



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 2021

Prohlašuji, že jsem tuto bakalářskou práci vypracoval 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 doc. Mgr. Martinu Nečaskému, Ph.D., 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 Jakubu Klímkovi, RNDr., Ph.D. za zapůjčení počítače za účelem získání testovacích dat pro tuto práci.

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., Katedra softwarového inženýrství

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. Takto spočítaná data se využijí v aplikaci, která vznikne pro webové rozhraní, kde zobrazí aktuální polohy spojů a rozšiřující informace o nich do mapového podkladu. Aplikace bude aktivně interagovat s uživatelem. Všechny komponenty byly otestovány. A bylo ukázáno, že otevřená data lze využít k dosažení větších cílů.

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., Department of Software Engineering

Abstract: This thesis deals with analysis of real-time data of public transport vehicles over open data in Prague. Its main purpose is to create statistics and improve estimation, based on historical data, of a vehicle delay on its route between reference points. For demonstration of the computed data a front-end web app is created. This interactive app is able to show current vehicles locations over a map and other useful information. All components were tested. Open data from Prague public transport company were used to demonstrate that open data can be used for achieving high goals.

Keywords: delay open data public transport

Obsah

Úvod	3
1 Analýza	6
1.1 Úvod	6
1.2 Popis problému odhadu zpoždění	6
1.2.1 Příklad nelineárního profilu trasy	7
1.2.2 Současná řešení	9
1.3 Analýza zdroje dat	10
1.3.1 Přístup k datům	11
1.3.2 Obsah dat	12
1.3.3 Analýza statických dat	16
1.3.4 Funkční a kvalitativní požadavky	20
1.4 Analýza vizualizačních nástrojů	21
1.4.1 Mapové podklady	21
1.4.2 Současná řešení	22
1.4.3 Funkční požadavky	23
1.4.4 Kvalitativní požadavky	23
2 Návrh řešení	24
2.1 Úvod	24
2.2 Architektura	24
2.3 Zpracování vstupních dat	24
2.3.1 Databáze	25
2.3.2 Plnění databáze	27
2.4 Algoritmus odhadu zpoždění	28
2.4.1 Základní předpoklady	28
2.4.2 Návrh modelování	29
2.4.3 Využití modelů	34
2.5 Vizualizace dat	34
2.5.1 Návrh grafiky a UI	34
3 Implementace	43
3.1 Úvod	43
3.2 Zpracování dat	44
3.2.1 Konstrukce objektů vozidel	44
3.2.2 Odhad zpoždění	45
3.2.3 Kompletace dat a jejich uložení	45
3.3 Konstrukce modelů	47
3.3.1 Čtení dat	47
3.3.2 Příprava dat	49
3.3.3 Práce s modely	50
3.4 Vizualizace dat	51
3.4.1 Klientská část	51
3.4.2 Serverová část	54

4 Testování a evaluace	57
4.1 Testování softwarového řešení	57
4.1.1 Unit testy	57
4.1.2 Integrační testy	57
4.1.3 Testy kvality	57
4.2 Evaluace výsledků	61
4.2.1 Konstrukce modelů	61
4.2.2 Odhady zpoždění	64
4.3 Statistiky	71
Závěr	74
Seznam použité literatury	76
Seznam obrázků	77
Seznam tabulek	79
A Přílohy	82
A.1 První příloha	82

Úvod

Městská hromadná doprava v Praze a Středočeském kraji je jedním z hlavních 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, zda by nebylo možné určit s jakou pravidelností, pokud s nějakou, taková zpoždění vznikají. A zda by nemohl být včas informován o vzniklé anomálii a vzniklém zpoždění.

Ve vymezené oblasti operuje spousta soukromých i městských dopravců. Ti, kteří spadají do naší zájmové oblasti, zastřešují organizace ROPID a IDSK, které objednávají jednotlivé spoje. Pro naši práci je důležité, že tyto organizace zadaly 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 ve vlastnictví hlavního města Praha. Takových spojů, o kterých máme všechna požadovaná data, je v pracovní den vypraveno necelých deset tisíc.²

Definice problému

Cílem této práce je zpřesnit odhad zpoždění vozidel veřejné hromadné dopravy (dále jen VHD), zejména autobusů, na trase mezi dvěma sousedícími zastávkami. Dále pak tyto výsledky vizualizovat v mapových podkladech.

V době návrhu práce, kvůli právním komplikacím a složitosti informačního systému (Guryčová (2019)), nebyla k dispozici real-time data od majoritního dopravce na území Prahy DPP. Jelikož je práce zaměřena na odhad zpoždění spoje mezi dvěma sousedícími zastávkami na trase, má tedy větší význam odhadovat zpoždění mezi zastávkami, mezi kterými je větší vzdálenost. A to jsou převážně spoje jedoucí mimo Prahu. Proto tato data z DPP nenabývají takové důležitosti, jako data od dopravců operujících mimo Prahu. Vzhledem k tomu, že zbylí dopravci využívají k přepravě cestujících převážně autobusy, bude práce vypracována pouze s ohledem na autobusovou dopravu.

V práci se tedy pokusíme využít dostupná otevřená real-time data k získání informací o zpoždění spojů na trase a využít je k lepším odhadům zpoždění. Řešení ovšem není založeno pouze na real-time datech, ale využívá také statická data o jízdních řádech nebo zastávkách hromadné dopravy, jejichž zdrojem je přímo ROPID³ a také mapové podklady. Ty jsou potřeba zejména pro vizualizaci zastávek a jízdních řádů a vykreslení trasy spoje přímo do mapy. Avšak i tato 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

³pid.cz/o-systemu/opendata/

Protože disponujeme daty o aktuálních polohách vozidel VHD, která navíc rozšíříme o lepší odhadu zpoždění, nabízí se jejich využití tím, že budou zanesena do mapy, čímž 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 jeho žádost zobrazí více infomací o daném spoji nebo vybrané zastávce.

Souhrn

Klientská aplikace bude vytvořená pro webovou platformu, kód bude napsán ve standardních jazycích pro vývoj webových aplikací. Jsou jimi zejména HTML a JS.

Celá serverová část je napsána v jazyce Python3. Server bude sloužit pro stahování a ukládání dat do SQL databáze, konkrétně implementace MySQL. Další část back-endu bude zpřístupňovat data z databáze, určené pro klienckou aplikaci.

Jádro celé práce tvoří modul odhadu zpoždění. Zde používáme knihovny jazyka Python 3 pro strojové učení scikit-learn a pro práci s velkými objemy dat NumPy. Zpoždění spojů budeme odhadovat pomocí pravděpodobnostních modelů natrénovaných na historických datech.

V průběhu celé práce se setkávame s grafy a statistikami. Ty vyplývají z dat, dle kterých byla celá práce testována. Konkrétně byla data sbírána ve dnech 20. 2. 2020 až 23. 2. 2020. Dále v textu, kde se takové statistické údaje vyskytují, je vždy uvedeno i z jaké podmnožiny všech testovacích dat vycházejí. Všechny grafy, obrázky a statistické údaje uvedené v této práci, u kterých není explicitně uveden zdroj, byly vytvořeny jako součást této práce a jsou založeny na datech a poznacích získaných v rámci této práce. Vstupní data jsou analyzována v kapitole 1.3.

Struktura dokumentu

Tento dokument je rozdělen do 4 hlavních kapitol.

Analýza

V této kapitole se zaměříme na detailní definici a popis řešeného problému. Analyzujeme vstupní data a jejich zdroj. Dále si ukážeme příklady již funkčních nástrojů vizualizující polohy vozidel a jiná data.

Návrh řešení

V této kapitole definujeme funkční požadavky na dílo. Navrhнемe databázi, do které budeme ukládat data, včetně algoritmu plnícího databázi. Poté navrhнемe algoritmus, kterým budeme odhadovat zpoždění. A na závěr kapitoly definujeme front-endovou aplikaci včetně serverové části.

Implementace

Zde popíšeme implementaci konkrétních softwarových částí práce. Tedy zpracování dat, algoritmu odhadu zpoždění a klientské aplikace.

Testování a evaluace

Ukážeme, že aplikace je řádně otestována po všech stránkách a především porovnáme řešení této práce se stávajícím řešením odhadu zpoždění. Na závěr si ukážeme zajímavé statistiky o jízdách vozidel VHD.

Definice pojmu

Tato práce zabíhá do velkého detailu problematiky VHD. Protože je z hlediska softwarového vývoje velmi důležité ujasnit si terminologii a předcházet tak různým nedorozuměním, definujme si pojmy, které mohou běžně splývat.

- Nástupiště – Nástupiště je nejmenší bod, který jsme schopni rozlišit. Od něj odjíždějí konkrétní vozidla.
- Zastávka / Stanice / – Rozdíl mezi zastávkou a stanicí není jasně definován. Obecně se chápe, že stanice je většího rozsahu, tedy že má více nástupišť nebo je obsluhována více druhý dopravních prostředků. Stanicím a zastávkám pak přísluší i pojmenování.
- Jízdní řád – Je definován posloupností zastávek, kterým naleží časy příjezdu a odjezdu. Čas projetí zastávky je počítán vždy od půlnoci, tedy není vázán na konkrétní den.
- Spoj – Spoj je definován jízdním řádem, dále mu přísluší informace o jeho konečné stanici a číslo. Z definice jízdního řádu vyplývá, že jeden spoj může jet v libovolný počet dnů, ale vždy ve stejný čas a po stejné trase.
- Jízda – Jízdou myslíme jednu konkrétní realizaci spoje.

V této práci na softwarové úrovni budeme výhradně pracovat s nástupišti. Avšak pro zlepšení čitelnosti textu běžně používáme slovo zastávka i ve smyslu nástupiště.

1. Analýza

V této kapitole je detailně popsán řešený problém a způsoby navrženého a současného řešení. Dále je popsán zdroj dat, se kterým budeme v této práci pracovat.

1.1 Úvod

Spoje, které zajišťují hromadnou dopravu, jezdí podle jízdních řádů, které definují jejich trasu. Zastávky na trase těchto spojů 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 nezřídka 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 20. 2. 2020.

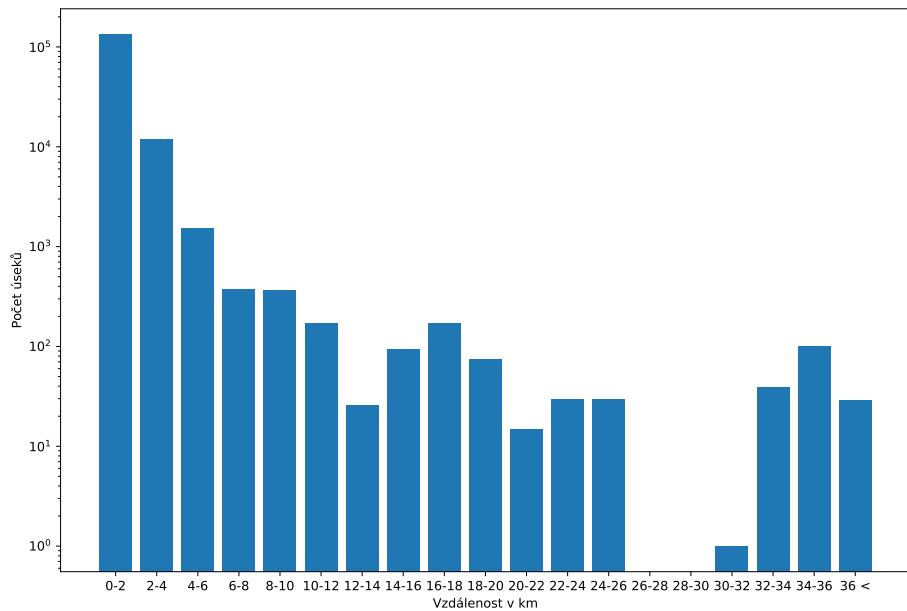
1.2 Popis problému odhadu zpoždění

Řešený problém se týká případu, kdy vozidlo projíždí 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 a při vjezdu do dalšího města zase zpomalí. 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 zpřesňuje odhad zpoždění. Odhad tímto novým způsobem by měl být mnohem přesnější než současné odhady, které předpokládají, že vozidlo jede konstantní rychlostí po celou dobu jízdy, více rozepsáno v kapitole 1.2.2. Také je možné brát jako aktuální zpoždění spoje poslední změřené zpoždění při průjezdu nějakým referenčním bodem (zastávkou, návěstidlem).

Přidaná hodnota je tedy v tom, že práce navrhne takové modely, které nebudou penalizovat zvyšováním zpoždění za pomalou jízdu v úsecích, které se pomaleji projíždějí vždy. A také naopak zvýhodňovat snížením zpoždění za rychlou jízdu v úsecích, které se obvykle projíždějí rychleji. Pokud bychom se tedy

¹Podle dat pro spoje jedoucí v 20. 2. 2020 je medián vzdáleností mezi zastávkami, mezi kterými 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 15 000.



Obrázek 1.1: Počet úseků mezi bezprostředně následujícími zastávkami a vzdálenost mezi nimi. Každý průjezd úsekem je započítán zvlášť.

podívali na změny zpoždění na trase mezi dvěma referenčními body, v ideálním případně by měly být nulové.

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

Pro vyloučení všech pochybností je třeba uvést, že se naše práce nesnaží předpovědět zpoždění, které spoj může nabrat vzhledem k dosavadnímu průběhu trasy. Tedy např. nijak nezohledňuje 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 odhadnout zpoždění v daném bodě na trase vzhledem k obvyklému profilu jízdy. Tedy např. pokud by výše uvažovaná kolona byla pravidelná, práce ji zohlední ve statistických modelech.

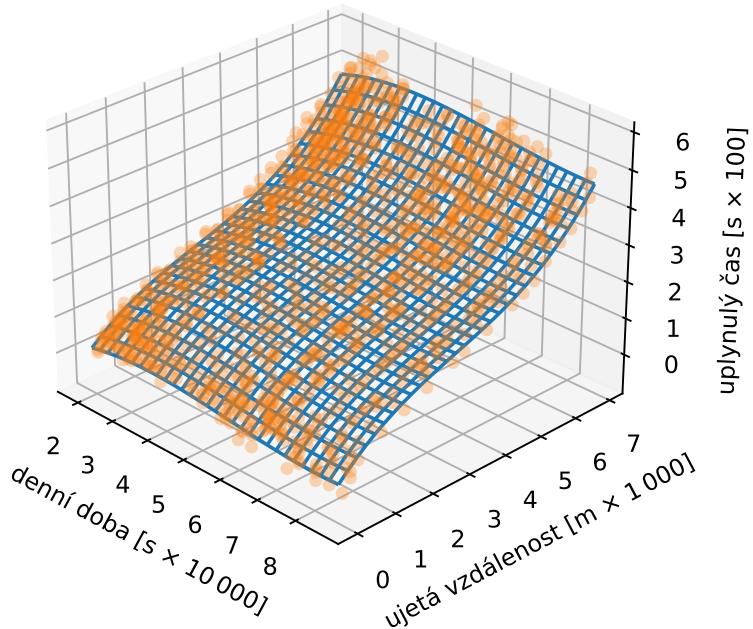
1.2.1 Příklad nelineárního profilu trasy

Celé ilustrováno na příkladě jízd mezi dvěma zastávkami K Letišti a Zličín, kde je nelineární profil jízdy vidět velice dobře. Jedná se totiž o trasu přesně odpovídající situaci popisovanou výše.

Popsané rozdíly v rychlosti a nelineární profil trasy je patrný na grafu 1.2. Tento typ grafů se vyskytuje v průběhu celé práce a zobrazuje podstatu problému a jeho řešení.

Popišme si tedy detailně jeho význam. Na ose vlevo dole je počet sekund od půlnoci, vpravo dole je vzdálenost od poslední projeté zastávky, na vertikální ose

K Letišti → Zličín



Obrázek 1.2: Vzorky poloh a model profilu jízdy mezi zastávkami K Letišti a Zličín. Data jsou ze dnů 20.–21. 2. 2020

je pak čas od vyjetí z poslední projeté zastávky. Oranžové body znázorňují jednotlivé vzorky poloh vozidel, z nichž jsme schopni jednoznačně zjistit čas pořízení vzorku, vzdálenost od poslední projeté zastávky, zpoždění v této zastávce a podle jízdního řádu i čas pravidelného průjezdu touto poslední zastávkou. Tento soubor dat nám umožňuje je vynést do popsaného grafu. Modrá plocha je pak vizualizace modelu, popisující profil jízdy vozidla mezi dvojicí zastávek. Jeho konstrukce je popsána v kapitole 3.3. Vlevo dole jsou tedy vzorky poloh zaznamenané krátce po vyjetí ze zastávky. Vpravo nahoře pak vzorky poloh těsně před příjezdem do následující zastávky.

Na tomto grafu stojí za povšimnutí viditelné zpomalení průjezdů v raní špičce, 7.–9.² hodina ráno.

Tento příklad dále podrobněji analyzován v kapitole 2.4.2.

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

Rozbor trasy

Celá tato trasa má necelých 7 km a její průjezd podle jízdního řádu VHD trvá 10 minut. Prvních 600 metrů je vedeno po obecní komunikaci přes křižovatku

²časy jsou v UTC



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

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 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 . Do této průměrné rychlosti se započítává i jízda ve všech výše popsaných úsecích, tedy skutečná průměrná rychlosť v tomto posledním úseku byla výrazně nižší.

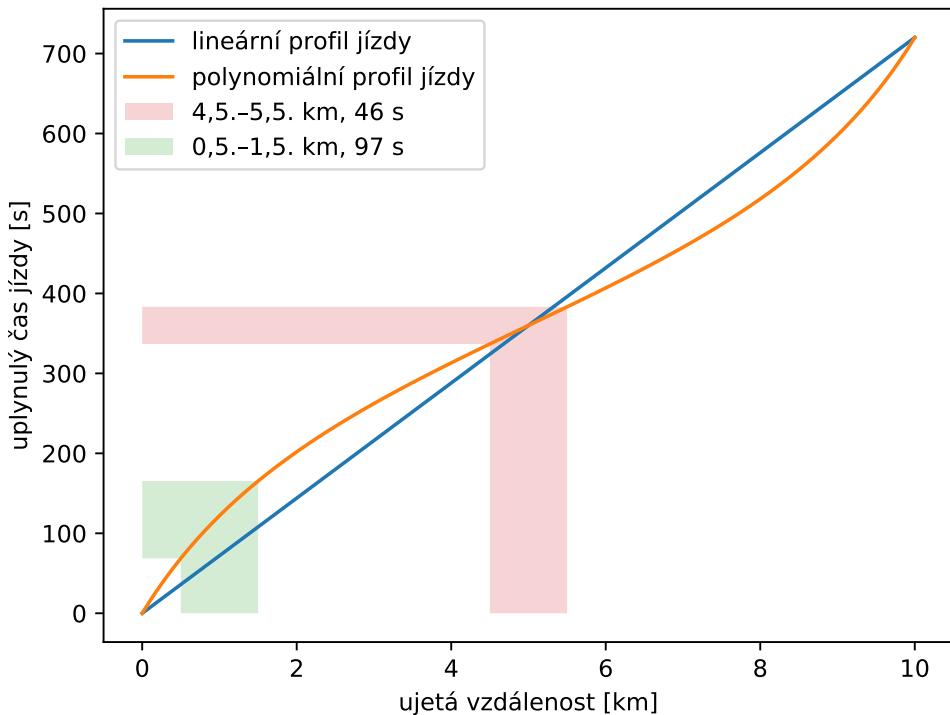
1.2.2 Současná řešení

Algoritmus na odhad aktuálního zpoždění mezi dvěma referenčními body již existuje a je součástí Datové platformy Golemio, ze které se čerpají data pro tuto práci. (Detailní popis dat uveden v kapitole 1.3.)

