Univerzita Karlova v Praze

Přírodovědecká fakulta

Katedra aplikované geoinformatiky a kartografie

Studijní program: Geografie (navazující magisterské studium)

Studijní obor: Kartografie a geoinformatika

Tomáš LOUKOTKA

MODELY VEŘEJNÉ HROMADNÉ DOPRAVY V PROSTŘEDÍ GIS

MODELS OF PUBLIC MASS TRANSPORTATION IN GIS ENVIRONMENT

Diplomová práce

Vedoucí diplomové práce: Mgr. Stanislav Grill

Praha 2011

Prohlášení

Prohlašuji, že jsem závěrečnou práci zpracoval samostatně a že jsem uvedl všechny použité informační zdroje a literaturu. Tato práce ani její podstatná část nebyla předložena k získání jiného nebo stejného akademického titulu.

V Praze dne 8. 8. 2011 .....

podpis

Poděkování

Na tomto místě bych rád poděkoval vedoucímu mé práce Mgr. Stanislavu Grillovi, který, i přes obtíže s dojížděním, obětoval konzultacím velkou část svého času, a poskytl mi mnoho cenných rad. Dále bych rád poděkoval rodině a přátelům za podporu při psaní práce.

Modely veřejné hromadné dopravy v prostředí GIS

Abstrakt

Práce popisuje a modely veřejné hromadné dopravy. Na základě kombinace těchto modelů a myšlenek autora je vytvořen vlastní model. Tento model pracuje pouze s intervaly a nikoli s jízdními řády. Proto v něm nelze s jistotou určit chování cestujícího. Model pracuje kromě sítě linek i s uliční sítí. Nad tímto modelem je vytvořena aplikace, která umožňuje editaci sítě hromadné dopravy a provádění analýz dostupnosti a nejkratší cesty. Použité technologie jsou: ExtJS, OpenLayers, CGI, Shapely, STORM, PostgreSQL. Tato aplikace je prakticky využita při vytvoření sítě hromadné dopravy pro Prahu 6 a provádění analýz nad touto sítí.

Klíčová slova: modely hromadné dopravy, síťové analýzy, GIS, programování, webové mapování

Models of Public Mass Transportation in GIS Environment

Abstract

The thesis tries to describe and categorize models of public mass transportation. A new model is built upon these models and based on authors' thoughts. This model is constructed to work with headways only instead of timetables. Therefore one cannot precisely estimate the passengers' behaviour. The model works also with street network aside from public transportation lines. An application is built upon these model and allows network editing and processing of shortest-path analysis and accessibility analysis. Technologies used: ExtJS, OpenLayers, CGI, Shapely, STORM, PostgreSQL. This application is then used for building a public transportation network of Prague 6 region and for performing analysis upon this network.

Keywords: mass transportation models, network analysis, programming, GIS, web-mapping

OBSAH

SEZNAM OBRÁZKŮ A TABULEK

KAPITOLA 1: ÚVOD

V současné době je čím dál obvyklejší zapojení aparátu geoinformatických systémů (dále jen GIS) do nejrůznějších oborů lidských činností. Doprava mezi těmito obory hraje významnou roli. Je tomu tak z důvodu, že doprava má k prostoru velmi těsný vztah. Hromadná doprava pak, i přes rychlý rozvoj dopravy individuální, rozhodně na významu neztrácí a je dost možné, že její význam v budoucnosti naopak poroste.

1.1. Motivace a cíle práce

Motivací k sepsání této práce byla, kromě autorova dlouhodobého zájmu o sítě hromadné dopravy, také nedostupnost jednoduchých nástrojů, kterými by se dala síť modelovat. Tento nedostatek autor již pocítil při sepisování jedné ze semestrálních prací. Vektorizace sítě pro Prahu 6 zabrala spoustu času, a stejně s touto sítí nebylo možné v analýzách zohlednit všechna specifika hromadné dopravy. Tato vektorizace vyžadovala mnohdy zbytečné duplikace geometrií a atributů. Z tohoto důvodu pak jakákoliv následná editace byla velmi nesnadná. Autor chtěl tedy tyto nedostatky ve své práci napravit.

V současné době jsou oblíbené přístupy, které pro sestrojení modelů potřebují konkrétní jízdní řády. Pro potřeby jednoduchého a rychlého modelování nejsou příliš vhodné, ačkoliv samozřejmě poskytují větší přesnost. Autor tedy přichází s myšlenkou se od tohoto předpokladu odpoutat a věnovat se pouze práci s intervaly linek městské dopravy. Takto navržený model je dále inovativní v tom, že propojuje síť hromadné dopravy s uliční sítí, a tím se více přibližuje realitě.

Definujme prvotní úlohu, kterou se autor v práci pokusí řešit. Mějme existující síť linek hromadné dopravy a ulic na daném území a zjistěme časovou dostupnost libovolného bodu. I přes zdánlivou jednoduchost tohoto zadání se záhy ukázalo, že je sice možné použít stávající softwarové nástroje k vyřešení této úlohy, nicméně řešení je natolik časově náročné a neflexibilní, že se vyplatí přemýšlet o jiném. Řešení pomocí stávajících systémů by spočívalo v nadefinování síťového modelu, který by musel obsahovat linie pro každou linku a mezistaniční úseky, dále propojení linek mezi sebou, napojení na uliční síť v nejbližším bodě a další prvky. Ruční editace takovéto sítě by

byla velmi časově náročná, a i po jejím dokončení by analýzy nad touto sítí prováděné provázelo mnoho otázek týkajících se přesnosti výstupů.

Cílem práce bude tedy navrhnout takový model, který bude snadné plnit, udržovat a provádět nad ním analýzy. To vše s přihlédnutím ke specifikům, které vyvstávají z faktu, že při práci bez konkrétních jízdních řádů panuje o chování uživatelů hromadné dopravy určitá nejistota a tudíž nelze přesně předpovídat jejich chování.

Autor si klade následující otázky:

- Jak definovat datový model tak, aby byl snadno editovatelný a aby v něm nebyly zbytečné duplicity?
- Jak přistupovat k počítání časů přepravy z místa na místo bez znalosti jízdních řádů?
- Jak za takovýchto podmínek počítat časové dostupnosti jednotlivých míst?
- Existují vhodné softwarové nástroje, které by nám k vyřešení těchto otázek alespoň částečně pomohli?
- Jakým způsobem stávající nástroje spojit v jeden celek, aby bylo možné provádět bez většího úsilí výše zmíněné analýzy?

Aby tyto otázky mohly být zodpovězeny, klade si autor za cíl vyvinutí aplikace, která umožní síť MHD vytvářet a editovat a posléze nad ní provádět analýzu nejkratší cesty a analýzu časové dostupnosti. Samozřejmě, že vývoji takovéto aplikace musí taktéž předcházet definice datového modelu i obecnějších teorií přístupu k problému, stejně jako vývoj vlastních algoritmů, které jsou založené na stávajících řešeních.

Základní funkčnost aplikace v bodech:

- Možnost snadno vytvářet a editovat prostorové i neprostorové složky modelu, tak aby uživatel nebyl nucen k zadávání duplicitních údajů
- Možnost prohlížet dosud vytvořený model v přívětivém a přehledném uživatelském prostředí
- Možnost provádět v aplikaci analýzy časové dostupnosti a analýzy nejkratší cesty a jejich výsledky vizualizovat v mapě

1.2. Popis obsahu práce

Po tomto úvodu bude v práci následovat kapitola, která se věnuje nejprve popisu základních pojmů a poté kategorizaci existujících modelů veřejné dopravy. Zde také budou následovat konkrétní příklady těchto modelů. Hlavním dělícím kritériem je zde především deterministické či stochastické zaměření modelu, či případně obecnější dělení modelů dle jejich náplně.

Poté se autor věnuje samotnému popisu teorie, dle které jsou posléze sestrojovány algoritmy a datový model. Klíčovým problémem je zde úvaha nad tím, jakým způsobem počítat čas mezi 2 danými místy za panující nejistoty o konkrétních jízdních řádech. Autor tento problém řeší pomocí přiřazování pravděpodobnosti různým variantám. Po definici teorií pak práce popisuje datový model, který slouží pro ukládání dat. Je zde řešen v obecnější rovině, bez vazeb na konkrétní programovací jazyky či databázové platformy.

V další poměrně rozsáhlé kapitole se autor věnuje implementaci jednotlivých algoritmů i s popisem postupů. I v této kapitole se autor snaží o odstup od konkrétních technologií použitých pro programování. Tato kapitola čtenáři umožní více proniknout do myšlenkových pochodů autora i do principu fungování celé aplikace.

Nyní se již práce může věnovat samotnému návrhu aplikace. Nejprve je potřeba na základě požadavků na funkčnost aplikace vybrat vhodné technologie. V dnešní době existuje mnoho variant pro dosažení totožného cíle. Autor se proto snaží alespoň trochu poodhalit množství existujících řešení, i když to samozřejmě není možné v plné míře a znalost všech technologií čtenář ani nemůže od autora očekávat, i s přihlédnutím k zaměření práce. Postupně jsou probírány technologie na klientské části aplikace, serverové části aplikace a nakonec databázového serveru. Samozřejmostí je popis propojení těchto vrstev.

Dále je popsána aplikace z hlediska funkčnosti. Jedná se o stručný popis kroků, které je třeba provést za účelem vytvoření sítě, její editace a posléze k provádění analýz. Po této kapitole se aplikace zabývá konkrétním technickým návrhem aplikace z hlediska členění tříd, rozdělení zodpovědnosti za činnosti mezi klientskou a serverovou část, dělení na metody atd.

Pro ověření funkčnosti a výkonu se práce v poslední kapitole věnuje aplikaci programu na konkrétní data, a to data městské hromadné dopravy na území Prahy 6. Zkoumána je snadnost, s jakou lze takovouto síť vytvořit a editovat. Posléze se práce věnuje časům jednotlivých analýz při změnách parametrů, které mají vliv na přesnost a výkon. Následuje popis vybraných výstupů z analýz a jejich interpretace a zhodnocení toho, na kolik je pravděpodobné, že tyto výsledky odpovídají skutečnosti.

Shrňme nyní cíle práce. Cílem práce je shromáždit údaje o existujících modelech a na jejich základě vyvinout model nový. Tento nový model usnadní vytváření i editaci sítě, jeho výhodou bude snadná změna parametrů. Nad tímto modelem pak budou prováděny analýzy. Pro provádění všech těchto činností je vyvinuta aplikace, která kromě vytváření a editace sítě umožní provádět analýzu nejkratší cesty a analýzu časové dostupnosti.

KAPITOLA 2: PŘEHLED EXISTUJÍCÍCH MODELŮ

V této kapitole budou popsány vybrané existující modely hromadné dopravy, zároveň s jejich členěním. V úvodu kapitoly lze nalézt základní vymezení pojmů se kterými se v práci pracuje. Následuje popis dělení modelů, které jsou z pohledu autora zajímavé. Posléze je toto dělení názorně předvedeno na vybraných existujících modelech. Na závěr kapitoly se autor věnuje popisu různých prostředků reprezentace těchto modelů v prostředí GIS. Model popisovaný v práci by byl zařazen při těchto různých děleních do následujících skupin v hierarchické posloupnosti: modely jevů – modely idealizované a fenomenologické - modely hromadné linkové dopravy – modely určené k analýze – modely používající jízdní řády, beroucí užitkovou funkci jako deterministickou, přiřazení poptávky nabídce pak stochasticky – modely uvažující informovaného cestujícího. Popis jednotlivých těchto skupin následuje dále.

2.1. Vymezení pojmů

2.1.1. Model

Pro potřeby práce je nejprve potřeba si ujasnit co je to model. Modely lze rozdělit na modely, které něco reprezentují a modely teoretické. Reprezentující modely pak dále na modely reprezentující nějakou část reálného světa (modely jevů) a modely dat (FRIGG, HARTMANN, 2009). V práci bude řeč o modelech, které se věnují modelování jevů reálného světa, v tomto případě všem částem světa, které mají co do činění s dopravou.

FRIGG a HARTMANN (2009) posléze dělí tyto modely na další skupiny. První z nich jsou modely měřítkové. Jejich podstatou je zmenšení nebo zvětšení reality. Tyto modely by měly realitu co nejvěrněji kopírovat. Samozřejmě kopírovat jen v některých oblastech. Je jasné, že např. zmenšený model auta nebude vyroben z kovů, ale např. z plastů, nicméně jeho tvar bude stejný.

Další skupinou modelů jevů jsou tzv. idealizované modely. V tomto případě se z reality postupně odebírají pro dané potřeby nepodstatné součásti a vzniká zjednodušený model. Takováto idealizace se nazývá Aristotelovská. Existuje také Galvaniho idealizace, která do modelu vkládá předpoklady, o kterých víme, že ve

skutečnosti nejsou platné stoprocentně. Tyto 2 idealizace samozřejmě mohou existovat i současně. Příkladem idealizovaného modelu jsou např. různé modely trhů v oblasti ekonomie, ale daly by se sem zařadit i dopravní modely.

Další skupinou modelů jevů jsou modely analogické. Zde se realita přirovnává k nějaké jiné, již existující a popsané realitě. Např. interakce plynů v atmosféře se přirovnává k chování kulečníkové koule apod.

Poslední zmiňovanou skupinou jsou modely fenomenologické. Ty se zaměřují pouze na popis pozorovaných skutečností bez vazby na popis příčin. Někdy se popisují jako modely nezávislé na teorii.

Z výše uvedeného výčtu lze usoudit, že hlavním předmětem práce budou modely idealizované a modely fenomenologické.

2.1.2. Doprava

Podle The Concise Oxford English Dictionary (1999) je doprava chápána jako přemisťování z jednoho místa na druhé za pomoci dopravních prostředků. Rozlišujeme různé způsoby dopravy – lidskými silami, zvířecí silou, železniční, letecká, silniční ad.

Doprava může používat infrastrukturu k realizaci tohoto pohybu (silnice, kanály, železnice, letiště apod.). Dále se k dopravě mohou (ale nemusí, např. v případě obyčejné chůze) využívat různé dopravní prostředky (vlak, automobil, autobus ad.).

2.1.3. Veřejná doprava

V terminologii existují rozdíly mezi výrazy veřejná doprava, hromadná doprava, městská doprava a linková doprava. Protože tyto termíny budou v práci užívány, je třeba si jejich význam ujasnit.

Jako veřejnou dopravu lze chápat každou dopravu, která je přístupná za určitých podmínek komukoliv. Nejedná se tedy např. o individuální dopravu v autech. Jako příklad, který naplňuje význam termínu lze uvést provozování taxislužby, provozování linek metra ve městě, provozování nákladní dopravy za pomoci kamionů atd.

Jako hromadnou dopravu lze chápat takovou dopravu, kdy se v jednom prostředku přepravuje více jak jeden člověk (do důsledků vzato, i více jak jedna komodita). Dle této definice by do takovéto dopravy mohla spadat i individuální automobilová doprava při cestě více jak jedné osoby. Typickým příkladem takovéto dopravy je např. doprava pomocí tramvají, autobusů, apod.

Jako linkovou dopravu definujeme takovou, kde se dopravní prostředky pohybují dle předem stanovených jízdních řádů. Opět se nemusí jednat jen o dopravu osob, ale i o dopravu nákladů.

Osobní doprava je pak taková, při které se přepravují osoby. Otázkou je, jak vyřešit smíšené případy (vlak s poštovními vagóny apod.). Za osobní dopravu bude brána každá doprava, při které se přepravuje minimálně jedna osoba.

Z výše zmíněných definic lze nyní definovat, jakými druhy doprav se bude práce zabývat. Bude se jednat zejména o osobní veřejnou hromadnou linkovou dopravu. Její hromadnost nemusí být úplně nutná, i když je obtížné najít příklady linek, které převážejí jen 1 osobu. Taktéž by v případě nesplnění podmínky hromadnosti vyvstala nutnost modelovat velmi omezenou kapacitu dopravního prostředku, což není cílem práce. Veřejnost dopravy je nutná zejména proto, že v případě zahrnutí i neveřejných prostředků by různí jedinci měli různě obtížný přístup k dopravě. Modelovat takové situace není cílem práce. Obecně si však lze jen těžko představit dopravu, která bude osobní a linková a zároveň nebude veřejná. Dále se v textu bude pracovat s termínem modely linkové hromadné dopravy (MLHD), a přitom se předpokládá, že tato doprava bude zároveň i osobní a veřejná.

2.1.4. Dopravní modely

Nyní je potřeba si ujasnit chápání pojmu dopravní model. Dostupná literatura se zaměřuje spíše na popis jednotlivých modelů vhodných pro nějaký konkrétní účel či na popis modelů, použitelných pro více účelů. Problém je s vymezením toho, odkdy již lze o modelu reality hovořit jako o dopravním, když jeho součástí jsou i jiné jevy. Jako příklad lze uvést komplexní model města. Doprava v něm jistě hraje svoji nezastupitelnou úlohu, ale rozhodně netvoří rozhodující část takového modelu. Z těchto důvodů je vymezení pojmu problematické. V popisu pojmu model bylo též zmíněno, že model může i fyzicky existovat a být zmenšeninou či zvětšeninou reality. Lze v případě např. modelové železnice mluvit o dopravním modelu? Ano lze, ale tento příklad ukazuje, že nic jako dopravní model nelze jednoznačně vymezit. Můžeme však modely týkající se dopravy rozčlenit do 3 skupin:

- modely, kde dopravní složka hraje jen doplňkovou roli (např. modely celého města a jeho fungování, modely krajiny, apod.)
- modely, kde je doprava hlavní složkou, přičemž se bere jako fenomén, a neřešíme zdroje toho, jak vzniká – toto by se dalo chápat jako modely fenomenologické (viz výše)
- modely, kde je doprava hlavní složkou a zkoumáme příčiny jejího vzniku. Takové modely pak musí být komplexnější a doprava, potažmo dopravní síť, v nich nemůže být jedinou složkou.

Práce se bude zabývat především modely, kde je doprava hlavní složkou, ale nebude zkoumat příčiny – ty už budou dané (je dána poptávka po dopravě, není třeba ji odvozovat z hustoty zalidnění, funkčního využití prostoru apod.). Nyní lze přistoupit

k definici modelu od nejobecnějšího popisu po přesnější, pomocí postupného přidávání dalších subjektů či vlastností subjektů.

Mezi dopravními prostředky jsou takové, jejichž potřeby infrastruktury jsou při určitém zobecnění téměř nulové. Jedná se např. o leteckou dopravu (doprava probíhá ve vzduchu, kde k tomu již nepotřebuje nic), pěší dopravu (člověk se dostane v podstatě kamkoliv) ad. Základní jednotkou pro tyto druhy dopravy je tedy zcela obecný trojrozměrný prostor. Samozřejmě, že model takovéhoto prostoru bez dalšího upřesnění lze těžko vydávat za dopravní model. Přidejme k němu tedy subjekty, které se pohybují, tj. v případě letecké dopravy letadla, v případě pěší dopravy lidi. Jedná se v tuto chvíli o dopravní model? Dle autorova názoru už do jisté míry ano. Na nejjednodušší úrovni lze tedy konstatovat, že model sestává pouze ze subjektů v prostoru, jejichž jedinou vlastností je poloha v čase. Abychom pak mohli hovořit o dopravním modelu musí se tato poloha v čase měnit.

Nyní lze model rozšířit. Prostoru můžeme přidat vlastnost: průchodnost. Tato průchodnost může nabývat buď dvou hodnot – průchozí, neprůchozí, nebo plynule svůj obor hodnot měnit a charakterizovat tedy, jak se dané subjekty mohou danou částí prostoru pohybovat. Stejně tak můžeme přidávat atributy i pohybujícím se subjektům, např. maximální rychlost, přepravní kapacita atd.

Tato práce se bude zabývat modely, kde doprava je hlavní složkou a nejsou řešeny její zdroje. Minimální požadavky na dopravní model jsou následující: jeho součástí musí být alespoň jeden pohybující se subjekt a prostoru musí být nějakým způsobem přiřazena jeho průchodnost.

2.1.5. Osobní linková hromadná doprava

Linková hromadná doprava má oproti obecně pojaté dopravě některá specifika, které je třeba zmínit. Hlavním z nich je fakt, že určující pro čas a směry dopravy není vůle cestujícího, ale nabídka spojů dopravců. Ti určují jaké budou intervaly linek, vedení linek, kapacity vozů atd. Při modelování je toto třeba vzít v potaz.

Modely linkové hromadné dopravy (MLHD) se mohou zabývat 3 okruhy – nabídkou dopravy, poptávkou po dopravě a způsobu přiřazení poptávky k nabídce.

Nabídkou jsou zde myšleny dopravní prostředky a jejich vlastnosti, chování, intervaly, přesné jízdní řády, jízdní vlastnosti, dopravní síť jako celek, vedení linek ad. Nabídka většinou vzniká na základě poptávky, ale některé její složky mohou být ovlivněny i např. historicky (např. síť tramvají, síť železnic jako pozůstatek po průmyslové revoluci ad.). V úlohách, kde je MLHD plánována tato síť ještě není známa (nebo není známa její část) a na základě poptávky a předpokládaného způsobu přiřazení poptávky k nabídce je navrhována.

Poptávku představují v modelech cestující se svojí výchozí destinací, cílovou destinací, časovými preferencemi, se svými vlastnostmi, díky kterým některé druhy dopravy preferují před jinými, preferují jízdu bez přestupů před jízdou s přestupy apod., se svým chováním přímo v terénu, se svou informovaností ad. V plánovacích úlohách lze předpokládat, že tato složka je známá, v úlohách, kde je cílem cestující informovat, se tato složka ignoruje. Pak jsou samozřejmě také úlohy, kdy je cílem tuto složku odhadnout. Tyto úlohy a modely k jejich řešení už nelze nazývat MLHD, i když se jednou z jejich složek zabývají. Je tomu tak proto, že na vznik poptávky nemá linková hromadná doprava jako taková vliv.

Důležitým ve všech modelech je způsob, kterým je přiřazena poptávka k nabídce. Může se jednat o přidělení konkrétního požadavku na cestu konkrétním jednotlivým dopravním prostředkům, nebo se mohou jak poptávky tak dopravní prostředky agregovat. Záleží na tom jak model zdůrazní či naopak oslabí preference cestujícího. Samozřejmě, čím přesněji se budou uvažovat uvažovat preference cestujícího (bude se brát v potaz, např. fakt, že určité procento nerado cestuje tramvajemi, určité procento nechce přestupovat, někteří preferují jen co nejnižší čas, někteří nejnižší cenu apod.), tím přesnější výsledek vyjde. Toto však platí za předpokladu, že se podaří tyto preference úspěšně modelovat. V opačném případě pak se zesložiťováním modelu přichází i ztráta přesnosti.

Pokud bude toto rozdělení vztaženo k této práci, lze dospět k závěru, že práce navrhne způsoby, jakými ukládat a editovat nabídkovou část modelu, možnost vkládat do modelu poptávkovou část ve smyslu zadání bodu, ze kterého se bude počítat dostupnost či zadání 2 bodů pro výpočet cesty. Především pak práce bude řešit, jakým způsobem bude přiřazena poptávka nabídce.

2.1.6. Důvod vzniku modelů

Jak již bylo naznačeno v předchozích odstavcích, MLHD vznikají za účelem plánování této dopravy a pro případné revidování těchto plánů. Další možností jejich využití je např. informování cestujících, operativní řízení dopravy ad. Nejkomplexnější jsou modely založené na plánování sítě, v ostatních případech se již předpokládají existující síť a provádí se úlohy nad touto sítí. Při plánování sítě je využívána znalost (nebo spíše odhad) poptávky a zvolený přiřazovací mechanismus k tomu, aby byla plánována nabídka (vedení linek, intervaly ad.). Při informování je využívána pouze nabídková část modelu, při analýze sítě pak jak nabídková, tak poptávková část modelu a v popředí je princip přiřazení, na jehož základě pak z takovéhoto modelu lze získávat data.

Plánování se provádí za účelem minimalizovat náklady při maximálním uspokojení zákazníků (cestujících, občanů města, oblasti apod.) (BORNDÖRFER, GRÖTSCHEL, PFETSCH, 2007) Jak je vidět, jdou tyto dva principy (maximalizovat užitek cestujících

vs. minimalizovat náklady) proti sobě.

Při plánování linkové hromadné dopravy se postupuje v 5 krocích (GUILHAIRE, HAO, 2008) :

- naplánování sítě
- naplánování frekvencí
- stanovení konkrétního jízdního řádu
- přiřazení dopravních prostředků k JŘ
- přiřazení řidičů k dopravním prostředkům

V dalších částech práce se bude autor věnovat pouze problematice plánování a analýze dopravní sítě a frekvencí.

2.2. Dělení modelů

Modely linkové hromadné dopravy lze rozdělit na skupiny dle různých kritérií. (viz např. NUZZOLO, CRISALLI, 2009 či FAN, MACHEMEHL, 2006) Na následujících stránkách budou tyto skupiny popsány a vždy uvedeny nějaké příklady. Základní obrysy rozdělení byly již nastíněny v předchozích kapitolách. Nyní tedy uveďme vybrané způsoby dělení modelů:

- důvod vzniku: plánování, revize, analýzy, informování
- agregující přístup vs. neagregující přístupy
- náklady pouze časové vs. komplexní
- užitková funkce generována stochasticky x deterministicky
- se zadaným jízdním řádem, se zadanými intervaly, výjimečně bez těchto informací
- přiřazení určováno stochasticky vs. deterministicky
- cestující neinformován, informován o intervalech, informován o jízdním řádě
- modely s linkami, které mají velké vs. malé intervaly

V důsledku toho, že je často zmiňováno dělení na stochastické a deterministické modely, je třeba tyto termíny obecně objasnit. Deterministický model je takový, kdy při stejných vstupech do modelu jsou i stejné, přesně dané výstupy. Stochastický model pak přesné výstupy nahrazuje pouze jejich pravděpodobnostním odhadem. V případě deterministicky stanovené funkce je pro každého cestujícího dosažený užitek či ztráta při zachování stejných podmínkách konstantní. Znamená to, že např. náklady na čekání 1 minutu na zastávce, či 2 minut v autobuse jsou stejné, ať se jedná o jakéhokoliv cestujícího. V případě stochasticky stanovené funkce jsou parametry různé. Znamená to,

že 1 cestujícímu může více vyhovovat cestovat 1 minutu v tramvaji než druhému. Celkové náklady, a tedy i užitkovou funkci můžeme za pomoci pravděpodobnosti poté jen odhadovat. Nutno zmínit, že pokud tyto náklady jsou brány jako určené stochasticky, pak i přiřazení poptávky k nabídce bude stochastické (tj. o jedné poptávce nelze s jistotou říci, jaké nabídce bude přidělena). Deterministicky stanovené přiřazení poptávky nabídce znamená, že jednomu OD páru (poptávka na přepravu z určitého místa (origin) na jiné místo (destination)) je přiřazena právě jedna trasa, která je pro daného cestujícího s danými informacemi optimální. Při stochasticky stanoveném přiřazení se poptávka za pomoci pravděpodobnosti rozdělí mezi více tras.

Modely se podle účelu mohou dělit takto:

- plánování a následná revize plánů
- informování subjektů
- analýza

Modely určené k plánování kladou důraz na zmapování poptávky a na způsob přiřazení této poptávky nabídce. Výsledkem je posléze naplánování sítě, frekvencí, či přímo rovnou jízdních řádů a posléze i naplánování dalších detailů provozu. Vzhledem k prozatímní neexistenci dopravní sítě je jejich konstrukce ze všech nejsložitější. Podobné si jsou činnosti revize plánů a analýza. Respektive analýza předchází revizi. U analýzy se používá již existující síť a jízdní řády (intervaly) k zjištění nějakých faktů (dojezdových dob, zaplněnosti vozidel apod.), na základě kterých může být revize provedena. Opět je zde velice důležitý způsob přiřazení poptávky nabídce. Samotnou stranou poptávky se zabývat lze, ale není to nutné. Modely je též možné použít pouze jako zdroj pro informování cestujících. V takovém případě se pak spíše jedná o problém jakým způsobem navrhnout síť tak, aby potřebná data poskytovala. Jedná se tedy o datový model.

Modely se dělí taktéž dle svého přístupu ke straně poptávky a nabídky. U obou lze zvolit přístup agregující, či brát úvahu jednotlivé cestující, či spoje nebo oba přístupy kombinovat. Při agregaci na straně nabídky se neberou v úvahu jednotlivé spoje, ale jejich skupiny. Kdykoliv není přístup přímo ke konkrétním jízdním řádům, jedná se o tuto agregaci. Tento vztah ale neplatí opačně, lze agregovat jednotlivé spoje i ve chvíli, kdy se počítá s jízdním řádem.

Dále lze modely rozdělit dle přístupu k nákladům. Ty jednodušší berou jako náklady pro cestující pouze čas, ty složitější rozlišují i různá cenová pásma, pohodlnost jednotlivých dopravních prostředků, pohodlnost přestupů atd. Často vznikají tyto náklady jako pokuty při nesplnění dosahovaných časových cílů (např. cestující chce někde být ve 12:00, ale dorazí ve 12:05, oněch 5 minut nelze jednoduše přičíst k celkovým časovým nákladům, protože jeho újma je velmi pravděpodobně vyšší). Tyto náklady taktéž vznikají při příliš brzkém dojezdu (např. cestující chce někde být ve

12:00, dorazí v 11:30, cesta mu trvala 30 minut, ale jeho náklady budou určitě vyšší a budou se blížit nákladům odpovídajícím dojezdu ve 12:00).

Dále je významné dělení na modely pracující s jízdním řády a modely pracující pouze s intervaly. Toto dělení těsně souvisí i s dělením podle informovanosti cestujícího, frekvencí spojů a způsobu přiřazení poptávky nabídce (stochasticky či deterministicky).

Modely pracující pouze s intervaly používají ke svému fungování pouze část informace obsažené v modelech pracujících s jízdními řády. Tyto modely mají následující výhody (viz. např. CONSTANTIN, 1998, či NIELSEN, 2000):

- jednodušší naplnění databáze
- vyšší rychlost při tvorbě analýz
- možnost jednoduše je upravovat a tím pádem snadná simulace různých scénářů

Na rozdíl od nich mají modely pracující s jízdními řády následující výhody:

- v případě přesné znalosti preference cestujících jednoznačné (tedy deterministické) přiřazení poptávky k nabídce
- přesněji se modelují přestupy
- obecně vyšší přesnost

Pokud model pracuje s jízdními řády a každá poptávka v sobě nese přesné časové údaje o odjezdu a příjezdu, lze z toho odvozovat následující fakta. Je – li cestující o těchto řádech informován (většinou při nižší frekvenci spojů), a je – li užitková funkce určena deterministicky, pak je přiřazení poptávky nabídce rovněž deterministické a užitek (náklady) je optimalizován. Deterministické přiřazení znamená, že pro jednu poptávku, existuje právě jedna nabídka, která této poptávce bude přidělena. V případě stochastického přiřazení toto nelze s jistotou určit a je třeba pracovat s pravděpodobnostmi jednotlivých variant. Pokud za stejných předpokladů cestující o jízdních řádech informován není, zná jen intervaly (většinou při vyšších frekvencích spojů), je jeho cesta taktéž jasně určena (determinována), protože volí vždy variantu, u které očekává nejnižší náklady. Nicméně nutno dodat, že tato varianta vzhledem k reálnému jízdnímu řádu již nemusí být optimální. Tento případ by se modeloval poměrně obtížně. Toto všechno samozřejmě neplatí ve chvíli, kdy není přesně známa užitková funkce cestujícího. V tom případě je přiřazení vždy stochastické.

Pokud model s jízdními řády nepracuje, mohou nastat následující situace. Užitková funkce je známa a cestující je informován o jízdním řádu. Z důvodu, že v modelu tato informace není, nemůžeme předpokládat, pro jakou variantu se v závislosti na čase rozhodne. Přidělení poptávky nabídce je tedy stochastické. Pokud v tomto případě cestující zná pouze intervaly nastává následující situace. Deterministicky si zvolí

výchozí stanici, ale jeho další cestu už ovlivňuje např. to, který spoj přijede dřív, takže přidělení poptávky nabídce probíhá opět stochasticky.

O vlivu informovanosti na způsob přiřazení poptávky již byla řeč. Co se týká deterministického či stochastického způsobu přiřazení poptávky, z předcházejících řádků vyplývá, že toto dělení je vlastně ovlivněno dalšími parametry modelu, jejichž volba určuje i tento parametr. Dále se o roli informací v procesu rozhodování zmiňuje např. LARSEN a ØYVIND (2008).

