovány pomocí webové aplikace (klientská část) a ta bude stahovat data ze serverové části. Komunikační mapa ilustrující propojení těchto částí je zobrazena na diagramu 2.1.

3.4.1 Klientská část

Webová aplikace bude napsána pomocí jazyků a nástrojů vhodných pro vývoj webových aplikací. Používáme tedy značkovací jazyk HTML pro strukturu samotné webové stránky, pro stylování objektů je použit jazyk CSS. Hlavní vlastnosti stránky, jako je zobrazení entit do mapy je použitý jazyk JS, zejména pak jeho možností pro zacházení s DOM elementy. Pro připojení a načítání dat ze serveru se používá technologie AJAXových dotazů.

Koncepce klientské aplikace je taková, že žádná data nezpracovává ani nepře-počítává a zobrazuje jen data taková, která obdržela od serverové strany typicky ve formátu GEOJSON. Pro aktualizi dat je potřeba vyvolat nový dotaz, typicky se stejnými parametry.

⁸v této práci se vždy využívá časová zo'na UTC

Webová aplikace bude v pravidelných intervalech aktualizovat obraz všech vozidel. Dále pak bude reagovat na uživatelské vstupy v podobě klikání na vybrané elementy. Ty potom vykreslí do mapy odlišeně nebo stáhne přídavná data k zobrazení.

Mapbox API

Nejprve si popišme jaké funkce budeme využívat z knihovny Mapbox.

Prostředí Mapbox je široce využívaný multiplatformový nástroj pro zobrazení mapového podkladu a umožňuje do něj zanést širokou škálu různých geometrických útvarů. Takže mapové prostředí intuitivně interahuje s uživatelem a vývojáři mohou využít jednoduchého API pro zobrazení žádoucích dat do mapy.

Webová aplikace této práce využívá naprostoto základní funkcionality, které mapbox přináší. Popis jejich využití včetně načtení prostředí Mapboxu do webové stránky za předpokladu, že jsou splněny základní HTML požadavky webové stránky je následující.

Rozhraní se do webové stránky importuje pomocí:

```
<script src='https://api.tiles.mapbox.com/
    mapbox-gl-js/v1.4.0/mapbox-gl.js'></script>
<link href='https://api.tiles.mapbox.com/
    mapbox-gl-js/v1.4.0/mapbox-gl.css' rel='stylesheet' />
```

Dále je potřeba vytvořit element s identifikátorem webové stránky, kde bude mapa zobrazena.

Po naimportování je v JavaScriptu k dispozici knihovna jménem `mapboxgl` pomocí, které se ovládá celé mapové prostředí. Tedy je teď možné vytvořit samotnou mapu.

```
var map = new mapboxgl.Map({
  container: 'map', // identifikátor HTML elementu
  style: 'mapbox://styles/mapbox/streets-v11',
  center: [14.42, 50.08], // střed mapy při inicializaci [lng, lat]
  zoom: 10 // zoom při inicializaci
});
```

Nyní stačí jen vytvořit HTML element za pomocí JS a po té může být přidán do mapy následující funkcí. Nyní se nám již takový element zobrazuje v mapě na zvolených souřadnicích.

```
new mapboxgl.Marker(element)
  .setLngLat([Lng, Lat]) // zeměpisná výška a šířka
  .addTo(map);
```

Pro vykreslení složitějších objektů, jako je třeba lomená čára se využívá funkce `addLayer`. Tato funkce přijímá data ve formátu GEOJSON tedy není třeba dělat žádnou transformaci dat.

```

map.addLayer({
  "id": id, // identifikátor vrstvy
  "type": "line", // geometrický útvar k zobrazení
  "source": {
    "type": "geojson", // formát zdrojových dat
    "data": data // zdroj dat
  },
  "paint": {
    "line-color": "#BF93E4", // barva
    "line-width": 5 // šířka
  }
});

```

K manipulaci s objekty typu **Layer** se používají následující funkce.

```

map.getLayer(id);
map.removeLayer(id);

```

To je vše co potřebujeme k naplnění cíle vizualizace dat. Autobus na mapě budeme reprezentovat kolečkem s číslem spoje a zastávku jako špendlík, toto jsou HTML elementy. Lomené čáry trasy spoje vykreslíme jako vrstvu funkcí `addLayer`.

Běh aplikace

Se serverovou částí se komunikuje pomocí GET requestů a server vrací JSONové soubory. Webová aplikace používá knihovnu na parsování tohoto formátu a tedy můžeme se k nim chovat jako k mapám.