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

Proto je potřeba tento odhad zpřesnit, což je cílem naší práce. K tomuto cíli jsme byli nasměrováni v rámci schůzky s pracovníky společnosti OICT, kde bylo řečeno, že tento problém není řešen v současném řešení.

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.



Obrázek 1.4: Srovnání současného řešení s navrhovaným, modelový příklad

Současné řešení ve srovnání s navrhovaným řešením je zobrazeno na grafu 1.4. Zde je zobrazen modelový příklad trasy vozidla jedoucí trasu dlouhou 10 km za 720 s. Při použití současného řešení se na vozidlo nahlíží, jakož jede rychlostí 50 km/h po celou trasu. Zatímco navrhované řešení modeluje obvyklé výkyvy rychlosti, a tedy zohledňuje při odhadu zpoždění čas jízdy jednotlivých úseků celé trasy. Konkrétně v tomto modelovém příkladě uvažované vozidlo na začátku své trasy projelo úsek mezi 0,5. km a 1,5 km za 96 s (37,5 km/h), což odpovídá např. výjezdu z města. Úsek mezi 4,5. km a 5,5. km vozidlo ujelo za 46 s (78 km/h), což může být jízda mezi městy. Pokud tedy vozidlo pojede podle modelu jízdy tak, jak je na grafu zobrazeno oranžovou křivkou, budeme mu po celou dobu jízdy předpovídat konstantní zpoždění. Naopak podle současného lineárního modelu se odhad zpoždění v průběhu trasy může pohybovat v rozmezí několika desítek sekund.

1.3 Analýza zdroje dat

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

1.3.1 Přístup k datům

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

Taková data pak přímo putují k provozovateli systému na monitorování vozidel, kterým je společnost Chaps spol. s.r.o.⁵ jakožto partner ROPIDu. Ten však tato data zpracovává a posílá ke zveřejnění na platformě Golemio. Bohužel při tomto procesu zpracování se vytratí informace o události, v jaké byla data pořízena. Tedy informace o příjezdu nebo odjezdu ze zastávky je zjistitelná pouze z GPS souřadnic a následném odhadu pozice vozidla na trase daného spoje.

Poté co jsou tato data přenesena 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 a zpoždění v poslední zastávce.

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

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

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 platforma vůbec neodpovídala na dotazy, nebo vracela prázdné datasety.

Současně s využívanou verzí API, je nasazená i pokročilejší API ve verzi 2, která 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 platformy⁶ je sama o sobě poměrně zastaralá. Její aktuální sada parametrů neodpovídá dokumentaci a neobsahuje žádné popisy nebo vysvětlení atributů. 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.

⁴Řečeno zaměstnancem OICT na schůzce 4. 5. 2019

⁵<https://pid.cz/dopravci-a-partneri/chaps-spol-s-r-o/>

⁶Golemio: <https://golemioapi.docs.apiary.io>

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 get požadavku daného URL adresou, parametry a jeho hlavičkou. Upozorněme, že pro úspěšný přístup k datům je potřeba se na Datové platformě zaregistrovat a do hlavičky požadavku vždy umístit i vygenerovaný token.

Ačkoli se dokumentace tváří tak, že data jsou exportována ve formátech JSON nebo GEOJSON, v průběhu práce s daty jsme se setkali s tím, že formát dat nebyl vždy přesně podle specifikace těchto formátů. Například může být uveden atribut `wheelchair_accessible`, který je typu `boolean` a je nastaven na hodnotu `True`, nicméně podle specifikace se tyto hodnoty píší s malým písmenem⁷. Pro tuto práci to sice nepředstavuje komplikaci, jelikož tento atribut není potřeba, ale mohlo by se stát, že některé parsery formátu JSON vyhodnotí řetězec jako nevalidní a skončí chybou.

Celá Datová platforma Golemio je pojatá jako Open Source projekt⁸. Tedy je možné její zdrojový kód vylepšit či opravit, nebo také čtením kódu detailně porozumět, jak zde popisované zpracování dat funguje. Avšak takový rozbor zdrojového kódu je mimo rozsah této práce.

1.3.2 Obsah dat

Data důležitá pro tuto práci jsou ve zdroji dat rozdělená do několika souborů. Tyto soubory pak nesou podstatné údaje, které si rozebereme v této sekci. V dokumentaci Datové platformy je popsána celá řada souborů s informacemi vztaženými k hromadné dopravě, v nichž se dá vyhledávat pomocí hodnot atributů tak, že lze např. vyžádat data pro konkrétní spoj.

Pro tuto práci budeme využívat výhradně soubor aktuálních poloh vozidel, který má název All Vehicle Positions.

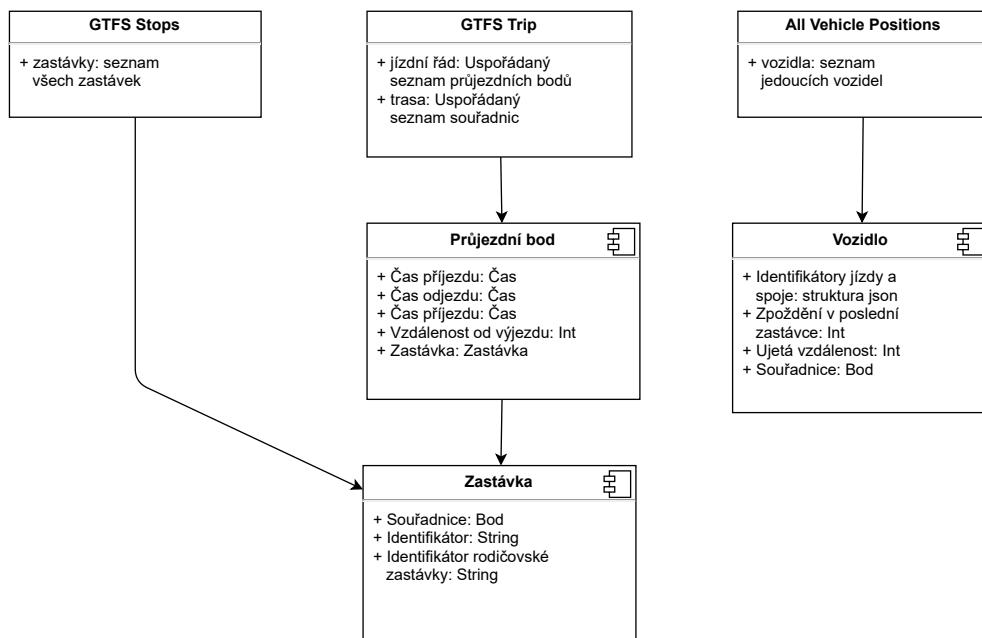
Informace o nově nalezených spojích v souboru s polohy vozidel si pak vyžádáme jednotlivě pro každý spoj ze souboru jménem GTFS Trip, který obsahuje jízdní řád spoje i jeho cestu.

Dále je možné využít soubor zastávek se jménem GTFS Stops, který by měl obsahovat všechny vyskytující se zastávky. Tento soubor je vhodné zpracovat na začátku běhu aplikace. Musíme však být připraveni na možný výskyt nové zastávky, která nebyla nalezena v tomto souboru zastávek, ale je nalezena v jízdním řádu spoje a být schopni vložit takovou zastávku do databáze. Proto pro nás nebude tento soubor příliš důležitý. Datový formát reprezentace zastávek je však stejný v souboru zastávek i souboru jízdního řádu spoje.

Diagram zobrazující strukturu dat v Datové platformě je zobrazen na obrázku 1.5. Uvádíme zde pouze datové sady potřebné pro tuto práci, to jsou výše popsané

⁷V průběhu tvorby této práce byla chyba opravena.

⁸Programátorská dokumentace je dostupná na <https://operator-ict.gitlab.io/golemio/documentation/>



Obrázek 1.5: Struktura vstupních dat a relace mezi nimi

soubory. Stejně tak popisujeme asociace mezi daty tak, jak je využíváme v této práci, např.: každé vozidlo může být na vyžádání dolněno o jízdní řád, tuto relaci však nevyužíváme.

Polohy vozidel

Ze zveřejněných dat na této platformě patří mezi nejdůležitější data pro tuto práci polohy vozidel, jelikož chceme vozidla zobrazovat na mapě. Tato data budou také sloužit pro trénování statistických modelů, pomocí kterých budeme následně odhadovat zpoždění. Jelikož se jedná o real-time data, tato data rychle zastarávají. Proto je nutné je velmi často (každých 20 sekund) stahovat tak, aby vizualizace reálné situace ještě dávala smysl. A také aby nám neunikly žádné vzorky poloh vozidel. Mohlo by se totiž stát, že poslední zaznamenaný vzorek polohy bude přepsán novějším vzorkem.

Využívané atributy u každého vozidla ze souboru All Vehicle Positions jsou:

- `coordinates` – aktuální GPS souřadnice vozidla
- `origin_timestamp` – čas zachycení polohy vozidla, v časovém pásmu UTC
- `gtfs_trip_id` – unikátní identifikátor tripu pro spárování s jízdním řádem
- `gtfs_shape_dist_traveled` – vzdálenost vozidla ujetá od začátku jízdy 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 atributy, je níže. Řada z atributů je pro tuto práci zbytečná a jsou vynechána. Dále je možno si povšimnout atributu `all_positions` (na konci příkladu), 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_real_agency_name": "ČSAD Česká Lípa"
        "gtfs_route_id": "L467",
        "gtfs_trip_id": "467_252_200105",
    },
    "last_position": {
        "bearing": 20,
        "cis_last_stop_id": 21393,
        "cis_last_stop_sequence": 28,
        "delay": 261,
        "delay_stop_departure": 287,
        "gtfs_shape_dist_traveled": "64.1",
        "lat": "50.41881",
        "lng": "14.91724",
        "speed": 20,
        "tracking": 2,
    },
    "all_positions": {
        "features": [],
        "type": "FeatureCollection"
    }
},
"type": "Feature"

```

Spoje

Dále jsou k dispozici data o každém spoji. To je popis trasy vozidla, včetně zastávek a časů příjezdů a odjezdů do/z nich. Také může být vyžádáno k informacím o jízdě připojení celého detailního nákresu trasy, tj. lomená čára kopírující celou trasu daného spoje po povrchu Země.

Míra unikátnosti identifikátorů těchto spojů je předmětem dohadů a zřejmě jsou pod správou plánovačů VHD, nicméně pro účely této práce můžeme předpokládat, 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 spoje pro spárování s real-time daty, pravděpodobně odpovídá atributu **gtfs_trip_id**

Navíc pro každý spoj může být vyžádáno zaslání seznamu zastávek, kterými projíždí. Zde jsou k dispozici informace vázající se k danému průjezdu zastávkou. Každá uvedená zastávka je doplněna o kompletní informace, které ji definují. Tedy má stejnou informační hodnotu jako samostatný dotaz na jednotlivé zastávky z datového souboru zastávek.

Zastávky

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

Příklad dat popisujících jednu jízdu včetně zastávek je uveden níže. Seznam zastávek a body trasy jsou zkráceny vzhledem k objemu dat. Nepotřebné atributy byly smazány pro větší přehlednost. Za povšimnutí stojí, že geometrie zastávky (tedy její souřadnice) mají hodnotu **null** a souřadnice jsou uvedeny v jiném atributu. Takových zmatečných situací je ve vstupních datech celá řada a před zahájením implementace zpracování těchto dat je nutné, se vždy důkladně ujistit, že potřebné informace jsou opravdu součástí datové sady. Stejně tak některé atributy, které by mohlo být užitečné přebírat přímo ze zdroje, nemůžeme použít, protože v reálných datech jsou vždy nebo většinou bez hodnoty. Takové chování se však nedá vyčítat Datové platformě nebo ROPIDu, protože kvalita dat je vždy závislá na jejich zdroji, v tomto případě samotných vozidlech a záznamových zařízeních v nich.

```
"route_id": "L421",
"trip_headsign": "Kolín, Nádraží",
"trip_id": "421_225_191114",
"stop_times": [
    {
        "arrival_time": "14:14:00",
        "departure_time": "14:14:00",
        "shape_dist_traveled": 0,
        "stop_id": "U2033Z5",
        "stop_sequence": 1,
        "trip_id": "421_225_191114",
        "stop": {
            "geometry": {

```