Frekvence spojů mohou ovlivnit nejen to, jestli cestující dobrovolně zvolí informovanost či neinformovanost, ale při vysokých frekvencích je nutno uvážit i to, že cestující uvažuje nad zaplněností vozidel a jejich kapacitou. Těmto aspektům se však práce nevěnuje. Jedním z důvodů je to, že ve středoevropské realitě nejsou případy, kdy by se cestující nevešli do dopravního prostředku časté a také proto, že by bylo tyto případy obtížné modelovat.

V práci se autor bude věnovat modelu, který je zaměřený na provádění analýz. Model pracuje s deterministicky stanovenou poptávkovou funkcí cestujících a stochastickým přiřazením poptávky nabídce, v důsledku toho, že model pracuje pouze s intervaly. Model předpokládá, že cestující je informovaný a jeho jediným kritériem volby spoje je čas.

2.3. Popis vybraných modelů

V této podkapitole jsou popsány vybrané modely, rozdělené do několika skupin. Ve většině příkladů se jedná o modely, jejichž cílem je upravit síť hromadné dopravy tak, aby byly lépe uspokojeny požadavky zákazníků, tedy cestujících. Na začátek jsou však uváděny příklady, jejichž cílem je v prvním případě naplánovat vedení linek a v druhém případě stanovit optimální vzdálenosti.

2.3.1. Modely pro plánování

V modelu pro plánování linek (BORNDÖRFER, GRÖTSCHEL, PFETSCH, 2007) se pracuje se 3 předpoklady. První z nich je tzv. rozdělení systému, které vyžaduje aby byla dopředu známá síť a pro každou část sítě aby byla známa její rychlostní charakteristika (např. pro železnice bude rychlost vyšší než pro autobus ve městě apod.). Poté při předpokladu, že lidé budou chtít co největší část své cesty ujet nejrychlejším dopravním prostředkem, lze jednotlivým částem sítě už dopředu přiřadit počty pasažérů. Dále je uveden předpoklad, že optimální linky budou vybírány jen z omezeného počtu možností, a posledním často užívaným předpokladem plánování je ten, že systém je optimální ve chvíli, kdy je minimalizován počet přestupů. Nemusí se tedy brát v úvahu přestupní časy. Tyto předpoklady dokáží zjednodušit plánovací proces. Samozřejmě, že díky nim vznikají jisté nepřesnosti, které se složitějším modelováním při nevyužití těchto předpokladů dají odstranit.

Jednoduchý model, který má za cíl optimalizovat vzdálenosti mezi zastávkami publikoval LOWSON (2004). Představuje zde základní model, jehož součástí je jen koridor se zastávkami. Dospívá k tomu, že území, která budou příslušet dané zastávce, jsou v závislosti na rychlosti dopravního prostředku posunuty proti směru jízdy, což je logické. Pokud se cestující bude nacházet přesně mezi 2 zastávkami, vydá se na tu, která je ve směru jeho jízdy. Ve druhém případě uvažuje model, jenž je tvořen sítí linek ve směru sever-jih a východ-západ. Pro každý OD pár pak lze dospět do cíle s max. jedním přestupem. Výsledkem zkoumání obou těchto modelů jsou optimální vzdálenosti mezi zastávkami při dané maximální rychlosti vozidla.

2.3.2. Modely pro úpravu sítě

Další příklady lze v zásadě rozdělit na ty, které se spíše věnují datové reprezentaci modelů a těm, které zajímá spíše způsob přiřazení poptávky nabídce, ať už čistě teoreticky, v algoritmickém vyjádření či přímo s implementačními detaily. Pak existují práce, které se věnují problému komplexně.

U modelu (FLORIAN, 1999) je pro specifikaci datového modelu volena poměrně jednoduchá reprezentace. Pro každou linku jsou specifikovány intervaly, čas po začátku periody kdy linka vyjíždí poprvé, počet běhů linky a pak pro každý segment jízdy (mezi 2 zastávkami) čas a pro každou zastávku čas, kdy cestující nastupují a vystupují. Pro každou jízdu (OD pár) je pak ještě definován požadovaný čas v cíli, případně také parametry, které udávají jak moc se tento čas může od reálně dosaženého času lišit (v obou směrech, pro každý parametr jeden), tu samou specifikaci obsahuje i u požadovaného času odjezdu. Zde je navíc specifikována tzv. granularita, tj. v jakých intervalech může cestující na cestu vyrazit. Samozřejmě, že nejblíže realitě je granularita blížící se 0, ovšem to velmi zatěžuje výpočetní výkon. Také je možné nastavit faktory, kterými se bude posuzovat pozdní či moc brzký dojezd. Algoritmus v této práci zmíněn není.

V práci věnující se modelu hromadné dopravy Vilniusu (UŠPALYTĖ-VITKUNIENĖ, BURINSKIENĖ, GRIGONIS, 2006) byla k dispozici dopravní data ve formě OD matic (matice start – cíl), a hledal se vhodný model z několika vybraných. Účelem použití takovýchto modelů nejspíše bude revize současného vedení linek, intervalů apod. Bylo vytvořeno 230 oblastí kterým byly přiřazeny jednotlivé body (starty či cíle). Z těchto oblastí byly poté zkonstruovány spojnice na nejbližší zastávky s ohledem na terénní překážky atd. Dále byla vzata do úvahy data z výzkumu, který zkoumal preference cestujících (tj. jestli upřednostňují jízdu bez přestupu apod.) a na základě těchto dat byly stanoveny koeficienty pro výpočet užitkové funkce. Byly hodnoceny následující modely: tzv. TSys model, kde se berou do úvahy pouze linky bez vazby na jejich intervaly. Stanoví se čas, který je potřebný na přestup a pak už se jen pro danou OD matici najde nejkratší cesta. Přiřazení je v tomto případě deterministické, stejně jako užitková funkce. Výsledky jsou dost vzdálené od reality.

Dále byl prověřen model, kde se zavádějí k linkám jejich intervaly. V takovém případě už není přidělení poptávky nabídky deterministické, ale stochastické (proč tomu tak je, viz výše). Model se již více přibližují realitě, ale co se týká např. počtu přestupů stále zaostává za modelem, který pracuje s jízdními řády.

Konečně modely s jízdními řády byly zkoumány 4. Lišily se v tom, jakým způsobem je z koeficientů odvozována užitková funkce. Všechny 4 předpokládaly úplnou informovanost cestujících. Zejména v případě modelu, který používal Kirchhoffovo rozdělení užitkové funkce byl model docela blízký realitě.

Další model (O'SULLIVAN, MORRISON, SHEARE, 2000) má za cíl provádění analýz dostupnosti pro vlaky a autobusy. Data jsou reprezentována v následující formě. Pro vlaky je pro každé zastavení určen jeden záznam v tabulce. Ten obsahuje číslo linky, pořadí stanice, identifikaci stanice a čas (v minutách, symbolizuje čas po celé hodině kdy vlak staví ve stanici, použitelné za předpokladu, že tyto časy jsou pro každou hodinu stejné). Dále tabulku stanic s jejich umístěním a pak tabulku jednotlivých autobusových spojů, které jsou ve formátu číslo linky, sekvence linií, trvání cesty a interval. Model předpokládá u vlaků informovaného cestujícího a u autobusů neinformovaného cestujícího. Z libovolného bodu je vždy nalezena nejbližší stanice vlaku, nebo autobusová linka (ten nemá definovány zastávky) buď metodou Voronoi diagramů (jsou zmiňovány i jiné metody), či v případě autobusů nejkratší vzdáleností. Algoritmus, který provádí analýzu dostupnosti jednotlivých míst je pak implementací Dijkstrova algoritmu.

V dalším modelu (JANSSON, RIDDERSTOLPE, 1992) pracuje systém jen s údaji o intervalech. Uvažují se 4 základní varianty. U první z nich je předpoklad, že cestující je informován o intervalech, o přesných časech odjezdu, i o jízdních časech. U druhé cestující nemá informace o přesných časech odjezdu. V třetím případě předpokládáme, že interval mezi odjezdy stejné linky je konstantní a odjezdy různých linek na sobě nejsou závislé. Ve 4. případě platí to samé, ale odjezdy jsou koordinovány. V případě že pasažér zná časy odjezdů, nemůžeme s jistotou říci na jakou zastávku se vydá, protože toto rozhodnutí závisí na čase kdy se rozhoduje, model tedy pro tyto případy počítá pravděpodobnosti volby jednotlivých tras. V případě, že cestující časy nezná, se rozhodne vždy pro jednu variantu. V práci není zmíněno jaký datový model se používá. Jedním z předpokládaných zjištění je to, že v případě, kdy cestující má plné informace, je jízdní doba v průměru nižší, tj. se zde objevuje termín „cena informace“. Tento model se velmi podobá výsledkům autora této práce (v době návrhu modelu nebyl autorovi znám tento zdroj).

HUANG (2007) přichází s novou metodou procházení sítě, a to tzv. „pattern search first“. Pattern, tedy vzor, zde představuje posloupnost zastávek s konkrétními časy, lze ho tedy chápat jako průjezd jednoho vozidla jedné linky po dané trase. Další součástí sítě jsou ještě zmiňované zastávky a přestupní body. Nejprve je inicializována síť pro

práci s kompletním jízdním řádem a pak na základě časových atributů jednotlivých OD párů je vždy inicializována část takto vytvářené sítě a označena jako aktivní. Nad ní poté probíhá algoritmus podobný Dijkstrovu algoritmu.

2.4. Reprezentace modelů v prostředí GIS

2.4.1. Základní dělení reprezentací

V této kapitole bude objasněno jak jsou dopravní modely uloženy v prostředí GIS a jak je s nimi v tomto prostředí pracováno. Prostorová data v GIS lze reprezentovat dle (THILL, 2000) pomocí 3 různých modelů:

- model polí, neboli rastrový model

- diskrétní model

- síťový model

V rastrovém modelu jsou jednotlivé části povrchu reprezentovány pixely, které nesou informaci o svém umístění a nějakou hodnotu (případně hodnoty). Hodí se pro zobrazení spojitých jevů (nadmořská výška, typ krajinného pokryvu, hustota zalidnění)

U diskrétního modelu jsou v popředí entity (body, linie, polygony) a jejich umístění v prostoru, ovšem bez vzájemného provázání.

U síťového modelu jsou základními stavebními kameny uzly a hrany. Hrany jsou nějakým způsobem vázány na liniové geometrické prvky a uzly na bodové geometrické prvky. Každá hrana pak sestává z 1-2 uzlů a každý uzel má 1-n hran.

2.4.2. Prostorová složka modelů

Nyní je potřeba si položit otázku, jaký z výše uvedených modelů použít, či je-li možné použít nějaký úplně jiný model. Nejprve je třeba si ujasnit, zda-li je vůbec potřeba datový model, ve kterém je nějakým způsobem přítomna prostorová složka. Např. systém IDOS pro vyhodnocování požadavků uživatelů na spojení může v nejjednodušší verzi pracovat i jen s neprostorovou databází, kde jsou k linkám uloženy stanice, jejich řády ad., ale není zde potřeba mít přístup k prostorovým datům. Pokud bude toto zobecněno, tak prostorová data nejsou potřeba v následujících případech: tam, kde je poptávka na přepravu vyjádřena pouze názvy, či kódy stanic či ulic; a zároveň tam, kde jsou spolu jednotlivé entity propojeny jinak než prostorově (každá ulice má kód nejbližší zastávky, nějakým způsobem jsou propojeny vlakové stanice s autobusovými zastávkami apod.). Jak je vidět, tyto předpoklady mohou být celkem pohodlně naplněny při zadání poptávky jako 2 stanic. Při zadání poptávky např. formou adresy již nelze dosáhnout uspokojivých výsledků a tento postup samozřejmě nefunguje, když se zadá poptávka jako bod. Také možnost provádět některé analýzy je omezena. Např. při analýze dostupnosti z jednoho místa sice existuje možnost spočítat

dostupnost i pro ostatní místa, ale nelze žádným způsobem provádět např. interpolaci.

Ve většině případů je tedy užitečné mít v datech nějakým způsobem přítomnou prostorovou složku. Rastrová reprezentace zde nepřichází v úvahu. Diskrétní model je velmi jednoduchý na vytvoření, nicméně aby mohla být prostorová data využita k analýzám, je při každé analýze potřeba nad tímto modelem dopočítávat vzájemné průsečíky linek a další vlastnosti sítě. V dnešní době, kdy velikost úložných prostor výpočetních jednotek je v podstatě neomezená, zatímco jejich výkon stále značně omezen, se toto nejeví jako ideální řešení, i když pro některé případy může být určitě dostačující. Řešením je tedy nějakým způsobem implementovaný síťový model.

2.4.3. Reprezentace nabídky, poptávky a přiřazení v prostředí GIS

Na tomto místě následuje rozbor možnosti, jak lze poptávku, nabídku a způsob přiřazení nabídky poptávce reprezentovat. Poptávka je požadavek na přepravu z místa na jiné místo doplněný o další atributy (čas, preference cestujícího, atd.). Jedná se tedy o situaci, kdy je potřeba nějakým způsobem uložit a zpracovávat dva body v daném pořadí. Na to se jeví být nejvhodnějším diskrétní model. Samozřejmě, že tento přístup opomíjí faktory, které ke vzniku takto jasně definované poptávky vedly, tedy např. hustota zalidnění, funkční využití prostoru ad. I když i tyto faktory mohou být součástí MLHD, jejich reprezentace v GIS už není primárně záležitostí těchto modelů, ale komplexnějších modelů, proto tu nebude řešena.

Stranu nabídky lze reprezentovat různými způsoby. Pro linkovou hromadnou dopravu je rastrový model neobvyklý, přesto je třeba ho uvést, ačkoliv je o něm jen velmi málo zmínek v literatuře v této souvislosti a dle (THILL, 2000) je zdaleka nejpoužívanějším pro MLHD, stejně jako pro ostatní modely dopravy, síťový model.

Pokud bude požadavek modelovat linkovou dopravu, resp. její nabídkovou část v prostředí GIS, je třeba si uvědomit několik specifik. Základem pro takovéto modelování bude síť, tak jak byla představena v předchozí kapitole. Hrany budou reprezentovat úseky na kterých doprava probíhá, uzly budou většinou představovat zastávky, a přemostění budou představovat jednotlivé linky. Je důležité podotknout, že v neupravené verzi takovéhoto modelu pro potřeby provádění síťových analýz je nutné, aby každá linka měla své vlastní geometrické entity. K takto definovaným součástem sítě bude samozřejmě zapotřebí definovat atributy. V případě hran se bude jednat např. o jízdní dobu, délku úseku, současný stav úseku (v provozu, v opravě), maximální rychlost ad. V případě uzlů lze např. definovat dobu, po kterou se cestující na uzlu zdrží.

U datové reprezentace síťového modelu se lze setkat s problémy a jejich řešením, které vznikají například potřebou omezit volbu směrů zatáčení, či definovat jednotlivé linky na trasách. Tyto datové reprezentace spoří místo a zlepšují výkon při vykonávání síťových algoritmů.

Příkladem takové reprezentace jsou např. duální sítě (AŇEZ, DE LA BARRA, PÉREZ, 1996). Pokud by bylo potřeba na klasicky vybudované síti omezit na některých uzlech možnosti volby hran (zjednodušeně si lze představit jako případ, kdy je nutno na křižovatce zamezit odbočení), vzniká podstatně složitější model, s více uzly i hranami. Místo jednoho centrálního uzlu (křižovatky) je nutno na každé hraně vytvořit nový uzel a ten pak propojit s takto vytvořeným uzlem hran, u kterých je volba této hrany (odbočení) povolena. Pro složitější pravidla tento fakt omezuje čitelnost modelu a zvyšuje jeho složitost. Aby se těmto důsledkům zabránilo vzniká druhá síť, kde jednotlivé uzly jsou představovány hranami původní sítě a hrany této nové sítě určují povolené volby hran původní sítě (zatáčení). Takto zkonstruovanou síť lze např. použít i při konstrukci modelu linkové dopravy, kde hrany druhé sítě představují možné přestupy mezi jednotlivými linkami (samozřejmě se této hraně přiřadí cena, která určuje cenu takového přestupu). Jako další příklad lze uvést použití diachronických sítí, tentokrát bez podrobného popisu.

Způsob přiřazení nabídky poptávce je na úrovni GIS řešen algoritmem, který je na základě poptávky schopen určit, po jakých trasách ze strany nabídky se cestující budou přepravovat. Základními algoritmy zde jsou síťové algoritmy, které budou zmíněny v části zabývající se konkrétní implementací algoritmů.

2.5. Shrnutí

V této kapitole byl vymezen termín „model linkové hromadné dopravy“ a dále z dostupných zdrojů literatury řešeno rozdělování takovýchto modelů a jejich příklady. Kritérií dle kterých lze modely rozdělovat do skupin je mnoho, zmiňme základní dělení na modely pracující s jízdním řádem a modely pracující jen s intervaly, které je pro potřeby práce důležité. Z tohoto rozdělení pak částečně i vychází rozdělení modelů na stochastické a deterministické.

KAPITOLA 3: KONSTRUKCE VLASTNÍHO MODELU

Tato kapitola bude obsahovat popis autorem sestrojeného modelu pro editaci sítě a provádění analýz. Proces vytvoření tohoto modelu se skládá z objasnění teoretických základů a předpokladů, na kterých bude model založen a pak ze samotného popisu datového modelu. Obě fáze budou vycházet z již popsaných modelů z předchozí kapitoly, ale v žádném případě se nedá říct, že by nově formulovaný model byl kopií nějakého z nich. Spíše by bylo možné ho brát jako kombinaci více těchto modelů s inovativními prvky autora.

Trochu netradičně bude nejprve probrán přístup k analýzám a až poté datový model. Je to z důvodu, že datový model musí reflektovat požadavky, které vyplývají z formulace toho, jak budou probíhat analýzy.

Pokud se hovoří o analýzách, má autor na mysli v první řadě analýzu nejkratší a nejpravděpodobnější cesty. Z této analýzy pak bude vycházet poté i analýza dostupnosti jednotlivých míst. Jako vstupní data této analýzy poslouží dva vstupní body, které se nacházejí na uliční síti, dopravní síť nadefinovaná v databázi a vybrané parametry analýzy. Výstupem z této analýzy je sada cest, u každé cesty včetně popisu použitých linek, úseků absolvovaných pěšky, čekacích časů, popisu geometrie cesty. Nejdůležitějšími parametry každé cesty, která bude výsledkem analýzy je vždy pravděpodobnost, se kterou cestující tuto cestu zvolí a čas, který mu cesta zabere.

3.1. Vstupní předpoklady

Nyní je nutno stanovit předpoklady za kterých budou analýzy probíhat. Prvním předpokladem, již několikrát zmíněným, je fakt, že jsou k dispozici pouze intervaly jednotlivých linek, namísto konkrétních jízdních řádů. Dále je známa jízdní doba mezi 2 zastávkami linky, stejně jako rychlost pohybu chodce po uliční síti. U cestujících se předpokládá, že se o cestě dopředu informují, tudíž ve chvíli kdy vyrážejí na cestu už přesně vědí, jak tato cesta bude vypadat. Zanedbávají se jakékoliv odchylky od normálního stavu jakými jsou např. zpoždění jednotlivých spojů. Platí, že cestující při jednou zvolené cestě už tuto cestu nemění. Dále se u cestujícího předpokládá, že nebere v úvahu jiné náklady než časové, a že se snaží být v cíli své cesty vždy co nejrychleji, bez ohledu na to, že vlastní doba strávená na cestě může být delší než v jiných případech. Počítá se s tím, že cestující optimalizuje nástup do prvního vozidla tak, aby

minimalizoval jízdní dobu. To v praxi znamená, že si nechá první spoj na trase ujet v případě, že druhým spojem dojede do cíle ve stejný čas. U mezistaničních úseků, po kterých jede více spojů se předpokládá jejich rovnoměrné rozložení.

Na tomto místě je nutné zmínit některá specifika analýzy. Při tradičních analýzách nejkratší cesty je výsledkem vždy časově nejkratší (či jiným způsobem nejefektivnější cesta). V tomto případě je ovšem výsledků více. Proč se uvažuje i o jiných cestách než o té nejkratší? Následuje krátký příklad: Linka 1 absolvuje cestu z bodu A do bodu B za 20 minut a jezdí každých 60 minut. Linka 2 tuto cestu absolvuje za 30 minut a jezdí jednou za 10 minut. Bylo by správné usuzovat, že cestující vždy využije linku 1, protože s ní cestuje o 10 minut kratší dobu? Bylo by správné uvažovat vzájemnou dostupnost bodů A a B jako 20-ti minutovou? To velmi záleží na úhlu pohledu. Člověk, který je časově absolutně flexibilní, vždy upřednostní linku 1 s tím, že na místo dorazí v čas, který má jen malou možnost ovlivnit. Jeho čas dojezdu se v tomto případě řídí jízdním řádem. Druhým extrémem je člověk, který je časově absolutně neflexibilní. V praxi to znamená, že vždy preferuje takovou možnost dopravy, při které dorazí na cílové místo v předepsaný okamžik, či co nejdříve před tímto okamžikem. Opět následuje příklad: Cestující chce někde být přesně v 9:00 či dříve. Má možnost jet buď linkou 3 v 8:20 a být na místě v 8:55, nebo jet linkou 4 v 8:35 a být na místě v 9:01, volí linku 3, i když mu cesta zabere o 9 minut déle. Jak bylo uvedeno v předchozím odstavci, právě předpoklad neflexibilního cestujícího bude základem modelu popsaného v následujících odstavcích. V případě níže popisovaného modelu se neflexibilita chápe především jako vůle cestujícího vyrazit v přesně stanovený čas a zvolit trasu takovou, která ho dovede do cíle co nejrychleji od této chvíle startu. Vzhledem k tomu, že se uvažuje o cestujícím, který je informovaný, bude běžný scénář následující: Cestující vyjádří vůli cestovat ve zvoleném budoucím čase, vyhledá spoj, který mu umožní dorazit do cíle co nejdříve od tohoto určeného času, a poté tohoto spoje využije. Přičemž se bere, že čas strávený čekáním na první spoj nebude do celkového času započítán, a to z důvodu, že tento čas může cestující strávit např. v pohodlí domova, i když to samozřejmě nemusí být podmínkou.

3.2. Výpočet pravděpodobnosti užití variant

V několika následujících odstavcích se autor věnuje metodice výpočtu pravděpodobností, se kterou cestující zvolí danou linku (či celou trasu, složenou z několika linek). Vzhledem k tomu, že se počítá s modelem informovaného cestujícího, ví se také, že jeho cesta je pevně určena již ve chvíli, kdy se o ní dopředu informuje. Z pohledu analytika ovšem přesně známá nebude nikdy, protože nejsou známé přesné jízdní řády. Může se tedy jen odhadovat s jakou pravděpodobností nastanou takové podmínky, aby využil jednotlivých tras. Při následujícím výpočtu se bude vycházet z výše uvedených předpokladů. V následujících kapitolách uvedené možnosti představují postupné zesložiťování výpočtu od 2 variant cesty mezi danými body A a B se stejnými

jízdními dobami až po libovolný počet variant s různými jízdními dobami i intervaly. V grafech se nachází na ose X čas do odjezdu linky, a na ose Y pravděpodobnost, že cestující danou linku využije v případě, že v daný čas dorazí na zastávku.

3.2.1. Dvě varianty se stejnou jízdní dobou a různým intervalem

Lze začít s nejjednodušším případem, kdy z bodu A do bodu B může cestující volit mezi právě 2 linkami a cesta oběma linkami trvá stejně dlouhou dobu. V tomto případě ve chvíli, kdy bude chtít vyrazit volí vždy linku, která pojede dřív. Nyní bude na tuto situaci nahlíženo z pohledu časové osy jednotlivých linek.

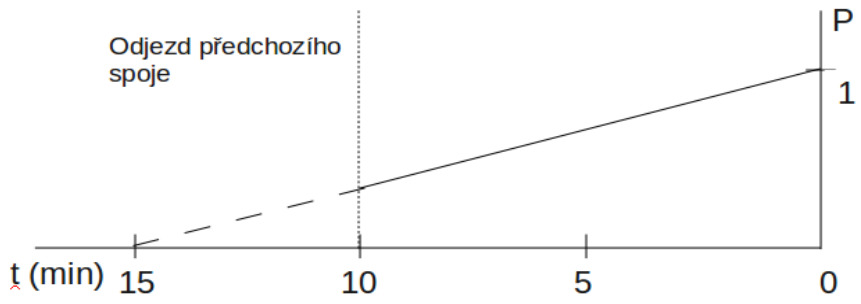
Příklad 1:

Vzdálenost mezi body A a B pro linku 1: nepodstatná např.10 min.

Vzdálenost mezi body A a B pro linku 2: nepodstatná, ale stejná jako u linky 1=10 min.

Interval linky 1: 10 min

Interval linky 2: 15 min



Obr. 3.1: Pravděpodobnost využití linky – stejné jízdní doby, 1 alternativa

Na obr. 3.1 lze vidět situaci z pohledu linky číslo 1. Časová osa je v tomto případě dlouhá 10 minut, což odpovídá intervalu linky. Lze uvažovat následovně: Linka 1 bude zvolena v takový okamžik začátku cesty, kdy od doby, kdy je vyjádřena vůle zahájit cestu po skutečný čas, kdy linka 1 jede nepojede linka 2. V případě, že by jela, byla by zvolena. Je jasné, že zbývá-li do odjezdu linky 1 čas 2 minuty, je pravděpodobnost, že v této době pojede linka č. 2 2/15, obecně $2/I_2$, tedy pravděpodobnost, že bude užita linka 1 je 1-2/15, tedy 13/15. V každém časovém okamžiku tedy lze vyjádřit pravděpodobnost že se využije linka 1 jako: $1-t/I_2$, kde t je čas do odjezdu linky 1. Z toho lze odvodit, že i v případě kdy bude cestující chtít začít cestu přesně ve chvíli, kdy linka 1 odjíždí, má stále pravděpodobnost cca. 33%, že linku 1 zvolí. To je totiž pravděpodobnost, že v následujících 10 minutách linka 2 vůbec nepojede. Uváží-li se

interval jako celek a zeptáme-li se v kolika minutách z tohoto intervalu využijeme tuto linku, dospějeme k názoru, že se jedná o integrál výše zobrazené funkce.

Příklad 1 – pravděpodobnost, že v daný okamžik zvolím danou linku

(1) $P = 1 - \frac{t}{I_{alt}}$, kde I_{alt} je interval druhé (alternativní linky), viz obr. 3.1

Příklad 1 – v jakém časovém úseku z intervalu dané linky tuto linku využiji

(2) $T = \int (1 - \frac{t}{I_{alt}})\,dt = [\,t - \frac{t^2}{2I_{alt}}]_0^I = I - \frac{I^2}{2I_{alt}}$, kde I je interval dané linky, viz obr. 3.1

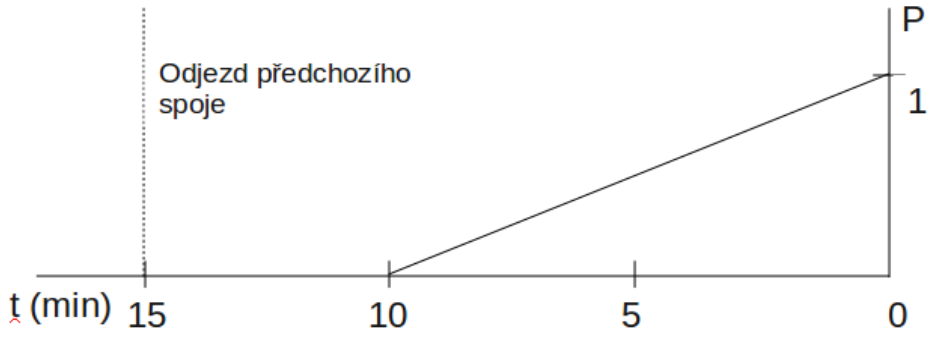
Příklad 1 – pravděpodobnost, že linka bude využita v libovolném čase

(3) $P = \frac{T}{I} = \frac{I - \frac{I^2}{2I_{alt}}}{I} = 1 - \frac{I}{2I_{alt}}$

Příklad 1 – dosazení

$$P = 1 - \frac{I}{2I_{alt}} = 1 - \frac{10}{2.15} = 0.\bar{6} = 66.\bar{6}\,\%$$

Pro ověření nahlédněme na tuto situaci z pohledu časové osy linky 2 (obr. 3.2). Časová osa bude opět odpovídat intervalu, tedy 15 minutám.



Obr. 3.2: Pravděpodobnost využití linky – stejné jízdní doby, 1 alternativa, druhý pohled

Z obrázku 3.2 lze usuzovat, že do 5 minut po odjezdu linky 2, je jasné, že bude zvolena linka 1. Je to z důvodu, že do následujícího odjezdu linky 2 chybí více jak 10 minut a tedy je jistota, že v tomto čase přijede linka 1, která bude upřednostněna. Nyní je nutno poupravit vzorec (1), tak aby reflektoval skutečnost, že pravděpodobnost musí nabývat hodnot mezi 0 a 1:

(4) $P=min\left(0,max\left(1{,}1-\frac{t}{I_{alt}}\right)\right)$, viz obr. 3.2

Ve chvíli, kdy se bude počítat určitý integrál, je tedy třeba rozdělit interval linky na čas od 0 do 10, kdy lze použít výše vypočtený určitý integrál a na čas od 10 do 15, kdy je určitý integrál roven 0, protože pravděpodobnost je rovna taktéž 0.

Příklad 1 – dosazení, pravděpodobnost využití linky 2, vychází z (2)

$$P=[t-\frac{t^2}{2\mathrm{I}_{alt}}]_0^I=\frac{(10-\frac{100}{2.10})}{15}=0.\bar{3}=33.\bar{3}\,\%$$

Lze vidět, že pravděpodobnosti obou linek dávají dohromady 100, což lze brát jako jedno z potvrzení, že vzorec je platný. Pokud je na situaci linky číslo 2 nahlíženo zjednodušeným pohledem, lze dojít k závěru, že 5 minut z 15 ji nevyužijeme nikdy a ve zbylých 10 minutách je šance přesně 50%. I tímto výpočtem lze dospět k číslu 33,33%.

3.2.2. Dvě varianty s odlišnou jízdní dobou a různým intervalem

Příklad 2:

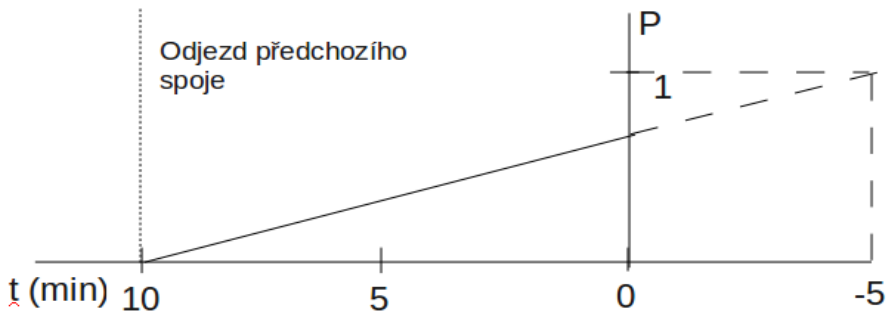
Vzdálenost mezi body A a B pro linku 1: 15 min.

Vzdálenost mezi body A a B pro linku 2: 10 min.

Interval linky 1: 10 min

Interval linky 2: 15 min

Nyní situaci lehce zkomplikujme a stanovme, že linka číslo 1 bude mít jízdní dobu 15 minut. K čemu nyní dojde? Je jasné, že pokud bude cestující chtít vyrazit, a linka 1 pojede v čase t a linka 2 v čase t+3, počká další 3 minuty a poté využije linku 2. Do cíle dorazí o 2 minuty dříve. Pokud bude chtít nastoupit cestu těsně před časem odjezdu linky 1, stále není jisté, zda-li v následujících 5 minutách nepojede linka 2 – v tom případě by ji upřednostnil. Situace na časové ose vypadá následovně (obr. 3.3):



Obr. 3.3: Pravděpodobnost využití linky – různé jízdní doby, 1 alternativa

Příklad 2 – pravděpodobnost, že v daný okamžik bude zvolena daná linku

(5) $P = 1 - \frac{t - D}{I_{alt}}$, viz obr. 3.3

kde I_{alt} je interval druhé (alternativní linky) a D je časový rozdíl mezi jízdními časy linek. V případě, že má daná linka tento čas nižší než alternativní, je D kladné

Příklad 2 – v jakém časovém úseku z intervalu dané linky tato linka bude využita

(6) $P = \int (1 - \frac{t}{I_{alt}} + \frac{D}{I_{alt}})\, dt = [\, t - \frac{t^2}{2\mathrm{I}_{alt}} + \frac{D.t}{I_{alt}}]_0^I$, viz obr. 3.3

kde I je interval dané linky, v intervalu, kde je pravděpodobnost menší než 0, či větší než 1, probíhá integrace těchto hodnot (0 a 1), v ostatních intervalech je využit uvedený vzorec.