Po inicializaci prostředí Mapboxu popsanou výše následuje inicializace naší aplikace. Především se pak spustí smyčka aktualizující aktuální polohy vozidel.

```

var vehicles = new Set(); // elementy vozidel v mape
var active_trips = {}; // vybraná vozidla
var vehicles_elements = {}; // html elementy vozidel
var no_stop_chosen = true; // indikátor vybrání zastávky

// inicializační stažení poloh vozidel
getFileByAJAXreq("vehicles_positions", showBusesOnMap);

// hlavní smyčka
window.setInterval(function(){
  getFileByAJAXreq("vehicles_positions", showBusesOnMap);

  // aktualizace ocasu všech vybraný vozidel
  for (var trip in active_trips){
    active_trips[trip].update_tail();
  }
}, 10000);

```

Po načtení poloh vozidel probíhá jejich vykreslování do mapy. Nejprve se odstraní z mapy všechny staré vozidla a pro každé nové vozidlo se konstruuje nový HTML element. Každý tento element reprezentující vozidlo poslouchá na kliknutí.

Tedy pokud vozidlo není vybráno vytvoří se nový element a následně se vykreslí do mapy a zároveň se stáhnou další informace o vozidle, které se též následně zobrazují. Jsou jimi trasa vozidla, jízdní řád a zpoždění. Vybrané vozidlo se v ko'du reprezentuje vlastní třídou `Active_trip`, každá instace této třídy se přidá do promněné `active_trips`. Tato třída pak obsahuje metody obstarávající zobrazení dalších informací, stejně tak jejich odstranění nebo aktualizaci.

Pokud vozidlo již bylo vybráno jednoduše se odstraní z množiny vybraných vozidel `active_trips` a z mapy.

V případě, že je nějaké vozidlo vybráno, zobrazuje se i jeho zastávky. Každá zobrazená zastávka reaguje na kliknutí a na přejetí myši. Po kliknutí se vyberou všechny spoje projíždějící zastávkou a po přejetí myši se zobrazí název zastávky. Tyto funkcionality se HTML elementu přiřadí následujícím ko'dem.

```
// zobrazí všechny spoje projíždějící zastávkou
el_c.addEventListener('click', function() {
    no_stop_chosen = false;
    show_trips_by_stop(getFileByAJAXreqNoCallback(
        "trips_by_stop." + marker.name
    ), marker.name);
});

// přidá do mapy název zastávky po najetí myši
el_c.addEventListener("mouseover", function(){
    var el_s = document.createElement('div');
    el_s.innerText = marker.name;
    el_s.setAttribute("class", "stop_pin_sign");
    new mapboxgl.Marker(el_s)
        .setLngLat(marker.geometry.coordinates)
        .addTo(map);
});

// odebere všechny názvy zastávek z mapy po vyjetí myši
el_c.addEventListener("mouseout", function(){
    var signs = document.getElementsByClassName('stop_pin_sign');
    while(signs[0]) {
        signs[0].parentNode.removeChild(signs[0]);
    }
});
```

Vybraná vozidla podle zastávky se pak chovají stejně jako když je vybrané pouze jedno vozidlo. Tedy do promněné `active_trips` se vloží více vozidel.

3.4.2 Serverová část

Tak jak je řečeno příchozí požadavky od klineta jsou odpovídáný skriptem na serverové straně. Který je napojený na databázi a z ní extrahuje potřebná data.

Data jsou posílána v textové podobě ve formátu `GEOJSON`, které skrip konstruuje z dat získaných z databáze.

Server reaguje na 4 typy požadavků:

- `get_vehicle_positions` vrátí aktuální polohy všech vozidel,
- `get_tail.id_trip` vrátí lomenou čáru popisující pohyb vozidla v uplynulých n minutách, vozidla podle identifikátoru jízdy,
- `get_shape.id_trip` vrátí lomenou čáru popisující trasu spoje podle id spoje, vozidla podle identifikátoru jízdy,
- `get_stops.id_trip` vrátí seznam zastávek pro spoj podle jeho id, vozidla podle identifikátoru jízdy.

Celý server je stejně jako jádro systému naprogramováno v jazyce Python3.

Server knihovna

Server je naprogramován pomocí Pythonové knihovny `simple_server`, která slouží pouze k debugování, jak se píše v její dokumentaci⁹. Protože se nepočítá s reálným nasezením této aplikace, není potřeba programovat robustní server. Pro demonstrační účely je však toto řešení dostatečné.