```

    "coordinates": [
        null,
        null
    ],
    "type": "Point"
},
"properties": {
    "stop_id": "U2033Z5",
    "stop_lat": 49.87486,
    "stop_lon": 14.9078,
    "stop_name": "S\u00e1zava,Aut.st.",
},
"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"
},
...
]

```

1.3.3 Analýza statických dat

Sběr dat probíhal ve dnech 20. 2. 2020 – 23. 2. 2020 z Datové platformy Golemio. Data byla stahována po souborech tak, jak jsou popsána výše.

Pro stahování těchto souborů byl využit skript napsaný v jazyce Bash, který po dobu 4 dnů periodicky každých 20 sekund stahoval 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. Po dokončení tohoto stahování byly staženy soubory o nalezených spojích.

Data byla přebírána pouze z datových souborů o každém jednotlivém spoji a aktuálních polohách vozidel stahovaných každých 20 sekund. Tedy pokud ur-

čitou 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 zpozdění spoje v poslední zastávce a zcela nenalezený spoj ve zdroji dat (tedy chybějící jízdní řád). Nicméně jedná se o zanedbatelné množství spojů. Práce totiž nemůže počítat s poškozenými daty a není jejím úkolem držet všechny zastávky, ale pouze ty, kterými nějaký spoj projízdí. To z důvodu přehlednosti zastávek při jejich vizualizaci, kde ukazovat nepoužívané zastávky nemá smysl. A také není smysluplné počítat odhady zpozdění pro tyto neobsluhované zastávky.

Zastávky

Do databáze bylo celkem vloženo 5820 nástupišť, 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 opakováně v následující den⁹.

Vstupní soubory

Pro testovací účely bylo celkem pořízeno 15 794 obrazů dopravní situace zaznamenávající aktuální polohy vozidel.

Protože zpracování všech vozidel v jednom souboru stojí netriviální množství času, analýza dat v těchto souborech je důležitá. Poslouží nám k definici technických požadavků aplikace a později pro její testování.

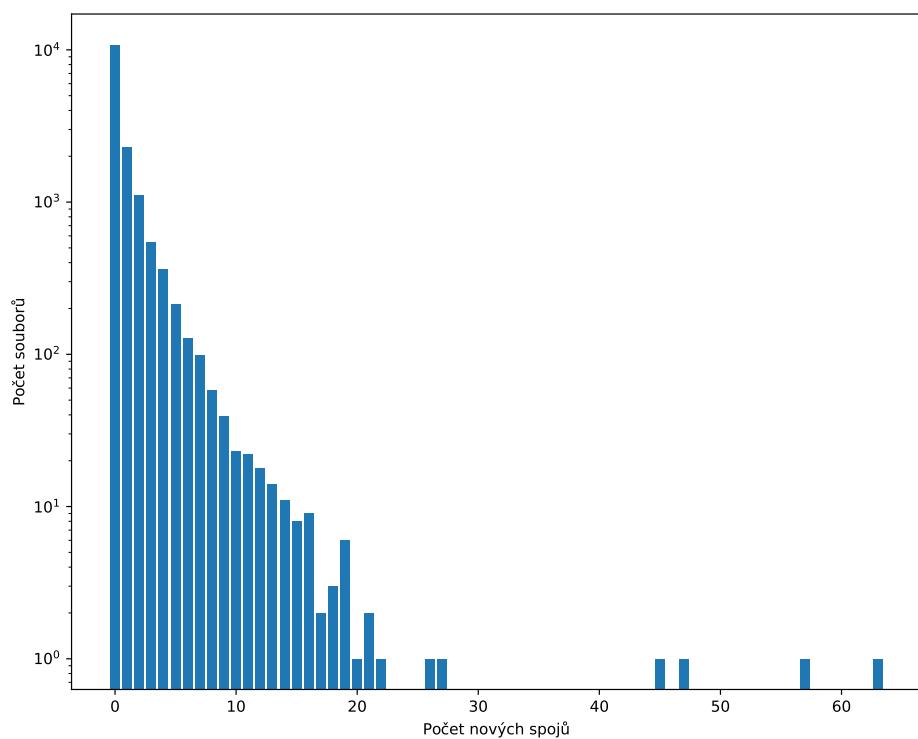
Z celkového počtu nalezených spojů vyplývá, že počet nově nalezených spojů v jednom obrazu je méně než jeden. Kompletní histogram počtu souborů poloh vozidel, které obsahují určitý počet nově objevených spojů je zobrazen na grafu 1.6.

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 1.7. Z tohoto grafu vyplývá, že velká většina vstupních souborů z celkového počtu 15 793 obsahuje do 200 vozidel v každém souboru.

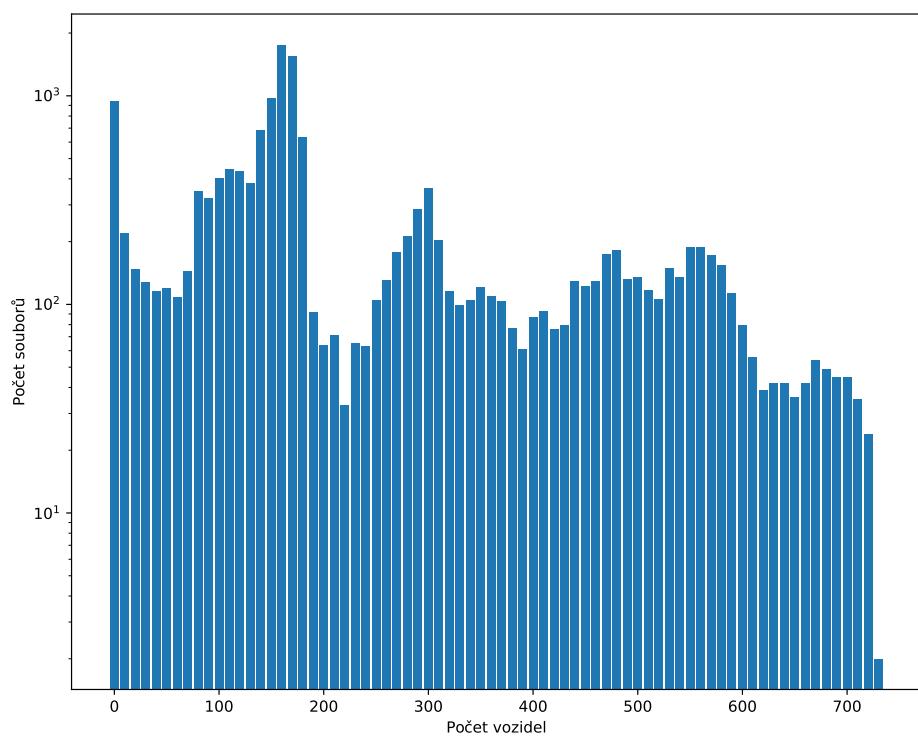
Avšak ne všechna vozidla v každém souboru jsou nová nebo dostala změny oproti předešlému záznamu. To z důvodu, že je zdroj dat nastaven tak, aby vysílal polohu vozidla jako aktuální, i když už je zastaralá několik minut. V takovém případě se zpracovává vzorek polohy vozidla opakováně.

Další rozbor dat na grafu 1.1 a v kapitole 4.3.

⁹Ze 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 trasu.



Obrázek 1.6: Počet souborů s počtem nově nalezených spojů v nich



Obrázek 1.7: Počet souborů s celkovým početem vozidel v nich

1.3.4 Funkční a kvalitativní požadavky

Nejprve specifikujme požadavky na systém, na kterém se pak bude zakládat konkrétní návrh řešení jádra celé aplikace. Funkční a kvalitativní požadavky vyházejí z analýzy problému a jeho řešení a jsou založeny na analýze vstupních souborů v kapitole 1.3.3.

V této sekci jsou specifikovány pouze moduly stahování dat a výpočtu modelů. Front-endová aplikace a část serveru, která komunikuje s touto aplikací jsou popsány níže v kapitole 2.5.

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 nejrychleji¹⁰.
- Data o polohách vozidel VHD v Datové platformě jsou aktualizována nejpozději každých 20 sekund. Tedy pro minimalizaci rychlosti zastarávání dat a získání všech existujících vzorků dat o polohách je nutné, data stahovat alespoň každých 20 sekund.
- Odhad zpoždění se bude provádět na základě historických dat z posledních vyšších jednotek dnů¹¹. Tím se sníží dopad mimořádné události na předpovědní model, která může na trase vzniknout. Zároveň by však neměla být 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, jelikož samotné jízdní řády se mohou lišit¹² a také se do velké míry liší hustota dopravy, která má velký vliv na profil jízdy.
- Zpracování historických dat bude probíhat vždy 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.

¹⁰Průměrná 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 grafu 1.1.

¹¹Pro demonstrativní účely této práce jsou využívána historická data pouze ze 4 dnů (2 pracovní a 2 výkendové).

¹²Ve dnech pracovního volna se v některých případech liší doba jízdy mezi zastávkami pro stejnou dvojici zastávek. A to je porušení základního předpokladu z kapitoly 2.4.1 Základní předpoklady.

- Uložená historická data budou strukturovaná tak, aby nad nimi mohly být prováděny statistické výpočty minimálně o frekvencích jízd spojů, vzdálostech tras a zpoždění spojů.

Kvalitativní požadavky

- Řešení bude schopno při jedné aktualizaci zpracovat alespoň 1 000 vzorků poloh vozidel, z čehož 10 % vzorků může být o dosud 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 počítat odhad zpoždění lépe (až na výjimku 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 okamžiku zveřejnění by již nebyl platný, nedává smysl zpoždění odhadovat pokročilejší metodou.

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 s vlastními daty. Takové služby mohou být provozovatelem zpoplatněny, ale pro naše demonstrační účely, kdy budeme využívat tuto službu velmi málo, bývá od poplatků většinou upuštěno.

Jedním z těchto poskytovatelů je společnost Google, která má propracované mapové podklady a prostřednictvím služby Google Maps poskytuje pro tuto práci požadovanou službu nazývanou Google My Maps¹³.

Další platforma je Mapbox¹⁴, která poskytuje s využitím dalších knihoven velmi podobné služby jako Google My Maps. Nicméně narozdíl od Googlu využívá jako mapový podklad OSM (otevřená geografické data).

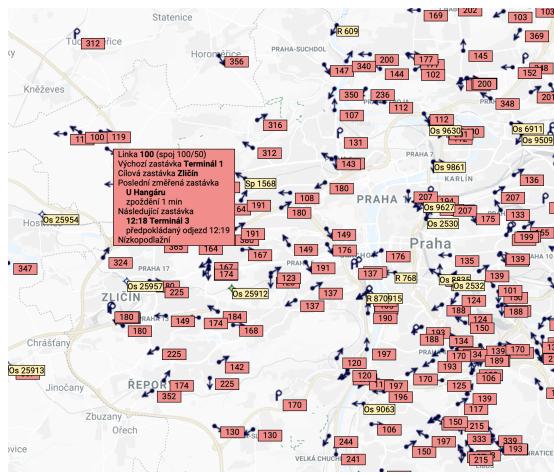
Protože smyslem práce je v co největší míře využít otevřená data, je žádoucí využít právě službu Mapbox.

¹³<https://www.google.com/maps/about/mymaps/>

¹⁴<https://www.mapbox.com>



Obrázek 1.8: Mapa z golemio.cz.



Obrázek 1.9: Mapa z www.tram-bus.cz.

1.4.2 Současná řešení

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

Golemio

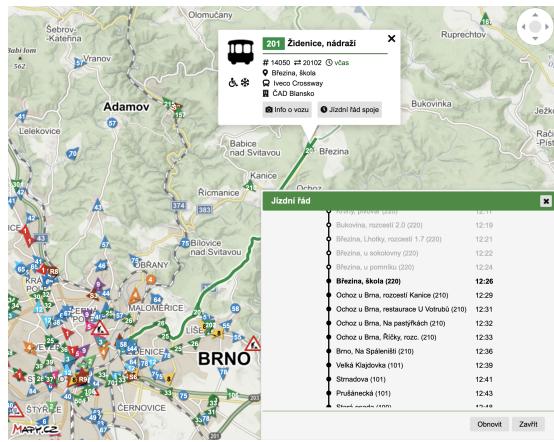
Takovou mapu zobrazuje i samotný provozovatel datové platformy. Nicméně nejsou zde vidět ani čísla linek zobrazených autobusů, natož pak nějaké další informace. Příklad vizualizace je uveden na obrázku 1.8.

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 zpozdění a nejbližší zastávce. Pozn.: na mapě jsou vidět spoje DPP, protože v době psaní této práce již byla data veřejná. Příklad vizualizace je uveden na obrázku 1.9.

IDSJMK

Mimo Prahu je velice pěkně zpracovaná aplikace pro zobrazení vozidel IDSJMK (Integrovaný dopravní systém Jihomoravského kraje). Ten ihned po na-



Obrázek 1.10: Mapa z mapa.idsjmk.cz.

čtení stránky zobrazuje všechny dopravní prostředky, tedy tramvaje, autobusy i vlaky, vše s čísly linek. Dále pak umožňuje, po kliknutí na vybraný spoj, zobrazit více informací včetně jízdního rádu. Příklad vizualizace je uveden na obrázku 1.10.

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

1.4.3 Funkční požadavky

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

- Aplikace vykreslí interaktivní mapu Prahy, Středočeského kraje a širšího okolí obsluhovaného pražskou integrovanou dopravou (dále jen PID) do webového rozhraní, kterou bude možné posouvat či zoomovat.
- V této mapě budou zobrazena 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, 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.

1.4.4 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.

Serverová část včetně databáze bude schopná obsloužit desítky dotazů za sekundu, a to pouze pro demonstrační účely. Účelem práce tedy není budování robustního back-endového řešení schopného produkčního nasazení.

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 databáze, 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.

2.2 Architektura

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

Návrh architektury vychází z popsaných funkčních požadavků a analýzy dat.

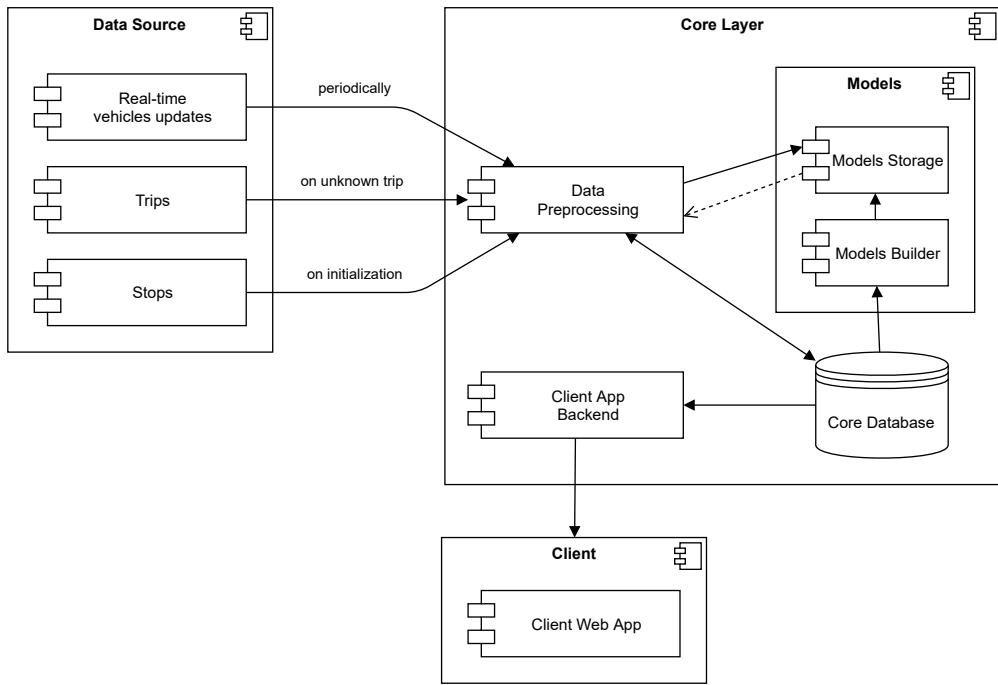
Moduly Data Preprocessing a Client App Backend musí být odděleny, protože přistupují do databáze nezávisle. Z logiky věci: stahování dat databázi plní a backend klientské aplikace data z databáze čte.

Modul starající se o modely je také oddělen od zbytku systému, protože bude volán zcela nezávisle na stahování dat i back-endu klientské aplikace. Konstrukce modelů bude číst data z databáze a vytvořené modely ukládat na disk počítače. Program plnící databázi pak bude pouze číst tyto vytvořené modely a pomocí nich odhadovat zpoždění.

Odhadnuté zpoždění se spočítá v modulu Data Preprocessing při zpracování každého jednotlivého spoje, poté se spolu s ostatními daty o spoji vloží do databáze jako aktuální zpoždění daného spoje.

2.3 Zpracování vstupních dat

Struktura uložení dat se bude zakládat na struktuře zdrojových dat popsaných v kapitole 1.3.



Obrázek 2.1: UML diagram návrhu aplikace

Na datové platformě jsou historická real-time data o vozidlech dostupná řádově v jednotkách minut, což je naprosto nedostatečné pro jakékoliv 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 ze 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.

2.3.1 Databáze

Za tímto účelem tato práce bude využívat relační databázi obsluhovanou dotazovacím jazykem SQL. Struktura databáze je navržena na ER diagramu 2.2. Tento návrh vychází ze struktury vstupních dat a požadavků na softwarové řešení. Tato databáze se skládá z 5 tabulek. Jsou jimi:

- **trips** – všechny objevené spoje
 - **id_trip** – unikátní identifikátor používaný v databázi
 - **trip_source_id** – identifikátor spoje převzatý ze zdroje dat
 - **id_headsign** – identifikátor nápisu pro daný spoj
 - **current_delay** – aktuální zpoždění spoje
 - **shape_dist_traveled** – aktuální vzdálenost ujetá od výchozí stanice
 - **last_updated** – čas poslední aktualizace, převzatý ze zdroje dat
 - **trip_no** – číslo dané linky

- **headsigns** – nápisy nad vozidlem, cílová stanice
 - **id_headsign** – unikátní identifikátor nápisu
 - **headsign** – text nápisu
- **trip_coordinates** – všechna historická real-time data
 - **id_trip** – identifikátor spoje, 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 spoje
- **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** Je tabulka jízdního řádu každého spoje, tvořená seznamem zastávek s časy odjezdů a příjezdů. Pořadí zastávek, v jakém jsou spojen obslouženy, je určeno časem příjezdu, resp. odjezdu nebo atributem **shape_dist_traveled**.
 - **id_trip** – identifikátor spoje
 - **id_stop** – identifikátor zastávky
 - **arrival_time** – čas příjezdu spoje do zastávky
 - **departure_time** – čas odjezdu spoje ze zastávky
 - **shape_dist_traveled** – vzdálenost zastávky od výchozí zastávky spoje

Ve zdroji dat jsou atributy se jménem **source_id** unikátní identifikátor entit, nicméně z dokumentace zdroje nevyplývá, jakého jsou datového typu. Také proto je tento identifikátor ukládán jako textový řetězec, ačkoli je tvořen pouze číslicemi a podtržítky, není nikde zaručeno, že jej lze jednoduše převést na číselný kód. 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. Přirozeně indexovány jsou primární klíče, což jsou ve všech tabulkách unikátní identifikátory typu INT. Dále indexujeme také unikátní identifikátory přebíráné ze zdroje dat. A všechny atributy, podle kterých vyhledáváme v databázi, ať už dotazem přímo z kódu aplikace nebo některou z funkcí napsanou přímo v jazyce SQL. Takové indexy jsou:

- `headsigs.headsign` – unikátní index, textová hodnota nápisu
- `trips.id_headsign.` – pro vyhledání všech spojů s danou konečnou stanicí
- `trips.trip_source_id` – pro vyhledávání spoje podle zdrojového id
- `rides.shape_dist_traveled` – pro setřídění jízdního řádu
- `trip_coordinates.shape_dist_traveled` – pro vybrání vzorků poloh mezi dvojicí zastávek

Databáze je nastavená tak, aby umožňovala získat všechny potřebné informace o vozidlech, hlavně přístup k historickým real-time datům, a to separovaně pro dvojici 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.

Konečnou strukturu vytvořené databáze najdeme na EER diagramu 2.3. SQL dotazy na sestavení celé databáze jsou definovány v příloženém souboru `database.sql`. Pro testovací, debuggovací a demonstrační účely jsou vytvořeny navíc i jiné databáze, které jsou strukturou totožné jako produkční databáze.

2.3.2 Plnění databáze

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

Algoritmus:

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

Pokud se vyskytne jízda, která postrádá některou z požadovaných informací, je automaticky zahozena. Jelikož všechny informace ukládané do databáze, jsou důležité pro hlavní cíl této práce. Vkládání do databáze se provádí na několika místech v kódu aplikace, zachování konzistence dat je proto řešeno pomocí databázových transakcí tak, aby stav databáze byl vždy konzistentní.

Databázové 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 nebo zjistíme, že nám podstatné informace chybí, můžeme v jakékoli fázi provádění změn všechny dosud provedené 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.

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 přijímána vizualizačním nástrojem ve formátu GEOJSON, tedy tato data není nutné vůbec transformovat a není nutné je držet v hlavní databázi.

2.4 Algoritmus odhadu zpoždění

Z formulování problému vyplývá, že se má odhadovat zpoždění mezi dvěma referenčními body a jediné takové jsou zastávky na trase daného spoje. Cíl algoritmu může být tedy formulován, jako vytvoření popisu průběhu trasy mezi každou dvojicí zastávek, kterou alespoň jeden spoj obsluhuje a jsou bezprostředně sousedící ve sledu zastávek ve směru jízdy tohoto spoje. Nechť se všechny dvojice 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 označují jako S_{ab} . První zastávku libovolné dvojce zastávek z AB označíme a a následující zastávku b .

Z definice problému chceme modelovat jízdu vždy mezi danou dvojicí zastávek. Ke každé dvojici zastávek (a, b) bude náležet jeden model, resp. modely pro pracovní dny a dny pracovního volna popisující průběh jízdy mezi nimi. Tyto modely budou vycházet z historických dat průjezdů mezi těmito zastávkami.

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

2.4.1 Základní předpoklady

Na začátku 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 nástupiště¹ – pro každý směr jedno. Pokud má zastávka více nástupišť, je tak v případech, kdy ze zastávky odjíždí spoje do více směrů, a tudíž pro každé 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 odhadu zpoždění a to tak, že termín zastávka a termín nástupiště splývají.

¹Rozepsáno v kapitole 1.3.3