Příklad 2 – dosazení, D= -5, integrovaný výraz je v intervalu <0;1> pro všechna t

$$P = [\, t - \frac{t^2}{2\mathrm{I}_{alt}} + \frac{D.t}{I_{alt}}]_0^I = \frac{I - \frac{I^2}{2\mathrm{I}_{alt}} + \frac{DI}{I_{alt}}}{I} = 1 - \frac{I - 2\mathrm{D}}{2\mathrm{I}_{alt}} = 1 - \frac{10 + 10}{2.15} = 0.\bar{3} = 33.\bar{3}\,\%$$

Změna v jízdní době 5 minut, tedy znamenala otočení preferencí obou linek. Nyní častěji bude využita linka číslo 2. Při analýze situace z pohledu linky 2 lze postupovat analogicky jako v předchozím případě a výsledek bude po sečtení s tímto dávat 100%. Opět nutno dbát na to, že hodnota funkce, která určuje pravděpodobnost využití dané linky se vždy musí pohybovat v oboru hodnot 0-1. Také dodejme, že tímto výpočtem lze situace řešit i v případě, že existuje linka, která má i kratší interval i kratší jízdní dobu, jen se příslušně upraví dané koeficienty. V případě, kdy je rozdíl jízdních dob větší než interval linky, která má kratší jízdní dobu, nastává situace, kdy je jisté, že vždy bude upřednostněna tato linka, před druhou linkou. Je to z důvodu, že se vždy vyplatí počkat klidně i po dobu celého intervalu linky a vždy do cíle cestující dorazí dříve.

3.2.3. Více variant s odlišnou jízdní dobou a různým intervalem

Příklad 3:

Vzdálenost mezi body A a B pro linku 1: 15 min.

Vzdálenost mezi body A a B pro linku 2: 10 min.

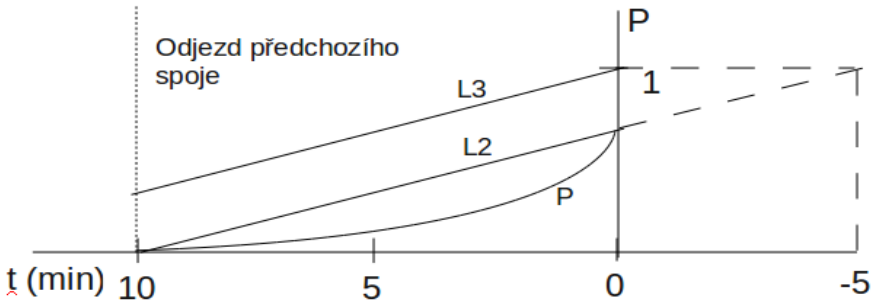
Vzdálenost mezi body A a B pro linku 3: 15 min.

Interval linky 1: 10 min

Interval linky 2: 15 min

Interval linky 3: 15 min

Nyní situaci více přibližme reálným situacím. Bývá obvyklé, že do daného místa se nelze dostat pouze 2 způsoby, ale způsoby více, berme v úvahu, že těchto možností může být jakýkoliv počet. Podívejme se opět na situaci z pohledu jedné linky. Dle odstavců výše lze vždy spočítat pravděpodobnost, s jakou tuto linku v daný moment cestující využije ve srovnání s jinou linkou. U tohoto přístupu zůstaňme, nicméně pro danou linku si do časové osy zakresleme všechny kombinace.



Obr. 3.4: Pravděpodobnost využití linky – více alternativ

Jak nyní situaci vyhodnotit? Například v čase 2 minuty před odjezdem linky 1, je pravděpodobnost, že cestující upřednostní linku 1 před linkou 2 8/15 , před linkou 3 13/15, což odpovídá hodnotám, které lze zjistit z grafu (obr. 3.4). Lze říci, že v tomto čase (2 minuty před odjezdem) cestující využije linku 1 tehdy, pokud do té doby nepojede linka 2 a zároveň nepojede linka 3 (je nutno ještě provést úpravu o rozdíl v jízdních dobách).

Příklad 3 – pravděpodobnost využití linky 1 při příchodu 2 minuty před jejím odjezdem

$$P = \frac{13}{15} . \frac{8}{15} = \frac{104}{225} = 0.4\overline{6} = 46.\overline{6}\,\%$$

Příklad 3 – analogicky - pravděpodobnost využití linky 1 při příchodu 4 minuty před jejím odjezdem

$$P = \frac{11}{15} . \frac{6}{15} = \frac{66}{225} = 0.29\overline{3} = 29.\overline{3}\,\%$$

Celková pravděpodobnost tedy vznikne vynásobením všech pravděpodobností pro dvojice linek (viz. obr. 3.4). Analogicky k postupu uvedenému výše můžeme čas po který v průměru využijeme linku 1 vyjádřit jako určitý integrál součinu všech pravděpodobnostních funkcí po celé délce intervalu. Vyjádření takovéhoto integrálu matematicky znemožňuje fakt, že funkce nemusí být ve všech bodech hladké. Pokud je jasné že konkurenční linku upřednostní cestující vždy, (hodnota pravděpodobnostní funkce je 0), je pak tedy i jasné že danou linku nevyužije ani v konkurenci více linek,

což odpovídá tomu, že součin, kdy jeden z činitelů je 0, je roven taktéž 0. V opačném případě, kdy existuje jistota, že upřednostní danou linku před konkurenční (hodnota funkce je 1), ovšem celkovou pravděpodobnost mohou ovlivnit ještě další konkurenční linky.

Z důvodu, že výše popsanou funkci, kterou chceme integrovat nelze popsat matematicky (resp. lze, ale jen za pomoci funkcí jako min či max, které nejsou integrovatelné), je třeba si pomoci buď rozdělením intervalu linky do skupin, a integrací v rámci těchto skupin či aproximací. Aproximací je myšleno načítání hodnot vždy po určitém časovém úseku. V tomto případě uvažujme postupné načítání součinů po jedné minutě. Načítání je nutno provádět vždy tak aby bylo symetrické z obou stran časové osy. V tomto případě tedy zvolme poloviny minut. Je jasné, že tímto výpočtem dochází k určité nepřesnosti. Jak však bude vidět z následujících výpočtů, je tato nepřesnost pro potřeby aplikace zanedbatelná a lze dosáhnout ještě dalšího jejího zmírnění. Z důvodu, že se jedná o funkce vyššího řádu,jejichž míra růstu se neustále zvyšuje, při postupném načítání vyjde součet pravděpodobností pro všechny linky menší než 100. Tato hodnota se odchyluje od 100 tím více, čím kratší jsou intervaly a čím více se uvažuje linek. Samozřejmě, že tuto odchylku lze zmírnit taktéž zvětšením počtu načítaných hodnot, např. po čtvrtminutě. Další možností, která se nabízí je proporcionální rozdělení procent, která zbývají do 100, mezi jednotlivé linky. Přímo v aplikaci se ani jedna z těchto metod neuplatňuje, protože výkon dnešních počítačů umožňuje načítání po malých úsecích a tedy dosažení výsledků, které se liší od výsledků získaných integrováním jen nepatrně.

3.3. Výpočet jízdního času a intervalu

3.3.1. Dvě na sebe navazující linky

Prozatím byly řešeny případy, kdy se za jednu variantu považovala pouze jedna linka bez přestupu. V takovém případě je jak jízdní doba, tak interval zřejmý. Jaká ale nastane změna, pokud do cíle cesty povedou i varianty s přestupem? Jak se změní uvažovaná jízdní doba a interval? Začněme jednoduchým propočtem pro variantu s jedním přestupem. Předpokládejme, že cestující si dopředu zjistil informace, lze také uvažovat čekací dobu na první linku jako nulovou, případně nějakou konstantní, přičemž tato konstanta bude blízká 0. Poté co cestující vystoupí z první linky ho nyní čeká čekání na druhý spoj. Vychází se z následujících předpokladů: odjezdy spojů nejsou nijak koordinovány, pravděpodobnost, že v dané chvíli spoj dorazí je v daném časovém období konstantní a jediné co na ni má vliv je interval linky. V takovém případě bude cestující na spoj čekat v průměru polovinu jeho intervalu. Celková jízdní doba bude tedy odpovídat součtu jízdních dob obou linek zvětšenému o prvotní konstantní čekací dobu a interval druhé linky. Za interval pro výpočet pravděpodobnosti volby se v takovém případě bere interval první linky. Níže autor uvádí proč je tento

přístup nesprávný.

Příklad 4:

Interval linky 1: 10 min, odjezdy: 15:05,15:15,15:25,...

Interval linky 2: 30 min, odjezdy: 15:00,15:30,16:00,...

Vzdálenost mezi body A a B pro linku 1: nepodstatná, např. 10 min.

Vzdálenost mezi body A a B pro linku 2: nepodstatná, např. 10 min.

Posloupnost cesty: linka 1 – přestup – linka 2

Dle výše uvedeného výpočtu by byla celková jízdní doba rovna součtu jízdních dob obou linek + poloviny intervalu druhé, tedy 10+10+15 = 35 minuz a takovouto variantu by bylo možno využít každých 10 minut. V reálu se ovšem situace nejspíše vyvine tak, že cestující si zjistí i odjezd následného spoje a tomu uzpůsobí odjezd spoje prvního. Uvažujme cestujícího, který bude chtít zahájit svou cestu v čase 15:20. Pokud nebere v úvahu navazující spoj, cestující vyrazí v 15:25 a v 15:35 dorazí na přestupní bod. Zde bude čekat 25 minut na navazující spoj. Racionálně uvažující cestující však odjede první linkou až ve 15:45, a na navazující spoj bude čekat 5 minut (jede v 16:00). Získá tím 20 minut času např. doma. Shrneme-li tento příklad, lze konstatovat: při cestování s jedním přestupem se jízdní doba vypočte jako součet jízdních dob obou spojů plus polovina kratšího intervalu. Celkový interval takovéhoto spojení je pak roven delšímu z intervalů.

3.3.2. Více na sebe navazujících linek

Příklad 5:

Jako příklad 4 + dále stanoven interval linky 3

Interval linky 3: 5 min

Nyní uvažujme nejobecnější případ, kdy na cestě do cíle je využito libovolné množství přestupů. Na tomto místě zároveň nutno připomenout předpoklad o tom, že spoje nejsou nijak koordinované. Vyjděme z předchozího příkladu. Předpokládejme, že po cestě linkou 2 je třeba přestoupit ještě jednou na linku 3. Může cestující nějakým způsobem směřovat svůj plán přesně ke konkrétnímu spoji této 3. linky? Nemůže, je limitován oběma předchozími linkami, které mají větší interval. Jiná situace by nastala, kdyby třetí linka měla interval např. 50 minut, v tuto chvíli se dá předpokládat, že cestující nějakým způsobem svůj příjezd k tomuto spoji plánují. Tedy v případě, že předcházející linka má interval 30 minut se nestane, že budou na tuto navazující linku čekat déle než těchto 30 minut (pokud by totiž měli čekat déle, načasují své odjezdy tak, aby stihli až další spoj 2. linky a tím pádem zkrátili své čekání na 3. linku o 30 minut. V případě intervalů 10-30-50 minut tedy bude čekací doba na linky: 0-5-15 minut

Zamysleme se nad situací, kdy intervaly budou 30-10-50 minut. Čekací doby pak budou 0+5+15 minut. Jak dospět k těmto číslům? Čekací doba na první linku bude vždy 0 (či nějaká konstanta blízká 0). Čekací doba na linku 2 je dána polovinou jejího intervalu. Čekací doba na linku 3 by za normálních okolností byla 25 minut, ale cestující uzpůsobí odjezd linkou 1 tak, aby na linku 3 čekal max. 30 minut. Proto bude průměrná čekací doba na linku 3 jen 15 minut.

Shrneme-li všechny výše zmíněné poznatky, dospějeme k poznatku, že čekací doba je vždy minimem ze dvou čísel: poloviny intervalu dané linky, pro kterou chceme čekací dobu zjistit a polovinou doby, která je maximem všech dosavadních intervalů. Pro první linku je pak čekací doba rovna konstantě (pro zjednodušení uvažujme číslo 0). Uveďme ve stručnosti několik stručných příkladů:

Intervaly 50-10-50 minut: čekací doby 0-5-25 minut

Intervaly 5-40-20-20 minut: čekací doby 0-2,5-10-10 minut

Intervaly: 30-10-10-30-10 minut: čekací doby: 0-5-5-15-5 minut

Do výpočtu pravděpodobnosti využití jednotlivých variant spojení tedy vstupují tyto jízdní doby (čekací doby + doby jízdy konkrétní linkou) a interval, který je spočten jako nejvyšší interval z intervalů použitých linek. Do úvahy se v tomto případě bere i první linka, tedy v případě, že intervaly jsou 60 a 5 minut, jsou sice čekací doby pouhých 0+2,5 minuty, ale tomuto spojení je přiřazen interval 60 minut.

Na začátku kapitoly byl zmíněn předpoklad, že cestující je časově neflexibilní, tedy vždy chce být na místě určení co nejdříve od současného okamžiku. Pokud by tomu tak nebylo, bylo by možné zavést faktor časové přizpůsobivosti, který by se pohyboval mezi 2 extrémy. Prvním extrémem by byla neflexibilita, se kterou se pracuje ve všech předchozích výpočtech, druhým pak absolutní flexibilita. V tu chvíli by cestující bez ohledu na interval vždy preferoval spojení při kterém na cestě stráví nejkratší čas. Tato varianta v aplikaci nijak řešena není, ale určitě by se jednalo o zajímavý parametr, který by mohl výsledky ovlivnit.

3.3.3. Více linek jedoucích ve stejných úsecích

Dalším faktorem, který je důležité vzít v úvahu je jízda stejných linek mezi stejnými zastávkami. Při ní zůstává sice čas jízdy stejný, ale snižuje se čekací čas. Samozřejmě, že ve chvíli, kdy cestující pokračuje dále, než linky mají společné zastávky, tuto redukci čekacího času uplatnit nelze. Uveďme si na krátkém příkladu, proč se nedá při společné jízdě více linek, uvažovat o každé lince samostatně.

Příklad 6:

Interval linky 1: 8min

Interval linky 2: 8 min

Vzdálenost mezi body A a B pro linku 1: nepodstatná, např. 10 min.

Vzdálenost mezi body A a B pro linku 2: nepodstatná, např. 10 min.

Posloupnost cesty: linka 1 – přestup – linka 2

Mějme linky 1 a 2 jedoucí mezi zastávkami A a B 10 minut, obě s intervalem 8 minut. Pokud se nevezme do úvahy souběh linek, výsledkem bude, že cestu mezi zastávkami A a B lze vykonat dvěma způsoby a každý z nich má čas 14 minut (uvažujme situaci, kdy do času je započtena i čekání na linku, což za normální situaci neplatí). Každé z těchto variant využije cestující v 50% a průměrná doba tedy bude taktéž 14 minut. Realita je ovšem taková, že za předpokladu, že odjezdy linek budou rovnoměrně rozloženy lze říci, že cestu mezi zastávkami A a B lze vykonat jen jedním způsobem, a to čekáním na linku 1 nebo 2 a jízdou linkou 1 nebo 2, celkový čas v tuto chvíli bude 12 minut. Samozřejmě ve chvíli, kdy by linky odjížděly ve stejný čas tak se tato časová výhoda ztratí. Pro potřeby práce se bude předpokládat, že odjezdy linek jsou synchronizované. Pseudo-interval této společné linky se spočte následujícím vzorcem.

(7) $$I=\frac{1}{\frac{1}{I_1}+\frac{1}{I_1}+\ldots+\frac{1}{I_n}}$$, kde I_1 až I_n jsou původní intervaly linek, viz. Odvození v textu

Příklad 7:

Interval linky 1: 6 min

Interval linky 2: 8 min

Pro vysvětlení výpočtu opět zvolme modelovou situaci. Uvažujme dvě linky, první s intervalem 6 minut, druhá s intervalem 8 minut. Za 6.8=48 minut pojede celkem 48/6 + 48/8 spojů. Pseudo-interval bude tedy roven 48/(48/6+48/8).

$$I=\frac{48}{\frac{48}{6}+\frac{48}{8}}min=\frac{1}{\frac{1}{6}+\frac{1}{8}}min\simeq 3.4285min$$

Příklad 8:

Interval linky 1: 2 min

Interval linky 2: 4 min

Interval linky 3: 6 min

Uvažujme 3 linky o intervalech 2,4,6 minut. Za 2x4x6=48 minut pojede první linka 48/2x, druhá 48/4x, třetí 48/6x. Pseudo-interval pak bude roven 48/(48/2+48/4+48/6).

$$I=\frac{48}{\frac{48}{2}+\frac{48}{4}+\frac{48}{6}}min=\frac{1}{\frac{1}{2}+\frac{1}{4}+\frac{1}{6}}min\simeq 1.0909min$$

Číslo 48 (či jakékoliv jiné v jiných případech) se vždy vykrátí a pseudo-interval lze

zapsat ve tvaru (7).

Objasněme nyní použití odvozených vzorců v analýzách. Při analýze nejkratší cesty bude nejobecnější výsledek takový, že bude existovat několik variant cesty mezi 2 body a jejich pravděpodobnosti. Z takovýchto údajů poté bude možné zjistit pomocí výše uvedených výpočtů střední čas cesty mezi těmito 2 body. Zajímavější však spíše bude pouze dosažené varianty vizualizovat. U analýzy dostupnosti jednoho místa naopak tato možnost je nereálná, protože se v podstatě zabývá velkým počtem analýz nejkratší cesty a vizualizace jednotlivých těchto cest by byla nereálná. V tuto chvíli je pak nutné přistoupit ke spočtení průměrných časových nákladů pro každý bod a až tento výsledek nějakým způsobem vizualizovat.

3.4. Datový model

Pro potřeby aplikace byly navrženy dva modely, první je modelem, v němž jsou data uložena v databázi a tedy uchovávána permanentně. Druhý model je síťový model, který slouží pro provádění analýz nad sítí. Tento model je uchováván v paměti a vytvářen vždy při každé analýze. Platí, že druhý model vznikne vždy z prvního. Tvorba tohoto druhotného modelu je taktéž ovlivněna vstupními parametry každé z analýz a to v tom smyslu, že je potřeba vzít počáteční a cílové body, a též je přidat do tohoto druhotného modelu. Vzhledem k tomu, že druhý model má silnou vazbu na konkrétní implementaci analýz v programovacím jazyce, bude popsán až v kapitole, která se zabývá algoritmy.

3.4.1. Východiska návrhu

V následujících odstavcích tedy bude popsán datový model, se kterým pracuje aplikace a který je reprezentován tabulkami v databázi.

Model byl navrhnut tak, aby se přiblížil realitě zejména v následujících ohledech:

- zohlednění chůze jako důležitého faktoru pro analýzy. Při vynechání tohoto faktoru, bychom museli předpokládat, že cestující nasedá vždy na nejbližší zastávce, což mnohdy bývá předpoklad chybný

- zohlednění intervalů jednotlivých linek, což má vliv nejen na doby přestupů, ale také na pravděpodobnosti s jakou jsou tyto linky využívány

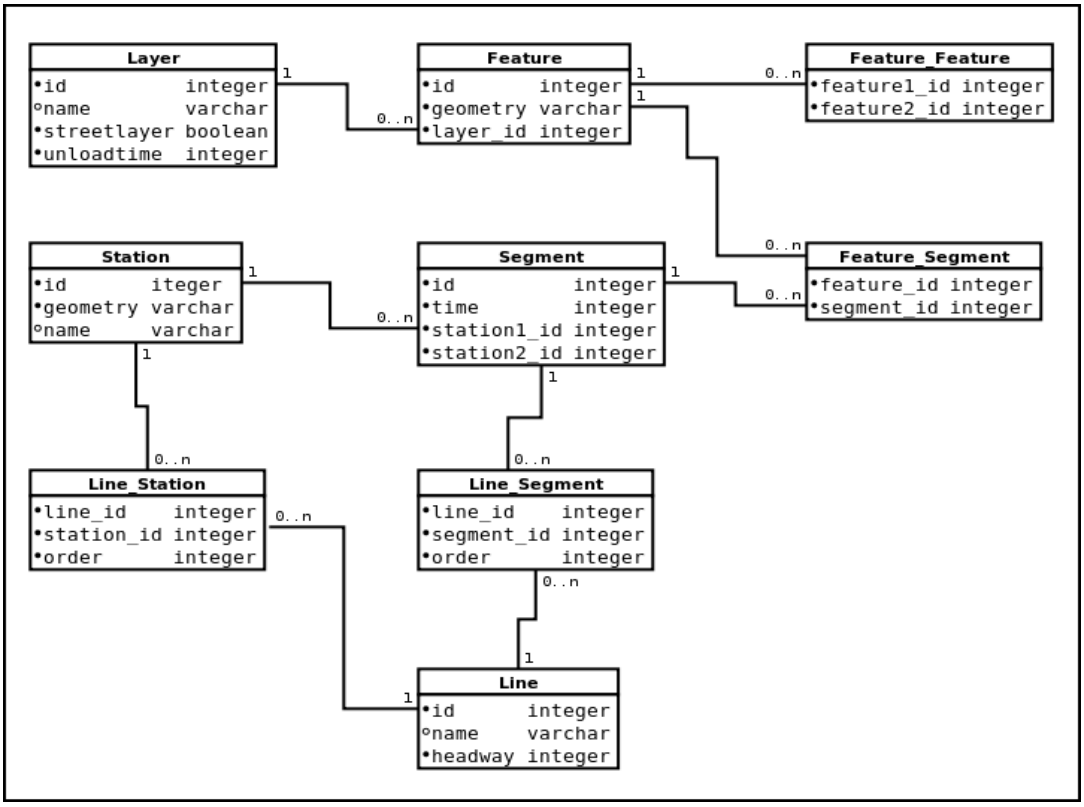
Naopak některé faktory byly pro zjednodušení a lepší možnosti modelování vynechány. Jedná se o tyto faktory:

- nedefinované jízdní řády, což sice může přinášet nepřesnosti, ale mnohem snáze se pak mění parametry modelu

- různé linky na stejné vrstvě (ulice, tramvaje, vlaky) zvládnou cestu mezi dvěma danými zastávkami za stejnou dobu, což však ve většině případů realitě odpovídá

- zjednodušený přístup k preferencím cestujících, kdy není zkoumáno kdo má v oblibě jaký typ dopravy, kdo preferuje chůzi atd.

Z hlediska správného návrhu databáze bylo důležité najít kompromis mezi omezením duplicity dat a výkonu + jednoduchosti správy. Autor se rozhodl, že duplicita bude omezena tím, že liniové prvky budou unikátní a ostatní entity, pokud to bude potřeba, budou na tyto prvky navázány. ERD diagram struktury modelu se nachází na obr. 3.5.



Obr. 3.5: Datový model aplikace

3.4.2. Popis tabulek

Základní jednotkou a nejjednodušší tabulkou v modelu je tabulka „layer“, neboli vrstva. Ta přibližně odpovídá i pojetí mapových vrstev v prostředí GIS. Jednotlivé geometrické entity jsou umisťovány právě do vrstev a pouze v rámci své vrstvy se na sebe vážou. Příkladem vrstvy je např. vrstva ulice, tramvajová síť, vlaková síť. Samozřejmě, že pro další analýzu je třeba tyto vrstvy nějakým způsobem propojit, to se však děje až při budování druhého „analytického“ modelu. Tabulka „layer“ má následující vlastnosti. Jednoznačný identifikátor ID, jméno, a pak také označení, zda-li se jedná o uliční vrstvu. Toto označení je důležité z důvodu, že základem pro další analýzy jsou právě body na uliční síti a také z důvodu, že jen po liniích této vrstvy se lze

pohybovat chůzí, zatímco po liniích jiných vrstev je pohyb umožněn pouze v rámci nadefinovaných linek. Dále se u vrstvy ukládá taktéž čas výstupu. Tím je myšlen čas, který trvá výstup z dopravního prostředku jedoucího po dané vrstvě. To se uplatní u dopravních prostředků, jakými je např. metro, kde nelze předpokládat, že po vystoupení z metra bude cestující plynule pokračovat po uliční síti.

Asi nejobjemnější součást databáze představuje tabulka „feature“. Každá entita této tabulky představuje nejmenší nepřerušovaný liniový prvek. Tento prvek může představovat část ulice, část tramvajové trati ad. Nutně nemusí platit a neplatí, že tato entita je shodná s mezistaničním úsekem. Každá entita této tabulky obsahuje identifikátor ID, a geometrii ve formátu WKT jako textový řetězec a následně odkaz na vrstvu, která obsahuje daný prvek.

Předpokladem pro správné fungování aplikace je požadavek na topologickou čistotu, zde uložených prvků. V praxi to znamená, že žádné dvě entity v této tabulce náležící stejné vrstvě se nesmí křížit, jen jejich krajní body se dotýkají. Autor řešil dilema, zda-li se tyto vztahy mají zaznamenávat již na úrovni tohoto primárního databázového modelu, nebo až v modelu sekundárním. Nakonec se rozhodl pro první řešení, tedy zaznamenání vztahů pomocí vazební tabulky. Výhodou tohoto řešení je nižší zátěž výpočetního jádra při samotných analýzách a samozřejmě nepatrně vyšší zátěž při editaci jednotlivých prvků. Toto řešení se jeví vhodnější z důvodu, že zatímco při jedné analýze by bylo potřeba hledat vztahy pro všechny entity, při editaci či postupném přidávání prvků je nutné toto provést pouze jednou, navíc o velkou část této práce se již na klientské straně stará knihovna OpenLayers. Tento přístup přináší problémy pouze tehdy, když je ze souboru nahráváno mnoho entit, a toto zachycení vztahů je nutné provést najednou. Co se týká tabulky „feature_feature“, její existence vychází z faktu, že tyto vztahy mezi jednotlivými liniemi jsou typu m:n, tudíž je použití vazební tabulky nezbytné.

Další tabulku představuje tabulka stanic, neboli „station“. Každá entita této tabulky představuje zastávku na některé z vrstev včetně uliční. Platí to, že stanice se nachází vždy na okraji linie v dané vrstvě. Toho je docíleno tím, že při přidání každé stanice je příslušná linie zrušena a vytvořeny dvě nové, navzájem na sebe navázané přesně v bodě zastávky. Zastávka má identifikátor ID, jméno, a odkaz na obě linie („feature“) se kterými sousedí. Dále obsahuje i bodovou geometrii ve formě WKT řetězce. Je jasné, že tato informace je v systému nadbytečná, protože daná geometrie se dá zjistit dotazem na průsečík obou linií. Ovšem cena za tuto duplicitu, není nijak vysoká. Geometrie bodů nepředstavují objemná data a jejich uložení v databázi výrazně urychluje další zpracování dat. V průběhu testování aplikace byly do této tabulky přidány ještě sloupce „feature_nearest_id“ a „distance“, které jsou využívány při analýzách k rychlejšímu vyhledání nejbližší linie na uliční vrstvě.

Dále je v databázi tabulka linek, neboli „line“. Tato tabulka obsahuje údaje o jednotlivých linkách v systému. V této tabulce není zahrnut žádný geometrický atribut, ty jsou přítomny pouze zprostředkovaně. Mezi vlastnosti této entity patří identifikátor ID, název a interval linky. Informace o vedení linky je pak uložena ve vazebních tabulkách „line_segment“ a „line_station“. K vysvětlení jejich funkcí je třeba nejprve objasnit pojem „segment“.

Segment je též tabulkou v databázi, která představuje úsek mezi 2 zastávkami na lince. Jeden segment může mít přiřazeno i více linek, to v případě že úsek mezi dvěma zastávkami je těmito linkami využíván. Entity v této tabulce obsahují identifikátor ID, název a čas segmentu. Toto je jízdní doba mezi 2 zastávkami. Je jasné, že toto zjednodušení nemusí vždy vystihovat realitu, protože je možné, že dvě různé linky jedou mezi dvěma zastávkami různou dobu – např. rychlík vs. lokální železniční spoj po společném úseku železnice. Nicméně tyto případy v praxi nebudou příliš časté díky faktu, že každý druh dopravy by měl mít svoji vrstvu a dá se předpokládat, že stejné druhy dopravy (např. tramvaje) v danou dobu zvládnou úsek za stejný čas. Samozřejmě, že jízdní doba se může v závislosti na denní době měnit. Toto je však vyřešeno tak, že model v jednom svém stavu může zachytit pouhý jeden časový okamžik (resp. časový úsek, kdy jsou stejné intervaly, jízdní doby ad.). Na segmenty jsou navázány přes tabulku „feature_segment“ liniové prvky. Platí, že segment se skládá z 1-n liniových prvků. Toto spojení je opět díky vazbě m:n řešeno vazební tabulkou. Na tomto místě je důležité připomenout, že toto napojení slouží výhradně k vizualizaci jednotlivých tras linek, či spojení z bodu do bodu. Síťové analýzy tuto informaci nepotřebují, dále bude vysvětlen princip, jakým se tyto informace získávají. Pokud se vrátíme ke zmiňovaným tabulkám „line_station“ a „line_segment“ - obě dvě tyto tabulky představují samotnou konfiguraci trasy linky z dvou různých pohledů. Samozřejmě, že pro definici trasy by stačila pouze jedna tabulka, ale tento přístup byl zvolen kvůli jeho přehlednosti a kvůli úspoře výpočetního výkonu. Obě dvě tyto tabulky představují vazební tabulky, opět kvůli vazbám m:n. Linka se může skládat z více zastávek, ale zároveň i zastávka může být součástí více linek a stejné je to s vazbami linka-segment. Každá z těchto tabulek ještě navíc obsahuje vlastnost „order“ neboli pořadí v trase linky. Tuto informaci by totiž nebylo jinak možné zjistit, či by se zjišťovala velmi obtížně. Platí tedy, že segment s pořadím např. 3, se skládá ze 2 stanic, které mají v tabulce „feature_station“ pořadí 3 a 4. Segment 0 se skládá ze stanic s pořadím 0 a 1 atd. Právě těchto informací ve spojení s informacemi o poloze stanice je využíváno při tvorbě druhotného modelu.

Z předchozích odstavců vyplývá, že se podařilo připravit takovou strukturu databáze, která omezuje duplicitu dat, tam, kde je to kritické (prostorová data) a naopak se této duplicitě nebrání tam, kde to zvýší výkon (informace v tabulkách, ve kterých je řádově méně záznamů, jako např. stanice či linky).

KAPITOLA 4: ALGORITMY

Na následujících stranách bude popsána implementace algoritmů klíčových pro běh aplikace. Autor se nakonec rozhodl popisovat algoritmy v textové strukturované podobě. Zároveň jsou součástí práce jako přílohy ukázky zdrojového kódu. I u těchto ukázek však z důvodu rozsáhlosti byla provedena redukce obsahu. V textovém popisu kódu jsou používány třídy objektů datového modelu v českých ekvivalentech, tam kde je to možné. „Feature“ jakožto nejmenší součást sítě je zde nazývána linií. Na tomto místě je třeba upozornit, že když v kódu bude zmíněna linie, jedná se o tento objekt. Pokud je potřeba pracovat s objektem geometrie „LineString“ je v textu zdůrazněno, že se jedná o objekt geometrie, aby nedocházelo k záměnám.

4.1. Používané algoritmy pro prohledávání sítě

V některých autorem navržených algoritmech se nacházejí i menší či větší části, které připomínají, či přímo představují, již existující algoritmy pro procházení grafu - sítě. Konkrétně se jedná o BFS, tedy prohledávání grafu do šířky a jeho specifičtější verzi – Dijkstrův algoritmus, která se používá pro nalezení nejkratší cesty v síti.

Algoritmus na bázi BFS autor využívá při konstrukci sítě z údajů z databáze. Postup vypadá následovně. BFS začíná s jedním uzlem, k němuž jsou nalezeni všichni sousedi. Ti jsou přidáni do fronty. Z této fronty jsou postupně uzly vytahovány a postup se opakuje (opět nalezení sousedů uzlu a přidání do fronty) (BAYER, 2008).