Vytvoření serveru pomocí této knihovny se v jazyce Python3 udělá následovně.

```
httpd = make_server("", self.PORT, self.server)
thread = threading.Thread(target=httpd.serve_forever)
thread.start()
```

Kde `server` je funkce, která je vždy volána když server obdrží dotaz. Upozorněme, že tento způsob startu serveru je odlišný od popisu v dokumentaci.

Volaná funkce `server` je kompletní glsWSGI aplikace, jež příjmá argumenty `environ`, což je dotaz a atribut `startresponse`, který reprezentuje hlavičku odpovědi. Dále se v této funkci nachází veškerá logika serveru, tedy reaguje na parametry dotazu.

Zpracování dotazu funguje pro všechny kombinace parametrů dotazu podobně. Vždy se data čtou z databáze a transformují se do formátu /glsgeojson. Uvedeme si na příkladu jak probíhá zpracování dotazu na zastávky daného spoje. Po té co obdržíme dotaz se z něj přečtemy parametry. Pro dotaz na zastávky musí být parametr ve formátu `get_stops.id_trip`. Parsování dotazu a volání příslušní interní funkce se provádí následovně.

```
elif "stops" == request_body.split('.')[0]:
    response_body = json.dumps(self.get_stops(
        request_body[request_body.index('.')+1:]))


```

Funkce `get_stops` přijímá identifikátor jízdy jako parametr a podle něj čte data z databáze pomocí SQL dotazu. Po přečtení dat vytváří mapu, která se snadno převede na řetězec ve formátu GEOJSON. Tělo funkce tedy vypadá takto:

⁹https://docs.python.org/3/library/wsgiref.html#module-wsgiref.simple_server

```

stops = self.database_connection.execute_fetchall("""
SELECT
    stops.lon,
    stops.lat,
    rides.departure_time,
    stops.stop_name
FROM rides
INNER JOIN stops ON rides.id_stop = stops.id_stop
WHERE rides.id_trip = %s
ORDER BY rides.shape_dist_traveled"",
(id_trip,)
)

stops_geojson = []
stops_geojson["type"] = "FeatureCollection"
stops_geojson["features"] = []

for stop in stops:
    stops_geojson["features"].append({
        "name": stop[3],
        "departure_time": stop[2].total_seconds(),
        "geometry": {
            "coordinates": [float(stop[0]), float(stop[1])]
        }
    })

return stops_geojson

```

4. Tesstování a evaluace

V této kapitole je popsáno jak je celá aplikace otestována. Dále pak porovnání odhadů zpoždění se stávajícím řešením.

4.1 Testování softwarového řešení

Ko'd práce popsaný v kapitole 3 je otestován unit testy. Propojení tohoto softwaru s databází i zdrojem vstupních dat je testováno integračními testy.

4.1.1 Unit testy

Unit testy testují spravnou funkčnost jednotlivých metod všech softwarových komponent této práce.

Pro ověření správné funkčnosti některých metod jsou vygenerována vstupní či výstupní data. To z důvodu, že tyto metody pracují z komplexní datovou strukturou nebo s velkým objemem dat, který není možno zadat jako vstupní přímo v ko'du testu, resp. je potřeba porovnat výstup tetované metody a ze stejných důvodů není možné uvádět výstupní hodnoty pro porovnání přímo v ko'du testu. Typickým příkladem takové vstupní struktury je model profilu jízdy, ptotož je potřeba otestovat funkce, které s takovým modelem pracují.

4.1.2 Integrační testy

TODO Popisovat co testují testy? Není v tom žádná složitá logika. Dále otázka k celému textu jak odkazovat na jednotlivé soubory s kodem?

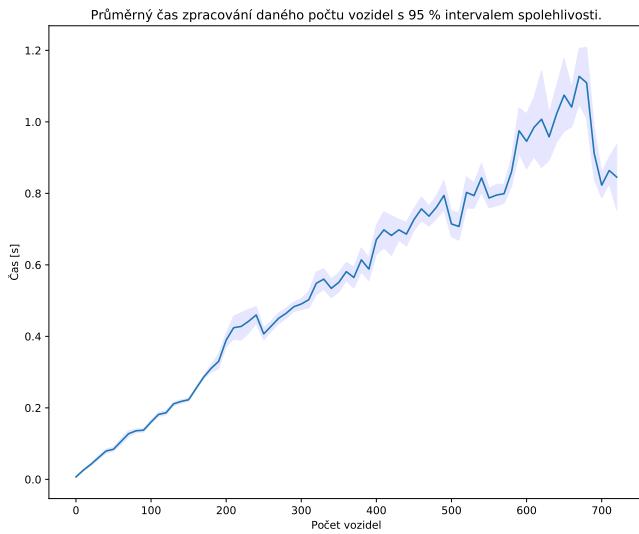
4.1.3 Výkonostní testy

Všechny následující výkonostní testy jsou prováděny na osobním notebooku s technickými parametry uvedenými v tabulce 4.1.3¹, kde všechny procesy aplikace včetně databáze běží paralelně.