Všechny spoje S_{ab} bez ohledu na linku nebo dopravce jedou ze zastávky a do zastávky b po stejné trase a vzdálenost je tedy konstantní. – Předpokládá se, že žádný 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, že žádný z dopravců nedisponuje právem přednosti v jízdě před jiným dopravcem nebo výrazně výkonnějším vozidlem. Dojezdové časy mohou být ovlivněny jen charakterem řidiče, avšak toto není zjistitelné z poskytnutých dat a zároveň se předpokládá, že charakterystika řidičů jsou rovnoměrně rozloženy mezi všechny dopravce a linky. Podle jízdních řádů některé linky jedou ve stejnou denní dobu rychleji než jiné, avšak skutečná doba jízdy je stejná. Tím, že některý spoj zastávku pouze projíždí, a tedy je rychlejší než jiný spoj, není porušení tohoto předpokladu, protože se jedná o dvě různé dvojice zastávek.

Během jízdy mohou nastat mimořádné události, 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.

2.4.2 Návrh modelování

Na úvod uvedeme, že ve všech grafech a následně i pro počítání modelů, byly všechny zobrazené vzorky poloh vozidel v grafech zarovnány tak, aby jejich jízda ze zastávky a vždy začínala se zpožděním 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 poskytovatele dat. Takové chyby však vznikají spíše výjimečně, a proto ovlivňují statistické výpočty jen málo. Pro eliminaci těchto chyb je pak implementován modul, který se pokouší očividně chybné vzorky najít a smazat. To také z důvodu vizualizace vzorků v grafech, kde jeden vzorek, zcela mimo škálu jiných vzorků, rozhodí celý graf, který se tak stává nepřehledným.

Dále si uvedeme druhy profilů jízd a způsoby jejich modelování.

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 dvojicí zastávek a a b .

Ačkoli je snaha tento model nahradit lepším, v některých situacích může jeho použití i nadále dávat smysl. Zejména pak v případech, kdy není k dispozici dostatek dat, nebo je vzdálenost dvou zastávek natolik malá, že nemá smysl ani jakýkoliv odhad zpoždění dělat.

Polynomiální profil jízdy

Po analýze dat poloh vozidel a možných vlivů ovlivňující profil jízdy je patrné, že v průběhu jednoho dne dochází na trase nejvýše k několika výkyvům rychlosti jízdy. Viditelné jako vlny v průběhu dne na grafu 2.4), v 8:30 hod². 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 zrychlí a dále opět vjede do města. Takový model se hodí spíše na delší trasy s plynulou jízdou.

Stejně tak po analýze profilu jízdy v závislosti na ujeté vzdálenosti je na grafu 2.5 vidět, že čas jízdy narůstá také v jistých vlnách.

Jak pozorujeme na grafech výše, je patrné, že vzorky poloh vytváří jisté vlny. Takové vlny se dají dobře popsat modely získané pomocí lineární regrese, resp. polynomiální regrese. Ve strojovém učení se k výpočtu polynomiální regrese využívá algoritmus lineární regrese s upravenými vstupními daty (viz Gareth James a Tibshirani, 2013, Strany 265–268).

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 2.6 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 páár vzorků, které zcela vybočují mimo největší shluky vzorků. V těchto případech se 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 2.7. To je způsobeno tím, že doba jízdy trasy je natolik krátká, že jeden spoj trasu projede za minutu, ale druhý spoj, který se zdrží o zanedbatelný čas (z pohledu problému řešeného v této práci) přijede do následující stanice až za dvojnásobnou dobu. Dále je vysoký rozptyl vzorků způsoben nepřesnostmi při měření polohy vozidel a dalšími možnými problémy. Jelikož se ale jedná o velmi krátké trasy, nemá smysl jejich specifika vůbec řešit, protože spočítaná data by zastarala ještě před zveřejněním.

²časy jsou uvedeny v UTC

Na ukázku uvedme ještě příklad grafu 2.8, kde jedna jízda dosáhla výrazného zpoždění oproti ostatním jízdám. Současně zde můžeme vidět poměrně častou chybu ve vstupních datech, kdy jedné jízdě přísluší vzorky poloh zcela mimo škálu grafu. Po bližším přezkoumání se jízda jeví, jakoby jela opačným směrem. V naší práci, ale nebudeme zkoumat zdroj chyby, ani se snažit data nějak opravit a rovnou tyto vzorky voloucíme.

Model konkávním obalem

Na dalším grafu 2.9 je zobrazen příklad, kdy na trase existuje bod, který určité procento projíždějících spojů zdrží o netriviální dobu. Něco takového nastane, pokud spoje projízdí světelnou křížovatkou nebo místem kde se náhodně tvoří kolona vozidel. Zde dochází ke skokové změně průběhu bodové funkce. Spojité modely, jakým je polynomiální model, by s okolím tohoto kritického místa měly 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 teoreticky 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. Uvažme, 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 $c_y + c_x = c$ sekund³. 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 případného zpoždění spoje je možno rozhodnout až po projetí druhého bodu. Zatímco polynomiální model by svými odhady jen uváděl uživatele v omyl.

Ilustrujme výše popsané na modelovém příkladu na grafu 2.10. Uvažujeme vozidlo jedoucí na trase mezi dvojicí zastávek, které jsou ve vzdálenosti 10 km. Pravidelný čas jízdy je 720 s. Na trase jsou 2 kritické body ve vzdálenosti 1,5 až 2 km a 8 až 8,5 km, ty si můžeme představit jako pomalu pojíždějící kolonu. Na tomto grafu máme vyobrazené 4 možné průjezdy této trasy: vozidlo projede bez zdržení, zdrží se pouze v bodě x , zdrží se pouze v bodě y a zdrží se v obou bodech x i y . O předjetí vozidla, které projede zcela bez zdržení, můžeme rozhodnout až po projetí bodu y . Stejně tak o zpoždění vozidla, které se zdrží v obou bodech, můžeme rozhodnout až v okamžiku zaseknutí se v bodě y . Všechny vzorky vozidel, které by se promítly do prostoru vyznačeného oranžovou barvou, mají z našeho pohledu nezměněné zpoždění.

Jinými slovy na grafu času jízdy 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 u všech spojů, které přijely do následující zastávky včas.

³ c_x značí nabrané zpoždění v bodě x , pro y analogicky

Nejprve k samotnému konkávnímu obalu je potřeba říct, že na množině bodů není definován jednoznačně, jak je vidět na obrázku 2.11 Saeed Asaeedi a Mohades (2013). Pro účely 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. Proto je potřeba přijít se zjednodušením úlohy. Tedy počítat obal pouze pro dvoudimenzionální prostor. Toho se nedá dosáhnou jinak, než diskretizací úlohy a počítání obalu pro každou hodinu zvlášt, 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 spojích, které přijely do cílové zastávky s co nejmenším zpožděním, ale je nutné mít na paměti, že příjezdy podle jízdního řádu nemusí vůbec odpovídat realitě. Proto se zdá být nejlepším řešením 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 určité procento nejbližších spojů k tomuto odhadu.

Z předchozího popisu řešení ovšem vyplývá, 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 sečtením odchylky průměrného příjezdu od příjezdu podle jízdního řádu a následně přičtení této konstanty k odhadnutému 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 zpoždění.

Naštěstí pro nás se v průběhu analýzy dat o polohách spojů nepodařilo najít jediný případ dvojce zastávek, mezi kterými 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 vznikajících 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 je 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 zdrží v obou kritických bodech stejně v časech s vysokou úrovní dopravy. Tyto situace jsou pokryty

v popisu 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, že se použije jistý druh práva přednosti v jízdě pro spoje VHD, čímž se myslí ovládání světelných křížovatek ve prospěch těchto vozidel, čímž se eliminuje dopad na zpoždění průjezdu křížovatkou.

Případ, kdy by model podle konkávního obalu přinesl zlepšení, je na grafu 2.12, který zobrazuje spočítaný model pomocí polynomiální regrese. Polynomiální regrese se sice se skokovou změnou nevypořádala, jak jsme předpokládali. Ale jediný problém nastane na trase před kritickým bodem, kdy model bude všem vozidlům přisuzovat vyšší předjetí. Jak je ale vidět z grafu, odhad se oproti středu vzorků odchyluje nanejvýš o desítky sekund.

Návrh algoritmu pracujícího s konkávním obalem je následující. Nejprve ukažeme konstrukci 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 poslední_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;
poměr = vzdálenost bodu od spodní hranice obalu
    ku vzdálenosti bodu od horní hranice obalu;
odhad_příjezdu = velikost_okna * poměr + 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;

2.4.3 Využití modelů

Z popsaných modelů v naší aplikaci budeme používat lineární model a polynomiální model. Rozhodnutí, který model využít, uděláme podle následujících faktorů.

Pro všechny dvojice zastávek, které jsou blízko sebe, se použije lineární model, protože pro takovou vzdálenost nemá smysl počítat komplikovaný odhad a ani se zabývat možným způsobem odhadu. První důvod je, že takto krátké vzdálenosti mezi zastávkami jsou častým jevem ve městech, ale tento případ nechceme řešit ze stejných důvodů. Dále na této krátkých trasách, které mají i krátký čas jízdy velmi vynikají nepřesnosti v měření. Např. odchylka v měření 10 sekund na trase, jejíž projetí trvá 100 sekund, se projeví velmi výrazně.

Lineární model zvolíme také v případě, že nemáme dostatek dat pro výpočet polynomiálního modelu. Kolik dat je dostatečných se počítá jako funkce délky trasy a délka části dne ve které jsou zastávky obsluhované.

Na závěr, pokud i přes počítání polynomiálního modelu zjistíme, že takový model má horší výsledky než model lineární, nemá smysl využívat polynomiální model.

2.5 Vizualizace dat

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

2.5.1 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.

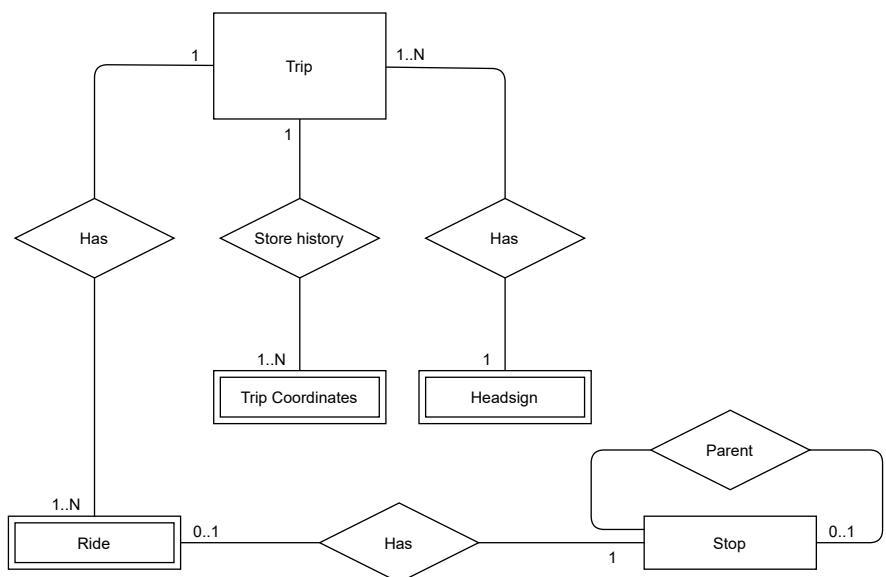
Externí mapové podklady mohou být zobrazeny v několika barevných variacích, umožňují zobrazení ortofoto a také zobrazení s důrazem na různé mapové vrstvy. Pro nás je nejvíce žádoucí zobrazit vrstvu ulic a cest a dále pak určité orientační body jako jsou budovy, vodní plochy, lesy atp. Z palety barev je pak dobrou volbou neutrální béžová barva. Příklad takové mapy je na obrázku 2.13.

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 2.14.

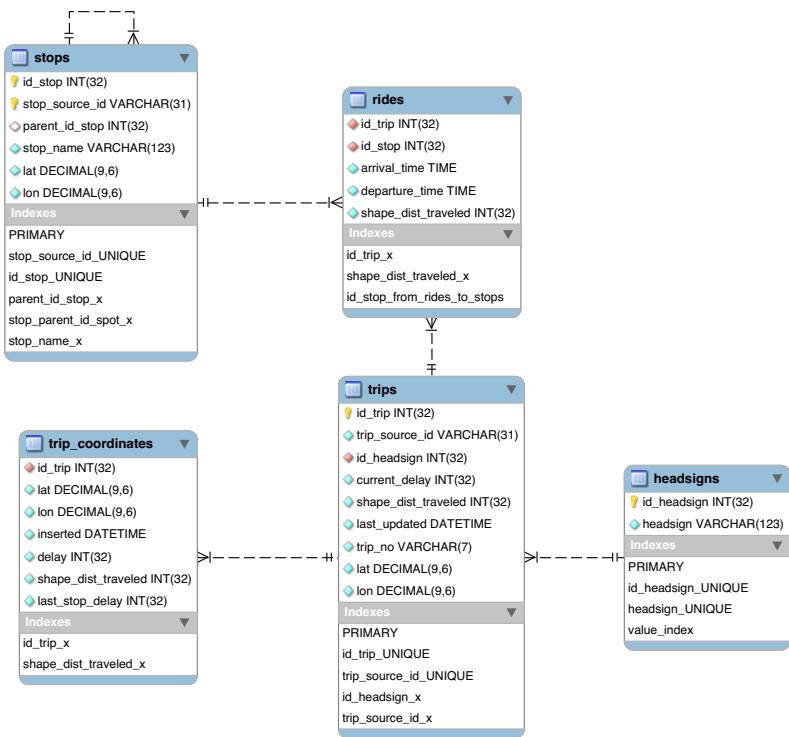
Pro zvolené vozidlo se vykreslí celá jeho trasa včetně všech zastávek. Trasa je reprezentována lomenou čarou a zastávky jako špendlíky v mapě, po přejetí symbolu zastávky se zobrazí i její název. Za zvoleným vozidlem je zobrazena i historie jeho jízdy za uplynulých několik minut, to pomocí barevné lomené čáry, tvořící ocas zvoleného vozidla. Rovněž k povšimnutí na obrázku 2.14.

Další informace o spoji, nebo zastávce se zobrazí v tabulce, která z části překryje mapu. Tato tabulka bude obsahovat informace o zvoleném vozidle, konkrétní konečné stanici, jeho zpoždění a celý jízdní řád. Respektive informace o zvolené zastávce – název zastávky a všechny spoje, které zvolenou zastávkou budou projíždět včetně jejich pravidelného odjezdu a aktuálního zpoždění. Po kliknutí na zastávku⁴ se tedy zobrazí její odjezdová tabule.

⁴V tomto případě myslíme opravdu celou zastávku se všemi jejími nástupiště a nikoli pouze příslušné nástupiště. Protože se nemůžeme spolehnout na vytvořené relace označující všechny rodičovské nástupiště ve zdrojových datech, budeme za nástupiště příslušné jedné zastávce považovat nástupiště se stejným názvem.

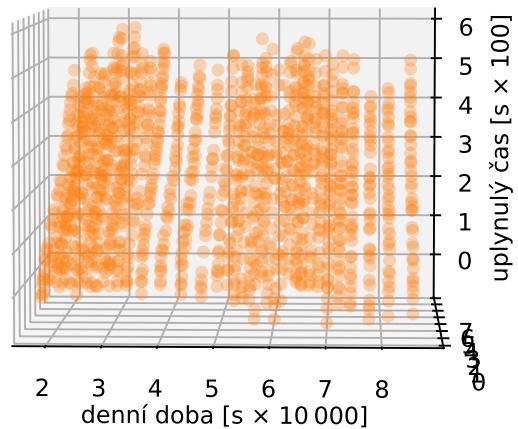


Obrázek 2.2: ER diagram návrhu databáze



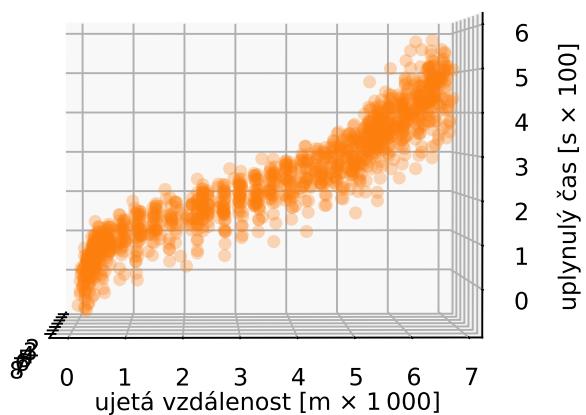
Obrázek 2.3: EER diagram návrhu databáze

K Letiště → Zličín



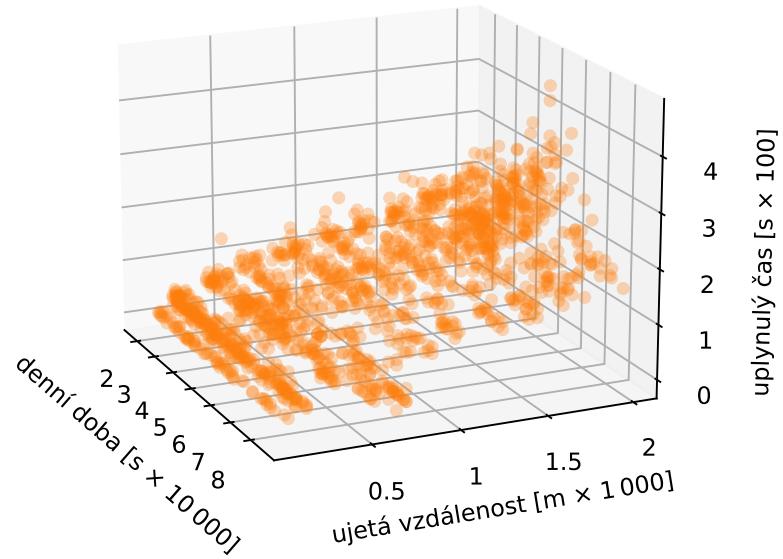
Obrázek 2.4: Variabilita délky jízdy v průběhu dne

K Letiště → Zličín



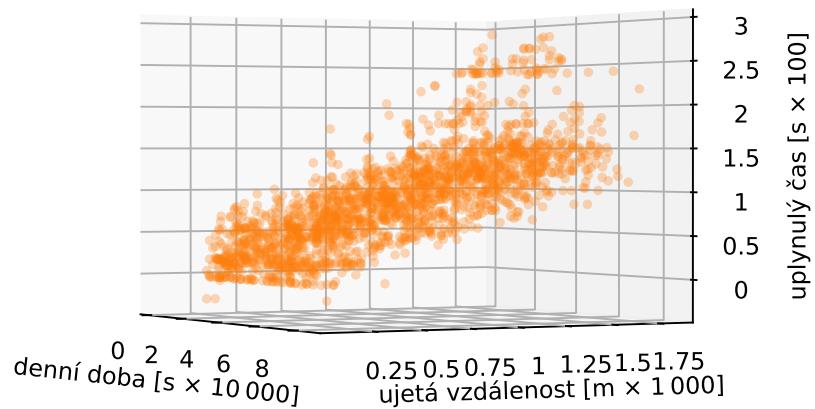
Obrázek 2.5: Variabilita času jízdy v závislosti na ujeté vzdálenosti

Černý Most → Chvaly



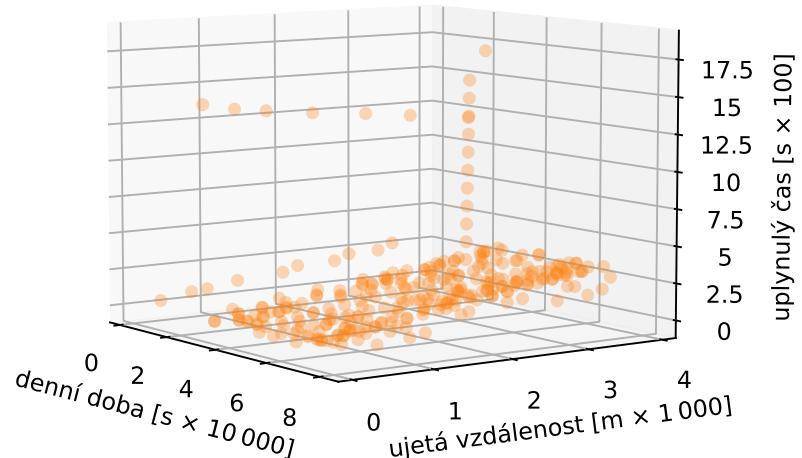
Obrázek 2.6: Úsek s nepravidelnostmi

Divoká Šárka → Nádraží Veleslavín



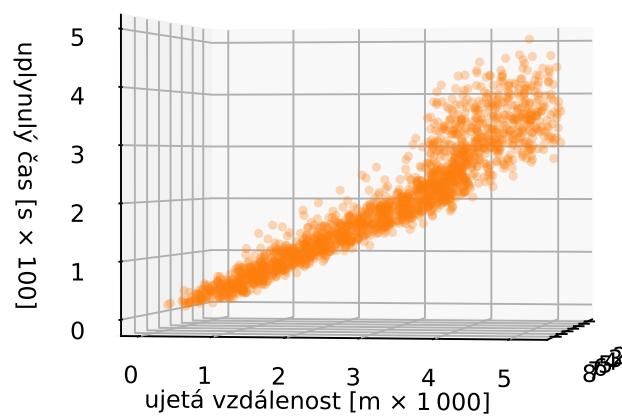
Obrázek 2.7: Úsek s nepravidelnostmi 2

Štěchovice → Slapy,Na Polesí

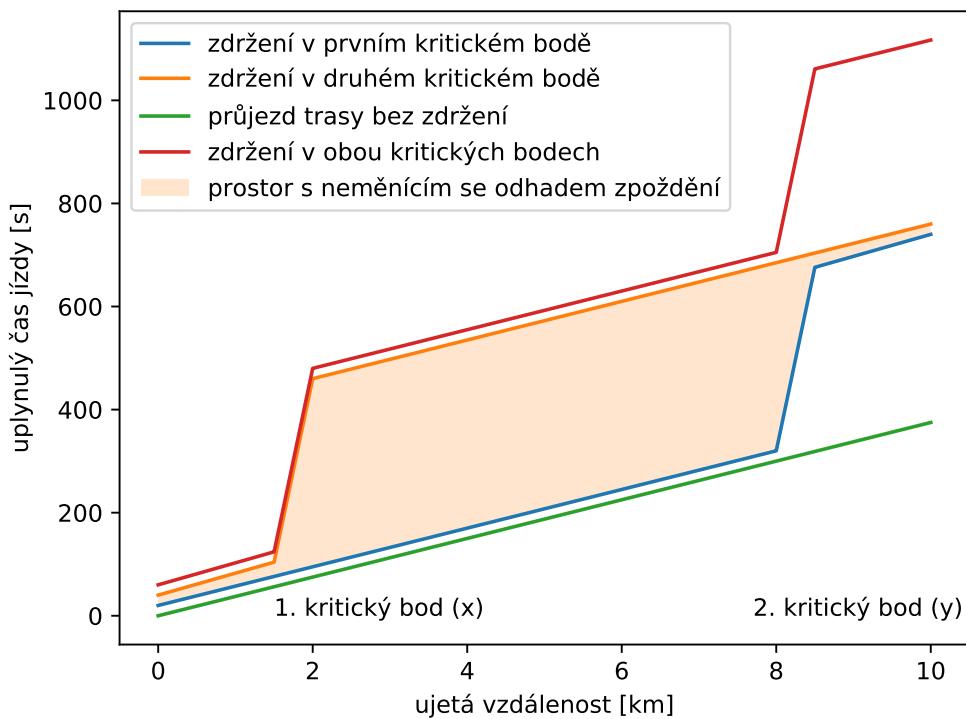


Obrázek 2.8: Výrazné zpoždění jedné jízdy a chybný směr jízdy

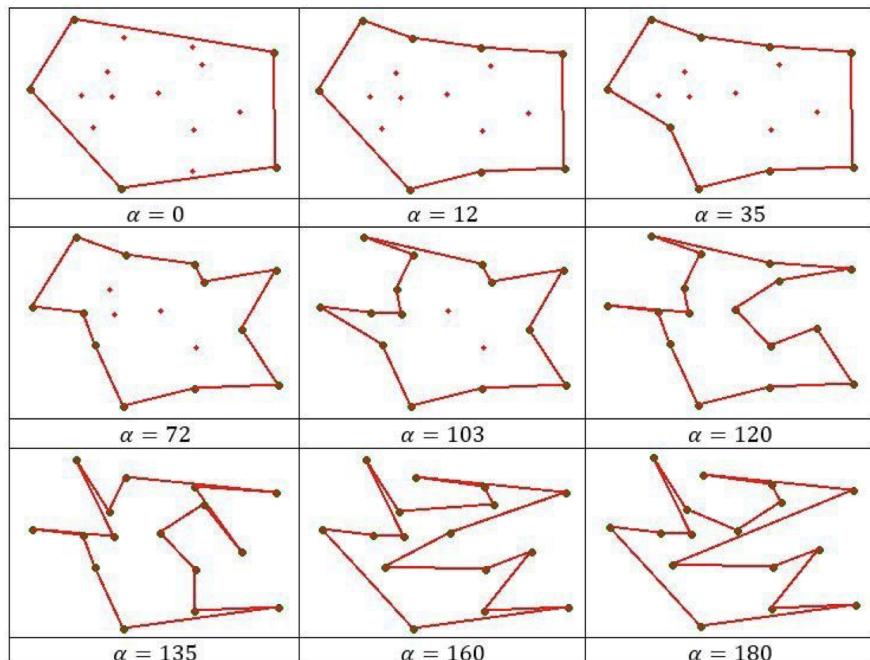
Měchenice,Rozc.k Žel.st. → Jíloviště,Výzkumný ústav



Obrázek 2.9: Úsek s nepravidelným zpožděním

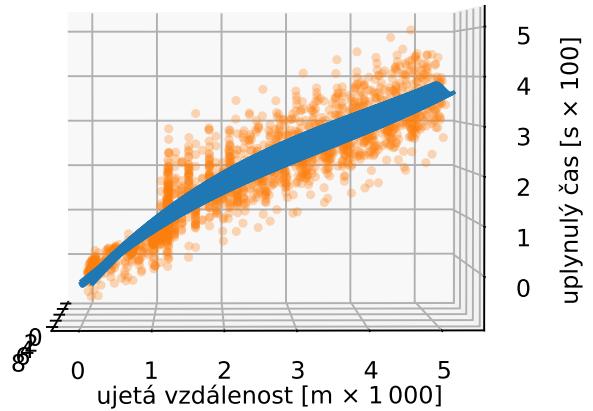


Obrázek 2.10: Modelový příklad profilu trasy dobře popsatelného konkávním obalem

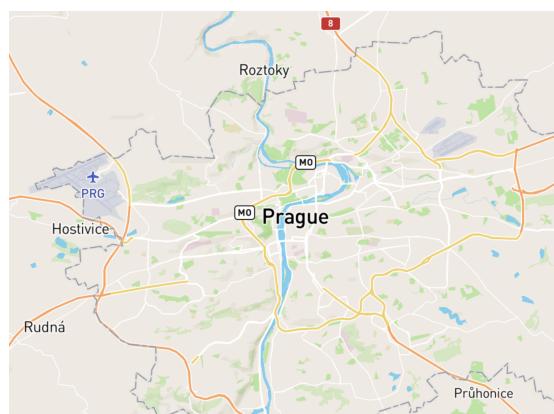


Obrázek 2.11: Nejednoznačnost konkávního obalu

Jíloviště, Výzkumný ústav → Měchenice, Rozc.k Žel.st.



Obrázek 2.12: Chyba polynomálního modelu



Obrázek 2.13: Mapbox Mapa, použitý styl mapy: streets-v11



Obrázek 2.14: Design zobrazení elementů v mapě, linka je 348 je vybrána

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 jazyka Python 3, konkrétně se jedná o knihovnu scikit-learn¹ a další nástroje pro práci s velkými daty jako je knihovna NumPy². Tyto knihovny implementují dobře známé algoritmy umělé inteligence, včetně optimalizací a pomocných funkcí zjednodušujících hledání nejlepšího modelu, dále pak i užitečné funkce pro počítání statistik. Navíc jsou tyto knihovny naimplementovány s ohledem na vysokou výkonost³ a využití grafických karet⁴. Proto nedává smysl jejich služeb nevyužít. Využití jazyka Python 3 je tedy pro jádro naší práce jasnou volbou.

Pro samotné zpracování dat žádné speciální požadavky nevyvstávají a je možné využít i jiné ověřené back-endové programovací jazyky a technologie. Nicméně pro zachování jednoty vývoje není důvod měnit prostředí a využijeme též jazyk Python 3. Může být namítnuto, že programy psané v jazyce Python 3 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.

Pro naši databázi jsme již vybrali MySQL⁵ implementaci. To zejména z důvodů, že se jedná o open source projekt a tato implementace je velmi často využívaná v celé řadě jiných projektů s velkou komunitou. Konkrétně pro Python 3 existuje knihovna MySQL Connector⁶ přes kterou je možné SQL databázi pohodlně obsluhovat.

Celá aplikace je ovládaná přes hlavní skript, který je 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 – vybrat si můžeme mezi demonstračními daty, vývojovými daty nebo real-time daty. Dále je možné spuštěním skriptu vytvořit model profilů jízd a také 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 popsaném v kapitole 1.3 je taková, že data se budou periodicky stahovat a ukládat do SQL databáze, tento postup je popsán v kapitole 2.3.

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 nezasahovaly do kódu implementující logiku systému. Stejně tak se tímto eliminuje výskyt paternů v celém kódu. Konkrétně se tím myslí komunikace s databází, komunikace se zdrojem dat a 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 zpoždění 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 protože 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. Dále se pak sadou funkcí hledá poslední projedána a následující 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 odhadnout zpoždění. To ovšem nehráje velkou roli, protože se tak stane pro každé vozidlo ihned po vyjetí z výchozí stanice, nebo ještě před vyjetím, kdy vozidlo stojí ve výchozí zastávce. V těchto případech nemá počítání odhadu zpoždění velký význam. V další iteraci již jízdní řády spoje budou známy, 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 zvlášť.

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 kód je volán pro každé vozidlo zvlášť.