Dijkstrův algoritmus pak představuje modifikaci tohoto algoritmu. Algoritmus je inicializován s frontou bodů, které mají nastavenu hodnotu na ∞, s výjimkou prvního uzlu, který má hodnotu 0. V každém kroku cyklu je z této fronty vždy vybrán uzel s nejnižší hodnotou. Jsou prozkoumány všechny jeho hrany a k nim příslušící druhé uzly. U těchto uzlů se upraví hodnota ve chvíli, kdy hodnota aktivního uzlu + hodnota hrany (vzdálenost uzlů) je menší než stávající hodnota druhého uzlu hrany. Zároveň je aktivní uzel odebrán z fronty a označen jako uzavřený. Pokud by byl požadavek na zpětnou rekonstrukci cesty z jednoho bodu do druhého, je zároveň potřeba při úpravě hodnoty uzlu uložit i odkaz na předcházející uzel. Díky tomu lze z cílového uzlu cestu zpětně zrekonstruovat (BAYER, 2008).

4.2. Tvorba sítě – geometrie

První skupinu algoritmů představují algoritmy použité při tvorbě sítě z geometrického hlediska. Tedy při přidávání jednotlivých linií a stanic. Zde je klíčovým algoritmus umožňující rozdělení již použité linie na více linií a přidávání linií nových, přičemž je stále zachovávána referenční integrita v databázi. Do této skupiny patří taktéž algoritmus přidávání linií ze souboru. Tyto dva algoritmy budou zmíněny. Dále lze uvést např. algoritmus použitý při mazání jednotlivých linií, který rozebrán nebude.

4.2.1. Přidávání a rozdělování nových linií

Při rozdělování a přidávání nových linií a stanic se pro tento úkon používá jedna společná metoda a to z důvodu, že jsou tyto činnosti úzce provázány. Při přidání stanice dojde zároveň k rozdělení původních linií, stejně jako k rozdělení dojde při přidávání nové linie (samozřejmě v případě, že to geometrie nové linie vyžaduje). Základním vstupním parametrem této metody je objekt pracovně nazvaný „splitter“, který obsahuje všechny relevantní informace pro výše uvedené úkony. Tento objekt je slovníkem, obsahujícím více klíčů. Pod klíčem „station“ lze nalézt geometrii stanice k přidání. Pod klíčem „layerId“ se nachází kód vrstvy. Pod klíčem „replace“ lze nalézt složitější objekt, který bude v dalších odstavcích popsán a pod klíčem „add“ se nachází seznam linií k přidání. V případě, že již klient znal identifikátor linií, se kterými se mají provádět, je tento uveden v příslušném parametru (např. při nahrazování existujících linií). V případě že tento identifikátor známý není je poslán jakýsi pseudokód, vždy začínající písmenem „i“. Po proběhnutí metody se pak na klientskou stranu vrací slovník, který mapuje tyto pseudokódy na nově vzniklé linie (části kódu viz. Příloha 5 – segment 2.1).

Pod klíčem „replace“ se ve slovníku nachází další slovník s klíči „add“ a „remove“. Pod těmito klíči jsou pak identifikátory a geometrie linií k přidání spolu s identifikátory sousedů(„neighborIds“), a identifikátory linií ke smazání.

1) Procházení objektů k nahrazení(„replace“), ke každému vždy nalezení odpovídajícího liniového objektu.
2) V rámci objektů k nahrazení procházení objektů k přidání(„add“), na základě těchto objektů vytváření příslušných instancí třídy linií.
3) Mapování ID nově vytvořených linií na původní id objektu. Zároveň ukládání odkazu na objekt („add“) do instance této linie.
4) Nalezení a procházení linií, které sousedí s linií, která bude nahrazena(„remove“). V rámci tohoto cyklu procházení nově vytvořených linií. Pokud se tyto entity (nová a sousední) dotýkají vytvoření vazby „Feature_Feature“ (linie na linii)
5) Smazání nahrazované linie z databáze

6) V případě, že „splitter“ obsahuje stanici, pak její vytvoření v databázi. Přiřazení 2 linií z nově vytvořených tomuto objektu (nově vytvořené jsou v tomto případě vždy právě 2 linie, a to linie, které vzniknou rozdělením původní linie v bodě, kde se nachází stanice).

7) Procházení objektů k přidání („add“) a vytváření příslušných objektů linií a vzájemné mapování těchto 2 objektů (viz (3))

8) Mapování nových linií mezi sebou (na základě údajů o sousedních objektech z klienta(„neighborIds“)). Každá linie obsahuje odkaz na původní objekt, z něho jsou pak získány informace o sousedech.

9) Pro každou novou linii procházení jejich sousedů. Ty obsahují buď přímo id linie v případě, že se jedná o linii, která již existovala v době volání metody, či pseudokód začínající na „i“, v opačném případě. V tomto případě je nutné ještě k tomuto kódu nalézt odpovídající nově vytvořenou linii. Posléze vytvoření vazby „Feature_Feature“

4.2.2. Nahrávání linií ze souboru

Dále zde bude popsáno nahrání linií ze souboru. Toto je důležité zejména při nahrávání uliční sítě, kdy ruční editace v aplikaci by mohla trvat dlouho a nemusela by být tak přesná. Soubor musí být ve formátu GeoJSON. Jedná se o formát vycházející z JSON. Nástroje tohoto formátu jsou schopny nezávisle na programovacím jazyku z textového řetězce vytvářet objekty a naopak. Na vstupu této metody je název souboru, ze kterého mají být linie přečteny. V současné době k této metodě neexistuje uživatelské rozhraní, takže není využitelná přímo v aplikaci (části kódu viz. Příloha 5 – segment 2.2).

1) Načtení obsahu souboru do objektu pomocí JSONu

2) Procházení všech entit a přidávání jejich geometrií do pole

3) Topologické čištění: Je vytvořen objekt vícenásobné geometrie a do něj je uložen první člen z původního objektu geometrií. Následně jsou procházeny všechny původní geometrie a objekt vícenásobné geometrie je vždy roven výsledku volání metody „union“ na původním objektu vícenásobné geometrie, parametrem je vždy původní geometrie, která je aktuálně iterována.

4) Následně je objekt vícenásobné geometrie přeměněn na pole geometrií a z něj vytvořeny nové objekty linií.

5) Na závěr jsou stylem porovnání „každý s každým“ mezi těmito liniemi vytvořeny vazby „Feature_Feature“

4.3. Tvorba sítě – linky

Následující skupina algoritmů je zaměřena na tvorbu linek z již existujících linií a stanic. Do této skupiny spadá jak přidávání stanic do linky, tak jejich odebírání z linek. V této práci bude popsán jen první algoritmus, u druhého lze najít spoustu podobností s prvním.

4.3.1. Přidání stanice do linky

Při přidání stanice do linky přichází z klienta následující parametry. Jedná se o parametr „stations“ v němž se nacházejí id jedné nebo dvou stanic. Pokud se zde nachází jen jedna stanice, jedná se o id přidávané stanice, a ta je zařazena na počátek řetězce. Pokud jsou zde 2 stanice, první id představuje stanici předcházející a druhé id stanici přidávanou. Dalším parametrem je id linky do které se budou stanice přidávat a na závěr je zde parametr „catchAll“, který umožňuje přepínat nastavení vyhledávače tras do dvou módů. Pokud je tento parametr roven „True“, přidávají se do linky všechny stanice, které se najdou na cestě mezi uvedenými 2-3 (třetí stanicí může být stanice následující po uvedené předchozí stanici, případně dosud první stanice ve chvíli, kdy žádná předchozí stanice uvažovaná není). V případě, že je parametr „False“ přidává se jen požadovaná stanice (části kódu viz. Příloha 5 – segment 3.1).

1) Pokud je na vstupu jen jedna stanice, bude první v pořadí. Nalezení následující stanice (která byla dosud první), pokud existuje.
2) Pokud jsou na vstupu stanice dvě, jedná se o předcházející a aktuální stanici. Nalezení následující stanice, pokud existuje.
3) Nalezení segmentů mezi stanicemi, tato hledání proběhnou 2, pokud známe předchozí i následující stanici, jedno pokud jednu z nich neznáme a žádné, pokud se jedná o první a zatím jedinou stanici linky.
4) Procházení segmentů (pokud zachytáváme všechny stanice na cestě, může jich být i více než počet hledání) a přidávání objektů „Line_Station“ do databáze.
5) Ověřování, zda-li daný segment v databázi existuje, případně jeho vytvoření.
6) Vytvoření nového objektu „Line_Segment“ buď z existujícího segmentu, nebo z právě vytvořeného segmentu.
7) Pokud jsme znali všechny 3 stanice, pak smazání úseku mezi předchozí a následující stanicí.
8) Posun atributu „order“ u objektů „Line_Station“ i „Line_Segment“, které jsou v pořadí za přidávanými objekty.

4.3.2. Hledání segmentů – vytvoření sítě

V tomto odstavci bude popsán způsob hledání segmentů. K tomu je nejprve potřeba vytvořit síťový model sestávající z uzlů a hran. Nad ním je pak proveden Dijkstrův algoritmus pro nejkratší cestu mezi stanicemi. Vstupem do této metody jsou výchozí a cílová stanice (části kódu viz. Příloha 5 – segment 3.2).

1) Vytvoření uzlu s výchozí stanicí a 2 hran, které představují linie, se kterými tato stanice sousedí. Uložení těchto 2 hran do pole.

2) Procházení pole hran: nalezení uzlů pro danou hranu, pokud jsou již 2, algoritmus skáče na další hranu. Všechny objekty linií hran, které vycházejí z již existujícího uzlu jsou uloženy do pole již inicializovaných linií.

3) Nalezení stanic, které se nacházejí na dané hraně.

4) Vytvoření nového uzlu, který se stane druhým uzlem pro hranu.

5) Procházení sousedních linií dané hrany. Pokud se taková linie nachází v poli již inicializovaných linií je v iteraci přeskočeno na další linii.

6) Pokud nový uzel zatím nemá přiřazenou stanici, je proveden pokus o její nalezení iterací stanic z bodu (3).

7) Pokud sousední linie již má přiřazenou hranu, pak je načtena ze slovníku, v opačném případě je vytvořena.

8) Přiřazení této hrany k novému uzlu.

4.3.3. Hledání segmentů – procházení sítě

Nyní je tradičním způsobem proveden Dijkstrův algoritmus, a poté je dokončeno vyhledávání segmentů. K provádění této metody je potřeba mít vytvořenou síť z předchozího odstavce a taktéž výchozí a cílovou stanici (části kódu viz. Příloha 5 – segment 3.3).

1) Inicializace uzlu, který náleží výchozí stanici časem 0 a přidání do zásobníku otevřených uzlů

2) Procházení otevřených uzlů: Pokud k uzlu patří stanice, která se shoduje s cílovou, pak ukončení cyklu

3) V rámci uzlu procházení jeho hran: Na každé hraně nalezení druhého uzlu. Pokud je čas druhého uzlu větší než čas původního uzlu sečteného s časem hrany, pak dochází ke změně času druhého uzlu. Zároveň je přitom uložen do druhého uzlu i odkaz na aktuální uzel pro zpětné procházení cesty.

4) Poté jsou zpětně procházeny uzly, které tvoří výslednou cestu. Pokud je na uzlu

stanice, vytváříme nový segment. Pokud ne, pouze k aktuálnímu segmentu přidáváme odkaz na „Feature“ hrany.

5) Otočení pořadí segmentů

4.4. Vytváření sítě pro analýzy

V této kapitole se autor bude věnovat vytváření síťového modelu na základě dat z databáze, který posléze bude sloužit pro provádění obou analýz. Nejprve bude třeba popsat strukturu tohoto modelu a poté bude popsán postup vytvoření sítě.

4.4.1. Model pro provádění analýz

Model se skládá především z entit třídy „Edge“ (hrany) a „Node“ (uzel). Platí že každá hrana obsahuje právě dva uzly a každý uzel obsahuje 1-n hran. Dále hrany obsahují atribut čas, který pak slouží při samotném výpočtu. Jak však bude popsáno dále, ne vždy do výpočtu vstupuje celý, tak jak je zde definován. Takováto definice sítě by se nijak nelišila od té, která je používaná např. v Dijkstrově algoritmu. Nyní však budou popsány atributy, které představují již nadstavbu této sítě.

Nejdůležitějším atributem hrany je tzv. „EdgeType“, tedy její typ. Rozlišuje se 6 typů. Prvním z nich je „STREET“, tedy uliční úsek žádným způsobem nedělený, dalším typem „STREET_SPLIT“, jedná se taktéž o uliční úsek, nicméně oproti úseku, který se nachází v databázi byl rozdělen na víc částí. Toto mohlo nastat díky 2 okolnostem. Zaprvé se v blízkosti nachází zastávka, kterou je potřeba navázat na uliční síť a proto v nejbližším bodě musí být síť rozdělena, druhou možností je to, že na dané původní ulici se nachází začátek či konec cesty, cesta tedy také musí být rozdělena. Časová cena za pohyb po tomto typu hrany je dána jeho délkou vynásobenou přednastavenou rychlostí chůze. Dalšími 2 typy jsou „LOAD“ a „UNLOAD“. Tyto hrany představují spojnice mezi uliční sítí, po které se lze pohybovat pěšky, a sítí linek hromadné dopravy. Časová cena za pohyb po hraně „UNLOAD“ je představována nejčastěji 0, ale dalo by se předpokládat, že pro určité vrstvy bude smysluplná její nenulová hodnota. Jako příklad může posloužit např. vrstva představující hluboce zanořené metro, kdy i výstup zabere třeba 3 minuty času. Časová cena za nástup je rozebrána níže. Prozatím předpokládejme, že tato hrana si nese jako atribut interval linky, ke které náleží. Informace o umístění těchto hran do prostoru nejsou u těchto hran k dispozici. Na tomto místě je třeba upozornit, že i zastávky, které se nacházejí na uliční síti jsou spojeny těmito hranami. V praxi to vypadá tak, že na uliční síti se nachází konec uličního segmentu, který má stejnou polohu jako zastávka, na kterou se pak dále vážou linky, nicméně mezi těmito dvěma uzly i přesto existují hrany „LOAD“ a „UNLOAD“. Dalším typem hrany je typ „LINE“. Ten představuje spojení mezi dvěma zastávkami jedné linky. Takováto hrana se pak váže na další, sousední hrany téže linky, a také na hrany typu „LOAD“ a „UNLOAD“. Pokud náleží zastávce více linek, přestup se chápe

jako výstup hranou typu „UNLOAD“ a nástup hranou typu „LOAD“ do jiné. Časovým nákladem této hrany je jízdní doba mezi danými dvěma zastávkami, která se získává z příslušné entity třídy „Segment“. Posledním typem hran je hrana pojmenovaná „LINE_SHARED“. Představuje úsek mezi dvěma zastávkami, po kterém jede více linek. Tato hrana existuje z důvodu nižšího intervalu ve chvíli, kdy cestující využije společný úsek více linek. Toto je blíže popsáno v předchozích odstavcích. Časový náklad je stejný jako v případě typu „LINE“.

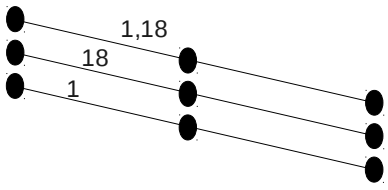
Hrana si taktéž drží odkazy, které umožňují provádění analýz i vizualizaci výsledků. Ke všem hranám kromě typů „LOAD“ a UNLOAD existuje odkaz na geometrii, hrany „STREET“ mají odkaz na původní objekt linie. Hrany typu „LINE“ si drží odkaz na objekt třídy linky a hrany třídy „LINE_SHARED“ na objekt, který představuje množinu objektů třídy linek.

Uzel si drží odkaz na geometrii, odkaz na stanici, pokud se jedná o uzly, mezi jejichž hranami se nacházejí též hrany představující linky či sdílené linky. Dále si uzel, pokud je to relevantní, drží informaci o intervalu linky, pokud je s nějakou spojen. Mezi další atributy uzlu pak patří pole „visitors“ (návštěvníků), které se při tvorbě sítě nepoužije, zato se používá při analýzách.

4.4.2. Vytváření sítě

V této kapitole bude popsán způsob vytvoření sítě na základě údajů v databázi a vstupních bodů analýzy. Postup je takový, že nejprve je vytvořena síť pouze z linek v databázi, následně jsou k uzlům těchto linek přidány hrany typu „LOAD“ a „UNLOAD“, poté jsou propojeny mezi sebou linie uliční sítě a na závěr jsou uzly hran typu „LOAD“ a „UNLOAD“ přidány do uliční sítě.

Tato pasáž obsahuje popis vytváření sítě z linek v databázi. Vstupní údaje žádné potřeba nejsou, vše probíhá čtením z databáze. Po provedení níže zmíněných kroků vypadá síť následujícím způsobem (viz. obr. 4.1). Čísla 1 – 18 – 1,18 představují čísla linek, přičemž 1,18 je chápána jako sdílená linka. (části kódu viz. Příloha 5 – segment 4.2.1).



Obr. 4.1: Síť po inicializaci úseků linek

1) Iteracemi všemi linkami v síti, pro každou linku vytvoření prvního uzlu a v rámci tohoto cyklu iterace jednotlivými segmenty linky.

2) Vytvoření druhého uzlu a následné vytvoření hrany spojující první a druhý uzel.

3) Nalezení stanice, která náleží k danému segmentu a vytvoření objektu, který mapuje stanice na uzly, které danou stanici reprezentují (dle počtu linek a jejich kombinací jich může být i více)

4) Nalezení všech linek, které se pohybují po daném segmentu a vytvoření 2-n členných kombinací, kde n je počet linek, které sdílejí tento segment. Do těchto kombinací nejsou zahrnuty již prošlé linky.

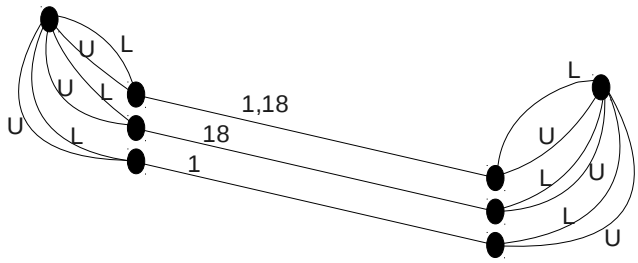
5) Iterace kombinacemi linek: Zpracovávají se pouze kombinace, které v sobě obsahují aktuální linku.

6) Pokus o nalezení již existující shodné kombinace, pokud existuje první uzel je použit z této kombinace, pokud neexistuje je nutné první uzel vytvořit. Posléze je vytvořen i druhý uzel pro daný segment a kombinaci a pro oba uzly je vytvořena hrana. Druhý uzel je též přiřazen dané stanici v objektu, který mapuje stanice na uzly. Ještě předtím je pro uzly spočten čas čekání, který se spočte dle vzorce (7) v kapitole 3.3.3.

7) Každá takto prošlá kombinace je uložena do pole. Objekt „LineComination“ obsahuje vždy množinu linek a také poslední uzel, kde byla tato kombinace použita.

8) Druhý uzel se stává prvním uzlem pro další segment, na který v cyklu posléze přecházíme.

V následující pasáži jsou k takto vzniklé síti přidány hrany typu „LOAD“ (L) a „UNLOAD“ (U). Na obr. 4.2 lze vidět výsledek po provedení. (části kódu viz. Příloha 5 – segment 4.2.2).



Obr. 4.2: Stav sítě po přidání nástupních a výstupních hran

1) Procházení všech stanic v objektu, který mapuje stanice na uzly.

2) Pokud se jedná o stanici na uliční síti, přidání do objektu, který mapuje linie na stanice a pokračování v cyklu s další stanicí.

3) Pokud se nejedná o stanici na uliční síti, nalezení nejbližší linii na uliční síti a vzdálenosti k ní, pokud tyto údaje jsou k dispozici pak jsou načteny z objektu stanice, pokud ne, je nutno je dopočítat.

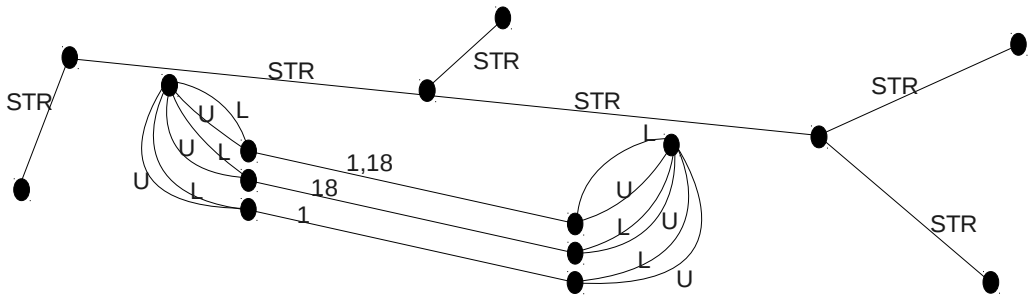
4) Vytvoření kružnice kolem stanice pomocí bufferu o poloměru, který je roven

nalezené minimální vzdálenosti. Na průsečíku této kružnice a nejbližší uliční linie leží bod, přes který se stanice napojí na uliční síť.

5) Vytvoření uzlu, reprezentujícího tento bod, a postupné vytvoření hran, které spojují vždy uzel příslušný zastávce a tento nově vytvořený. Hrany se vytvářejí 2 a to typu „LOAD“ a „UNLOAD“.

6) Přiřazení tohoto nově vytvořeného uzlu do objektu, který mapuje uliční linie na uzly.

Poté algoritmus pokračuje vytvořením uliční sítě (STR). Tento algoritmus má některé rysy podobné tomu, který již byl popsán v odstavci o přidávání stanic do linek. Výsledek po proběhnutí této části algoritmu lze vidět na obr. 4.3. (části kódu viz. Příloha 5 – segment 4.2.3).



Obr. 4.3: Stav sítě po vytvoření hran ulic

1) Nalezení jakékoliv uliční linie a jejích sousedů.

2) Vytvoření prvního uzlu na okraji této linie a zároveň vytvoření hrany s tímto jedním uzlem.

3) Ze sousedů vybrání těch, které náleží vytvořenému uzlu a z nich pak také vytvoření hrany. Přidání všech dosud vytvořených hran do pole, které se bude procházet.

4) Procházení aktivních hran, pokud hrana má již 2 uzly, pokračování v cyklu.

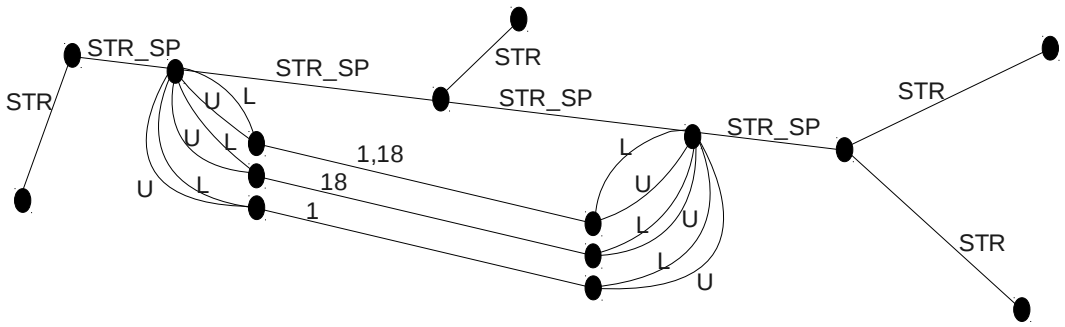
5) Vytvoření uzlu na konci právě procházené hrany a nalezení všech sousedních linií této hrany.

6) Procházení všech sousedních linií dané hrany a hledání těch, které náleží k právě vytvořenému uzlu. Pokud už pro sousední linii byla vytvořena hrana je načtena, pokud ne, je vytvořena a přidána do pole, kterým je iterováno.

7) Pokud se zároveň daná sousední linie nachází v objektu, který mapuje uliční linie na stanice, pak nalezení těchto stanic a jejich procházení.

8) Pokud se stanice nachází na daném uzlu, procházení všech uzlů z linkové sítě, které jí náleží a mezi uzlem na uliční síti a těmito uzly vytváření nových hran typu „LOAD“ a „UNLOAD“. Poté následuje odebrání stanice z objektu, který mapuje linie na stanice.

V tuto chvíli je tedy k dispozici kompletní síť linek, kompletní síť ulic, plus všechny hrany nástupu a výstupu, které ovšem na uliční síti obsahují uzel, který není na tuto sít zatím nijak navázán. Proto je nutné toto navázání pomocí následujících postupů provést. Po provedení bude síť vypadat dle obr. 4.4 (části kódu viz. Příloha 5 – segment 4.2.4).



Obr. 4.4: Stav sítě po spojení uličních hran a hran linek

1) Procházení všech linií na uliční síti, na které se bude vázat síť.

2) Vytvoření objektu, ve kterém se mapují geometrie na hrany. Do tohoto objektu je přidána geometrie linie na uliční sítě a hrana, která jí odpovídá

3) Procházení všech uzlů náležících dané uliční linii. Nalezení geometrie na které se nalézá uzel v objektu, mapujícím geometrie na hrany a rozdělení této geometrie v daném bodu.

4) Vytvoření nových hran, reprezentujících tuto rozdělenou geometrii, s tím, že jako jeden z uzlů je použit uzel původní hrany a jako druhý uzel je použit uzel právě aktivní procházený uzel.

5) Novým hranám jsou nastaveny atributy, typ hrany je nastaven na „STREET_SPLIT“ (STR_SP), z původní hrany jsou odebrány oba uzly a následně je celá tato hrana odebrána ze sítě

V tuto chvíli je inicializace sítě kompletní, zbývá jen do sítě dodat vstupní body pro každou jednotlivou analýzu.

4.5. Provádění analýz

4.5.1. Třída „Visitor“

Při analýze je využívána ještě další třída síťového modelu a to „Visitor“, neboli návštěvník. Objekty této třídy v sobě uchovávají údaje o cestě od počátku až do daného

bodu. Zároveň platí, že každý uzel má odkaz na unikátního návštěvníka. V praxi je tedy postup takový, že vždy je načten původní návštěvník pro původní uzel, vytvořen nový, do kterého je zkopírována většina atributů původního a tento nový návštěvník je pak dle charakteru hrany ještě modifikován do výsledné podoby. Jednou z modifikací je např. čas, kdy k původnímu údaji se přičítá čas hrany. Třída návštěvník má následující atributy: Interval – což je nejvyšší dosažený interval při cestě, z něho se posléze počítá pravděpodobnost volby cesty. Čas – což je čas od startu až do tohoto místa. Dále obsahuje odkazy na dosud použité linky, či společné linky, všechny použité hrany, uzly, odkaz na uzel, odkaz na předchozího návštěvníka a slovník, ve kterém je hrana mapována na čas.

Tento objekt obsahuje metodu, do které vstupuje hrana a uzel, do kterého má tento návštěvník vstoupit. Metoda probíhá tak, že je ověřováno, zda-li je návštěva dalšího uzlu oprávněná, pokud ano, tak dojde k vytvoření nového uzlu návštěvníka a poté k překopírování většiny atributů z návštěvníka původního do nového. Některé atributy jsou samozřejmě přepočítávány (části kódu viz. Příloha 5 – segment 5.1.1).

1) Nalezení předchozí hrany, po které se návštěvník pohyboval

2) Nedovolit návštěvu pokud: předchozí hrana byla typu „LINE“ nebo „LINE_SHARED“ a nyní hodnocená hrana je typu „LOAD“, stejně tak nedovolit návštěvu v případě, že předchozí hrana nebyla těchto typů a nyní hodnocená hrana je typu „UNLOAD“. Zamítnout návštěvu i v případě, že v uzlu už je tolik návštěvníků na kolik je nastaven maximální počet a taktéž v případě, že už nynější uzel byl tímto návštěvníkem někdy navštíven.

3) Vytvoření nového návštěvníka s tím, že většina vlastností je zkopírována z původního. V případě, že typ hrany je jakýkoliv jiný než „LOAD“, pak pouze přičítáme čas, v opačném případě je třeba složitějšího výpočtu.

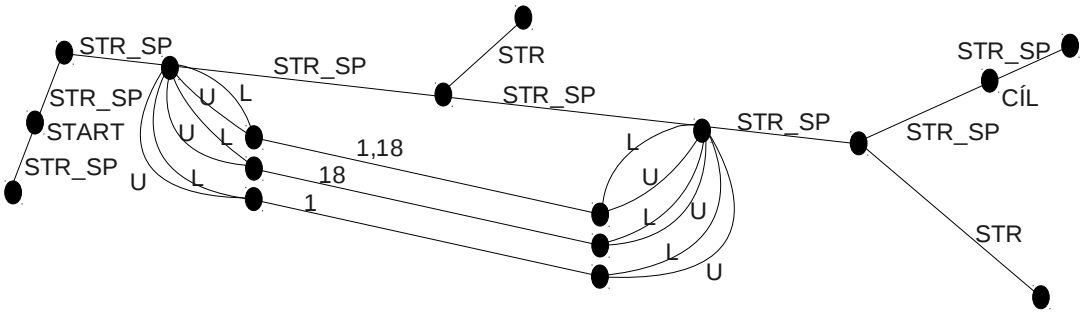
4) Vložení současné linky či kombinace linek do příslušných polí v objektu návštěvníka.

Poté je ještě prováděna druhotná kontrola vhodnosti, zde se ověřuje, zda předchozí návštěvníci nevylučují přijmutí aktuálního návštěvníka (části kódu viz. Příloha 5 – segment 5.1.2).

1) Nedovolit návštěvu v případě, že některý z ostatních návštěvníků uzlu má nižší čas a je splněna jedna z následujících podmínek: rozdíl časů je větší než zatím nejvyšší dosažený interval návštěvníka, všechny linky současného návštěvníka se nacházejí i v předchozím návštěvníkovi, předchozí návštěvník obsahuje jen chůzi, sdílené linky představují podmnožinu kombinací sdílených linek, či použitých linek současného návštěvníka.

4.5.2. Procházení sítě

Oba algoritmy (hledání nejkratší cesty i analýza dostupnosti) začínají podobně a to inicializací sítě (viz výše). Následuje přidání bodů analýzy, v případě analýzy dostupnosti se jedná pouze o cílový bod, v případě nejkratší cesty se jedná o dva body určující start a cíl. Postup je patrný z následujícího kódu. Jsou procházeny všechny hrany a pokud je zjištěno, že odpovídají jedné ze dvou linií, dochází k rozdělení dané hrany pomocí metody „splitGeom“ (části kódu viz. Příloha 5 – segment 5.2.1).



Obr. 4.5: Stav sítě po přidání výchozího a cílového bodu (START-CÍL)

1) Procházení všech hran, pokud hrana není uliční tak pokračování v cyklu

2) Pokud hrana obsahuje výchozí nebo cílovou linii, pak její rozdělení na 2 části v bodě, který odpovídá dané vstupní geometrii výchozího či cílového bodu

Obě analýzy též využívají pro svůj běh upravený Dijkstrův algoritmus. Vzhledem k tomu, že do uzlu může přijít i více návštěvníků je hlavní jednotkou právě návštěvník. V zásobníku se tedy nenacházejí uzly, ale návštěvníci, jinak je princip podobný. Vhodnost návštěvníka se neověřuje jen pomocí času, ale pomocí složitější metody (viz. předchozí odstavec).

1) Vytvoření prvního návštěvníka a přidání uzlu, který reprezentuje počáteční bod. Přidání návštěvníka do zásobníku návštěvníků

2) Procházení návštěvníku, vybrání ze zásobníku vždy takového návštěvníka, jehož čas je nejmenší. Procházení všech hran uzlu, který náleží návštěvníkovi

3) U každé hrany vytváření nového návštěvníka. Pokud není návštěva z nějakého důvodu vhodná (viz výše), pak se pouze pokračuje v cyklu. V opačném případě dochází k přidání návštěvníka do zásobníku a pokračování v cyklu

Dále je též využíván algoritmus hledající pro daný uzel a jeho návštěvníky pravděpodobnosti jejich využití. Proměnná „granularity“ určuje počet úseků, na který bude interval rozdělen (části kódu viz. Příloha 5 – segment 5.2.2).

1) Procházení všech návštěvníků cílového uzlu

2) Určení kroku, po kterém se postupně bude načítat čas, tento krok je spočten jako interval linky dělený konstantou. Prvotní čas je pak nastaven na polovinu kroku. Tento čas představuje čas do příjezdu spojení.