Parametr	Hodnota
Procesor	4x Intel(R) Core(TM) i7 CPU @ 2.70 GHz
Paměť	16 GB DDR3 RAM
Rychlosť zápisu na disk	1000–3000 MB/Sec
OS	macOS Big Sur
MySQL	version 8.0.18

¹

¹<https://9to5mac.com/2016/11/01/the-late-2016-entry-level-13-macbook-pro-has-a-ridiculously-fast-ssd/>



Obrázek 4.1: Průměrný čas zpracovávání daného počtu vozidel ze všech souborů se statickými daty. Světle modrá barva ohraničuje 95 % interval spolehlivosti. Počty vozidel jsou vždy zaokrouhleny dolů na celé desítky.

Pro potlačení zkreslení testů vlivem čekání na stažení dat z internetu jsou všechny data načítána z disku počítače.

Testování stejně jako funkční a kvalitativní požadavky na práci vychází z analýzy vstupních dat uvedené v kapitole 1.3.2.

Zpracování dat

Zpracování dat probíhá přečtením souboru s polohy vozidel a dále zpracovává každé vozidlo zvlášť. Přičemž pokud je vozidlo již nalezeno a jeho poloha se od poslední aktualizace změnila provedou se pouze 2 čtení z databáze, jeden záznam se aktualizuje a vloží se jeden nový záznam. Pokud je vozidlo již nalezeno a jeho poloha se od poslední aktualizace nezměnila provedese se pouze jedno čtení z databáze. Pokud ovšem je vozidlo obsluhující spoj nenalezeno musí se číst soubor s detailním daného spoje a všechna data se vkládají do databáze (jízdní řád včetně zastávek) navíc geografická lomená čára popisující jízdu se ukládá jako soubor.

Jak je ale vidět na grafu 4.1 i pro nejvyšší množství vozidel (720) celé zpracování trvá nanejvýš 1.2 sekundy. Z toho plyne, že samotné zpracování dat není nijak časově náročné a vzhledem k 20sekundové periodě aktualizace dat máme velkou časovou rezervu. Rychlosť zpracování jednoho vstupního souboru může více ovlivnit stahování dat z internetu, kde ale předpokládáme, že po většinu času nebude trvat stáhnout aktuální polohy vozidel déle než desítky milisekund.

Jediné delší prodlení může nastat ve chvíli, kdy je potřeba stáhnout velké množství dodatečných informací o novém spoji. Na grafu 1.4 je vidět, že až na jednotky vyjímek je počet nově nalezených spojů v jednom souboru nejvýše 20.

Aplikace je ale naimplementována tak, aby se tyto informace stahovaly asynchroně a tedy čákání na stažení dat je co nejkratší.

4.2 Evaluace výsledků

4.2.1 Sestrojení modelů

Po využití testovacích dat vzorků poloh vozidel zaznamenaných ve dnech 20.–24. února 2020 bylo podle krytéří, kterými jsou zejména vzdálenost zastávek a počet vzorků mezi nimi, sestrojeno celkem 1106 polynomiálních modelů. Z toho je 847 modelů pro pracovní dny, které jsou nejdůležitější. Přičemž celkový počet párů zastávek je 7230, ale zastávek ve vzdálenosti 1500 metrů² je pouze 2142. Z toho vychází, že u 40 % dvojic zastávek je dostatek dat, aby dával výpočet modelu smysl.

U zbylých dvojic zastávek se využívá lineární model.

4.2.2 Odhadý zpoždění

Z toho jak jsou definovány požadavky řešení v kapitole 2.1.1 pro změření kvality výsledků stačí porovnávat odhad zpoždění lineárního (původního) modelu a nového polynomiálního modelu. Přičemž odhad je lepší pokud má sekvence odhadů z celé jízdy mezi dvojcí zastávek menší rozptyl.