```
# najde model podle dvojice 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 asynchronně, 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 poslední projeté 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

```

3.3 Konstrukce modelů

Pro spočítání polynomiálního modelu se využívá knihovna sklearn konkrétně algoritmus zvaný Rigde, který sám o sobě hledá lineární závislosti. Nicméně vstupní hodnoty jsou mezi sebou náležitě pronásobeny tak, aby simulovaly 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 Python 3 za pomocí knihovny sklearn provede následujícím kódem. Omezení stupňů polynomiální regrese vyplývá ze zkušenosti, kdy modely pro vyšší stupně jsou více chybové, protože došlo k tzv. přeučení modelu.

```

for degree in [1, 2, 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)

```

V kódu funkce je možné si povšimnout, že model trénujeme na trénovacích datech a testujeme na testovacích datech. Do této podmnožin jsme rozdělili vstupní data ještě před zavoláním této funkce. Toto rozdělení nám slouží k nalezení nejlepšího stupně polynomu, tedy hyper parametru modelu (viz Ripley, 1996, Strana 365, validation set a training set). a provede se následující funkcí v balíčku sklearn.

```

X_train, X_test, y_train, y_test =
    train_test_split(input_data,
                     output_data, test_size=0.33, random_state=42)

```

3.3.1 Čtení dat

Samotné nalezení správného modelu se nyní může zdát jednoduché, nicméně nejsložitější prací pro jakoukoli úlohu z oblasti umělé inteligence a strojového učení je příprava dat a v naší práci tomu není jinak. Ať už se jedná o samotná zpracování dat popsané výše v kapitole 3.2, tak také je potřeba tyto zpracovaná

data dále transformovat 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 ve vstupních datech spousta a mohly by negativně ovlivnit správnost odhadů nalezených modelů.

Pro zkonztruování modelů popisujících profily jízd mezi všemi dvojicemi zastávek je nejprve potřeba zjistit všechny dvojice 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 2.3.1

Dále pokud máme všechny dvojice zastávek, je potřeba získat všechny oznámené polohy vozidel mezi nimi pro každou dvojici zastávek zvlášt. Tyto dva kroky 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 + 99
      BETWEEN schedule.shape_dist_traveled AND
           schedule.lead_stop_shape_dist_traveled - 99
ORDER BY id_stop, lead_stop, shifted_shape_trav

```

Tento SQL dotaz nejprve získá všechny dvojice 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 dvojice 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 od sebe vzdáleny méně nebo přesně 1500 m (v implementaci je pak vzdálenost určena parametricky), zdůvodnění této vzdálenosti je uvedeno 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 zkušeností, jak rychle se vyvíjí 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 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 dvojcí 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 výrazně mimo cluster všech ostatních, tak není odstraněn pouze tento jeden vzorek, ale rovnou všechny vzorky dané jízdy. To proto, že s vysokou pravděpodobností jsou ovlivněny chybou i ostatní vzorky, ale nesplňují poměrně volná kritéria na odstranění. Hledání chyb pak probíhá tak, že se spočítá poměr čas jízdy ku vzdálenosti všech vzorků 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()

# vydělí každý prvek pole,
# abych nepracovali s příliš malými čísly v následujícím kroku
norm_shapes = np.divide(self.norm_data.get_shapes(), 100)

# vydělí dvě pole podle vzorce r[i] = a[i]/b[i]
# a tím zkiskáme poměr času a vzdálenosti pro každý vzorek
rate = np.divide(coor_times, norm_shapes, where=norm_shapes!=0,)

# ošetření krajních případů
for i in range(len(rate)):
    if rate[i] == np.inf:
```

```

        rate[i] = 0.0
    if rate[i] is None:
        rate[i] = 0.0

# normalizace prvků v rate podle vzdálenosti
if max(norm_shapes) > 1:
    tmp = []
    for i in range(len(rate)):
        tmp.append(rate[i] * (1 - ((max(norm_shapes) -
            norm_shapes[i]) / max(norm_shapes)))))

rate = np.array(tmp)

# výběr indexů prvků pole výrazně převyšující rozptyl
high_variance = np.where((
    abs(rate - np.median(np.array(rate))) >
    rate.std() * 4 + (np.median(np.array(rate)))
).astype(int) == 1)[0]

# zjistí id spojů s vysokým rozptylem,
# vybere všechny vzorky daného spoje
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 uvedenému výše.

3.3.3 Práce s modely

Pro vložení odhadnutého zpoždění do databáze je potřeba využít před vypočítané 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 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 tomto případě 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, vůbec nejake nemění. Ošetření této situace se však může mít vliv, 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 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 nepřepočítává a zobrazuje jen data taková, která obdržela od serverové strany typicky ve formátu GEOJSON. Pro aktualizaci dat je potřeba vyvolat nový dotaz typicky se stejnými parametry.

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 odlišně nebo stáhne přídavná data k zobrazení.

⁸v této práci se vždy využívá časová zóna UTC

Mapbox API

Nejprve si popíšme, 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. Mapové prostředí intuitivně interaguje s uživatelem a vývojáři mohou využít jednoduchého API pro zobrazení žádoucích dat do mapy.

Webová aplikace této práce využívá naprosto 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 nainstalování je v JavaScriptu k dispozici knihovna jménem `mapboxgl`, pomocí které se ovládá celé mapové prostředí. Nyní je 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 poté může být přidán do mapy následující funkci. 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 můžeme se k nim tedy 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~mapě
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 ocasů 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 stará 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. Zároveň se stáhnou další informace o vozidle, které se následně zobrazují. Jsou jimi: trasa vozidla, jízdní řád a zpoždění. Vybrané vozidlo se v kódu reprezentuje vlastní třídou `Active_trip`, každá instance této třídy se přidá do proměnné `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 kódem.

```
// 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 proměnné `active_trips` se vloží více vozidel.

3.4.2 Serverová část

Dle příchozích požadavků od klienta odpovídá serverová strana skriptem, který je napojen na databázi a z ní extrahuje potřebná data.

Data jsou posílána v textové podobě ve formátu GEOJSON, které skript 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 Python 3.

Server knihovna

Server je naprogramován pomocí Pythoní knihovny `simple_server`, která slouží pouze k debbugování, jak se píše v její dokumentaci⁹. Protože se nepočítá s reálným nasazením 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 WSGI aplikace, jenž přijímá argumenty `environ`, což je dotaz a atribut `start-response`, 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 GEOJSON. Uvedme si na příkladu, jak probíhá zpracování dotazu na zastávky daného spoje. Poté, co obdržíme dotaz, se z něj přečtou parametry. Pro dotaz na zastávky musí být parametr ve formátu `get_stops.id_trip`. Parsování dotazu a volání příslušné interní funkce se provádí následovně.

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

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

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:

```
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. Testování a evaluace

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

Dosažené výsledky demonstrujeme v řadě případů na grafech. Tyto grafy byly vytvořeny pomocí knihovny jazyka Python 3 matplotlib a zdrojový kód je napsán ve složce TODO.

4.1 Testování softwarového řešení

Kód 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 správnou funkčnost jednotlivých metod všech softwarových komponentů této práce.

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

Unit testy jsou k nalezení v příloze ve složce tests/unit.

4.1.2 Integrační testy

Integrační testy testují propojení jednotlivých modulů. Dále také testujeme správnou funkčnost databáze a její funkce.

Dále se testuje i kompletní běh aplikace, kde se využívají stažená data.

Integrční testy jsou dostupné v příloze ve složce tests/integration

4.1.3 Testy kvality

Všechny následující výkonnostní testy jsou prováděny na osobním notebooku s technickými parametry uvedenými v tabulce 4.1.3¹.

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

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

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

Testování stejně jako funkční a kvalitativní požadavky na práci vychází z analýzy vstupních dat uvedené v kapitole 1.3.3.

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 dvě čtení z databáze. Jeden záznam se aktualizuje a vloží se jeden nový záznam². Pokud je vozidlo již nalezeno a jeho poloha se od poslední aktualizace nezměnila, provede se pouze jedno čtení z databáze. Pokud ovšem je vozidlo obsluhující spoj nenalezeno, musí se číst soubor s detailem 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.

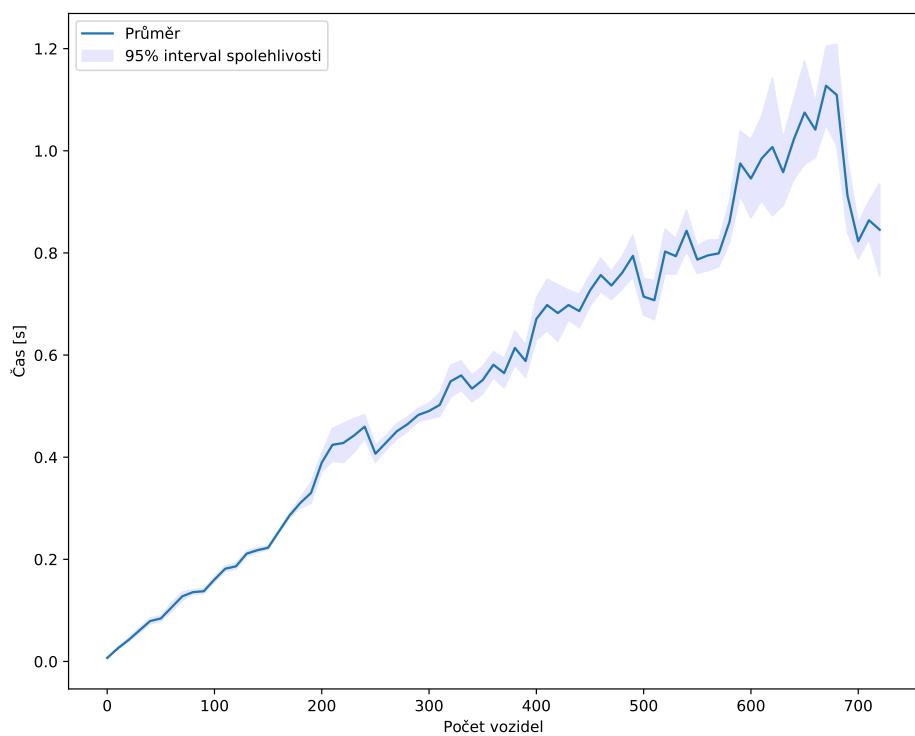
Jediné delší prodlení může nastat ve chvíli, kdy je potřeba stáhnout velké množství dodatečných informací o novém spoji. Na grafu 1.6 je vidět, že až na jednotky výjimek je počet nově nalezených spojů v jednom souboru nejvýše 20. Aplikace je ale naimplementována tak, aby se tyto informace stahovaly asynchronně, a tedy čekání na stažení dat bylo co nejkratší.

Konstrukce modelů

Po využití testovacích dat vzorků poloh vozidel zaznamenaných ve dnech 20.–24. 2. 2020 bylo podle kritérií, kterými jsou zejména vzdálenost zastávek a počet vzorků mezi nimi, sestrojeno celkem 1 106 polynomiálních modelů. Z toho je 847 modelů pro pracovní dny, které jsou nejdůležitější. Přičemž celkový počet párů zastávek je 7 230, ale zastávek ve vzdálenosti 1 500 metrů³ je pouze 2 142. Z toho vychází, že u 40 % dvojic zastávek je dostatek dat, aby dával výpočet modelu smysl.