3) Dokud je čas menší než interval spojení, je pro něj spočtena pravděpodobnost a následně je k němu přičtena hodnota kroku

4) Na závěr je pravděpodobnost spočtena jako doteď nasčítaná pravděpodobnost vydělená počtem kroků

V předchozí metodě bylo zmíněno počítání pravděpodobnosti pro daný čas, daného návštěvníka, a všechny alternativní návštěvníky. Tato metoda je v následujícím odstavci v krocích popsána (části kódu viz. Příloha 5 – segment 5.2.3).

1) Jsou procházeni všichni ostatní návštěvníci daného uzlu. Pro každého z nich je vždy zjištěn časový rozdíl mezi tímto návštěvníkem a původním návštěvníkem.

2) Na základě vzorce (6) z kapitoly 3 je zjištěna pravděpodobnost využití v daný čas a ta je pak z metody navrácena.

4.5.3. Analýza dostupnosti

Nyní lze přejít k popisu analýzy dostupnosti. Vstupními parametry této analýzy jsou souřadnice jednoho bodu a identifikátor linie na kterém tento bod leží. Nejprve jsou provedeny všechny výše uvedené postupy pro inicializaci sítě a procházení sítě, posléze lze přikročit k samotné analýze (části kódu viz. Příloha 5 – segment 5.3).

1) Jsou procházeny všechny uzly sítě, které se nacházejí na uliční síti.

2) Pro každý je spočtena tzv. pravděpodobnostní mapa, kde se návštěvníka váže čas spojení a pravděpodobnost využití takového spojení.

3) Do výsledku pro každý uzel je vložena bodová geometrie tohoto uzlu a čas, který je spočten jako vážený průměr časů pravděpodobnostní mapy, kde váhami jsou pravděpodobnosti.

4) Výstupem této analýzy je pole bodů, z nichž každý má přidělen čas. Takovýto výsledek je pak vracen na klientskou stranu aplikace.

4.5.4. Analýza nejkratší cesty

Druhou analýzou je analýza nejkratší cesty. Vstupními parametry této analýzy jsou souřadnice dvou bodů a identifikátory 2 linií na kterých se tyto body nachází. Taktéž je nejprve provedena inicializace sítě a poté její projití. Ačkoliv lze uvažovat o zapojení nějaké výstupní podmínky pro běh cyklu procházejícího síť, autor tak nečiní, protože výkonová penalizace v tomto případě je nepatrná (části kódu viz. Příloha 5 – segment 5.4).

1) Nalezení vazby návštěvníka a pravděpodobností a časů tohoto návštěvníka pro cílový uzel.

2) Procházení všech návštěvníků uzlu a zápis času a pravděpodobnosti do výsledku.

3) Procházení všech hran návštěvníka, pokud se jedná o typ hrany „LINE“ nebo „LINE_SHARED“ je do pole geometrií přidána její geometrie, zároveň je přičten čas k původnímu času a určeno jméno segmentu, který však bude vytvořen až při výstupu.

4) Pokud se jedná o hranu typu „LOAD“ je vytvořen segment „CHŮZE“ sestávající z dosud načtených časů a zároveň je vytvořen segment „ČEKÁNÍ“. Pokud se jedná o hranu typu „UNLOAD“ je vytvořen segment pro linku, ze které byl výstup proveden na základě údajů načtených při procházení hran linky.

5) Pokud se jedná o uliční hranu, pak je pouze přidána geometrie do pole geometrií a přičten čas k původnímu času

6) Na závěr je pak ještě ukončen závěrečný segment typu „CHŮZE“ a pole geometrií se mění v jednu vícenásobnou geometrii.

7) Výsledkem této analýzy je pole variant. Každá varianta obsahuje geometrii a pole segmentů cesty, z nichž každý obsahuje název segmentu a čas příslušný danému segmentu.

4.6. Shrnutí

V této kapitole byly popsány algoritmy, které slouží k vytváření a editaci sítě a pak k provádění analýz nad touto sítí. Hojně se zde uplatňují algoritmy pro síťovou analýzu BFS a Dijkstrův algoritmus. Hodně využíváno je též porovnávání geometrií. V následujících kapitolách budou popsány konkrétní technologie pro uvedení těchto algoritmů v život.

KAPITOLA 5: TECHNOLOGIE

Následující kapitola se bude zabývat volbou technologií pro předvedení výše nastíněného modelu. Pokud se zobecní požadavky na aplikaci, mělo by se jednat o systém, který umožní uživateli editovat model a provádět nad ním analýzy v uživatelsky přívětivém prostředí. Základní komponentou tohoto prostředí musí být mapa, kde se zobrazují jednotlivé prvky modelu.

5.1. Desktop vs. server

Už při formulaci takto obecného zadání vyvstala základní otázka, a to, jestli se má jednat o aplikaci webovou, či instalovanou přímo na konkrétní pracovní stanice (desktopovou). Každá z těchto dvou možností má svá pro a proti.

Výhodou lokální aplikace může být nižší potřeba výpočetního výkonu (z hlediska generování grafických prvků) a tedy i rychlost, ačkoliv v současnosti se tento rozdíl pomalu začíná vytrácet. Také možnosti tvorby uživatelského rozhraní jsou zatím v případě desktopových aplikací rozvinutější, ačkoliv webové aplikace tento rozdíl postupně také smazávají a jejich nasazování je nyní velmi populární. Pokud budeme uvažovat řešení pomocí webové aplikace je jasné, že bude muset být nějakým způsobem řešeno propojení prohlížeč - server s databází. Toto spojení navíc nebude přímé, ale bude mezi ním ještě vložen prostředník – aplikační server. Přímé přistupování z prohlížeče do databáze zatím není možné (zejména z bezpečnostních důvodů). Těchto aplikačních serverů v současné době existuje nepřeberné množství v různých jazycích.

Dalším důležitým kritériem při rozhodování byl požadavek na zobrazování mapy v aplikaci. V případě webové aplikace už jsou nyní možnosti pro zobrazení různých vrstev ze serveru rozšířené, desktopová aplikace má naopak výhodu při zobrazování lokálních dat. Tuto výhodu, ale nebude možné využít, protože prostorová data budou uložena v databázi.

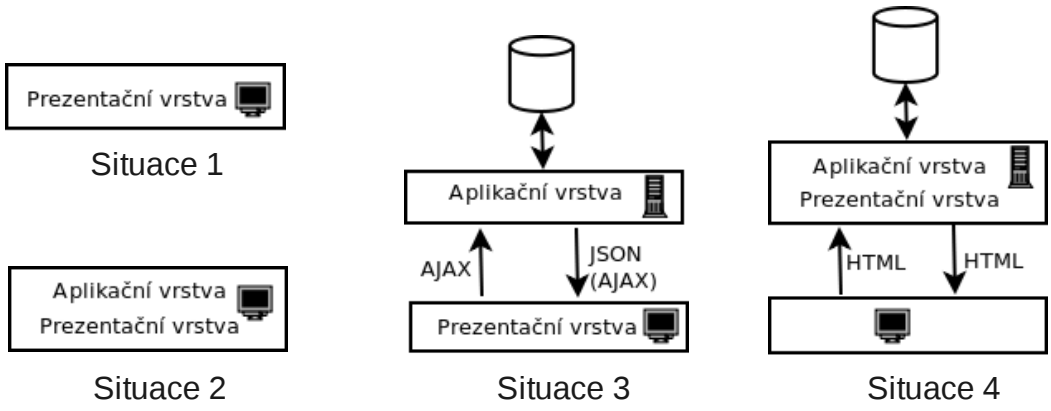
Nakonec bylo rozhodnuto, že aplikace bude naprogramována jako webová. Pro složitější výpočty a dotazování do databáze bude využívat aplikačního serveru, a spojení mezi tímto serverem a webovou aplikací bude zajištěno pomocí technologie AJAX.

5.2. Architektura aplikace

Obecně se většina složitějších webových aplikací skládá ze 3 částí (vrstev, v angličtině používaný termín 3-tier architecture): z datové (databáze), aplikační (zde je uložena většina logiky) a prezentační, která představuje to, co se ve finále zobrazí uživateli. Je jasné, že čím větší požadavek na interaktivitu aplikace, tím bude prezentační vrstva silnější a blíže k uživateli a naopak. Vzhledem k faktu, že základem aplikace bude práce s mapou, je nutno počítat s tím, že právě tato prezentační vrstva bude tvořit důležitou složku aplikace.

Dále bylo potřeba rozhodnout, jak moc bude v aplikaci přítomno skriptování na straně klienta. Už z výše uvedených faktů vyplývá, že aplikace se bez podpory kódu na klientské straně neobejde. Mluví-li se o kódu, či programovacím jazyku na straně klienta u webové aplikace, je zřejmé, že se jedná o Javascript. Alternativou by bylo využití nějakého řešení, kdy vývojář napíše jen HTML značky a v určitých místech odkazy na dynamický kontext. Takováto stránka je uložena na serveru, zde zpracována a vrácena s vyplněným dynamickým kontextem. Příkladem takové technologie může být např. JSP, neboli Java Servlet Pages, ovšem z výše uvedených důvodů vyplývá, že v tomto případě bude použito spíše volnějšího propojení prezentační vrstvy s aplikační pomocí technologie AJAX.

Na obr. 5.1 (níže) jsou zobrazeny různé varianty propojení serverové a klientské části. V prvním případě (situace 1) se jedná o webovou stránku s omezenou interaktivitou (ta je zde představována max. odkazy na místa na stránce, či na jiné stránky). V druhém případě (situace 2) se jedná o aplikaci či stránku, kde je veškerý kód vykonáván na straně klienta a nepotřebuje ke svému běhu databázi. Příkladem takové aplikace může být např. jednoduchá webová kalkulačka. Na třetím obrázku (situace 3) je vidět propojení serverové a klientské části pomocí technologie AJAX, kdy na klientské části se nachází prezentační vrstva včetně omezeného množství logiky, na serverové části pak většina aplikační logiky s případným řešením přístupu do databáze. Příkladem může být jakákoliv aplikace vyžadující přístup do databáze, nahrávání souborů či složitější výpočty, kdy je zároveň kladen důraz na interaktivitu uživatelského prostředí. Typickým případem mohou být aplikace pro práci s mapou a jejich nadstavby, či jakákoliv jiné aplikace vyžadující vyšší interaktivitu. Posledním příkladem (situace 4) je těsnější propojení aplikační a prezentační vrstvy, kdy se obě tyto vrstvy nacházejí na serveru. Toto řešení může nalézt uplatnění např. v různých systémech pro administraci, kdy se jednotlivé pohledy na data a případné tabulky apod. vytvářejí již na serveru a prohlížeč dostává již jen zpracované HTML s minimem Javascriptového kódu.



Obr. 5.1: Vybrané varianty architektury webových aplikací

5.3. Řešení klientské strany aplikace

5.3.1. Javascriptové frameworky

Vzhledem ke komplexním požadavkům na aplikaci je zřejmé, že pro její sestavení si autor nevystačí pouze s HTML, CSS a vlastním Javascriptovým kódem. Pozicování jednotlivých prvků, zdlouhavé psaní AJAX dotazů, řešení problémů s tabulkami, řešení problémů s různou interpretací stránek různými prohlížeči by se stalo téměř neřešitelným. Proto je nutné využít některý z Javascriptových frameworků, který výše zmíněné úkony usnadní. Tyto různé knihovny se liší především v různé míře podílu frameworku na komponentách grafického uživatelského rozhraní (GUI). Od frameworků, které programátorovi pouze usnadní manipulaci a procházení HTML prvků, či sestavování AJAX dotazů po pokročilé knihovny, které umožňují vývojáři sestavit komplexní aplikaci, která je téměř k nerozeznání od desktopové.

Jako příklad jednodušších frameworků lze uvést např. jQuery. jQuery je knihovna usnadňující vývojáři manipulaci s HTML prvky, událostmi a též s AJAX dotazy. Základem této knihovny je skupina metod, tzv. selektorů, které umožňují vybrat jednotlivé prvky v HTML např. na základě jejich identifikátoru, třídy, rodiče, souseda ad. V principu se syntaxe těchto selektorů podobá CSS selektorům, podle kterých se prvky HTML stylují. Tyto metody v jQuery však umožňují mnohem pokročilejší a snazší výběr prvků. S takto vybranými prvky lze následně pomocí další skupiny prvků manipulovat – upravovat jejich HTML obsah, vlastnosti, mazat je, přidávat k nim třídy, kopírovat je a mnoho dalších. Knihovna mj. Usnadňuje obsluhu událostí, kdy se tyto jednoduše vážou na pomocí selektorů vybrané prvky. Zjednodušeno je též volání metod serveru pomocí AJAXU. Alternativou k jQuery na této nejzákladnější úrovni jsou např. frameworky Prototype či MooTools.

Přistupme nyní k popisu pokročilejších knihoven, které kromě výše uvedených

knihoven poskytují i interaktivní grafické prvky, které obohatí vyvíjené stránky. Rozšířením pro již zmiňovanou knihovnu jQuery je nadstavba jQuery UI. Tato nadstavba umožňuje přiřadit prvku v HTML novou funkcionalitu. Jako příklad uveďme přiřazení funkcionality „automatické doplňování“ prvku textového vstupu. Dále lze vylepšit vzhled tlačítek, používat dialogy (pohyblivá okna) s proměnným obsahem, taby (podobný systém jako v moderních prohlížečích, kdy lze vidět v rámci prohlížeče vždy jednu stránku a na horní liště mezi nimi přepínat) a další. Pomocí těchto nástrojů lze již psát aplikace s pokročilejším uživatelských rozhraním. Stále tu však zůstává nutnost starat se o pozicování prvků pomocí CSS a tedy i o celkový layout aplikace. Opět uveďme alternativy k výše zmíněné knihovně. Jedná se např. o knihovnu script.aculo.us, která je podobnou nadstavbou pro Prototype jako jQuery UI pro jQuery. Dále lze zmínit např. framework Dojo Dijits, který je součástí obsáhlého frameworku Dojo.

Poslední skupinu tvoří frameworky, jejichž používání je podobné jako používání knihoven pro GUI pro desktopové aplikace (namátkou zmiňme např. Java Swing). Hlavním rozdílem oproti předchozím skupinám je fakt, že vývojář se již nestará o umisťování prvků přímo, ale přes jednotlivé třídy, které reprezentují objekty, jež se přidávají do HTML jako např. tlačítka, pole pro vkládání textů ad. Jejich pozice je pak dána umístěním do kontejneru. Celkový vzhled stránky je pak dán nastavením layoutu těchto kontejnerů. Je možné mít např. stránku s čistě sloupcovým layoutem, sestávající ze 3 panelů vedle sebe, které pak dále mohou obsahovat nějaký svůj vlastní layout. Protože se již nepracuje přímo s HTML elementy, ale s objekty tříd, je potřeba také přehodnotit práci s daty. Zatímco při klasickém přístupu např. HTML prvku „select“ lze nastavit možnosti výběru přidáváním HTML prvků „option“, v tomto pokročilém prostředí je nutné řešit předávání dat odlišným způsobem. Dochází k tomu, že data jsou oddělená od objektu, který je zobrazuje, tento objekt pak dostane jen odkaz na objekt obsahující data. Objektem obsahujícím data se myslí různé typy objektů, které nepředstavují jen pouhý výčet textových řetězců, ale obsahují třeba i více polí pro jeden záznam, informace o datových typech v jednotlivých polích atd. Je tedy možné mít takovýto datový objekt obsahující mnoho polí a v nějakém prvku stránky např. zobrazit pouze jedno pole. Nutno dodat, že používání těchto druhů frameworků se stává populárním až v dnešní době díky požadavkům na interaktivitu webů a lepšímu výkonu modernějších prohlížečů.

Jednou z těchto nejpokročilejších Javascriptových knihoven je knihovna ExtJS, která je založena na výše zmíněném principu. Na tomto místě nemá smysl popisovat všechny její funkce, jen namátkou, umožňuje bez jakýchkoli dalších pluginů vytvářet grafy, tabulky, formuláře, dialogová okna, stromy, spoustu jednodušších prvků jako jsou comboboxy, slidery, kalendáře a další. Uplatňuje se zde výše zmíněný princip oddělení dat od prvků, jež je zobrazují. Podobnými knihovnami jsou např. YUI, nebo qooxdoo, či již dříve zmíněné Dojo.

5.3.2. Výběr technologií

Když autor vybíral technologii pro klientskou část aplikace, vycházel především ze znalostí technologií které měl, a které uspokojivě dokázaly splnit to, co je od aplikace požadováno. Nejprve padla volba na kombinaci jQuery a jQuery UI v kombinaci s dalšími pluginy, zejména s pluginem pro tvorbu tabulek. S tím jak se aplikace rozrůstala se ovšem ukázalo, že pozicování jednotlivých prvků je čím dál obtížnější, až se stalo nezvladatelným. Autor proto dodatečně musel volit knihovnu, která by si poradila se složitějším layoutem, a zároveň dokázala v uživatelsky příjemném prostředí zobrazit zejména tabulky a mapu. Volba nakonec padla právě na ExtJS z důvodů autorovy zkušenosti s touto knihovnou a také díky rozšíření této knihovny jménem GeoExt. Tato nadstavba kombinuje knihovnu pro webové mapovací aplikace OpenLayers s knihovnou ExtJS. Výsledkem je pak řada grafických prvků, které nějakým způsobem souvisí s mapováním. Jedná se např. o legendu, prvky pro volbu měřítka, tabulky s obsahem geometrických prvků v mapě, tabulky s přehledem vrstev, které se dají zapínat atd. V samotné aplikaci se sice tato knihovna příliš nevyužívá, nicméně její existenci je třeba vzít v potaz pro případné rozšiřování aplikace.

V předchozím odstavci byla zmíněna knihovna OpenLayers. Nutno konstatovat, že co se týká takto komplexně pojatého klientského řešení, které je zdarma, OpenLayers v podstatě nemají ve své kategorii konkurenci. Jedná se o knihovnu, která umožňuje do určené oblasti HTML stránky umístit mapu, k té pak specifikovat její vrstvy a zároveň je možné i přidávat ovladače, které nad mapou vykonávají různé činnosti (posuny v mapě a zoom, měření vzdáleností, přidávání vektorů a mnoho dalších). Co se týká možností připojení vrstev, ty jsou nepřeberné, většina jich má samozřejmě původ na vzdáleném serveru (můžeme uvést OpenStreetMap, Google Maps, Yahoo Maps, dále jakákoliv WMS či WFS služba, nativní podpora pro MapServer), ale lze pracovat i s vektorovými vrstvami, jejichž objekty jsou přidávány v kódu. OpenLayers nativně podporují Mercatorovo zobrazení na kouli a WGS 84, podporu pro převod mezi dalšími zobrazeními lze získat přidáním knihovny Proj4JS. Nevýhodou této knihovny je fakt, že český uživatel si zde musí dodefinovat Křovákovo zobrazení, pokud chce provádět převody do a z tohoto zobrazení. Nutno dodat, že OpenLayers neumí transformovat rastr z jednoho zobrazení do druhého na straně klienta, proto potřeba převádět mezi zobrazeními vyvstává pouze u vektorových vrstev.

Při shrnutí volby technologií na klientské straně aplikace autor dochází k závěru, že vzhledem k rozsáhlosti aplikace a častému použití tabulek bude potřeba zvolit takový Javascriptový framework, který umožní i pohodlně kontrolovat layout stránek. Volba padla na ExtJS, který tyto požadavky splňuje a navíc k němu existuje knihovna GeoExt. Při volbě knihovny pro zobrazení map byly OpenLayers jasnou volbou, protože se autor domnívá, že co se týká bezplatných a otevřených řešení, tato knihovna v daném segmentu nemá konkurenci.

5.4. AJAX

Technologie AJAX je v aplikaci využívána pro přenos dat mezi serverovou a klientskou částí. AJAX je zkratkou pro Asynchronous Javascript and XML a v praxi funguje tak, že pomocí Javascriptu browser vysílá na server požadavek v určité podobě (nejčastěji textové) a vrací se mu opět nějaký blok dat, který je pomocí Javascriptu dále zpracováván. Od klasického načítání zdrojů v HTML se liší právě způsobem zpracování dat. Zatímco např. obrázky na webové stránce jsou po přijetí požadavku rovnou zobrazeny v prohlížeči, stránka je po odeslání formuláře znovu načtena apod., zde je potřeba nějakým způsobem v kódu odpověď zpracovat. Využívají se zejména 2 typy požadavků na server, a to GET a POST. U GET požadavku jsou data odesílána jako součást webové adresy, např. www.myserver.getData?id=10. To je sice z hlediska čitelnosti a širší použitelnosti např. i v adresních řádcích prohlížečů praktičtější, avšak přináší to s sebou omezení na délku takovéhoto požadavku a také na jeho typ – ten může být pouze textový. Na rozdíl od toho POST nese data přímo v těle požadavku. Lze také nastavit jakého typu tato data jsou. Tento typ se využívá ve chvíli, kdy chceme serveru zaslat obsáhlejší data, či např. soubor. U Ajaxu je ještě nutno zmínit JSON, což je standardizovaný způsob přepisu objektů v různých jazycích do textového řetězce a nazpět. Výhodou tohoto standardu je, že podporuje téměř všechny programovací jazyky a díky tomu umožňuje jejich vzájemnou komunikaci. Na následujících řádcích je popsáno převedení Python slovníku do JSON řetězce a jeho zpětné obnovení do objektu pomocí Javascriptu.

Python:

```
person = dict()
person['name']='John'
person['score']=[105,120,113]
text = json.encode(person)
```

V proměnné text se v tuto chvíli nachází tento řetězec:

```
{“name“:“John“,“score“:[105,120,113]}
```

Tento text je zaslán ze serveru na klienta, kde je načten opět do proměnné text, zpětné převedení:

```
var person = JSON.parse(text)
```

V tuto chvíli je v proměnné person uložen objekt s totožnou strukturou jako v Pythonu.

5.5. Řešení serverové strany aplikace

5.5.1. Obsluha požadavků na serveru

Nyní práce řeší volbu technologie pro provádění kódu na serverové straně aplikace. Požadavky byly následující: Technologie musí zvládat komunikaci s databází, a to ideálně na základě ORM principu. ORM je zkratkou pro objektově-relační mapování a

v praxi znamená, že s tabulkami v databázi pracujeme v programovacím jazyku jako s objekty. Dále musí řešení zvládat základní operace s geometrickými objekty. Technologií, které splňují dané zadání je více. Při hodnocení vhodnosti jednoltivých programovacích jazyků je nutno na prvním místě jmenovat Javu, která má jak pro ORM, tak pro práci s geometriemi dostatek knihoven. Nesmí být též opomenut Python, kde už jsou možnosti chudší, ale existují. Důležité je, že oba tyto jazyky lze nasadit na serveru. Každý z těchto jazyků má pak samozřejmě dále více možností, jakým způsobem bude na serveru fungovat. U Javy by to nejčastěji znamenalo zavedení dalšího, tzv. aplikačního serveru, jeho příkladem může být např. Tomcat. Na tento server se poté nahrávají zkompilované kódy a ty pak obsluhují přicházející požadavky. Samozřejmě, že možností řešení je více. Autor se však nakonec rozhodl pro implementaci v Pythonu z důvodu jednoduššího nasazování kódu na server (kód není nutné kompilovat, což sebou naopak přináší problém s nemožností jej účinně debugovat). U Pythonu lze v zásadě volit mezi komplexnějším řešením, kdy se větší část vzhledu stránek přesouvá na serverovou část (např. Django, Pylons), nebo řešení, které se nestará o nic jiného než jen obsluhu požadavků. Výše bylo uvedeno, že aplikace bude založena na silnějším klientovi, proto si v serverové části aplikace vystačí se základní obsluhou požadavků. Díky znalosti mod-pythonu se autor vydal touto cestou, ačkoliv je tato technologie již zastaralá. Dalšími technologiemi pro zpracování požadavků jsou CGI, FastCGI, WSGI. Rozdíly mezi těmito technologiemi v práci rozebrány nebudou. Mod-python funguje ve zkratce následovně. Technologie je nasazená přímo na http serveru Apache. V jeho konfiguraci se definuje, že o obsluhu určitých adres se bude starat právě Mod-python modul a zároveň se nastaví, jak bude která fáze požadavku obsloužena. Na to již existují předdefinované handlery (objekty zodpovědné za obsluhu požadavků). Je zde také možnost, že vývojář má nad požadavkem v každé fázi kontrolu. Autor využívá již předdefinovaných handlerů. Tyto handlery zaručují to, že parametry volání budou převedeny na volání dané metody, která se jmenuje stejně jako adresa daného požadavku. V Praxi tedy např. http://myserver.com/ajax.py/getName?id=100 zavolá metodu getName v souboru ajax.py a argumentem této metody bude parametr id nastavený na hodnotu 100. Pokud má vývojář zájem, může jako parametr metody obdržet též objekt představující samotný požadavek. To je využitelné ve chvíli kdy se zpátky neposílá jen řetězec, ale např. soubor. Samozřejmě, že výše uvedenou syntaxi lze použít jak pro požadavek GET tak i POST.

5.5.2. Použité knihovny jazyka Python

Jak již bylo řečeno, důležitá je pro aplikaci komunikace s databází. V nejjednodušší formě se dá řešit pomocí zasílání SQL dotazů a dostávání výsledků v podobě záznamů tabulky, ale autor chtěl uplatnit výše popsané ORM, aby mohl s tabulkami v databázi pracovat jako s objekty. Volba padla na knihovnu STORM, která se autorovi zalíbila pro

svojí relativně jednoduchou syntaxi a instalaci. Na druhou stranu nutno uvést, že dokumentace nepatří mezi nejsilnější stránky této knihovny a pokud se vývojář rozhodne vybočit ze zaběhlých postupů popsaných srozumitelně v návodech, může se dostat do problémů. Předpokladem práce s touto knihovnou je nadefinování tříd, které budou představovat tabulky v databázi, objekty těchto tříd pak budou představovat řádky dané tabulky. Každá třída obsahuje definované vlastnosti, které pak představují sloupce tabulky. Je také možné v dané třídě nadefinovat vztah k jinému objektu, např. třída Byt, by takto měla nadefinovaný odkaz na třídu Dům, ve kterém se byt nachází. Zároveň je pak vhodné mít v databázi nadefinovaná příslušná omezení pro cizí klíče, aby nemohlo dojít k chybě. Pokud je tedy z databáze získán objekt třídy Byt, lze pomocí odkazu na objekt Dům pracovat i s ním, aniž by bylo třeba se dále dotazovat. Právě toto je jednou z největších výhod ORM. Dotazy na objekty v databázi se zadávají přes k tomu určené metody, syntaxe je jednodušší než u SQL, ale ten, kdo je na SQL zvyklý, v ní může chvíli tápat. Samozřejmě je zde možnost zadávat dotazy i pomocí nativního SQL. Důležitým faktem je to, že tato knihovna není vázána na konkrétní databázovou platformu a lze tedy mezi nimi volně přecházet bez změny kódu v Pythonu. Databázi bude věnován jeden z následujících odstavců.

Nyní je na čase představit druhou významnou knihovnu a tou je Shapely. Jedná se o knihovnu, která umožňuje práci s geometrickými objekty. Proč používat takovéto knihovny ve chvíli kdy existuje pro Python knihovna OGR, která je určena pro práci s vektorovými daty? Je tomu především proto, že OGR spíše řeší převádění z různých formátu vektorů a z různých souřadných systémů do jiných, co se týká práce s geometrií jako takovou je silnější Shapely, který představuje port knihovny C++ GEOS do Pythonu, obě tyto knihovny vychází z knihovny JTS – Java Topology Suite, která slouží pro manipulaci s geometrií v jazyce Java, a je například součástí balíku GeoTools pro práci s geodaty. Shapely a výše zmíněné knihovny se nezabývají aspektem souřadných systémů, berou souřadnice tak jak jsou, bez převodů, a provádějí nad nimi operace jako sjednocení, průniky atd. Umožňuje též import a export geometrických objektů z textových souborů ať už ve formě WKT, či WKB. Každá geometrie je zde reprezentována objektem třídy. Třídy jsou např. následující Bod, Linie, Polygon, Uzavřený řetězec bodů, dále vícenásobné geometrie a kolekce geometrií. Základními vlastnostmi těchto objektů jsou pak souřadnice. Na tomto místě nutno konstatovat, že Shapely se zabývá pouze planární geometrií a proto neobsahuje metody pro práci s objekty ve třetím rozměru.

5.6. Databáze

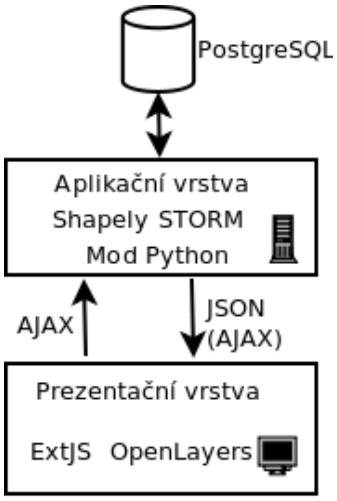
Další podmínkou pro běh aplikace je databáze. Při výběru konkrétní platformy, a to PostgreSQL, opět hrála roli autorova zkušenost, fakt že je tato platforma poskytována zdarma, i existence rozšíření PostGIS, které nakonec autor nevyužil. Toto rozšíření nebylo využito zejména z důvodu obtížnější integrace ORM s tímto rozšířením. Dá se

tak napsat, že autor se rozhodoval mezi využitím ORM nebo využitím PostGISu či jiné nadstavby pro práci s prostorovými daty. Samozřejmě i PostGIS data jsou uložená v textové formě (konkrétně WKB) a bylo by tedy možno k nim tak přistupovat a s pomocí Shapely pak z tohoto WKB tvořit objekty geometrií, nicméně to by posléze komplikovalo zasílání dat na klientskou část. Aby tak vývojář mohl těžit z výhod PostGISu bylo by nutné uvažovat úplně jinou architekturu aplikace, kde by výraznou roli hrály např. mapové servery. V takovéto architektuře by pak editace geometrických prvků mohly probíhat právě přes tento mapový server a služby WFS-T.

PostGIS	ORM bez použití PostGIS
Možnost prostorových dotazů	Přímější práce s geometriemi při přeposílání na klienta
Úspora diskového prostoru	Snadnější používání ORM
Možnost pracovat s desktopovými klienty	Jednodušší ladění programu

Tab. 1: Porovnání výhod PostGISu a ORM

Shrňme nyní závěry ke kterých bylo dosaženo v této části práce věnované technologiím. Aplikace se bude skládat ze 3 vrstev – databázové, aplikační a prezentační. Databázová vrstva bude představována platformou PostgreSQL, aplikační vrstva kódem v jazyce Python, který je schopen díky knihovně STORM pracovat s tabulkami a řádkami v nich jako s objekty, díky knihovně Shapely je schopen provádět operace s geometriemi a díky technologii Mod-python je schopen přijímat požadavky z klientské části aplikace a vracet na ně odpovídající odpovědi. Na klientské části pak najdeme technologie HTML, CSS, které ustupují do pozadí a nechávají vzhled a interaktivitu na Javascriptu. Ten především využívá knihovnu OpenLayers pro zobrazování map a knihovnu ExtJS pro celkový layout aplikace, zobrazování tabulek, grafických prvků a zasílání AJAX požadavků. Celkový návrh aplikace vystihuje obr. 5.2 (níže).



Obr. 5.2: Navrhovaná architektura aplikace a použité technologie

KAPITOLA 6: NÁVRH A FUNKČNOST APLIKACE

V této kapitole bude nejprve popsána funkčnost aplikace a posléze její návrh z programátorského hlediska, což obnáší v tomto případě zejména způsob členění kódu do souborů, protože algoritmy už byly probrány v minulé kapitole.

6.1. Funkčnost aplikace

Do této chvíle byla řeč obecně o aplikaci. V realitě ovšem pracujeme se dvěma aplikacemi. První umožňuje uživateli definovat síť a druhá provádět analýzy. Obě aplikace jsou spuštěny vstupem na webovou stránku, která se v případě obou aplikací nachází na odlišné adrese. Ihned po tomto vstupu je aplikace inicializována.

Obě aplikace mají společné mapové okno s podkladovou vrstvou a vrstvami představujícími ulice a jednotlivé druhy dopravních spojení. Jako podkladová vrstva slouží jedna z vrstev Google Maps, ale není problém tuto vrstvu změnit na jinou z Google Maps, či např. na OpenStreetLayers pomocí přepínače vrstev. V tomto přepínači lze též vypínat a zapínat i ostatní vrstvy.