Podívejme se tedy na porovnání odhadů zpoždění novými modely profilů jízd se stávajícím řešením pracujícím s předpokladem, že vozidla jedou celou trasu mezi dvěma zastávkami konstantní rychlostí.

Evaluaci výsledků budeme provádět s daty sesbíranými 20. 2. 2020, které použijeme jako trénovací data a s daty sesbíranými 21. 2. 2020, které použijeme jako testovací data. Toto je standardní postup pro hodnocení úspěšnosti predikcí modelů ve světě strojového učení. Modely nemohou být testovány na stejných datech jako na kterých byly trénovány, protože kdyby se trénovalo i testovalo na stejných datech, model by nemusel nic predikovat, ale stačilo by, aby si jen "zapamatoval" hodnotu z množiny trénovacích dat.

²zvolená minimální vzdálenost mezi zastávkama, mezi kterýma má ještě smysl odhadovat zpoždění

5. Porovnání se stávajícím řešením

TODO tabulka pro kazdy model s porovnanim odhadu zpodeni nyni a podle meho modelu

6. Navrhy na zlepšení

TODO

7. Statistiky

TODO neni to primo v zadani prace, ale ruzne statistiky nad zpozdenima by mohly byt zajimave, je to vhodne???

Závěr

Seznam použité literatury

Seznam obrázků

1.1	Graf počtu úseků mezi následujícími zastávkami a vzdálenotí mezi nimi.	6
1.2	Modrá plocha značí vymodelovaný profil trasy. Oražové body jsou jednotlivé vzorky polohy vozidel. Data pro graf jsou ze dnů 20.–21. 2. 2020	7
1.3	Trasa mezi zastávkama K Letišti a Zličín. Zdroj: mapy.cz	7
1.4	Počet souborů s počtem nově nalezených spojů v nich.	15
1.5	Počet souborů s počtem nově nalezených spojů v nich.	16
1.6	Mapa z golemio.cz.	17
1.7	Mapa z www.tram-bus.cz.	17
1.8	Mapa z mapa.idsjmk.cz.	18
2.1	UML diagram návrhu aplikace	20
2.2	EER diagram databáze.	22
2.3	Variabilita délky jízdy v průběhu dne	27
2.4	Variabilita času jízdy v závislosti na ujeté vzdálenosti	28
2.5	Úsek s nepravidelnostmi	28
2.6	Úsek s nepravidelnostmi 2	29
2.7	Úsek s nepravidelnostmi	30
2.8	Nejednoznačnost konkávního obalu	31
2.9	Mapbox Mapa, použitý styl mapy: streets-v11	34
2.10	Design zobrazení elementů v mapě, linka je 348 je vybrána	34
4.1	Průměrný čas zpracovávání daného počtu vozidel ze všech souborů se statickými daty. Světle modrá barva ohraničuje 95 % interval spolehlivosti. Počty vozidel jsou vždy zaokrouhleny dolů na celé desítky.	49

Seznam tabulek

Seznam použitých zkratek

Slovník

- AJAX** Asynchronous JavaScript and XML. 42
- API** rozhraní pro programování aplikací. 9, 43
- CSS** Cascading Style Sheets. 42
- DOM** Document Object Model. 42
- DPP** Dopravní podnik hlavního města Prahy, a.s. 8, 17
- GEOJSON** standardní formát navržený pro reprezentaci jednoduchých prostorových geografických dat, specifikace: <https://tools.ietf.org/html/rfc7946>. 9, 24, 42, 43, 45, 46
- GPS** Global Position System. 8, 10, 12
- HTML** Hypertext Markup Language. 42, 43, 44, 45
- HTTP** HyperText Transfer Protocol. 9
- IDSJMK** Integrovaný dopravní systém Jihomoravského kraje. 17
- INT** Celé číslo. 23
- JS** JavaScript. 42, 43
- JSON** JavaScript Object Notation, specifikace: <https://tools.ietf.org/html/rfc8259>. 9, 21, 36, 44
- Kapsch** Kapsch, rodinná firma. 8
- MHD** městská hromadná doprava. 4, 11
- OSM** OpenStreetMap. 16
- PID** Pražská integrovaná doprava. 33
- PID** Object Oriented Programming. 36
- ROPID** Regionální organizátor pražské integrované dopravy, p. o.. 3, 9
- SQL** Structured Query Language. 21, 35, 36, 38, 39, 46
- URL** Unique Resource Link. 9, 36
- UTC** Koordinovaný světový čas. 10, 42
- VHD** Veřejná hromadná doprava. 16, 32

A. Přílohy

A.1 První příloha