²ověření existence spoje v tabulce trips, načtení jízdního řádu z tabulky rides, aktualizace dat spoje v tabulce trips a vložení aktuální polohy vozidla do tabulky skladující historická data trip_coordinates

³zvolená minimální vzdálenost mezi zastávkami, mezi kterými má ještě smysl odhadovat zpoždění



Obrázek 4.1: Průměrný čas zpracovávání daného počtu vozidel ze všech souborů se statickými daty s 95 % intervalem spolehlivosti. Počty vozidel jsou vždy zaokrouhleny dolů na celé desítky.

U zbylých dvojic zastávek se využívá lineární model.

Čtení dat potřebných pro trénování modelů z databáze, kde jsou data ze 4 dnů, trvá přibližně 110 sekund. Čtení se totiž provádí komplikovaným SQL dotazem uvedeným v kapitole 3.3.1, ovšem na rychlosť provedení tohoto dotazu i konstrukce modelů celkem neklademe žádné časové nároky, protože přepočítávání modelů je plánováno na čas nejmenšího zatížení systému, což bývá typicky v noci.

Dále ověřme, že výsledek dotazu nezahltí paměť počítače. Dotaz sice umožňuje čtení dat po stránkách, ale v implementaci se tato vlastnost nevyužívá. Určení velikosti objektu jazyka Python 3 v paměti počítače není úplně triviální úloha, protože v paměti není objekt uložen na jednom místě, na části objektu se totiž ukazuje pointery⁴. Dobrý odhad nám, ale poskytne uložit objekt na disk pomocí knihovny `pickle`⁵. Takto uložený objekt zabírá necelých 10 MB prostoru na disku.

Samotné zpracování dat a trénování modelů trvá pro všechny 2 142 páry zastávek přes 2 minuty. Pro jeden páru zastávek je průměrná doba běhu 57 milisekund. Nejdéle trvá výpočet modelů 1,2 sekundy, a to v případě, kdy se zpracovává dohromady přes 10 000 vzorků poloh vozidel.

Serverová část uživatelské aplikace

Server odpovídá na všechny typy požadovaných dotazů.

Server je schopný odbavit přinejmenším 100 dotazů za sekundu. Na grafu 4.2 je zobrazeno jako dlouho trvalo serveru odpovědět na 100 paralelních dotazů, celkem v 1 000 případech. Průměrná odpověď trvala třetinu sekundy a jen v jednom případě jsme na odpověď čekali 0,8 sekundy, tedy stále ještě pod jednu sekundu.

Vizualizace

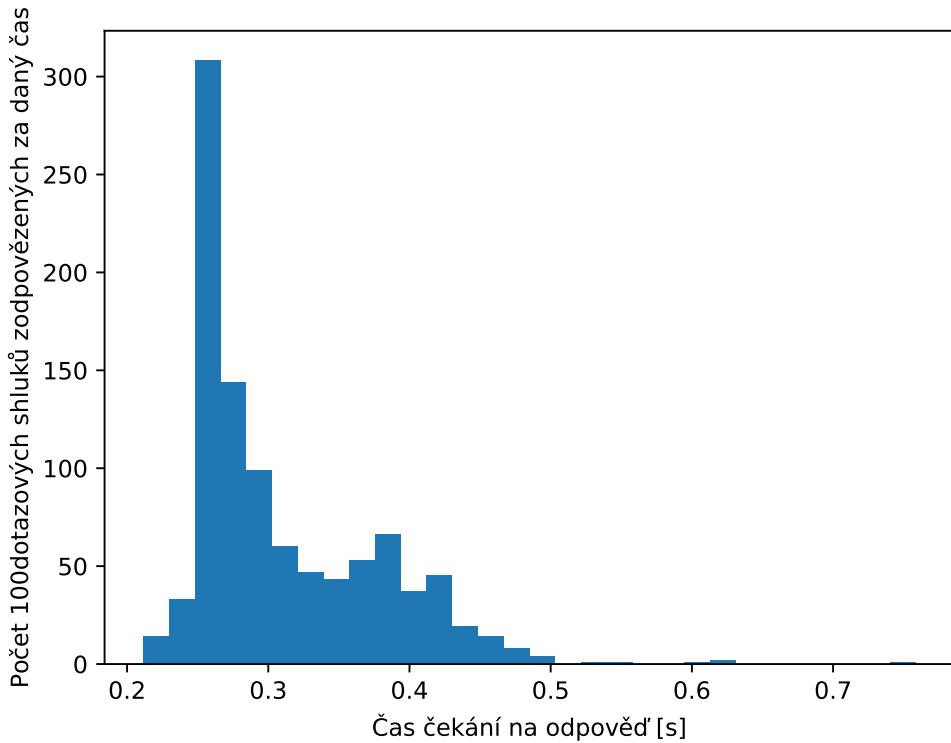
Zátěžové testování vizualizační aplikace se dělá poměrně obtížně. Prakticky je pro testování jakékoli front-endové aplikace vždy potřeba spustit aplikaci na konkrétním zařízení a pozorovat její chování a výkon.

V našem případě pro testování zobrazení vozidel na mapě se spokojíme s testováním ve webovém prohlížeči, konkrétně v Safari verze 14.0.3 a Google Chrome verze 90.0.4430.93. Testování probíhá na datech používaných pro demonstraci běhu celého řešení. Tato data jsou sesbírána za den 23. 2. 2020 večer a zachycují půlhodinový časový interval.

Oba prohlížeče bez jakýchkoli problémů zobrazily v jeden moment více než 100 vozidel a stejně tak zvládly i data aktualizovat. Po celou dobu testování aplikace běžela plynule. Ukázky grafického znázornění jsou zobrazeny na obrázcích 2.14, kde je vybrán jeden spoj a zobrazují se jeho zastávky a jeho trasa. Dále na obrázku 4.3 jsou vidět polohy vozidel tak, jak byly zaznamenány 23. 2. 2020 ve 21:30. Na obrázku 4.4 vidíme způsob zobrazení shluku vozidel, ikony reprezentující vozidla

⁴<https://docs.python.org/3/reference/datamodel.html#objects-values-and-types>

⁵<https://docs.python.org/3/library/pickle.html>



Obrázek 4.2: Doba odpovědi serveru na 100 paralelních dotazů (1000 vzorků)

se překrývají, zároveň ale překryvy nepůsobí nijak rušivě. Navíc po přejetí myší po jakékoli části i částečně skryté ikony jsou přeneseny do popředí tak, aby bylo číslo linky dobře viditelné a bylo umožněno vybrání vozidla klikem myši.

Dále se v pořádku zobrazily i přídavné informace o vybraném vozidle. Tedy všechny zastávky, kterými spoj projíždí (kterých může být i několik desítek) a tabulka s jízdním řádem. Na obrázku 4.5 je vidět zobrazení celé trasy spoje. Vybraný spoj je zvýrazněn a vždy se zobrazí přes všechny ostatní ikony jiných vozidel. Jízdní řád spoje se pak zobrazí v tabulce zobrazené na obrázku 4.6.

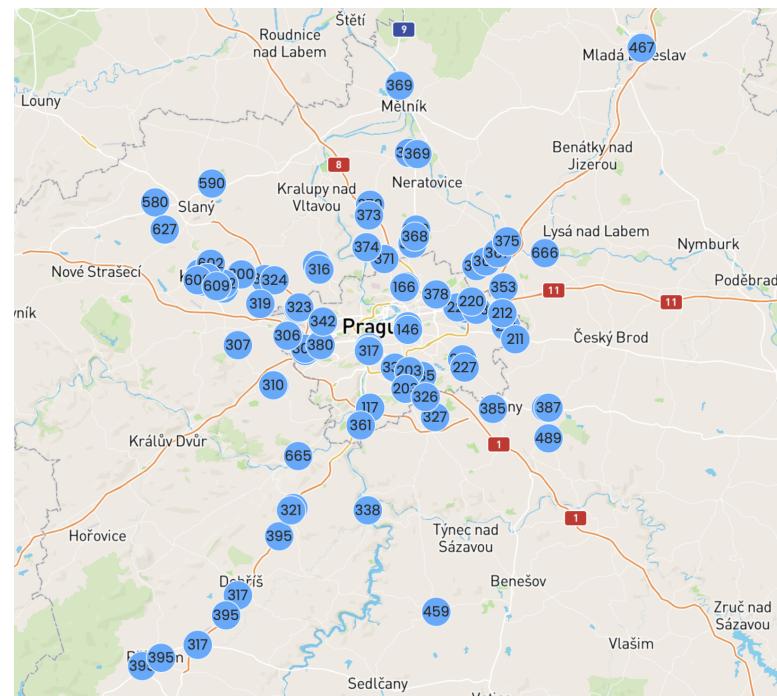
Pokud byla vybrána zastávka, zobrazila se odjezdová tabule a všechny spoje, které budou zastávkou projíždět⁶. Zobrazení více vozidel současně je ilustrováno na obrázku 4.7. Odjezdová tabule je zobrazena na obrázku 4.8

4.2 Evaluace výsledků

4.2.1 Konstrukce modelů

Zde popsaná data vychází z trénování modelů na datech sbíraných ve dnech 20. 2. 2020 – 23. 2. 2020, tedy ze 4 dnů (2 pracovních, 2 víkendových).

⁶V demonstrační aplikaci se zobrazí všechny spoje projíždějící zastávkou, protože časy jízdních řádů neodpovídají simulovaným časům pořízení vzorků poloh vozidel.



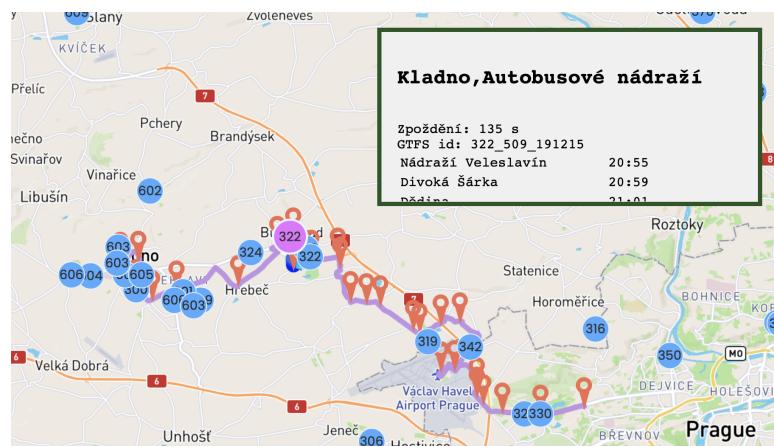
Obrázek 4.3: Mapa Prahy a okolí s polohy vozidel zaznamenány 23. 2. 2020



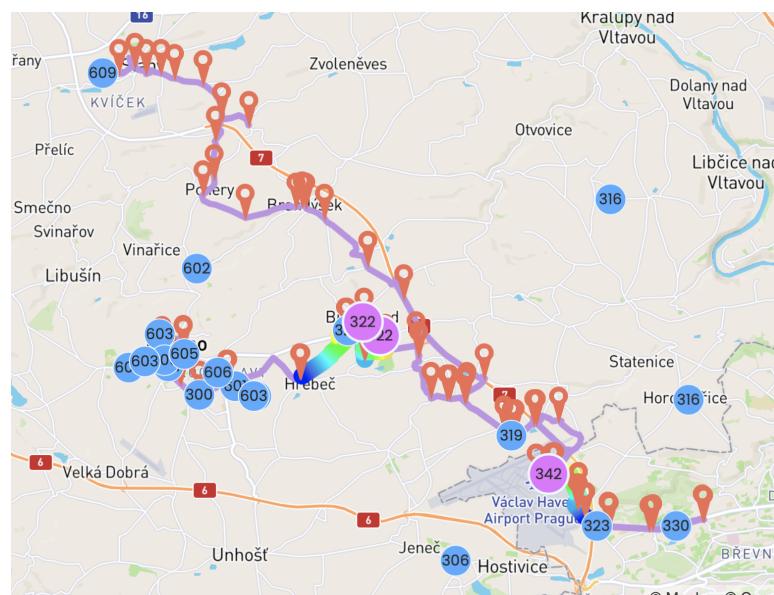
Obrázek 4.4: Shluk vozidel



Obrázek 4.5: Vyznačená trasa jízdy



Obrázek 4.6: Vyznačená trasa jízdy s jízdním řádem



Obrázek 4.7: Vozidla projíždějící vybranou zastávkou



Obrázek 4.8: Odjezdová tabule

Z 2 142 dvojic zastávek se alespoň pro jeden typ dne (pracovní, víkendový) vytvořil polynomiální model popisující profil jízdy mezi nimi pro 1 108 dvojic zastávek. Pro oba typy dnů se polynomiální model vytvořil ve 222 případech. Ve 190 případech se vytvořil model pouze pro pracovní dny, protože mezi danou dvojicí zastávek nejel žádný spoj ve víkendový den.

Tyto vytvořené polynomiální modely jsou součástí přílohy práce a jsou uložené ve složce TODO.

Chceme-li zjistit, jak přesně jsou profily jízdy odhadnutý pomocí vytvořených modelů, zaměříme se na RMSE u každého modelu. Tato chyba nám říká, o kolik sekund se v průměrném případě nás odhad plete od skutečnosti na testovacích datech⁷. Pro pracovní dny je průměrná RMSE necelých 20 sekund. Nejvyšší RMSE byla zaznamenána 200 sekund. Tak vysoké odchylky jsou, po prozkoumání jednotlivých případů způsobeny chybami ve vstupních datech, které se automaticky odstraňují jen velmi obtížně. Příklad je zobrazen na grafu 4.9. V tomto konkrétním případě jsou chybně uvedena zpoždění v poslední projeté zastávce. Histogram všech RMSE modelů z pracovních dnů je zobrazen na grafu 4.10. Zde je patrné, že nejčastěji je chyba do 30 sekund a zcela výjimečně překročí 60 sekund. Takto nízké chyby jsou nad očekávání dobré, protože průměrných 20 sekund je pro cestujícího čekající na autobus takřka nepostřehnutelnými.

Co se týče stupňů polynomiálních regresí, které využíváme při konstrukci modelů, tak nejčastěji se generoval model stupně 3 nebo 4. Což značí, že průběhy tras jsou poměrně tvárné. Modely vyššího stupně se nalezly také, ale spíše značí chybu v datech, protože se tak snaží vymodelovat různě rozházené vzorky, jako třeba na obrázku 4.9. Nakonec se nalezlo i nezanedbatelné množství modelů stupně 2, což také značí tvárnost, ale převážně jen v jedné ose. Vizualizace modelu druhého řádu je na grafu 4.11, třetího řádu na grafu 4.12

4.2.2 Odhad ypozdnení

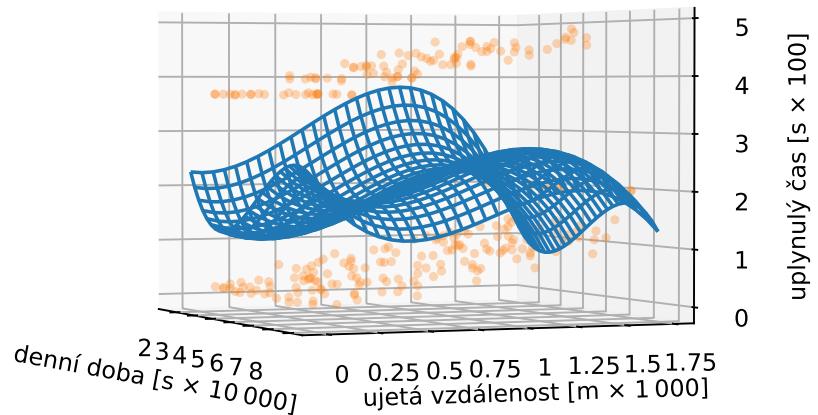
Z toho, jak jsou definovány požadavky řešení v kapitole 1.3.4 pro změření kvality výsledků, stačí porovnávat odhad zpoždění lineárního (původního) modelu a nového polynomiálního modelu. Přičemž odhad je lepší, pokud má sekvence odhadů zpoždění z celé jízdy mezi dvojicí zastávek menší rozptyl⁸.

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. Budeme porovnávat data pouze mezi dvojicí zastávek, mezi kterými byl spočítán polynomiální model. V jiných případech, kdy model spočítán nebyl, se vždy použil lineární model a není tedy co srovnávat.

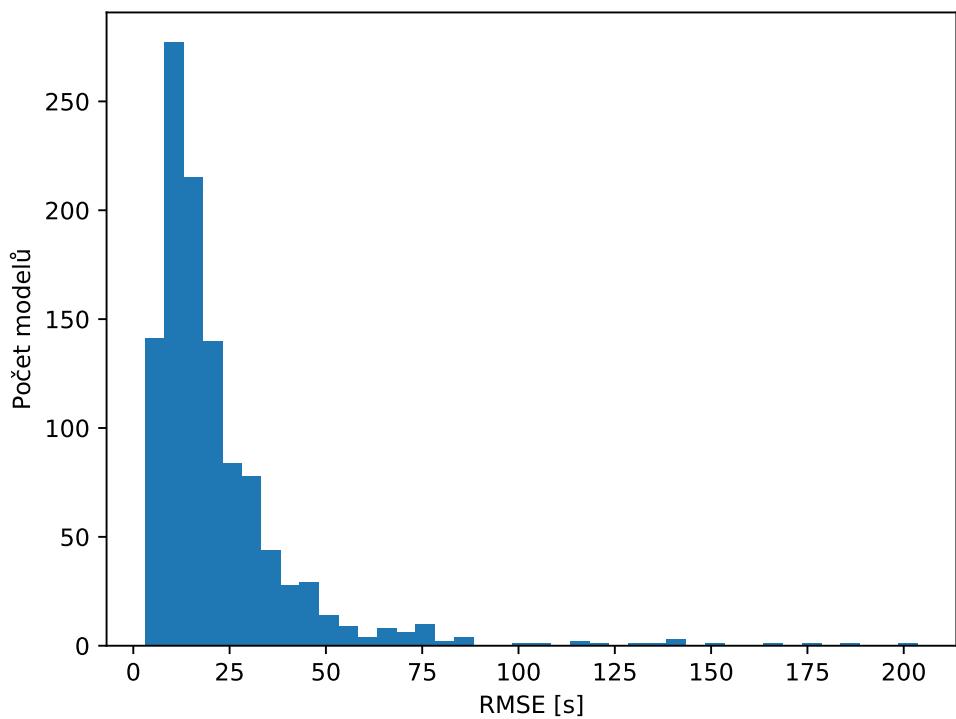
⁷Jedná se o testovací data, podle kterých se určil nejlepší stupeň polynomu. Kdy vstupní data pro funkci trénování modelu byla rozdělena na trénovací a testovací. Testovací data, podle kterých posuzujeme vlastnosti nově odhadnutých zpoždění v kapitole 4.2.2, jsou zcela jiná množina dat (viz Ripley, 1996, Strana 365, validation set a test set).

⁸Níže budeme pracovat s odmocninou z rozptylu, což se formálně nazývá směrodatná odchylka.