6.1.1. Editační aplikace

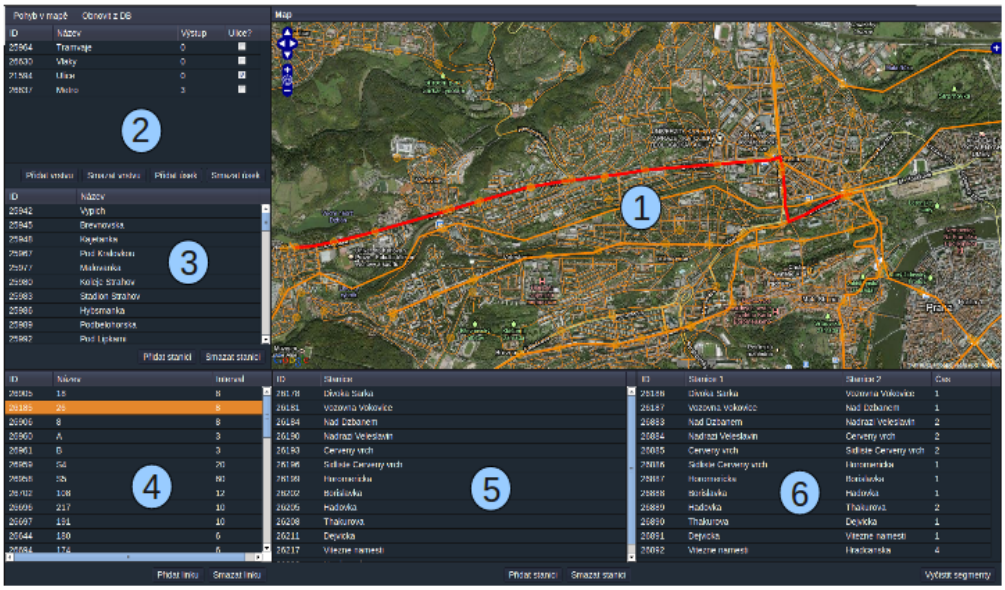
Nyní následuje popis aplikace pro tvorbu sítě. Funkce, které se v této aplikaci dají provádět by se daly rozdělit na modální – jedná se o funkce v mapě, z nichž může být aktivní v jednu chvíli pouze jedna a funkce okamžité, které se provedou okamžitě a poté lze pokračovat v práci s aplikací. Layout aplikace je rozdělen na hlavní mapové okno (na obr. 6.1 označeno číslem 1) a levý a spodní panel, které se dále dělí na další panely. V levém panelu se nachází panely pro manipulaci s vrstvami a základní mapovou manipulaci, panel zastávek a panel linek. Ve spodním panelu najdeme panel stanic pro linku a segmentů pro linku.

Modální funkce (popis následuje v textu):

- pohyb v mapě
- přidávání linií
- mazání linií

– přidávání stanic

– označování stanic, což se projevuje v jejich přidání k označené lince



Obr. 6.1: Vzhled editační aplikace

V panelu pro manipulaci s vrstvami se nachází tabulka, která zobrazuje vrstvy v databázi, jejich ID a označení (na obr 6.1 pod číslem 2), zda-li je daná vrstva vrstvou uliční. Tlačítka Přidat vrstvu a Smazat vrstvu jsou provedena a okamžitě a provádějí daný úkon. V případě přidání je přidána vrstva bez vyplněného jména, v případě mazání musí být samozřejmě nějaká vrstva vybrána. Dále je zde tlačítko Obnovit z DB, které načte stav databáze. Ve chvíli kdy ukládání změn neprobíhá transakčně, ale okamžitě, má toto tlačítko spíše formální význam ve chvíli, kdy se vyskytne nějaký neočekávaný stav aplikace, což by se samozřejmě mělo dít pouze při vývoji. Další tlačítka v panelu již jsou modální a aktivní může být pouze jedno – nejen v rámci panelu, ale v rámci celé aplikace. Toto tlačítko je ve chvíli kdy je aktivní podbarveno. Prvním z těchto tlačítek je Pohyb v mapě, který umožňuje pohyb po mapě pomocí myši. Tato funkce je aktivní při spuštění aplikace. Tlačítko Přidat úsek aktivuje přidávání linií do dané vrstvy. Pomocí klikání myši v mapě vznikají nové linie. Pokud sousedí již s existujícími liniemi, jsou na tyto linie navázány. Na tomto místě nutno upozornit, že vzhledem ke komplexitě uchovávání topologie linií může dojít k chybě při vytváření složitějších linií, které překrývají mnoho linií, již v mapě existujících. Uživatel by se měl proto vyvarovat přidávání linií, které kříží více jiných linií apod. Tlačítko Smazat úsek maže vždy linii na kterou je kliknuto. Řádky v tabulce je možné klikem na řádku vybrat, v některých případech je vyžadováno mít vybranou vrstvu. Při dvojkliku na pole ve sloupci Název ho lze editovat, taktéž při zaškrtnutí či odškrtnutí políčka „Uliční vrstva“ dojde k

automatickému odeslání změny do databáze.

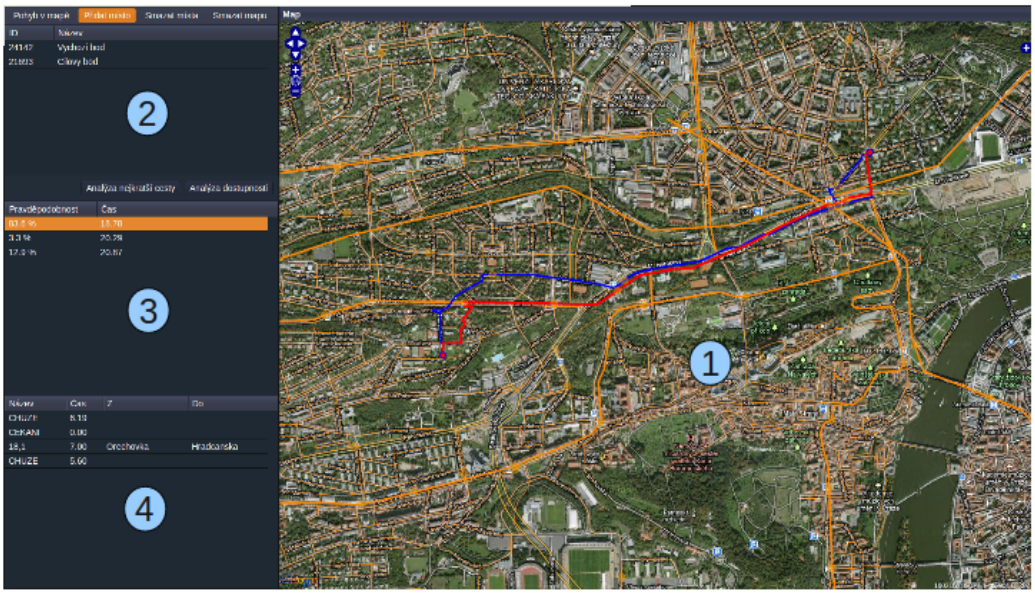
V panelu stanic (na obr 6.1 pod číslem 3) se nachází modální tlačítko Přidat stanici, které aktivuje přidávání stanic v mapě na kliknutí. Při aktivaci této funkce je zároveň zapnuto snapování na aktuálně vybranou vrstvu. Nová stanice musí ležet na této vybrané vrstvě. Po přidání stanice se stanice objeví i v tabulce, kde lze editovat její název. Při kliku na řádek v tabulce dojde k vysvícení zastávky v mapě. Panel stanic taktéž umožňuje mazat existující stanice. Vždy je proveden pokus o smazání aktuálně vybraného řádku v tabulce stanic. Smazání stanice je dovoleno jen v případech, kdy daná stanice není součástí žádné linky, potom mizí z mapy i z tabulky.

Dalším panelem je panel linek (na obr 6.1 pod číslem 4). Pomocí tlačítka Přidat linku je možné přidat řádek do tabulky linek. V tabulce linek lze editovat sloupce Název a Interval. Při kliku na řádek v této tabulce se vysvítí celá linka v mapě, a zároveň dojde k zobrazení všech stanic a segmentů linky. Pomocí tlačítka Smazat linku je možné odstranit linku, která se nachází na řádku v tabulce který je vybrán.

Dalším panelem je panel stanic linky (na obr 6.1 pod číslem 5), obsahující seřazené stanice pro danou linku. Pomocí modálního tlačítka Přidat stanici lze vybírat stanice v mapě. Ty se poté přidávají k dané lince. Na serveru při tom probíhá algoritmus, který hledá nejkratší cestu od předchozí stanice a zároveň cestu k následující stanici. Pokud se na cestě nacházejí další zastávky, přidá je také. Pokud není vybrána žádná stanice, přidává se vybraná stanice na první místo. Není možné vytvářet cyklické linky, ze serveru v tu chvíli přichází chybová odpověď. Pomocí tlačítka Smazat stanici lze vybranou stanici ze seznamu smazat. Při tom opět proběhne přepočet trasy.

V panelu pro segmenty linky (na obr 6.1 pod číslem 6) se nacházejí všechny úseky pro danou linku. Úsek je v tabulce identifikován názvy dvou stanic a taktéž časem, což je jediná interaktivní položka v této tabulce. Čas každého tohoto úseku lze modifikovat. Ve chvíli kdy se tak stane, změní se tento čas i pro ostatní linky obsahující daný úsek. Dále je zde tlačítko Vyčistit segmenty, které odstraní z databáze segmenty, které již nenáleží k žádné lince.

6.1.2. Aplikace pro provádění analýz



Obr. 6.2: Vzhled aplikace pro provádění analýz

Aplikace pro provádění analýz disponuje stejným mapovým oknem (na obrázku 6.2. pod číslem 1) jako aplikace editační. Ostatní panely se však již liší. Prvním z panelů je panel pro přidávání bodů (na obr 6.2 číslo 2) pro analýzu. V tomto panelu se také nacházejí jediná tlačítka, kterými se aplikace ovládá. Jedná se o modální tlačítka pro pohyb v mapě a přidání míst a tlačítka okamžité akce. Tlačítka okamžité akce jsou následující: Smazat místa, které smaže aktuální místa pro analýzu, Smazat mapu, které smaže výstupy analýz. Dále se jedná o tlačítka spouštění analýz a to: Analýza nejkratší cesty a Analýza dostupnosti. Při přidávání míst mohou být v mapě maximálně 2 místa. Pokud již 2 místa v mapě jsou, jedno z nich se maže a nové se přidává. Pro analýzu nejkratší cesty jsou potřeba 2 místa a pro analýzu dostupnosti 1 místo.

Po dokončení analýzy nejkratší cesty se v dalším panelu (na obr 6.2 číslo 3) zobrazí varianty cesty s jejich časem a pravděpodobnostmi využití těchto variant. Při kliknutí na konkrétní variantu dochází k jejímu podbarvení v tabulce i v mapě a k zobrazení údajů varianty v posledním panelu (na obr 6.2 číslo 4). Zde se pro každou variantu postupně zobrazují všechny úseky varianty. V názvu je vždy buď název trasy (např. „Linka 8“ nebo „ČEKÁNÍ“ pro čekání na spoj, „CHŮZE“ pro chůzi, případně „VÝSTUP“ pro výstup z prostředku linky, která se nachází na vrstvě, kde výstupní čas je nenulový. U daného názvu úseku je vždy čas, který tento úsek zabere a u použití linek se zde nachází i výchozí a cílová stanice pro daný úsek. Při vybrání záznamů z této tabulky již aplikace nijak nereaguje.

Co se týká svižnosti aplikace, většinu činností uživatel nezaregistruje, aplikace je plynulá. Problémy aplikaci působí pouze propočítávání tras na uličních sítích a analýzy, kde největší čas ukrajuje budování sítě z databáze. Samotné procházení této sítě již je rychlé.

6.2. Návrh programu

Tato kapitola se bude zabývat samotným návrhem aplikace, tedy nikoliv konkrétními algoritmy, ale spíše architekturou návrhu, členění metod do jednotlivých souborů či tříd, pravidly pro používání globálních proměnných atd.

Jak již bylo popsáno, program se dělí na část naprogramovanou pro serverovou část v jazyce Python a část klientskou interpretovanou webovými prohlížeči v jazyce Javascript a s použitím souborů HTML a CSS. Na obou těchto stranách je kód dělen pro větší přehlednost do více souborů. Zde je nutno podotknout, že v jazyce Python jsou objekty používány pouze pro manipulaci s databází a pak při síťových analýzách. Metody volané na serveru žádnou třídou obaleny nejsou, jednoduše proto, že to není potřeba a povaha těchto volání to nijak nevyžaduje. Jazyk Javascript obecně není uzpůsoben pro práci s předdefinovanými třídami objektů. Pravdou však je, že mnoho knihoven, včetně použité ExtJS tento aspekt jazyka řeší zaváděním vlastních mechanismů, které posléze umožňují pracovat s třídami a objekty tak, jak je vývojář zvyklý z jazyků, kde jsou definice tříd důležité. Záměrně se autor vyhýbá výrazu objektově-orientovaný jazyk z důvodu, že i Javascript je objektově orientovaný, ovšem se slabou (téměř žádnou) typovou kontrolou.

6.2.1. Zdrojový kód na klientské straně aplikace

Kód vykonávaný na straně serveru se nachází v adresáři py_script, který se nachází v kořenovém adresáři webové aplikace. Tento postup není obvyklý ani ideální z důvodu, že na tento adresář je vidět „zvenku“, což by se u kódu na straně serveru nemělo dít. Samozřejmě je možné tomu zamezit nastavením vhodných přístupových práv.

Na straně serveru je kód rozdělen do 3 souborů:

- features.py
- lines.py
- analysis.py

Obecně je ve všech těchto souborech přítomný stejný základ, který obsahuje definici datového modelu (jeho třídy). Toto by mělo být řešeno samostatnou třídou a jejím importem, nicméně autor práce při tomto importu řešil problémy s tím, že mimo importovanou třídu se daná třída neinterpretovala z neznámých důvodů správně. Proto bylo zvoleno toto řešení. V případě modulů „lines“ a „analysis“ jsou součástí definice ještě třídy pro práci se síťovým modelem. Tedy „Edge“ a „Node“, v případě analýzy

ještě rozšířené o třídu „Visitor“. Nutno dodat, že síťové modely se pro potřeby analýzy i editace liší, takže jsou tyto třídy rozdílné. Dále každá třída obsahuje deklaraci globálních proměnných stojících mimo třídy, což jsou nejčastěji slovníky, které se využívají ve chvíli, kdy je metoda dělena na více podmetod a neustálé posílání daného parametru (slovníku) by znepřehledňovalo kód. V každém ze souborů je navíc obsažena definice databázového připojení pro knihovnu STORM. Dále již následují metody pro obsluhu požadavků. Jednodušší požadavky jsou obslouženy přímo v dané metodě, složitější se pak skládají z více podřízených metod, či jsou obaleny blokem „try..except“ pro zachycení výjimek. První ze souborů, tedy „features.py“ řeší přidávání vrstev, linií a stanic do databáze a jejich případné mazání. Nejobsáhlejší metodami jsou zde metody, které zajišťují dělení a mazání linií. Je tomu tak z důvodu nutnosti přepracovat vazby mezi jednotlivými liniemi. Také se zde nachází metoda pro nahrávání linií ze souboru do databáze. Tato metoda nakonec není používaná z klientské strany a autor ji používal jen jako utilitu.

Dalším souborem je soubor lines.py, který se stará o vše spojené s linkami dopravy. Zde jsou nejobjemnější metody pro přidávání a odebírání stanic z linek. Tyto metody pak v sobě zahrnují volání dalších pomocných metod, které slouží zejména k vytvoření sítě, jež je pak procházena a tím vlastně vzniká trasa mezi 2 zadanými zastávkami.

Posledním souborem serverové části je soubor analysis.py a jak již název napovídá, je zodpovědný za provádění analýz. Nejvíce prostoru v něm ovšem zabírají metody na vytvoření sítě nad kterou tyto analýzy probíhají. To se děje postupně v krocích, přičemž každý krok je v jedné metodě. Samotné procházení je pak již jen modifikací Dijkstrova algoritmu, takže jeho implementace patří v rámci souboru k těm kratším.

Pro kód zpracovávaný na straně klienta je struktura standardní z většiny webových aplikací, tedy adresář s názvem projektu se nachází na UNIX serveru v adresáři /var/www a v tomto adresáři aplikace se pak nachází soubor index.html a poté adresáře js pro javascriptové knihovny a skripty a css pro soubory s kaskádovými styly.

6.2.2. Zdrojový kód na klientské straně aplikace

Soubory .js zdrojového kódu:

- gui.js
- gui_analysis.js
- controls.js
- map.js
- layer.js
- station.js

- analysis.js
- line.js

Na klientské straně jsou soubory s Javascriptem rozděleny víceméně podobným způsobem jako na straně serveru s několika rozdíly. Logika zobrazování a tomuto zobrazování podléhající struktura datových skladů (objekty třídy „Store“) jsou definovány v souboru gui.js pro editační aplikaci, resp. gui_analysis.js pro analytickou část. Je zde tedy definován celkový vzhled aplikace, podoba tabulek v jednotlivých panelech, propojení tlačítek v panelech s funkcemi, které budou volány po stisknutí tlačítka a další. Ostatní soubory se již používají pro obě aplikace společné, ačkoliv platí, že soubor analysis.js využije pouze analytická aplikace a např. soubor line.js pak výhradně editační část aplikace.

V souboru controls.js se nachází funkce, které mají na starost ovládání mapy. Jedná se o různé módy práce s mapu v závislosti na tom, které modální tlačítko je aktivní.

V souboru map.js jsou definovány základní vrstvy mapy a také metoda obsluhující obdržení údajů o mapě ze serveru. Tato metoda pak postupně volá metody v ostatních souborech a ty pak plní mapu i tabulky.

V souboru layer.js je řešeno nakládání s obdrženými vrstvami a liniemi v nich. Každé vrstvě jsou pak přiřazeny různé ovladače chování, např. přidávání, mazání, dělení, které jsou vždy aktivní při současném vybrání vrstvy a aktivaci příslušného modálního tlačítka. Nejobsáhlejší pasáže souboru pak tvoří obsluha události „rozdělení linie“, kde je potřeba připravit příslušný požadavek na server a posléze ho zpracovat. Problém nastává ve chvíli, kdy se na server posílají údaje o ještě neexistující linii a ze serveru se vrací již údaje o linii, která byla vytvořena. Je nutno nějakým způsobem zajistit mapování původní linie na tuto novou obdrženou ze serveru. I díky tomuto faktu je kód těchto metod a i celého souboru layer.js objemný.

Další soubor se jmenuje station.js a kromě přidání stanic při inicializaci aplikace zajišťuje i jejich přidávání v rámci editačního procesu. Toto přidávání posléze spouští metody v souboru layer.js a dochází k dělení linií. K dělení linií se využívá pomocná vrstva, do které je při vytvoření zastávky přidána linie, která s minimální délkou obsahující stanici. Tato linie pak slouží pro rozdělení již existujících linií vrstev.

Soubor line.js obsahuje metody pro práci s linkami. Vesměs se jedná o volání serverových metod a reakce na jejich úspěšné provedení. Tyto metody neobsahují nijak složitou logiku a starají se především o plnění tabulky linek, stanic linky a segmentů linky.

Konečně soubor analysis.js slouží pro provádění analýz. Je zde zahrnuto především volání příslušných analytických metod na serveru a reakce na obdržení výsledku. Tato reakce zahrnuje u analýzy nejkratší cesty naplnění tabulek a zobrazení linií v mapě a v

případě analýzy dostupnosti vytvoření bodové vrstvy obarvenou dle dojezdových časů a zobrazení legendy.

Aplikace, zejména na klientské části, užívá ve velké míře globálních proměnných a další její rozšiřování by bylo obtížné. Pokud by se jednalo o vývoj pro produkční nasazení, bylo by asi vhodnější zvolit nějakou pokročilejší architekturu, ale to v době návrhu aplikace přesahovalo jak zaměření práce, tak schopnosti autora.

KAPITOLA 7: PŘÍPADOVÁ STUDIE

Za účelem ověření funkčnosti aplikace i konceptu je součástí práce i případová studia. V rámci této studie bude aplikace provozována nad reálnými daty MHD pro oblast Prahy 6.

7.1. Popis území

Oblast Prahy 6 byla vybrána z důvodu, že svou velikostí odpovídá většímu krajskému městu ČR. Je jasné, že se zde budou uplatňovat jiné vzory konstrukce sítě linek než u krajských měst, protože se jedná o jednu z mnoha částí Prahy a to část, kterou nelze označit centrem. Logickým důsledkem tohoto faktu bude, že síť bude houstnout směrem k centru města, ale nikoliv směrem k centru oblasti. Dalším poměrně specifickým rysem území je to, že zde převládá vedení linek v západo-východním směru. To je dáno především geomorfologickými poměry na tomto území, které sestává z hřebenů a údolí právě v tomto směru.

Nejprve je nutno území přesněji vymezit, protože pro potřeby testování není vhodné se omezit striktně na administrativní hranice Prahy 6. Z těchto hranic bylo vyjmuto území pražského letiště, území katastrů Zličín a Sobín, plus některá území při severním okraji hranice Prahy, kde se nenachází zástavba vůbec, či jen v zanedbatelném množství. Naopak bylo území rozšířeno o části Prahy 5 zahrnující Motol a Smíchov, za účelem zajištění konektivity Řep s centrem. Dále bylo území rozšířeno o části Prahy 1 v katastru Malé Strany a Hradčan (v podstatě celé území Prahy 1 na levém břehu Vltavy), protože tvoří s Prahou 6 v těchto místech poměrně kompaktní celek. Ze stejného důvodu byly přidány i části Prahy 7 v oblasti Bubenče. Na takto definovaném území byly prováděny analýzy nad kompletní (definice viz. dále) uliční síti. Pro možnost sledovat dostupnost míst na Praze 6 i vzhledem k centru města byl model rozšířen ještě o trasy metra A z Malostranské na Můstek a trasy B z Anděla na Náměstí Republiky. Ze stejného důvodu bylo provedeno i rozšíření železniční sítě až na Masarykovo nádraží. V okolí těchto tras již se nenachází plnohodnotná uliční síť, ale jen její malá část, která umožňuje provádět analýzy. V praxi to vypadá tak, že tato síť prakticky kopíruje vedení výše zmíněných linek metra a vlaků.

Síť sestává z uliční sítě, po níž se pohybují spoje autobusových linek, ze sítě tramvajových, vlakových drah a drah metra. Zatímco uliční vrstva vznikla nahráním

souboru s úseky sítě na server, zbylé 3 vrstvy vznikly ruční editací přímo v prostředí aplikace. Jako zdrojová data v případě uliční sítě posloužila data OpenStreetMap, což je otevřený projekt pro shromažďování geodat. Výhodou tohoto projektu je snadná dostupnost dat a jejich množství, nevýhodou pak mnohdy jejich kvalita. To je dáno tím, že přispěvatelem se může stát v podstatě každý. I přes tuto nevýhodu se však autor rozhodl, že pro vytvoření sítě, která zatím nebude pracovat v produkčním prostředí je toto řešení dostatečné. Ostatní sítě pak byly vektorizovány přímo v aplikaci a to na základě map společností ROPID, DP Praha a na základě serveru idos.cz. Intervaly a časy segmentů byly počítány pro pracovní středu cca. 7:00 ráno, tedy ve chvíli, kdy se dá předpokládat, že frekvence spojů bude nejvyšší. Do úvahy byly brány pouze městské linky, příměstské linky do úvah zahrnuty nebyly, ačkoliv samozřejmě v některých úsecích mohou být doplňkem k nabídce linek městských. Zároveň se předpokládalo, že všechny linky linky staví ve všech projížděných zastávkách, což je také předpoklad, který ne vždy odpovídá realitě.

7.2. Příprava sítě

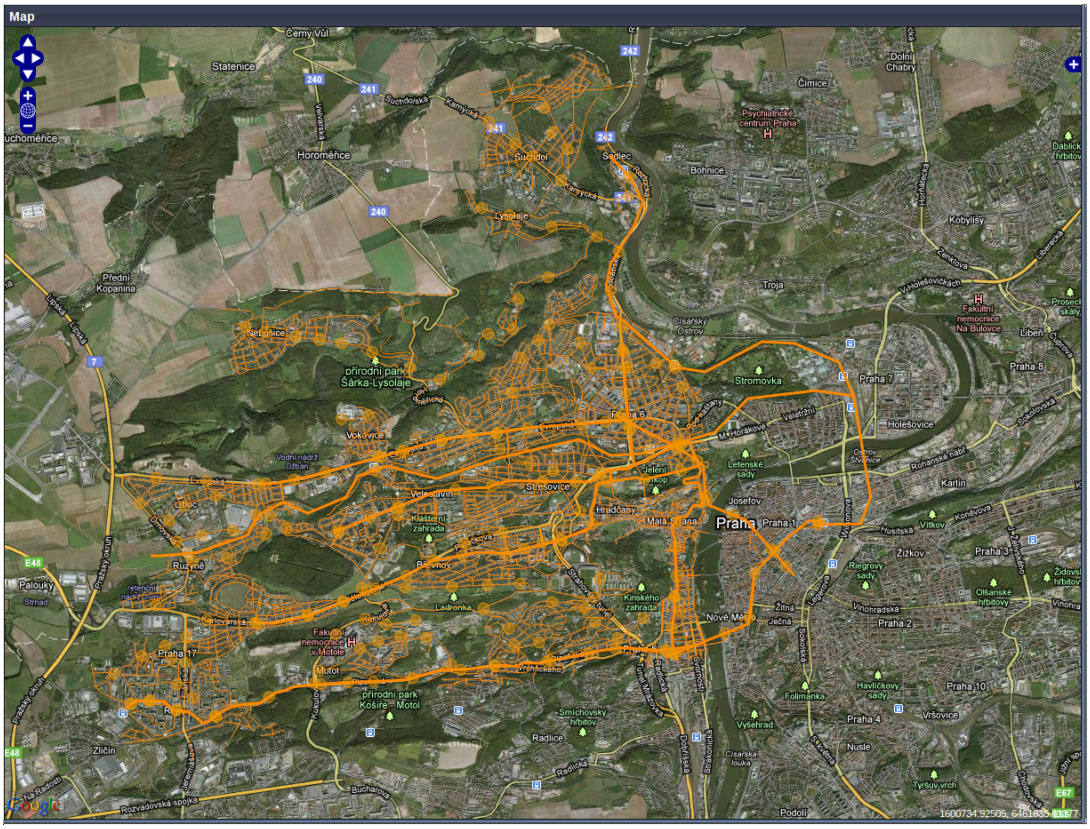
V této kapitole budou popsány všechny kroky, které vedly ke vzniku aplikačního prostředí fungujícího nad reálnými daty. V první fázi je potřeba vytvořit databázi a potřebné databázové tabulky. Příkazy k vytvoření databázových tabulek se nacházejí v elektronické příloze této práce. Pro provedení těchto úkonů byl využit program PgAdmin 3, který umožňuje správu databází. Poté bylo potřeba v editační aplikaci vytvořit příslušné 4 vrstvy (ulice, tramvaje, metro, vlaky). Nyní bylo možné přistoupit k nahrání uliční sítě. Nejprve bylo nutné data OpenStreetMap zpracovat. To proběhlo v programu QGIS následujícím způsobem. Byla nahrána vrstva představující všechny liniové prvky na území České republiky. Tato vrstva byla nejprve oříznuta na oblast, která nás zajímá a následně redukován počet linií v ní. Pravidlo bylo zvoleno takové, že budou zachovány jen linie typu ulice, které mají název. Tím se autor zamýšlel zbavit nepřesných, či omylem zanesených linií (předpokladem je zde to, že pokud už někdo pojmenuje ulici ve správném atributu, tak má alespoň základní ponětí o základech systému a v takovém případě nebude tak často docházet k chybám). Zároveň tímto krokem také byly odebrány linie marginálního významu jako jsou různé průchody, stezky, apod. To samozřejmě pomůže výkonu aplikace, na druhou stranu v některých případech to zkreslí výsledky analýzy, protože v některých místech jsou právě tyto cesty menších významů jedinými cestami spojujícími 2 místa, mezi nimiž se jinak musí obtížně cestovat. Další problém nastal s víceproudými ulicemi (např. Evropská) mezi nimiž byl jen na málo místech umožněn přechod. To neodpovídá realitě, kdy tyto lze přecházet na přechodech na více místech. Snahou bylo z těchto víceproudých ulic udělat smazáním jednoho proudu obyčejné. Poté však bylo nutno opravit ještě návaznosti bočních ulic na tyto ulice, které mnohdy končily na smazaném proudu. Taktéž byly doplněny editací spoje mezi místy, kde ve skutečnosti průchod existuje a je důležitý pro

pěší. Naopak na některých místech bylo nutné přebytečné úseky ještě ručně odmazat. Po těchto úpravách byla výsledná vrstva uložena jako GeoJSON v zobrazení EPSG:900913 (Mercator definovaný na kouli), které je používané v OpenLayers. Tento soubor pak byl přes skript nahrán do databáze, linie topologicky očištěny a vytvořeny vazby mezi sousedními liniemi (viz. Kapitola Algoritmy). Výsledky těchto editací jsou vidět v obr. 7.1, kde je červenou barvou síť stažená z OpenStreetMap a žlutou barvou síť po editaci.

Obr. 7.1: Původní a aktuální uliční síť

Nyní bylo možné přistoupit k samotné editaci v prostředí aplikace. Vzhledem k nízkému počtu jak tramvajových linek (4 hlavní tahy), tak metra (2 trasy) i vlaků (taktéž 2 trasy) byla tato editace otázkou několika minut. Následovalo vytváření stanic. Těch se na daném území nachází 209 a jejich vytvoření včetně zadávání jejich názvů zabralo 1-2 hodiny. Poté mohly být vytvořeny jednotlivé linky. Vytváření linek je usnadněno tím, že aplikace automaticky hledá všechny stanice mezi 2 zadanými. V extrémním případě tedy lze jednu linku obsahující desítky zastávek zadat pomocí 2 kliknutí. To je možné v případě, kdy neexistuje jiná kratší cesta, kterou by aplikace mezi danými 2 zastávkami našla. Je zřejmé, že tato funkce usnadní práci spíše při práci s jinými vrstvami než s uliční sítí. Ta je totiž velmi rozvětvená a v případě, že se uživatel spolehne na výpočet cesty aplikací, málokdy dojde k tomu, že tato cesta vede přes všechny požadované zastávky. Hledání této cesty nad uliční sítí může v závislosti na její velikosti nějaký čas trvat. Naopak ale stanice na uliční síti šetří čas při samotné analýze, protože již není třeba hledat nejbližší uliční linii. Oba problémy by samozřejmě byly řešitelné. V prvním případě přísnější optimalizací algoritmu a v druhém případě nějakým druhem serializace síťového modelu a tedy jeho vytváření jen ve chvíli kdy v síti dojde k nějakým změnám. Oba tyto aspekty však v práci řešeny nejsou, proto je s popsaným zdržením nutno počítat. Naopak přidávání stanic do linek mimo uliční síť proběhlo v podstatě okamžitě.

Pro srovnání lze uvést, že na uliční vrstvě se nachází 4507 linií, s cca. 8500 vazbami mezi sebou, zatímco na tramvajové vrstvě se nachází 96 linií, na vrstvě metra 16 linií a na vlakové vrstvě 11 linií. Přidělování stanic k jednotlivým linkám taktéž zabralo cca. 1-1,5 hodiny času. Poté bylo potřeba přidělit jednotlivým segmentům časy jízdy a jednotlivým linkám intervaly. To trvalo zhruba 1 hodinu. Celkově tedy vytváření sítě zabralo 5-7 hodin s tím, že takto vytvořená síť je velmi snadno editovatelná z hlediska časů jednotlivých segmentů, intervalů linek, přidávání dalších zastávek i dalších tratí. Problematickým bodem je zde jedině měnění tras v rámci uliční sítě, které může být časově náročnější. Výsledná síť je zobrazena na obr. 7.2.



Obr. 7.2: Vytvořená síť s podkladem

Při editaci se objevily některé zajímavé situace, které je zde třeba zmínit. Při přidávání stanic nastaly situace, kdy z mapy bylo zřejmé, že je zastávka jednosměrná, případně že zastávky v obou směrech jsou daleko od sebe. V takovémto případě vytvořil autor – tvůrce sítě stanici tak, aby její poloha byla průměrem obou stanic, avšak i s ohledem na morfologii sítě v daném místě.

Dále při vytváření linek metra nastala zajímavá situace při přidávání stanic, které mají 2 východy z podzemí. Zjednodušit situaci na přidání pouze jednoho východu už autor uznal jako příliš velký odklon od reality. V některých případech může vzdálenost

mezi těmito východy činit až 5 minut chůze. Situace je tedy řešena přidáním 2 stanic na místa východů. Obě tyto stanice jsou přidány jako součást linky a je mezi nimi nastavena délka segmentu 0.

Několikrát se též stalo, že při zadání po sobě jdoucích stanic na uliční sítě došlo k vypočtení trasy jinudy než autobusy ve skutečnosti jezdí. S tímto se v současné verzi aplikace nic dělat nedá, zároveň je však nutné dodat, že to také nijak nebrání funkčnosti. Důležité jsou jen body, kde jsou stanice a pak časová vzdálenost mezi stanicemi (čas segmentu), která se však nepočítá z projeté trasy, ale zadává se při editaci.

7.3. Analýza nejkratší cesty

Nyní přistupme k vyhodnocení obou analýz. Hodnocení můžeme provádět např. z časového hlediska, či z hlediska přesnosti. U časového hlediska bude hrát roli kromě velikosti území, která je neměnná, taktéž přesnost výpočtu pravděpodobností (krok integrálu, viz. Kapitola 3.2), kde platí, že čím menší krok, tím delší čas analýz. Dále lze omezovat počet maximálních návštěv v jednom uzlu, což samozřejmě urychluje analýzy, ale zmenšuje přesnost výpočtu. Cílem je najít takovou konfiguraci, kdy časové hledisko i hledisko přesnosti budou v rovnováze. Vzhledem k návrhu algoritmu nezáleží na vzdálenosti dvou bodů při analýze nejkratší cesty, stejně jako by nemělo záležet na umístění bodu při analýze dostupnosti.

Nejprve byla zkoumána analýza nejkratší cesty mezi místem, které se nachází a to s krokem integrálu 1/50 intervalu, 1/20 intervalu, 1/10 intervalu a s počtem max. návštěv uzlu 10, 5 a 3. Původní myšlenka byla tyto časy vizualizovat v tabulce, nicméně výsledky dopadly poněkud překvapivě – u všech variant stejně a to časy okolo 1,2 minuty. Tento čas se může jevit jako vysoký, nicméně právě fakt, že časy pro všechny tyto varianty byly stejné, že drtivou většinu tohoto času zabere příprava sítě a nikoliv její samotné procházení. Autor tedy na základě těchto zjištění upravil program tak, aby se do databáze vždy ukládala ke každé stanici vždy nejbližší linie na uliční síti a bod na této síti. Tato úprava zlepšila dobu provádění analýzy na 17 vteřin.

Co se týká přesnosti, lze hodnotit, nakolik se od sebe výpočty pravděpodobností, časů a tras liší a jaké odchylky jsou ještě přijatelné. Co se týká různého kroku integrálu, zde se výsledky neliší vůbec. Pokud by se v některém případě lišily, bylo by to jen v pravděpodobnostech, nikoliv v časech a trasách. Jediný rozdíl se nachází mezi analýzami, kdy byl omezen počet návštěv uzlu na 3 a analýzami, kde byl maximální počet návštěv vyšší. V tuto chvíli (bez ohledu na to, zda-li jsou je analýza omezena 5 nebo 10 návštěvami uzlu) přibývá 4. varianta.

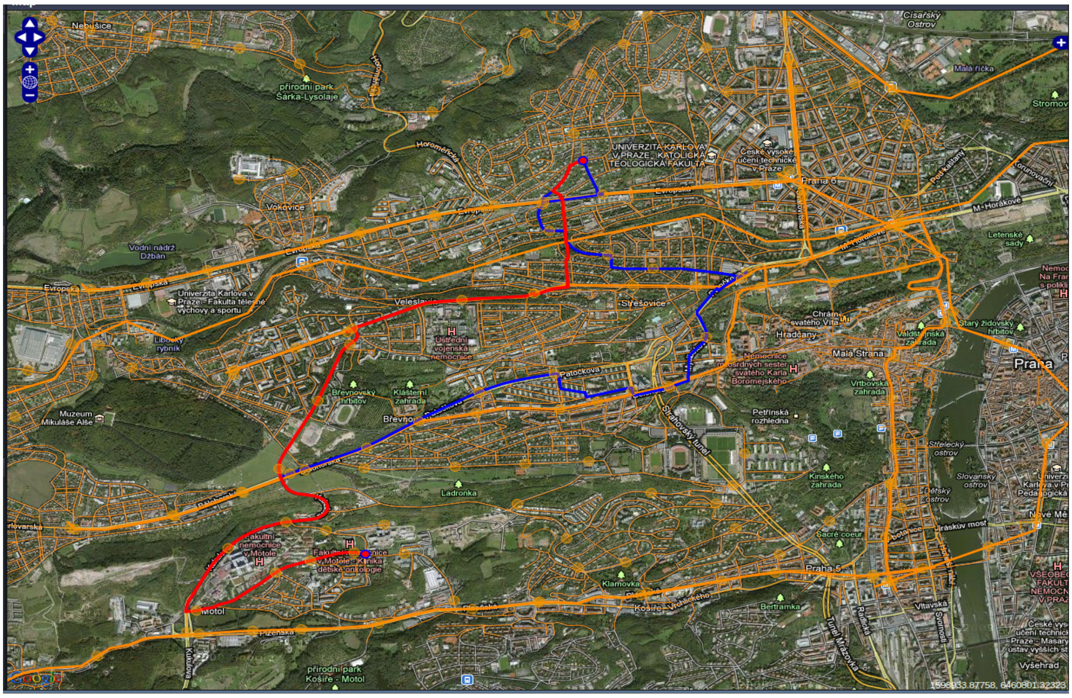
Jak lze vidět ze všech následujících obrázků, zkoumaná trasa začínala na Sídlišti Homolka a končila u Hanspaulky. Pravděpodobnosti prvních 3 variant se pohybují okolo 30% (viz. obr. 7.3), ačkoliv čas je v jednom případě o 5 minut delší. Paradoxně je to v případě, který má nejvyšší pravděpodobnost. Pak je zde ještě nepříliš

pravděpodobná varianta 4. V dalším textu se bude hovořit o variantách v pořadí od nejvyšší pravděpodobnosti k nejnižší.

Pravděpodobnost	Čas
29.9 %	35.31
31.3 %	35.12
1.7 %	43.07
37.0 %	39.94

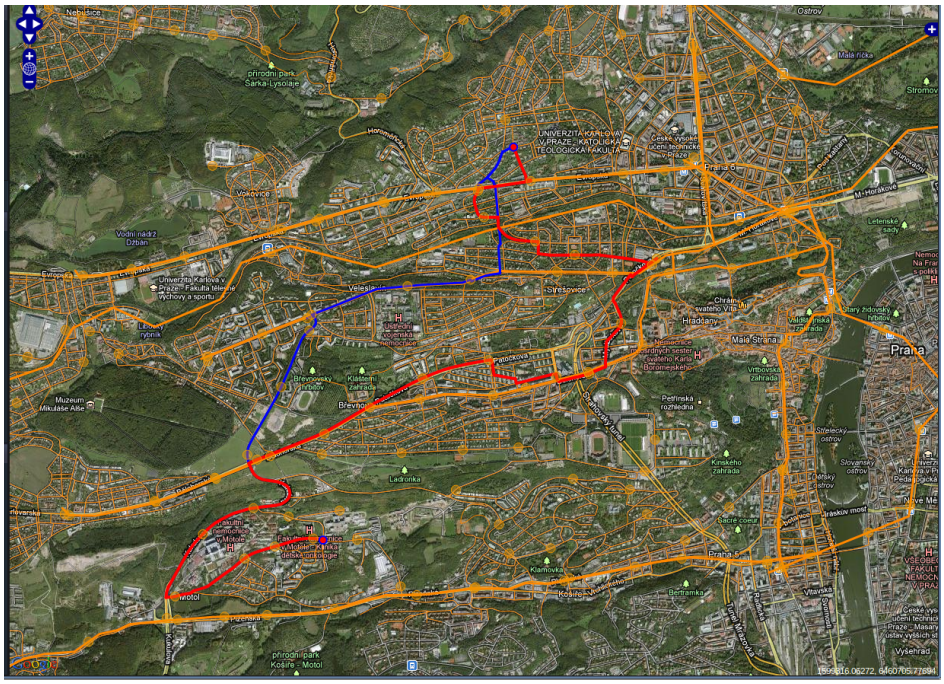
Obr. 7.3: Pravděpodobnosti jednotlivých variant

Na tomto obr. 7.4 lze vidět první variantu, tedy variantu nejpravděpodobnější. Podrobný popis cesty, který je taktéž součástí výsledku analýzy lze nalézt v Příloze 2. (pravděpodobnosti zde jsou počítány jen pro 3 varianty). Vzhledem k tomu, že v okolí startovního místa nejezdí jiná linka než 167, jsou první úseky absolvovány právě pomocí této linky. Dále je nutno přestoupit v Motole a zde využít jeden z autobusů 179,184 a dojet na Větrník, ani zde moc jiných variant neexistuje. Samozřejmě dalo by se uvažovat o vystoupení na Vypichu a posléze pokračování tramvají směrem na Dejvice. Ještě jednodušší by bylo jet přímo z Motola autobusy 174 či 180 směrem na Dejvice. Nicméně to tato varianta nezahrnuje. Z Větrníku se pokračuje linkami 1,2 či 18 na Baterie a odtud dále pěšky. Pokud se vezmou v úvahu intervaly těchto jednotlivých



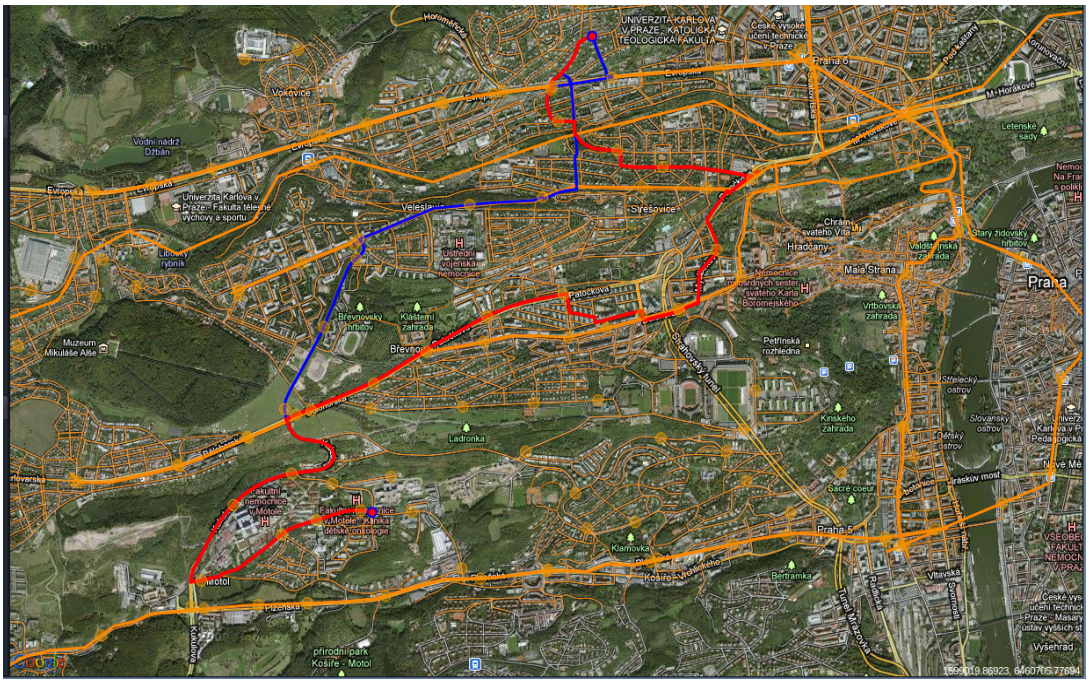
Obr. 7.4: Analýza nejkratší cesty - varianta 1

spojení, dospěje se k tomu, že nejvyšším intervalem je 6 minut u sdílených linek 179 a 184. Tento nízký interval je také důvodem, proč je pravděpodobnost využití tohoto spojení tak vysoká vzhledem k vyššímu času (viz. Kapitola 3.2).



Obr. 7.3: Analýza nejkratší cesty - varianta 2

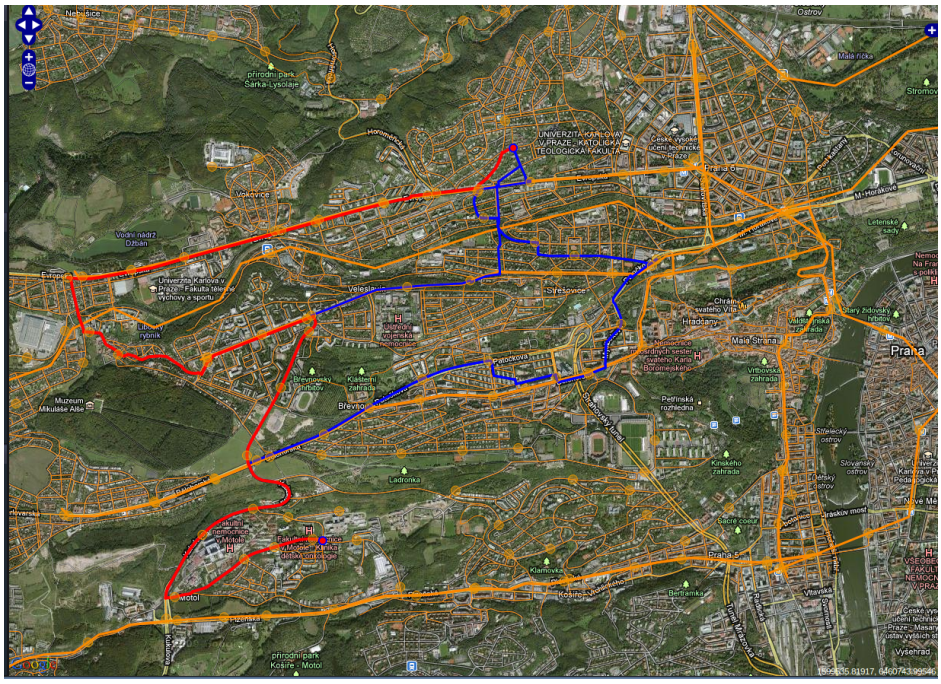
V druhém případě (obr. 7.5) jsou z Motola využity již zmiňované autobusy 174, 180 k přepravě na zastávku Vozovna Střešovice. Zde následuje přestup na autobus 216, pokračuje se na Bořislavku a odtud tramvají na Hadovku a pak až do cíle. Nevýhodou tohoto spojení je kromě vyššího počtu přestupů, což v algoritmu nijak penalizováno není, především fakt, že je využita linka 216, která i v ranní špičce má interval 20 minut. Je proto zřejmé, že nižší pravděpodobnost, které toto spojení dosahuje je zapříčiněna právě touto linkou.



Obr. 7.4: Analýza nejkratší cesty - varianta 3

Varianta 3 (obr. 7.6) je podobná variantě předchozí, akorát poslední úsek absolvovaný tramvají je nahrazen chůzí po mírně odlišné trase. Toto spojení je paradoxně méně pravděpodobné než předchozí, a to díky tomu, že má o cca. 20 sekund horší čas.

Poslední variantou (obr. 7.7) je varianta, která se objevila až ve chvíli kdy jsme povolili více jak 3 návštěvy jednoho uzlu. Z Motola se pokračuje autobusem 179 až na Divokou Šárku a odtud autobusy 218 či 119 na Horoměřickou a dále pěšky. Tuto variantu znevýhodňuje jak horší čas, tak i fakt, že linka 179 má ve špičce interval 12 minut, proto její použití nebude tak časté jako u jiných variant.

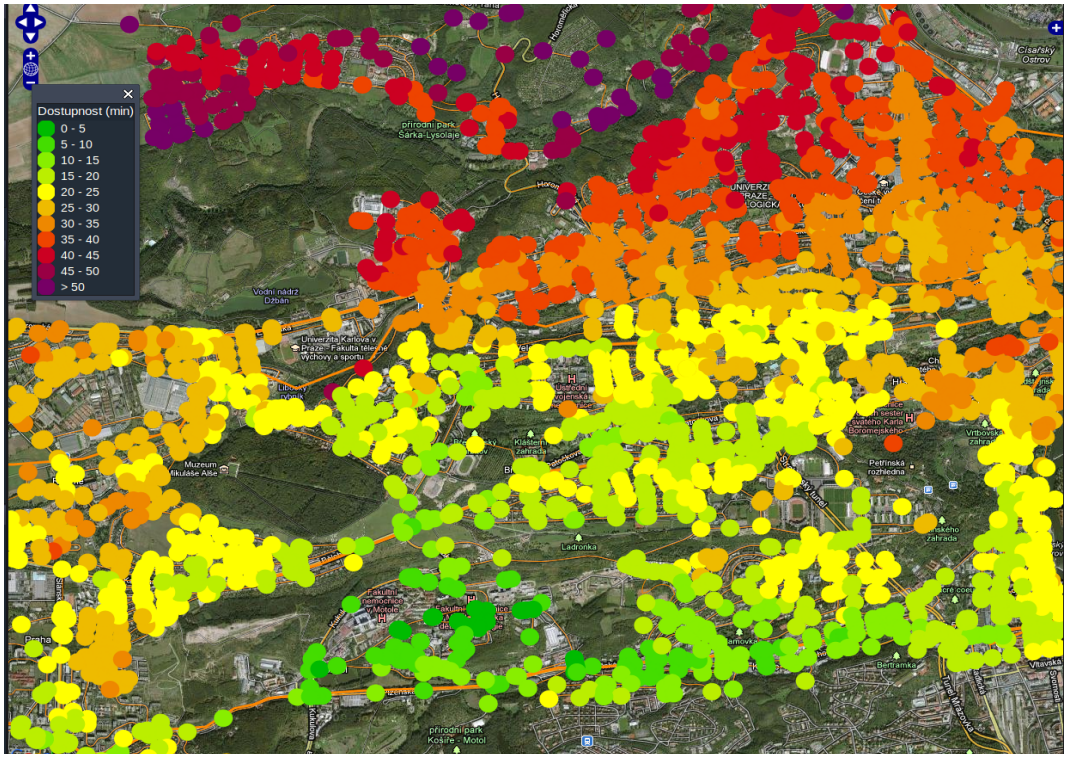


Obr. 7.5: Analýza nejkratší cesty - varianta 4

Srovnají-li se dosažené varianty s vyhledáváním na serveru IDOS, nutno konstatovat, že se v mnohém shodují a v mnohém liší. Problematický je zde fakt, že hodnocená aplikace využívá i chůze po uliční síti, proto nelze do IDOSu jednoznačně zadat zastávky. Pokud je tedy zadáno hledání Sídliště Homolka – Bořislavka, výsledek je podobný jako u varianty 3 (tedy bez posledního přestupu). Časy v tomto případě odpovídají. Variantu 1 v podstatě není šance vyhledat, protože zahrnuje dlouhý úsek, který je absolvován pěšky. Variantu 2 nelze nalézt ani při zadání cílové stanice Hadovka. IDOS upřednostňuje cestu přes Dejvickou v různých variantách. Někde se i objevuje cesta přes Anděl. Varianta 4 je pak k nalezení téměř vždy (při libovolné cílové zastávce). Lze tedy konstatovat, že IDOS nenalezne žádnou kombinaci, která by nebyla nalezena aplikací a časy odpovídají. Varianty, které zahrnují hodně chůze samozřejmě IDOS nalézt nemůže.

7.4. Analýza dostupnosti

Po analýze nejkratší cesty byla analyzována dostupnost. Pro ni bylo použito stejných 9 kombinací jako v minulém případě. Časy provádění analýz jsou opět stejné – 1.2 minuty, s výjimkou použití velkého počtu kroků integrálu (50 kroků na interval). V tom případě je čas 1.3 minuty. Je to z toho důvodu, že na rozdíl od analýzy nejkratší cesty jsou zde pravděpodobnosti počítány pro tisíce uzlů, a zde už se tento parametr projeví. Je však vidět, že nijak dramaticky. Po uložení některých údajů o síti do databáze se čas stejně jako v případě analýzy nejkratší cesty zkrátil na 17 vteřin.



Obr. 7.6:Dostupnost sídliště Homolka

Na obr. 7.8 lze vidět dostupnost bodu u zastávky Sídliště Homolka z minulé analýzy. Ve výsledné mapě jsou vidět očekávané vzorce, kdy podél hlavních tahů, dobře pokrytých linkami MHD je dostupnost lepší než v jejich okolí. Dobře se to projevuje např. podél tramvajové linky Petřiny – Vozovna Střešovice, či Sídliště Řepy – Anděl. V některých místech vystupují body s výrazně horší dostupností než v okolí. Je to z důvodu, že se jedná o slepé ulice, napojené do sítě jen ze strany sítě s horší dostupností.

Autor dále provedl analýzu dostupnosti pro místo, které se nacházelo u Václavského náměstí. Výsledek analýzy pak má reprezentovat dostupnost Prahy 6 z centra (viz. Příloha 3). Velmi dobře vynikají se svojí dobrou dostupností stanice metra Anděl, Malostranská, Hradčanská, Dejvická a Nádraží Bubeneč. Zároveň je vidět, že díky těmto dobře dostupným místům se pak místa s dobrou dostupností nacházejí též v Plzeňské ulici a okolí, Patočkově ulici a okolí a na Evropské třídě v důsledku dobrého tramvajového spojení.

Poslední prováděnou analýzou byla analýza spojení Dejvic (konkrétně stanice Dejvická), jakožto přirozeného centra Prahy 6 se zbytkem území (viz. Příloha 4). Z výsledku lze vyčíst, že nejlépe napojeny na Dejvickou jsou místa v západním, severním, jihozápadním a jihovýchodním směru. V případě západního směru se jedná o tramvajové linky kombinované s autobusovými mířící na Bořislavku a pak dále do Ruzyně, v případě severního směru se jedná o autobusové linky směřující na Suchdol. V

případě jihovýchodního směru je jasně patrný vliv metra, v jihozápadním směru pak poměrně frekventovaná doprava ve směru Střešovice, Břevnov. Naopak východní a jižní směr mají dostupnost zhoršenou.

Celkově lze hodnotit analýzy jako vydařené vzhledem k tomu, že odpovídají realitě a projevují se v nich očekávané vzory. Zejména u analýz nejkratší cesty se mnohdy projevuje i vliv různých intervalů, což je faktor, který do analýz není započítáván tak často.

KAPITOLA 8: ZÁVĚR

V této kapitole je potřeba shrnout poznatky, které vyplývají z práce. Nejprve budou představeny závěry a výsledky práce. V následující části pak budou tyto výsledky konfrontovány s cíli práce, zhodnoceny její přednosti a nedostatky a také nastíněny další možnosti rozvoje.

8.1. Výsledky a závěry práce

V práci je popisován způsob, jakým byl vyvinut model pro provádění analýz dostupnosti a analýzy nejkratší cesty. Nad tímto modelem pak byla vyvinuta aplikace, která teorii tohoto modelu uvádí do praxe a ověřuje jeho použitelnost.

Autor dochází k tomu, že existuje množství popsaných přístupů, kterým se lze postavit k problému provádění analýz. V souladu se zadáním práce je zvolen přístup, který vychází z již existujících teorií, kombinuje je, a přidává vlastní myšlenky. Výsledkem je model, který umožňuje provádění analýz bez použití konkrétních jízdních řádů. Výhodou modelu je to, že bere v úvahu i pohyb cestujících v uliční síti, snaží se realisticky odhadovat chování cestujících a co nejpřesněji popsat jízdní časy při pohybu v síti.

Pokud se autor při návrhu modelu inspiroval u jiných modelů a ty kombinoval a přidával vlastní myšlenky, při návrhu aplikace již převládaly originální postupy. Vyvinutá aplikace je kombinací mnoha technologií. Tuto kombinaci lze považovat za jedinečnou. Zmiňme technologie OpenLayers, ExtJS, GeoExt na klientské straně aplikace a CGI, Shapely, STORM, PostgreSQL na straně serveru. Výsledkem této kombinace jsou 2 aplikace, jedna určená pro editaci sítě, druhá pro provádění analýz.

V práci uvedená případová studie prováděná nad sítí Prahy 6 ukázala, že aplikace je použitelná v praxi. Po úpravě aplikace trvalo provádění analýz okolo 17 vteřin, což je výsledek, který autor pokládá za uspokojivý, ačkoliv zde samozřejmě existuje prostor pro další optimalizaci. Výsledné trasy se jevily jako reálně použitelné a byly podobné těm, které vyhledával server IDOS. Práce s aplikací je uživatelsky příjemná, a grafická podoba aplikace nabízí intuitivní a standardní ovládání.

8.2. Diskuze, naplnění cílů práce

Cílem práce bylo srovnat a popsat modely veřejné dopravy teoreticky a na příkladech vybraných měst. Na základě toho pak navrhnout aplikaci, která umožní editaci modelu a provádění analýz nad ním. Tato aplikace měla být otestována nad reálnými daty.

Tento cíl se dle autorova mínění povedlo naplnit. Nedostatkem je absence popisu používaných modelů v prostředí českých měst. Vzhledem k současnému rozsahu práce, který by byl již pouze obtížně rozšiřitelný, se autor zaměřuje spíše na obecnější teoretické modely a posléze popis vlastních postupů. Autor popsal teoretické modely jak bez návaznosti na místo, tak aplikovány na konkrétní místa. Tento teoretický základ se ukázal jako dostatečný pro sestavení vlastního modelu. V případě aplikace se povedlo naplnit cíle snadné editovatelnosti sítě a velkého zapojení automatizace do procesu jejího vytváření. Aplikace je snadná na používání, což bylo také jedním z cílů práce. Konečně i případová studie, jejíž vytvoření bylo jedním z cílů práce, ukázala, že koncept je využitelný v praxi.

Mezi hlavní přednosti práce autor řadí vytvoření vlastního modelu, který je kombinací stávajících s přidáním vlastních myšlenek. Další předností práce je to, že se povedlo vyvinout aplikaci na poměrně netradičních základech. Za výhodu lze považovat i fakt, že se jedná o aplikaci webovou, a tudíž snadno nasaditelnou.

Nedostatkem práce je nedostatečná pozornost věnovaná modelům používaným v českých městech. Za další nedostatek lze považovat architekturu aplikace a styl psaní kódu. Tyto nedostatky se začaly projevovat postupem času, jak aplikace rostla. Díky tomu aplikace není v některých činnostech tak výkonná a rychlá jak si autor původně představoval.

Možností rozvoje jak práce, tak s ní související aplikace je mnoho. Určitě je potřeba v první řadě zmínit nutnost provést mnoho optimalizací v kódu programu před tím, než by mohl být nasazen v ostrém provozu. Pokud by se povedlo tyto nedostatky vyřešit, otevřela by se pro vývojáře spousta možností. Bylo by možné srovnávat mezi sebou 2 různé sítě a zkoumat rozdíly v dostupnostech. Při přiřazení rastru s hustotou zalidnění by bylo možné časy dostupností počítat váženě vzhledem k této hustotě. Ve chvíli kdy by uživatelé měli k dispozici data, týkající se požadavků cestujících na přepravu (OD páry), bylo by možné určovat vytížení jednotlivých linek ad. Co se týká teoretického modelu, existuje zde mnoho možností jak zpřesnit výpočty pravděpodobnosti využití jednotlivých variant přidáním dalších koeficientů.

Autor si při vytváření této práce osvojil mnoho poznatků z teorie existujících modelů hromadné dopravy. Nejcennější jsou však pro něj znalosti a zkušenosti nasbírané při programování aplikace a seznamování se s novými technologiemi.