Pchery → Pchery,Theodor

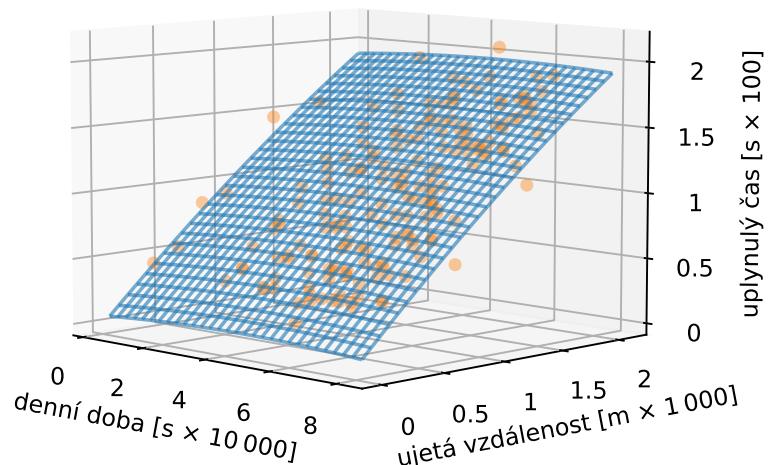


Obrázek 4.9: Modelování profilu jízdy s chybnými vstupními daty



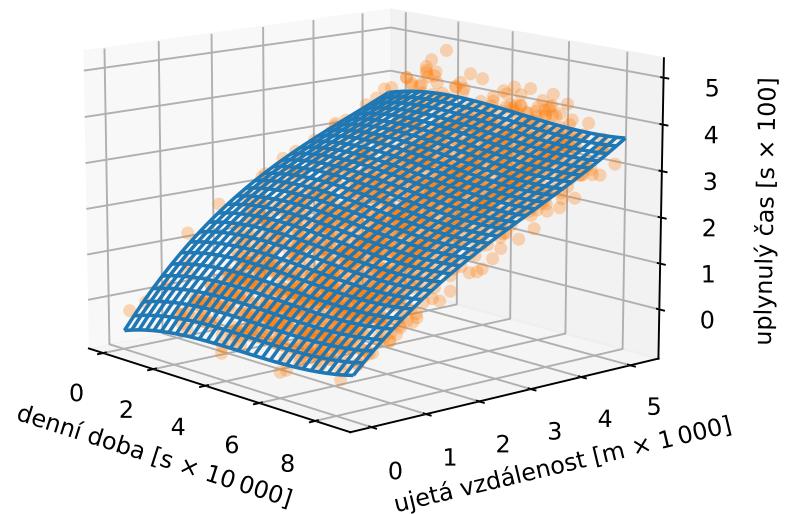
Obrázek 4.10: Histogram RMSE všech modelů

Chotilsko → Chotilsko, Prostřední Lhota



Obrázek 4.11: Vizualizace modelu druhého stupně

Jíloviště, Výzkumný ústav → Měchenice, Rozc.k Žel.st.



Obrázek 4.12: Vizualizace modelu třetího stupně

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 (viz Gareth James a Tibshirani, 2013, Strana 30).

Z dat sesbíraných 20. 2. 2020 se vytvořilo 284 modelů popisující průběh trasy. Tyto vytvořené polynomiální modely jsou součástí přílohy práce a jsou uložené ve složce TODO.

Budeme porovnávat zaznamenaná odhadnutá zpoždění mezi odhadem podle lineárního modelu a polynomiálního modelu. Pro lepší výsledky vyloučíme z porovnání vzorky poloh vozidel, které byly zaznamenány bezprostředně po vyjetí z první zastávky nebo před přijetím do druhé zastávky. Takové vzorky poloh vozidel obsahují často chybná data, která negativně ovlivňují výpočet modelů (data jsou vyloučena i při výpočtu) i porovnání jejich výsledků. Tyto chyby jsou převážně způsobeny zaokrouhlováním atributu ujeté vzdálenosti ve vzorku dat na celé stovky.

Ilustrujme si tento problém na příkladu z reálných dat. V tabulce 4.2.2 jsou zaznamenány vybrané polohy vozidla, které jelo podle jízdního řádu, jehož první dvě zastávky jsou uvedené v tabulce ???. Ačkoli je druhá zastávka spoje ve vzdálenosti 4 840 metrů o výchozí zastávky, podle vzorku poloh vozidla pořízeném ve vzdálenosti 4 800 metrů se změnilo zpoždění v poslední zastávce, což značí, že vozidlo již stojí v druhé zastávce. Avšak kdybychom brali všechny vzorky dat pořízené před projetím bodu 4 840 metrů (druhá zastávka), zahrneme tento bod do výpočtu modelu, resp. evaluace výsledků a negativně nám ovlivní výsledky. Dále vidíme, že po vyjetí ze zastávky má vozidlo konstantní zpoždění v poslední projeté zastávce, tedy vozidlo opustilo zastávku a je na trase.

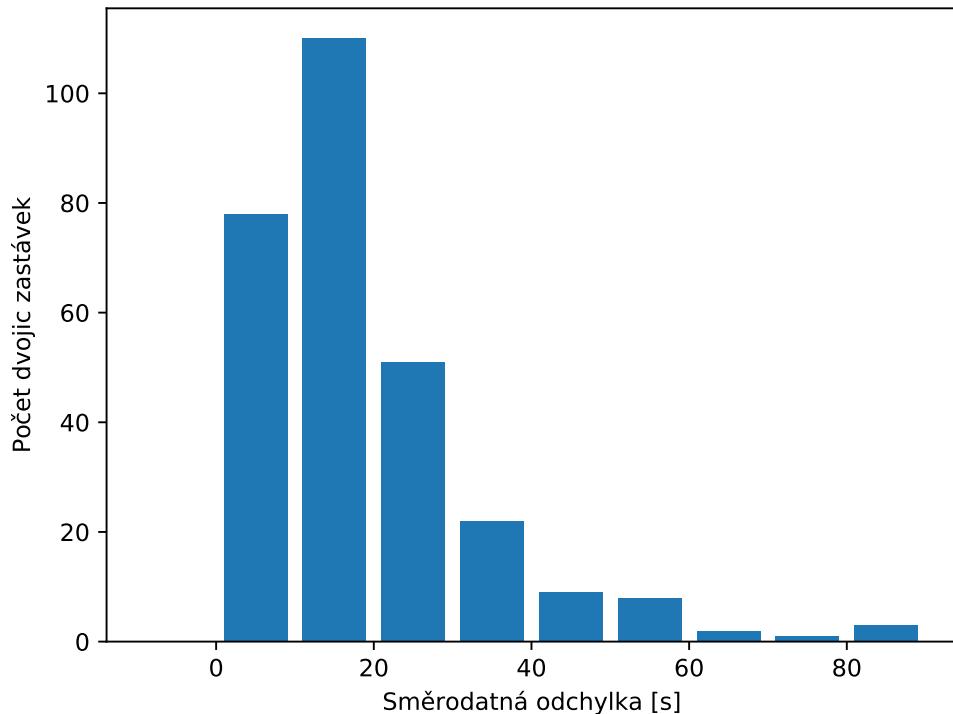
ID spoje	ujetá vzdálenost	zpoždění v poslední zastávce
336_89_181210	4400	61
336_89_181210	4700	61
336_89_181210	4800	84
336_89_181210	4900	92
336_89_181210	5300	92

Tabulka 4.1: Záznamy poloh spoje 336_89_181210.

Nyní se podívejme už na samotné srovnání rozptylů zpoždění, které jednotlivé modely odhadovaly na trasách mezi danými dvojicemi zastávek. Histogram rozptylů ke dvojicím zastávek podle lineárního modelu je vidět na grafu 4.13.

ID spoje	vzdálenost zastávky na trase
336_89_181210	0
336_89_181210	4840

Tabulka 4.2: Výtah z jízdního řádu spoje 336_89_181210.



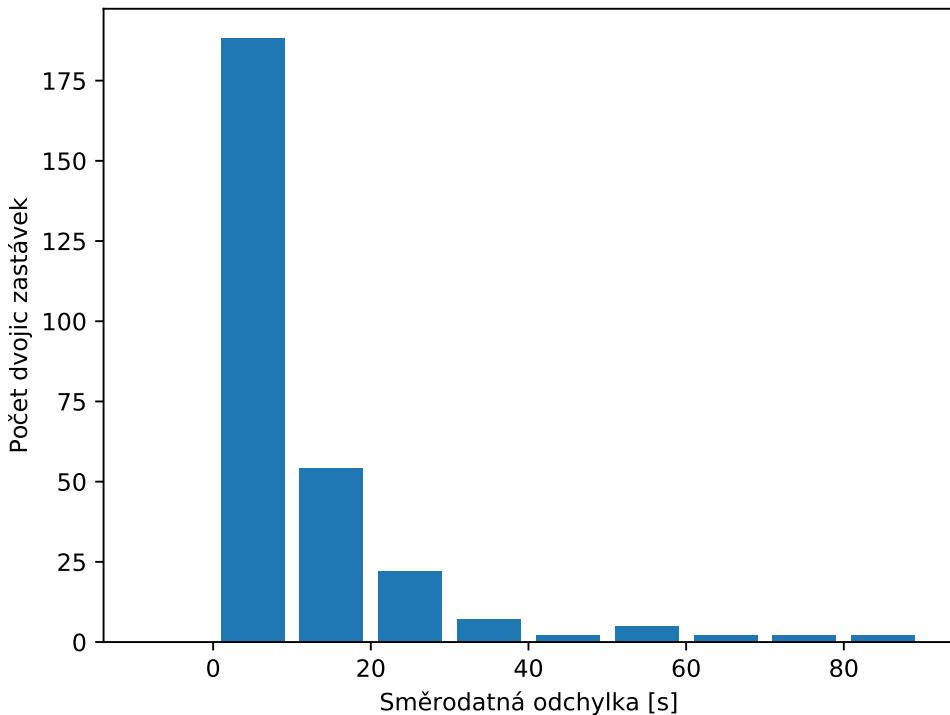
Obrázek 4.13: Histogram rozptylu zpoždění podle lineárního modelu, poslední sloupec značí hodnoty 80 a více

Histogram rozptylů zpoždění podle polynomiálního modelu je zobrazen na grafu 4.14. Z porovnání těchto grafů je jasné viditelné, že došlo k výraznému zmenšení rozptylu. Průměr rozptylů se snížil z 19 sekund na 12.

Tyto grafy nám ukazují, že v průměru došlo ke zlepšení odhadu, ale podívejme se ještě na porovnání rozptylů zpoždění mezi dvojicemi zastávek jednotlivě. Na histogramu 4.15 je patrné, že u největšího počtu dvojic zastávek došlo ke snížení rozptylu o 0 až 10 sekund, nezanedbatelného počtu dvojic o 10 až 20 sekund a k významnému snížení došlo u necelých 25 dvojic. Toto pozorování lze vyvodit i z grafů 4.13 a 4.14, ze kterých vychází, že k nejvýznamějšímu poklesu rozptylů došlo z 10–20 sekund na 10–20 sekund.

Příklad zhoršení odhadu zpoždění

Dále také na grafu 4.15 vidíme, že u 26 dvojic došlo ke zvýšení rozptylu, což je v této práci nežádoucí. Stalo se tak, ale pouze v necelých 10 procentech případů a z toho ve většině nedošlo k výraznému zhoršení. Avšak v jednotkách případů



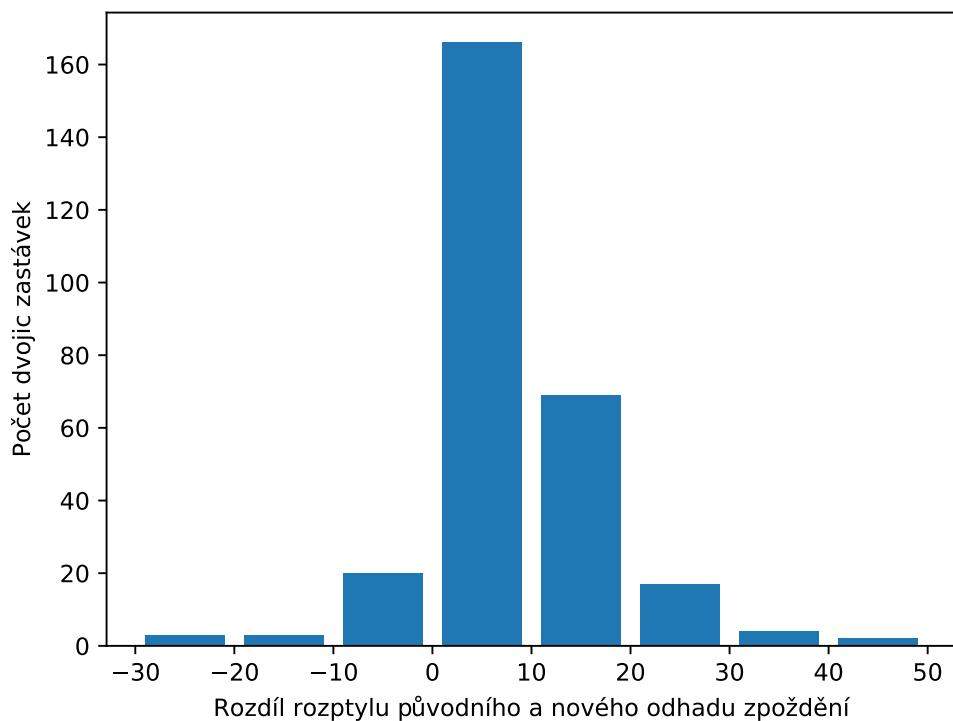
Obrázek 4.14: Histogram rozptylu zpoždění podle polynomiálního modelu, poslední sloupec značí hodnoty 80 a více

došlo k výraznému zhoršení i o několik desítek sekund. Rozeberme si tedy jeden příklad výrazného zhoršení, a proč k němu došlo.

Na grafu 4.16 je vizualizován vygenerovaný model pro dvojici zastávek, kde došlo k největšímu zhoršení oproti odhadu zpoždění lineárním modelem. Na první pohled se zdá, že takto model vypadá správně. Při bližším ohledání ale zjistíme, že dokonalá vlna vznikla z chybných vstupních dat. Konkrétně si povšimneme, že vzorky se seskupily do dvou clusterů, první ve vzdálenosti 0 až 3 000 metrů od výjetí a druhý dále.

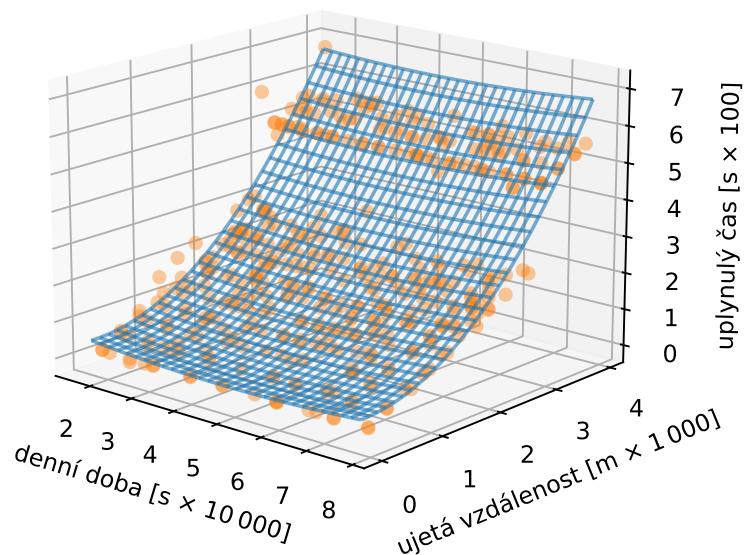
Chybná vstupní data, ze kterých vypočítaný model vychází, uvedeme ještě v tabulce 4.2.2 a část jízdního řádu dotčeného spoje v tabulce 4.2.2. V první tabulce si můžeme povšimnout, že zhruba ve dvou třetinách jízdy mezi zastávkami došlo ke změně zpoždění v poslední zastávce. Což je z našeho pohledu nevysvětlitelné. Tento skok ve zpoždění o více než 220 sekund je u všech jízd projíždějících mezi touto dvojicí zastávek.

Avšak tento skok by se měl projevit i u odhadu zpoždění lineárním modelem. Jak je tedy možné, že u lineárního modelu vyšel rozptyl velmi malý a u polynomiálního modelu v řádu vysokých desítek? K pochopení nám napomůže podívat se na záznamy tohoto spoje v den 21. 2. 2020, tedy na záznamy, které slouží pro



Obrázek 4.15: Histogram rozdílu rozptylu zpoždění podle lineárního modelu a polynomiálního modelu

Malé Kyšice,Poteplí → Malé Kyšice,Rozcestí



Obrázek 4.16: Zhoršení odhadu zpoždění vlivem špatných vstupních dat

ID spoje	ujetá vzdálenost	zpoždění v poslední zastávce
630_177_200210	6700	167
630_177_200210	7400	167
630_177_200210	7600	167
630_177_200210	7900	167
630_177_200210	8300	-62
630_177_200210	8600	-62
630_177_200210	9100	-62

Tabulka 4.3: Záznamy poloh spoje 630_177_200210.

ID spoje	vzdálenost zastávky na trase
630_177_200210	5225
630_177_200210	9187

Tabulka 4.4: Výtah z jízdního rádu spoje 630_177_200210.

evaluaci úspěšnosti. V tento den se totiž pořídily pouze 3 záznamy ve vzdálenosti 2 km od sebe. Tedy lineární model předpověděl všem téměř stejná zpoždění, ale polynomiální model vlivem natrénování se na špatných datech, odhadoval zpoždění zcela mimo realitu.

Z tohoto se dá usuzovat, že v případech, kde se rozptyl zpoždění výrazně zhoršil, jsou na vině závady ve vstupních datech nebo nám schází informace o procesu určování zpoždění v poslední projeté zastávce. Např.: i autobusy mohou projíždět návěstidly, kde se po průjezdu hodnota zpoždění upraví, avšak z jízdních řádů takovou informaci nejistíme.

Příklad zlepšení odhadu zpoždění

Na druhou stranu v našem modelovém případě profilu jízdy mezi zastávkami K Letiště a Zličín pro spoj 324_593_200106 došlo k výraznému zlepšení, a tedy snížení rozptylu z 60 sekund na 12. Kompletní porovnání záznamu vývoje zpoždění na trase je možné pozorovat na grafu 4.17.

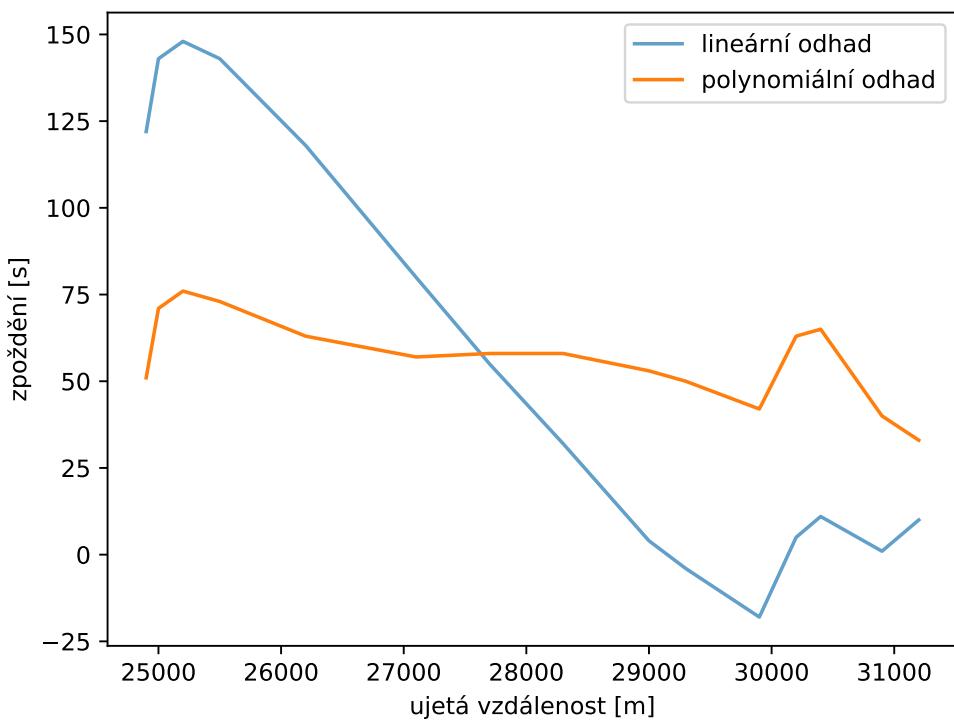
4.3 Statistiky

Ze sesbíraných dat je možné odvozovat i různé statistiky o spojích nebo o zpoždění jízd. Uvedeme si tedy některé z nich.

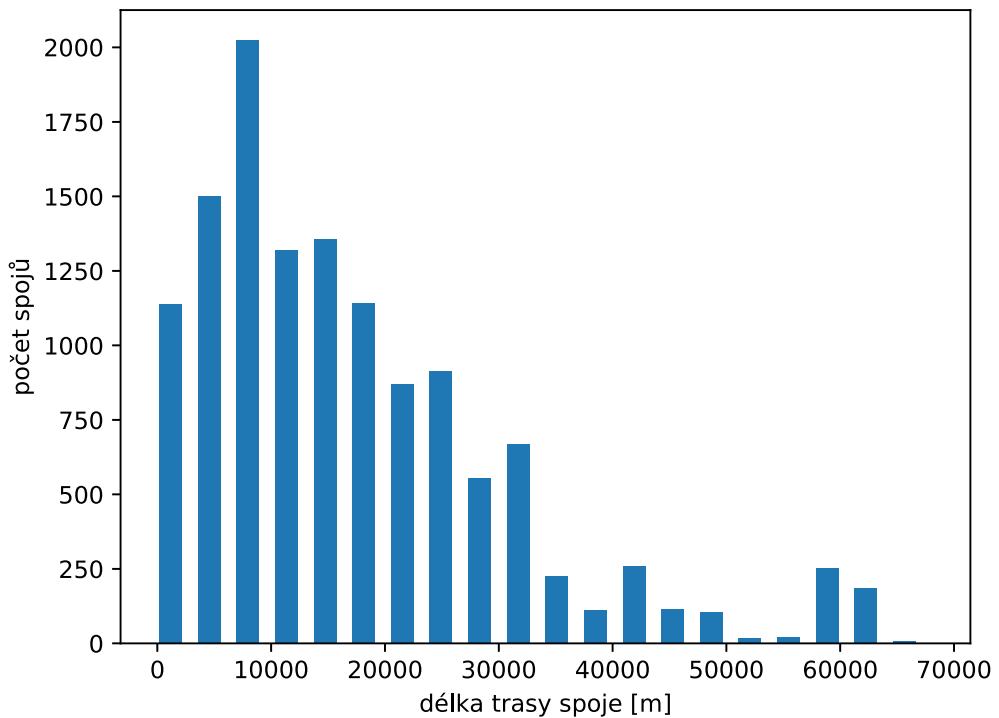
Všechny uvedené grafy a statistiky se zakládají na datech ze dne 20. 2. 2020.

Na prvním grafu 4.18 je znázorněn histogram celkových délek jízd všech spojů. Přičemž průměrná délka jízdy spoje byla 18 kilometrů.

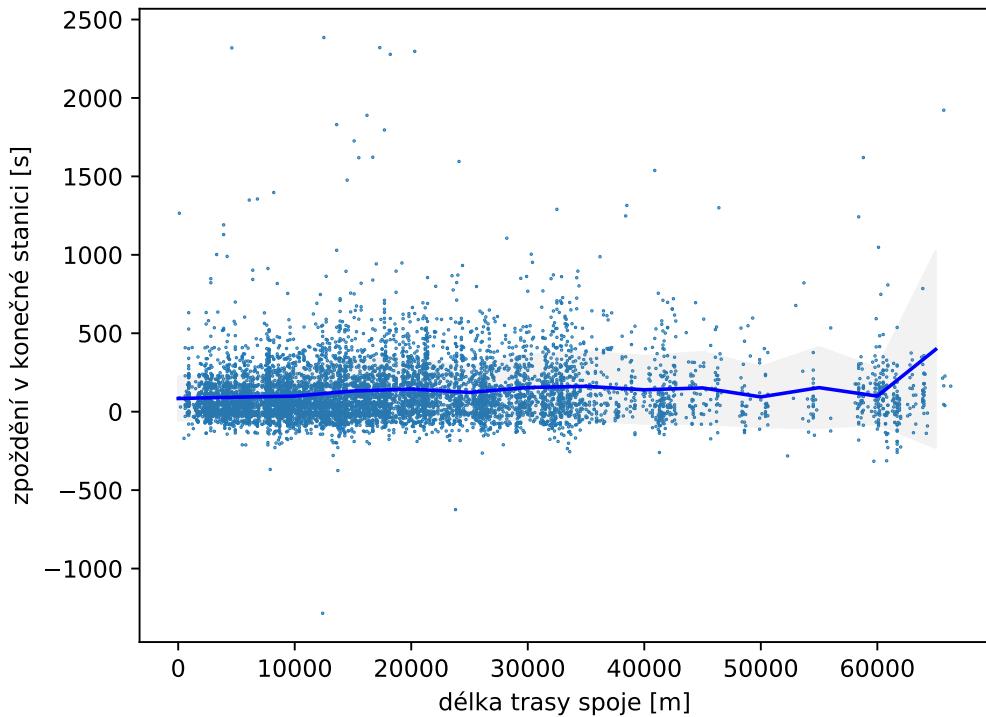
Dále si uvedeme, počet zastávek, které jeden spoj obsluhuje. Samotný histogram se podobá předešlému histogramu 4.18 zachycující délky jízd, proto jen v číslech. Průměrný počet zastávek jednoho spoje je 17. Rekordní počty zastávek jsou přitom 2 a 48. Celkem 2 zastávky obsluhuje 192 spojů, ale většina z nich



Obrázek 4.17: Porovnání průběhu odhadnutých zpoždění lineárním modelem a polynomiálním modelem mezi zastávkami K Letiště a Zličín



Obrázek 4.18: Histogram délek tras spojů.



Obrázek 4.19: Zpoždění v poslední zastávce jízdy vůči délce celé trasy jízdy, Průměr vyznačen modře a rozptyl šedě

jsou přívozy P1 a P2, které se také dostaly do naší databáze a 18 spojů autobusů také obsluhuje pouze 2 zastávky⁹. Naopak maximální počet zastávek na trase má pouze 8 spojů, které naleží jedné lince číslo 369.

Na grafu 4.19 je zobrazeno zpoždění všech jízd v jejich konečné stanici¹⁰ a celková délka jejich tras. Intuitivně by se mohlo říci, že s rostoucí délkou trasy bude zpoždění narůstat. Ale jak vychází z grafu průměrné zpoždění v závislosti na délce trasy je takřka konstantní. Průměrné zpoždění činilo 114 sekund a rozptyl byl 179 sekund. Z toho vychází, že většina jízd dojela se zpožděním, byť v řádu jednotek minut, ale zároveň tím dochází k minimalizaci počtu jízd, které přijedou v předstihu. V konečné zastávce nás sice toto může potěšit, ale musíme si uvědomit, že celkové předjetí se akumuluje po celou dobu jízdy a na zastávkách před konečnou stanicí by tedy vozidlo muselo čekat.

⁹Vysvětleme proč 2 přívozy tvoří většinu spojů oproti 18 spojům pozemní dopravy. Jedna linka je obsluhována několika spoji (spoje se váže na konkrétní jízdní řád), ale linka je daná sekvenčí zastávek. Protože přívozy jezdí poměrně často, jsou popsány velkým množstvím spojů. Ale dotčené autobusové linky jezdí méně často, proto jim přísluší menší počet spojů.

¹⁰Pokud zpoždění v konečné stanici nebylo dostupné ve vstupních datech, bylo použito zpoždění v předposlední stanici.

Závěr

Nejprve jsme se seznámili s daty a po jejich analýze se zkušeností a intuicí s vývojem dopravních situací jsme zvážili, zda je problém možné řešit a jaký dopad by mohl mít v reálném nasazení. Ačkoli požadavek na zlepšení odhadu nebo i předpovědi zpoždění vozidel VHD se zdá naprosto přirozený, žádné z dosud existujících řešení zpracovávající real-time data se o nic takového nepokouší. O to víc je to překvapující s přihlédnutím k množství cestujících v Praze i jinde na světě.

Dále jsme analyzovali jiné nástroje v České republice vizualizující polohy vozidel a definovali jsme problém, Následně jsme si zadali samotné požadavky na dílo.

Navrhli jsme procesy zpracování vstupních dat a jejich následné využití k pravděpodobnostnímu odhadu zpoždění. Také jsme navrhli algoritmus, který se používá v této práci pro odhad zpoždění a algoritmu, který řeší komplikovanější situace, než s jakými jsme se setkali v pražské dopravní síti. Ačkoli tento návrh algoritmu neimplementujeme, může sloužit jako základní kámen pro další výzkum a obdobné aplikace. Součástí návrhu je design front-endu aplikace a databázové struktury.

Zdrojový kód aplikace jsme zdokumentovali jako součást kódu a logiku běhu softwaru jsme popsali v kapitole implementace. Rozdeleně jsou popsány části zpracování dat, výpočet modelů sloužících k odhadu zpoždění a server-klient vizualizační aplikace. Pro serverovou část jsme popsali, jak se k ní připojit v případě vývoje jiné klientské aplikace.

Na závěr jsme celou aplikaci otestovali, a to od jednotlivých funkčních tříd až po aplikaci jako celek. Testování funkcí jsme provedli pomocí unit testů. Složitější celky včetně správné funkčnosti databáze jsme otestovali pomocí integračních testů. Server byl podroben zátěžovým testům, kdy jsme jej dotazovali velkým množstvím paralelních dotazů. Rychlosť zpracování dat byla měřena v bodě maximálního vytížení.

Velkou část testování tvořilo ověření výsledků, tedy odhadů zpoždění. Z kterého vyplynulo, že v nezanedbatelném množství případů došlo opravdu k výraznému zlepšení. Zvolená metoda se však nedá uplatnit ve všech případech, a to zejména z důvodu, že klasické řešení dosahuje dobrých výsledků a není tam tedy prostor ke zlepšení.

Návrhy na zlepšení

Tak jak je aplikace napsána, je schopná samostatného běhu, avšak protože nemůžeme otestovat aplikaci v dlouhodobém nasazení je potřeba vyřešit několik problémů s tím souvisejících. První problém může být objem dat, které je potřeba skladovat, a tím zpomalující se reakční doba databáze. V takovém případě navrhujeme historická data skladovat na odděleném místě od databáze, kde ukládáme

aktuální data, a do které se dotazuje server. Dále je potřeba také určit správnou periodu přepočítání modelů a jak vybírat data, která pro počítání modelů použít.

Protože jsme při návrhu aplikace rozhodli rozdělit vstupní data pro výpočet modelů do dvou podmnožin – zaznamená v pracovní den a víkendový den. Nabízí se vylepšení do dní pracovního volna započítat i státní svátky. Nebo ještě lépe pro výpočet jednoho modelu použít data jen z jednoho dne v týdnu a vázat tak model na konkrétní den v týdnu.

Vzhledem k tomu, že v této práci pracujeme s geografickými daty, bylo by vhodnější použít jinou SQL databázi, protože MySQL databáze není příliš uzpůsobená pro ukládání tohoto typu dat. Vhodnější implementace databáze se nabízí PostgreSQL¹¹ s rozšířením PostGIS¹².

Jako poslední návrh na zlepšení je samotné vylepšení modelů popisujících profily jízd. Ať už zvolením modelu jiného typu nebo zcela odlišného přístupu k problému. Aplikace je nastavena tak, aby se při vylepšování modelů nemusely měnit jiné části aplikace.

¹¹<https://www.postgresql.org>

¹²<https://postgis.net>

Seznam použité literatury

- GARETH JAMES, DANIELA WITTEN, T. H. a TIBSHIRANI, R. (2013). *An Introduction to Statistical Learning with Applications in R*. Osmé přepracované vydání. Springer. Praha. ISBN 978-1-4614-7137-0.
- GURYČOVÁ, K. (2019). Pražský dopravní podnik dál tají data o poloze tramvají. zveřejnění brání smlouva s dodavatelem. *iRozhlas*. URL https://www.irozhlas.cz/zpravy-domov/data-o-poloze-vozidel-dpp-mhd-tramvaje-autobusy-praha-hrib-soud-informace_1904290600_kno.
- RIPLEY, B. D. (1996). *Pattern Recognition and Neural Networks*. Osmé vydání. Cambridge University Press, Cambridge. ISBN 0-521-46086-7.
- SAEED ASAEDI, F. D. a MOHADES, A. (2013). Alpha-concave hull, a generalization of convex hull. *Department of Mathematics and Computer Science, Amirkabir University of Technology*. URL <https://arxiv.org/pdf/1309.7829.pdf>.

Seznam obrázků

1.1	Počet úseků mezi bezprostředně následujícími zastávkami a vzdálenost mezi nimi. Každý průjezd úsekem je započítán zvlášť.	7
1.2	Vzorky poloh a model profilu jízdy mezi zastávkami K Letišti a Zličín. Data jsou ze dnů 20.–21. 2. 2020	8
1.3	Trasa mezi zastávkama K Letišti a Zličín. Zdroj: mapy.cz	9
1.4	Srovnání současného řešení s navrhovaným, modelový příklad	10
1.5	Struktura vstupních dat a relace mezi nimi	13
1.6	Počet souborů s počtem nově nalezených spojů v nich	18
1.7	Počet souborů s celkovým počtem vozidel v nich	19
1.8	Mapa z golemio.cz.	22
1.9	Mapa z www.tram-bus.cz.	22
1.10	Mapa z mapa.idsjmk.cz.	23
2.1	UML diagram návrhu aplikace	25
2.2	ER diagram návrhu databáze	36
2.3	EER diagram návrhu databáze	37
2.4	Variabilita délky jízdy v průběhu dne	38
2.5	Variabilita času jízdy v závislosti na ujeté vzdálenosti	38
2.6	Úsek s nepravidelnostmi	39
2.7	Úsek s nepravidelnostmi 2	39
2.8	Výrazné zpoždění jedné jízdy a chybný směr jízdy	40
2.9	Úsek s nepravidelným zpožděním	40
2.10	Modelový příklad profilu trasy dobré popsatelného konkávním obalem	41
2.11	Nejednoznačnost konkávního obalu	41
2.12	Chyba polynomálního modelu	42
2.13	Mapbox Mapa, použitý styl mapy: streets-v11	42
2.14	Design zobrazení elementů v mapě, linka je 348 je vybrána	42
4.1	Průměrný čas zpracovávání daného počtu vozidel ze všech souborů se statickými daty s 95 % intervalem spolehlivosti. Počty vozidel jsou vždy zaokrouhleny dolů na celé desítky.	59
4.2	Doba odpovědi serveru na 100 paralelních dotazů (1000 vzorků)	61
4.3	Mapa Prahy a okolí s polohy vozidel zaznamenány 23. 2. 2020	62
4.4	Shluk vozidel	62
4.5	Vyznačená trasa jízdy	62
4.6	Vyznačená trasa jízdy s jízdním rádem	63
4.7	Vozidla projíždějící vybranou zastávkou	63
4.8	Odjezdová tabule	63
4.9	Modelování profilu jízdy s chybnými vstupními daty	65
4.10	Histogram RMSE všech modelů	65
4.11	Vizualizace modelu druhého stupně	66
4.12	Vizualizace modelu třetího stupně	66
4.13	Histogram rozptylu zpoždění podle lineárního modelu, poslední sloupec značí hodnoty 80 a více	68

4.14 Histogram rozptylu zpoždění podle polynomiálního modelu, poslední sloupec značí hodnoty 80 a více	69
4.15 Histogram rozdílu rozptylu zpoždění podle lineárního modelu a polynomiálního modelu	70
4.16 Zhoršení odhadu zpoždění vlivem špatných vstupních dat	70
4.17 Porovnání průběhu odhadnutých zpoždění lineárním modelem a polynomiálním modelem mezi zastávkami K Letiště a Zličín	72
4.18 Histogram délek tras spojů.	72
4.19 Zpoždění v poslední zastávce jízdy vůči délce celé trasy jízdy, Průměr vyznačen modře a rozptyl šedě	73

Seznam tabulek

4.1	Záznamy poloh spoje 336_89_181210.	67
4.2	Výtah z jízdního řádu spoje 336_89_181210.	68
4.3	Záznamy poloh spoje 630_177_200210.	71
4.4	Výtah z jízdního řádu spoje 630_177_200210.	71

Slovník

- AJAX** Asynchronous JavaScript and XML. 51
- API** rozhraní pro programování aplikací. 11, 52
- CSS** Cascading Style Sheets. 51
- DOM** Document Object Model. 51
- DPP** Dopravní podnik hlavního města Prahy, a.s. 3, 22
- EER** Extended entity–relationship model. 27
- ER** Entity–relationship model. 25
- GEOJSON** standardní formát navržený pro reprezentaci jednoduchých prostorových geografických dat, <https://tools.ietf.org/html/rfc7946>. 12, 28, 51, 52, 55, 56
- GPS** Global Position System. 11, 13, 15
- GTFS** General Transit Feed Specification. 12
- HTML** Hypertext Markup Language. 4, 51, 52, 53, 54
- HTTP** HyperText Transfer Protocol. 12
- IDSJMK** Integrovaný dopravní systém Jihomoravského kraje. 22
- IDSK** Integrovaná doprava Středočeského kraje. 3
- INT** Celé číslo. 26, 27
- JS** JavaScript. 4, 51, 52
- JSON** JavaScript Object Notation, <https://tools.ietf.org/html/rfc8259>. 12, 25, 44, 53
- OSM** OpenStreetMap. 21
- PID** Pražská integrovaná doprava. 23
- PID** Object Oriented Programming. 44
- RMSE** root-mean-square error. 64, 65, 77
- ROPID** Regionální organizátor pražské integrované dopravy, p. o.. 3, 11, 15
- SQL** Structured Query Language. 4, 25, 27, 43, 44, 46, 48, 49, 56, 60, 75

URL Unique Resource Link. 12, 44

UTC Koordinovaný světový čas. 13, 51

VHD Veřejná hromadná doprava. 3, 4, 5, 8, 14, 20, 21, 22, 33, 74

WSGI Web Server Gateway Interface. 55

A. Přílohy

A.1 První příloha