POUŽITÉ ZDROJE

AŇEZ J. - DE LA BARRA, T. - PÉREZ, B. Dual graph representation of transport networks [online]. Transportation Research Part B, 1996, vol. 30, no. 3, s. 209-216 [cit. 2011-01-12].
URL <http://www.sciencedirect.com/science/article/pii/0191261595000240>

BAYER, Tomáš. Algoritmy v digitální kartografii. 1. vydání. Praha: Karolinum, 2008, 251 s. , ISBN 978-80-246-1499

BORNDÖRFER, Ralf – GRÖTSCHEL, Martin – PFETSCH, Marc E. A Column-Generation Approach to Line Planning in Public Transport [online]. Transportation Science, vol. 41, no. 1, 2007, s. 123-132 [cit. 2011-01-10].
URL <http://www.springerlink.com/content/3v41181130t76165/>

CONSTANTIN Isabel. Deterministic ("Timetable") Transit Assignment [online]. INRO, 1998 [cit. 2011-01-12]
URL <www.inro.ca/en/pres_pap/international/ieug98/timetab.pdf>

FAN, Wei – MACHEMEHL, Randy B. Optimal Transit Route Network Design Problem: Algorithms, Implementations, and Numerical Results . Journal of Transportation Engineering, vol. 132, no. 1, 2006 [cit. 2011-01-20]
URL <http://ascelibrary.org/teo/resource/1/jtpedi/v132/i1/p40_s1?isAuthorized=no>

FLORIAN, Michael. Deterministic Time Table Transit Assignment [online]. In First Asian EMME/2 Users Group Meeting. Shangai, 1999 [cit. 2011-01-12]
URL <www.inro.ca/en/pres_pap/asian/asi99/paper3.pdf>

FRIGG, Roman – HARTMANN, Stephan. Models in Science. The Stanford Encyclopedia of Philosophy (Summer 2009 Edition), 2009 [cit. 2011-01-10]
URL <http://plato.stanford.edu/archives/sum2009/entries/models-science/>

GUILHAIRE, Valérie – HAO, Jin-Kao. Transit network design and scheduling: A global review [online]. Transportation Research Part A , 2008, vol. 42, no. 10, s. 1251-1273 [cit. 2011-01-10].
URL <http://www.sciencedirect.com/science/article/pii/S0965856408000888>

HUANG, Ruihong. A Schedule-based Pathfinding Algorithm for Transit [online]. Geoinformatica, vol. 11, no. 2, 2007, s. 269-285
URL <http://www.springerlink.com/content/x881468518766211/>

JANSSON, Kjell – RIDDERSTOLPE, Bosse. A Method for the Route-Choice Problem in Public Transport Systems, vol. 26, no. 3, 1992, s. 246-251 [cit. 2011-01-20]
URL <http://transci.journal.informs.org/cgi/content/abstract/26/3/246>

LARSEN, Odd I. - ØYVIND, Sunde. Waiting time and the role and value of information in scheduled transport . Research in Transportation Economics, vol. 23, no. 1, 2008, s. 41-52 [cit. 2011-01-20]
URL <http://www.sciencedirect.com/science/article/pii/S0739885908000425>

LENNON Joe. Compare JavaScript frameworks [online]. Poslední revize 2.2.2010,
URL <http://www.ibm.com/developerworks/web/library/wa-jsframeworks/>

LOWSON, Martin - Idealised models for public transport systems [online]. International Journal of Transport Management, vol. 2, no. 3-4, 2004, s. 135-147 [cit. 2011-01-20]
URL <http://www.sciencedirect.com/science/article/pii/S1471405105000212>

MAGNANTI, T. L. - WONG, R. T. Network Design and Transportation Planning: Models and Algorithms, vol. 18, no. 1, 1984, s. 1-55 [cit. 2011-02-05]
URL <http://transci.journal.informs.org/cgi/content/abstract/18/1/1>

NIELSEN, Otto Anker. A stochastic transit assignment model considering differences in passengers utility functions [online]. Transportation Research Part B, vol. 34, no. 5, 2000, s. 377-402 [cit. 2011-02-02]
URL <http://www.sciencedirect.com/science/article/pii/S0191261599000296>

NUZZOLO, Agostino – CRISALLI, Umberto. The schedule-based modelling of transportation systems: recent developments [online]. Operations Research/Computer Science Interfaces Series, vol. 46, 2009, s. 1-26 [cit. 2011-02-02]
URL <http://www.springerlink.com/content/p8n6t171805633g1/>

O'SULLIVAN, David – MORRISON, Alastair – SHEARER, John. Using desktop GIS for the investigation of accessibility by public [online]. International Journal of Geographical Information Science, vol. 14, no. 1, 2000, s. 85-104 [cit. 2011-01-20].
URL <http://www.tandfonline.com/doi/abs/10.1080/136588100240976>

SPIESS Heinz., FLORIAN, Michael - Optimal strategies: A new assignment model for transit networks, vol. 23, no. 2, 1989, s. 83-102 [cit. 02-02-2011]
URL <http://www.sciencedirect.com/science/article/pii/0191261589900349>

THILL, Jean-Claude. Geographic information systems for transportation in perspective [online]. Transportation Research Part C, vol. 8, no. 1-6, 2000, s. 3-12 [cit. 2011-01-20]
URL <http://www.sciencedirect.com/science/article/pii/S0968090X00000292>

UŠPALYTÉ-VITKUNIENÉ, Rasa – BURINSKIENÉ, Marija -GRIGONIS, Vytautas. Calibration of Vilnius public transport model [online]. Transport, vol. 21, no. 3, 2006, s. 201-206 [cit. 2011-02-05]
URL <http://www.mendeley.com/research/calibration-vilnius-public-transport-model/>

The Concise Oxford English Dictionary. Tenth Edition. New York: Oxford University Press, 1999. 1666 s. ISBN 0-19-860259-6. s. 1524

AJAX Tutorial [online],
URL <http://www.w3schools.com/ajax/default.asp>

CSS Tutorial [online],
URL <http://www.w3schools.com/css/default.asp>

CGI: Common Gateway Interface [online],
URL <http://www.w3.org/CGI/>

Django | The Web framework for perfectionists with deadlines [online],
URL <https://www.djangoproject.com/>

Ext JS 3.4 API Documentation [online],
URL <http://dev.sencha.com/deploy/ext-3.4.0/docs/>

FrontPage - STORM [online],
URL <https://storm.canonical.com/>

HTML Tutorial [online],
URL <http://www.w3schools.com/html/default.asp>

JavaScript Toolkit for Rich Web Mapping Applications [online],
URL <http://www.geoext.org/>

Javascript Tutorial [online],
URL <http://www.w3schools.com/js/default.asp>

jQuery: The Write Less, Do More, JavaScript Library [online],
URL <https://www.jquery.com/>

jQuery UI – Home [online],
URL <https://www.jqueryui.com/>

Mod_python - Apache/Python Integration [online],
URL <http://www.modpython.org/>

OpenLayers [online], poslední revize 9.9.2010,
URL <http://dev.openlayers.org/releases/OpenLayers-2.10/doc/apidocs/files/OpenLayers-js.html>

PostgreSQL: The world's most advanced open source database [online],
URL <http://www.postgresql.org/>

Python Programming Language – Official Website [online],
URL <http://www.python.org/>

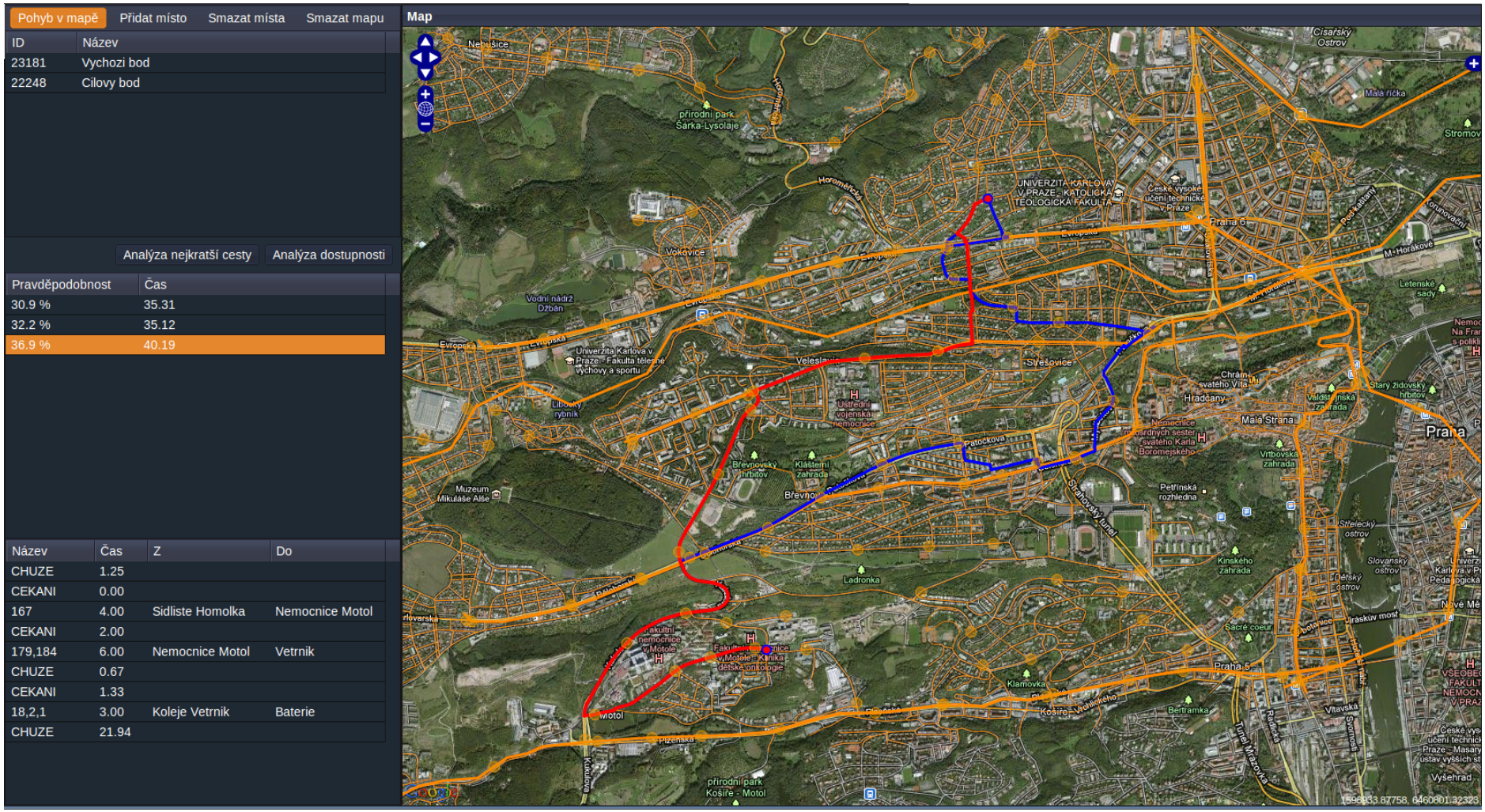
Shapely – GIS Python Lab [online],
URL <http://trac.gispython.org/lab/wiki/Shapely>

Unbeatable JavaScript Tools - The Dojo Toolkit [online],
URL <http://dojotoolkit.org/>

Welcome! - The Apache HTTP Server Project [online],
URL <http://httpd.apache.org/>

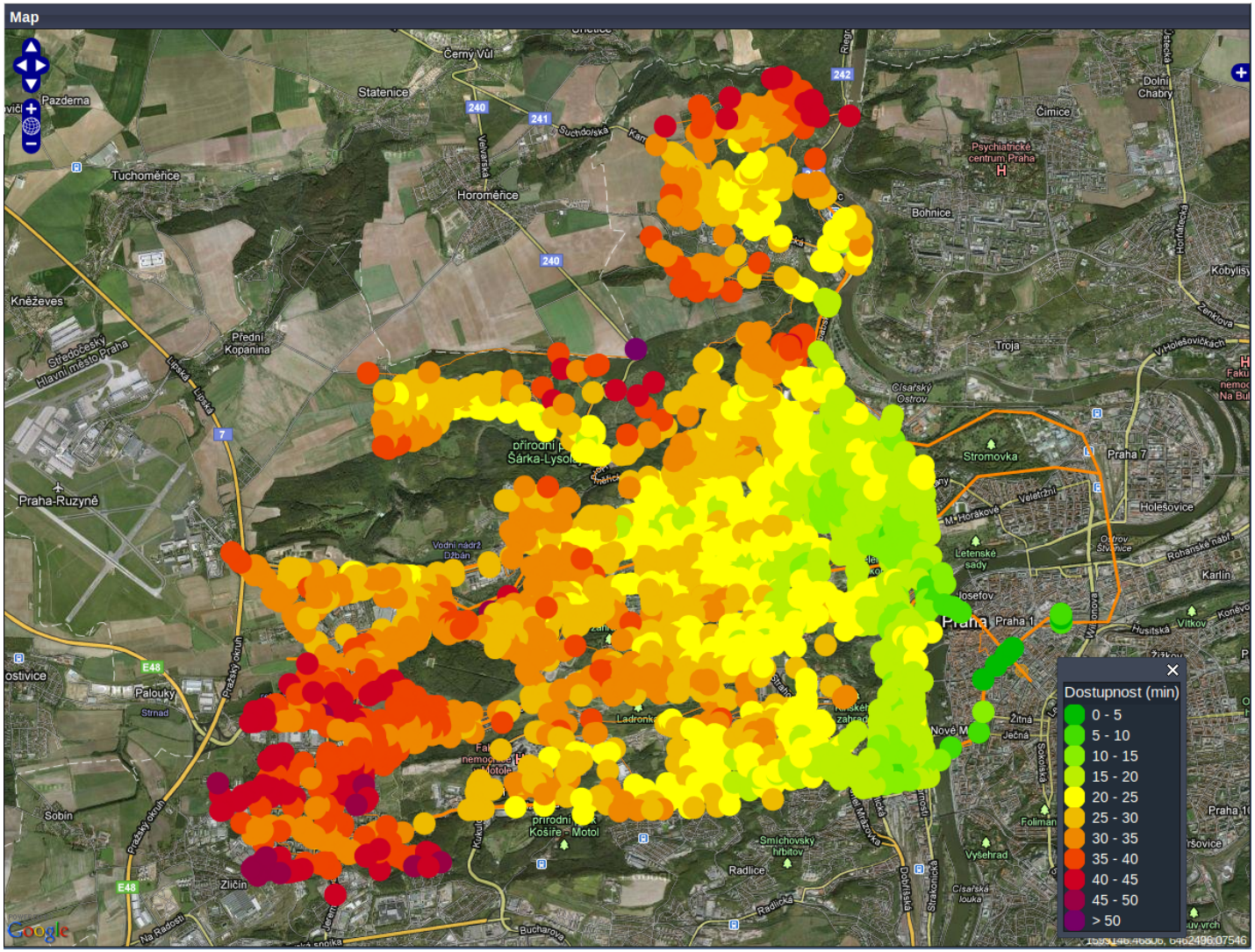
SEZNAM PŘÍLOH

Příloha 2: Analýza nejkratší cesty – varianta 1



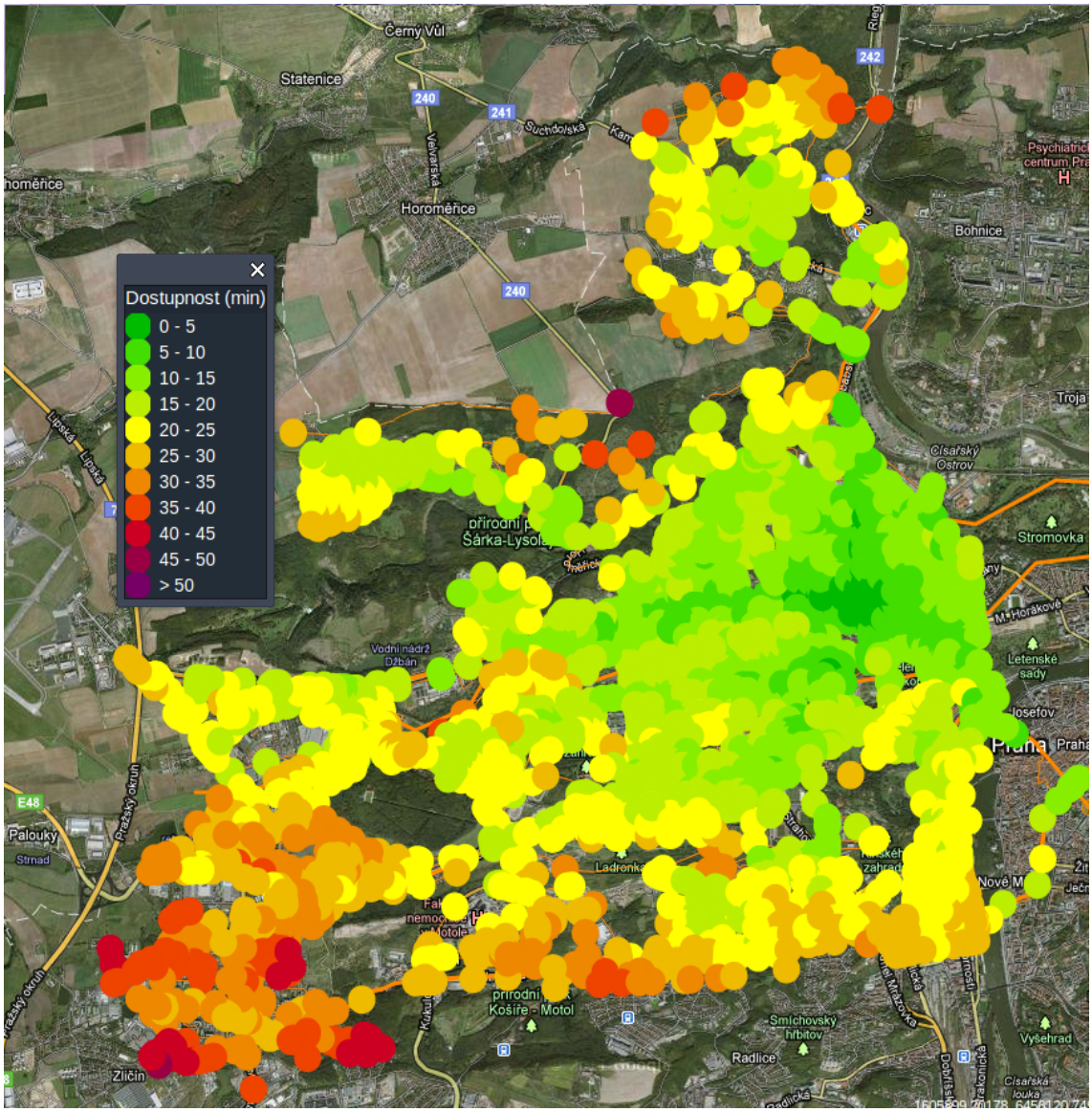
Zdroj: vlastní aplikace – varianta 1 červeně, ostatní modře

Příloha 3: Dostupnost Prahy 6 z Václavského náměstí



Zdroj: vlastní aplikace

Příloha 4: Dostupnost Prahy 6 ze stanice metra Dejvická



Zdroj: vlastní aplikace

Příloha 5: Zdrojové kódy

SEGMENT 2.1

```
# prochazeni objektu, ktere budou nahrazeny novymi
for replace in splitter['replace']:

    # feature k odstraneni
    oldFeature = store.get(Feature, int(replace['remove']))

    # objekty k pridani
    for objToAdd in replace['add']:

        feature = Feature()
        feature.layer_id = splitter['layerId']
        feature.geometry = unicode(objToAdd['geometry'])

        store.add(feature)
        # reference na objekt z klienta
        feature.incomingFeat = objToAdd;
        newFeatures.append(feature)
        # mapovani id objektu z klienta na novy objekt
        featMap[objToAdd['id']] = feature.id

    # nalezeni vsech sousednich features
    featuresToRefIds = store.find(Feature_Feature, Feature_Feature.feature1_id ==
      oldFeature.id).values(Feature_Feature.feature2_id)
    featuresToRef = store.find(Feature, Feature.id.is_in(featuresToRefIds))
    # prochazeni dotykajich se features
    for featureToRef in featuresToRef:
        # prochazeni nove pridavanych features
        for newFeature in newFeatures:
            # pokud se dotykaji vytvoreni nove vazby
            if (shapely.wkt.loads(featureToRef.geometry).touches(shapely.wkt.loads(newFeature.geometry))):
                f2f = Feature_Feature(newFeature.id, featureToRef.id)
                 store.add(f2f)
    # smazani puvodni linie
    deleteFeatureInternal(oldFeature.id, store, newFeatures)

#navazani novych 2 linii na stanici
if splitter.has_key('station'):
    station = Station()
    ...
    featMap['station'] = station.id

# pouhe pridavani linii
for objToAdd in splitter['add']:
    feature = Feature()
    feature.layer_id = splitter['layerId']
    feature.geometry = unicode(objToAdd['geometry'])
    feature.incomingFeat = objToAdd;
    featMap[objToAdd['id']] = feature.id

# referencovani novych features
for feature in newFeatures:
    if feature.incomingFeat.has_key('neighborIds') == False: continue
    # prochazeni sousedu z klienta
    for neighborId in feature.incomingFeat['neighborIds']:
        # soused se nachazi v nove pridanych features
        if str(neighborId).startswith('i'):
            neighborFeature = store.get(Feature, featMap[neighborId])
        # soused jiz exituje
        else:
            neighborFeature = store.get(Feature, neighborId)
        # pokud neexistuje vazba, vytvoreni vazby
        f2f = Feature_Feature(feature.id, neighborFeature.id)
        store.add(f2f)
```

SEGMENT 2.2

```
# nacteni geoJSONu
feat = json.loads(shpFile.read())
lineStrings = []# nacteni geometrii
for feature in feat['features']:
    lineStrings.append(asLineString(feature['geometry']['coordinates']))
# procisteni od krizicich se prvku, dle napr. navodu k JTS
nodedLineStrings = lineStrings[0];
for i in range(len(lineStrings)):
    nodedLineStrings = nodedLineStrings.union(lineStrings[i]);

lineString2Id = dict()
lineStrings = list(nodedLineStrings.geoms)
# iterace jednotlivymi linestringy a vytvareni feature do db
for lineString in lineStrings:
    feat = Feature()
    feat.layer_id = layerId
    feat.geometry = unicode(str(lineString))

adjLineStrings = list(lineStrings)
# hledani sousedicich linii
for lineString1 in lineStrings:
    adjLineStrings.remove(lineString1)
    for lineString2 in adjLineStrings:
```

```
        if lineString1.touches(lineString2):
            # vytvoreni vazeb
```

SEGMENT 3.1

```
# stanice je prvni
if (len(stations) == 1):
    #hledame stanici ktera byla prvni doted
    nextStationResult = store.find(Line_Station, And(Line_Station.order == 0, Line_Station.line_id == lineId))
    # pridavana stanice nebude posledni
    if nextStationResult.is_empty() == False:
        nextStationId = nextStationResult.one().station_id
        nextStation = store.get(Station, nextStationId)
    # umisteni nalezenych stanic na zacatek retezu
    order = 0
    station = store.get(Station, stations[0])
# stanice ma predchudkyni
else:
    prevStation = store.get(Station, stations[0])
    station = store.get(Station, stations[1])
    # umisteni nalezenych stanic za nalezenou stanici
    order = store.find(Line_Station, And(Line_Station.station_id == stations[0], Line_Station.line_id ==
      lineId)).one().order + 1

    # ziskani nasledujici stanice
    nextStationResult = store.find(Line_Station, And(Line_Station.order == order, Line_Station.line_id ==
      lineId))

    if nextStationResult.is_empty() == False:
        nextStationId = nextStationResult.one().station_id
        nextStation = store.get(Station, nextStationId)

# je znama predchozi i nasledujici stanice
if (prevStation != None and nextStation != None):
    segments = findSegments(prevStation, station, store, catchAll)
    segments.extend(findSegments(station, nextStation, store, catchAll))
    mode = 'both'
# pouze predchozi
elif (prevStation != None):
    segments = findSegments(prevStation, station, store, catchAll)
    mode = 'prev'
# pouze nasledujici
elif (nextStation != None):
    segments = findSegments(station, nextStation, store, catchAll)
    mode = 'next'
# stanice je prvni pridavana
else:
    # vytvoreni stanice pro linku, v tomto pripade se jedna o prvni stanici retezu
    lineStation = Line_Station()
    lineStation.order = 0
    lineStation.line_id = lineId
    lineStation.station_id = station.id
    lineStations.append(lineStation)
    orderShift = 1
    mode = 'none'

# prochazeni nalezenych segmentu
for segment in segments:
    # stanice je na konci, ci uprostred, pro nove lineStation pouzijeme station2
    if (mode == 'both' or mode == 'prev'):
        stationId = segment['station2'].id
        shiftSegment = -1
    # stanice je na zacatku pro nove lineStations pouzivame station1
    else:
        stationId = segment['station1'].id
        shiftSegment = 0

    # vytvoreni linestation
    lineStation = Line_Station()
    lineStation.order = order + orderShift
    lineStation.line_id = lineId
    lineStation.station_id = stationId

    # existuje dany segment?
    segmentResult1 = store.find(Segment, And(Segment.station1_id == segment['station1'].id, Segment.station2_id
      == segment['station2'].id))
    if segmentResult1.is_empty() and segmentResult2.is_empty():
        dbSegment = Segment()
        ...
    else:
        dbSegment = segmentResult1.one()

    # vytvoreni prislusneho line-segmentu
    lineSegment = Line_Segment()
    lineSegment.order = order + orderShift + shiftSegment
    lineSegment.line_id = lineId
    lineSegment.segment_id = dbSegment.id
    orderShift = orderShift + 1
# v tomto pripade jeste musime odstranit segment, ktery lezel tam kde vznikaly nove zastavky

… odstraneni segmentu# posunuti poradi u lineStations
followingLineStations = store.find(Line_Station, And(Line_Station.order >= order, Line_Station.line_id ==
      lineId))
for lineStation in followingLineStations:
    lineStation.order = lineStation.order + orderShift

#posunuti poradi u linesegments
followingLineSegments = store.find(Line_Segment, And(Line_Segment.order >= order, Line_Segment.line_id ==
      lineId))
```

```
    for lineSegment in followingLineSegments:
        lineSegment.order = lineSegment.order + orderShift
```

SEGMENT 3.2

```
    # prochazeni otevrenych hran
    while len(edgesToProcess) > 0:

        edge = edgesToProcess.pop(0)
        feature = edge.feature

        # pokud ma hrana jiz dva uzly, nezajima nas
        processedNodes = edge.nodes
        if len(processedNodes) > 1:
            continue
        # jediny uzel hrany
        processedNode = processedNodes[0]
        # a vsechny hrany se kterymi tento uzel souvisi
        for nodeEdge in processedNode.edges:
            closedFeatures.append(nodeEdge.feature)

        stations = store.find(Station, Or(Station.feature1 == feature, Station.feature2 == feature))

        # vytvoreni okrajoveho uzlu dane hrany
        anotherNode = Node()
        anotherNode.appendEdge(edge)

        # nalezeni sousedu
        adjacentFeaturesIds = store.find(Feature_Feature, Feature_Feature.feature1_id ==
(feature.id).values(Feature_Feature.feature2_id)
        adjacentFeatures = store.find(Feature, Feature.id.is_in(adjacentFeaturesIds))

        for adjacentFeature in adjacentFeatures:
            # zajimaji nas jen sousedi u spravneho uzlu
            if adjacentFeature in set(closedFeatures):
                continue
            # pokus o prirazeni stanice
            if anotherNode.station == None:
                # jen 1 ze 2 zastavek nalezi danemu nodu
                for station in stations:
                    point = loads(station.geometry)
                    line = loads(adjacentFeature.geometry)
                    if point.distance(line) < 0.1:
                        anotherNode.station = station
            # pokud uz pro feature hrana existuje, jeji nalezeni
            if feature2Edge.has_key(adjacentFeature):
                edge = feature2Edge[adjacentFeature]
            # pokud ne, jeji vytvoreni
            else:
                edge = Edge(loads(adjacentFeature.geometry).length, adjacentFeature)
                feature2Edge[adjacentFeature] = edge
                edgesToProcess.append(edge)

            anotherNode.appendEdge(edge)
```

SEGMENT 3.3

```
    # Dijkstra
    while len(openedNodes) > 0:
        openedNodes = sorted(openedNodes, key=operator.attrgetter('time'))
        currentNode = openedNodes.pop(0)
        currentNode.closed = True

        # podminka pro ukoneni vyhledavani
        if currentNode.station != None and currentNode.station == station2:
            break

        for edge in currentNode.edges:
            for edgeNode in edge.nodes:
                # podminka nejnizsiho casu
                if edgeNode.closed != True and currentNode.time + edge.time < edgeNode.time:
                    edgeNode.time = currentNode.time + edge.time
                    edgeNode.prev = currentNode
                    edgeNode.edgeGone = edge

    if catchAll == False:
        segment = dict()
        segment['station2'] = currentNode.station
        segment['features'] = []
    while currentNode.prev:
        if currentNode.station != None and catchAll == True:
            # pokud jiz je segment zacaty ukoncime ho a pridame do listu
            if segment != None:
                segment['station1'] = currentNode.station
                segment['features'].reverse()
                segments.append(segment)
            # zaciname novy segment
            segment = dict()
            segment['station2'] = currentNode.station
            segment['features'] = []
        segment['features'].append(currentNode.edgeGone.feature)
        currentNode = currentNode.prev
    # koncime posledni segment
    segment['station1'] = currentNode.station
    segment['features'].reverse()
    segments.append(segment)
```

```
# prevraceni poradi
segments.reverse()
```

SEGMENT 4.2.1

```
# nejprve prochazime vsechny linky a jejich useky
for line in lines:
    lineSegments = store.find(Line_Segment, Line_Segment.line_id == line.id).order_by(Line_Segment.order)
     lineStations = store.find(Station, Station.id == Line_Station.station_id, Line_Station.line_id ==
      line.id).order_by(Line_Station.order)

    # prvni node linky
    station = lineStations[0]
    if not(station in station2Nodes):
        station2Nodes[station] = []

    currentNode = Node()
    ...
    station2Nodes[station].append(currentNode)

    # prochazime useky linky
    for lineSegment in lineSegments:
        i=i+1
        station = lineStations[i]
        # nalezeni segmentu
        segment = store.find(Segment,Segment.id == lineSegment.segment_id).one()
        for segFeat in segment.features:
            geometries.append(loads(segFeat.geometry))

        # druhy node segmentu
        anotherNode = Node()
        ...
        station2Nodes[station].append(anotherNode)
        # vytvareni edge
        edge=Edge(currentNode,anotherNode)
        ...
        # prochazeni linek v danem segmentu
        linesForSegment = store.find(Line, Line_Segment.line_id == Line.id, Line_Segment.segment_id ==
              segment.id)
       # vytvoreni vsech kombinaci linek v danem segmentu (napr. 1,2;1,18;2,18;1,2,18), nesmi obsahovat jiz
      prosle linky
        combinations = createCombinations(linesForSegment,addedLines)
        for combination in combinations:
            combinationValid = False
            #zajimaji nas jen kombinace obsahujici danou linku
            for combLine in combination:
                if combLine == line:
                    combinationValid = True
            if not combinationValid:
                continue
            # je mozne ze dana kombinace uz je aktivni (tj. nezacina danym edge), v tom pripade musime navazat na
                   ni
            matchingLineCombination = getMatchingLineCombination(combination, lineCombinations)
            if matchingLineCombination:
                firstNode = matchingLineCombination.currentNode
                newLineCombinations.add(matchingLineCombination)
            # v opacnem pripade vytvarime novy node
            else:
                firstNode = currentNode.clone()
                firstNode.lineHeadway = getSharedLineTime(combination)
                station2Nodes[currentNode.station].append(firstNode)
                matchingLineCombination = LineCombination()
                matchingLineCombination.lines = combination
                newLineCombinations.add(matchingLineCombination)
            # nasledne vytvarime i druhy node spolecneho vedeni linek
            anotherCombinationNode = anotherNode.clone()
            anotherCombinationNode.lineHeadway = getSharedLineTime(combination)
            station2Nodes[station].append(anotherCombinationNode)
            edge=Edge(firstNode,anotherCombinationNode)
            ...
            # sem ukladame soucasny node, pro pripadne navazani dalsich segemntu
            matchingLineCombination.currentNode = anotherCombinationNode

        currentNode = anotherNode
        lineCombinations = newLineCombinations
    addedLines.add(line.id)
```

SEGMENT 4.2.2

```
# prochazime vsechny stanice
for station in station2Nodes.keys():
    # jeli stanice na ulicni siti, obe feature, ktere k ni nalezeji se pridaji do feature2Stations dict a dale se
            s nimi v teto       metode nepracuje
    if station.feature1.layer.streetLayer == True:
        if (feature2Stations.has_key(station.feature1)):
            feature2Stations[station.feature1].add(station)
        ...
        continue

    stationGeom = shapely.wkt.loads(station.geometry)
    # nachazime nejblizsi ulici a jeji vzdalenost
    nearestFeature, nearestDistance = findNearestFeature(stationGeom)
    # vytvarime okruh okolo stanice presne v pozadovane vzdalenost
    buffered = stationGeom.buffer(nearestDistance+1,100)
```

```
    # tento bod je nejblizsim bodem pro stanici v ulicni siti
    intersectGeometry = buffered.intersection(shapely.wkt.loads(nearestFeature.geometry)).centroid

    # novy nod na ulicni siti
    node = Node()
    node.geometry = intersectGeometry

    # napojime na nej vsechny nody, ktere patri dane stanici, jak load, tak unload
    for lineNode in station2Nodes[station]:
        lineEdge = lineNode.edges[0]
        edge = Edge(node,lineNode)
        edge.edgeType = 'LOAD'
        ...
        edge = Edge(lineNode, node)
        edge.edgeType = 'UNLOAD'
        ...
    # vazba ulicni feature na prislusny node
    if not(nearestFeature in feature2Node):
        feature2Node[nearestFeature] = []
    feature2Node[nearestFeature].append(node)
```

SEGMENT 4.2.3

```
# nalezeni nahodne feature
feature1 = store.find(Feature,Feature.layer_id == Layer.id, Layer.name == unicode('Upload')).any()
# nalezeni vsech sousedu

initialAdjacentFeatures = store.find(Feature, Feature.id == Feature_Feature.feature2_id, Feature_Feature
    .feature1_id == feature1.id)
# vytvoreni pvniho nodu na okraji nahodne feature
initialNode = Node()
initialFeatureGeom = loads(feature1.geometry)
# vytvoreni edge s jednim nodem
initialEdge = Edge(initialNode,None)
initialEdge.time = initialFeatureGeom.length/streetCnst
… další atributy initialEdge

# inicializace mapy a zasobniku
feature2Edge[feature1] = initialEdge
edgesToProcess = [initialEdge]
# geometrie, ktera reprezentuje prvni node
initialGeometry = shapely.geometry.Point((initialFeatureGeom.coords)[0])
initialNode.geometry = initialGeometry.centroid

# navazani edge na prvni node
for feature in initialAdjacentFeatures:
    geometry = loads(feature.geometry)
    if Not(geometry.touches(initialGeometry)):
        continue
    edge = Edge(node,None)
    … atributy edge
    feature2Edge[feature] = edge
    edgesToProcess.append(edge)
    # nyni jsou v edgesToProcess vsechny edge, ktere souvisi s initialNode, vcetne initialEdge

    while len(edgesToProcess) > 0:

    # kazda edge by mela mit prave jeden node
    edge = edgesToProcess.pop(0)
    feature = edge.featureRef
    # pokud ma dva nody, uz nas nezajima
    if edge.node2:
        continue
    processedNode = edge.node1
  # nebudeme se zabyvat features, ktere vychazeji z jiz inicializovaneho nodu, zajimaji nas features, ktere
      vychazeji z druheho          nodu

    for nodeEdge in processedNode.edges:
        closedFeatures.append(nodeEdge.featureRef)

    # vytvarime node na druhem okraji
    anotherNode = Node()
    anotherNode.edges.append(edge)
    edge.node2 = anotherNode
    featGeom = loads(feature.geometry)
    # pro souradnice tohoto nodu bereme bud zacatek ci konec geometrie
    if Point(list(featGeom.coords)[0]).intersects(processedNode.geometry):
        anotherNode.geometry = Point(list(featGeom.coords)[len(featGeom.coords)-1])
    else:
        anotherNode.geometry = Point(list(featGeom.coords)[0])

    # nalezeni vsech features sousedici s timto edge
    adjacentFeatures = store.find(Feature, Feature.id == Feature_Feature.feature2_id, Feature_Feature.feature1_id
              == feature.id)

    for adjacentFeature in adjacentFeatures:

        # nezajimaji nas ty sousedici s initialNode, viz vyse
        if adjacentFeature in set(closedFeatures):
            continue

        # v pripade, ze uz jsme sousedni feature zkoumali, nevytvarime novou edge, ale bereme jiz vytvorenou
        if feature2Edge.has_key(adjacentFeature):
            edge = feature2Edge[adjacentFeature]
            anotherNode.edges.append(edge)
            edge.node2 = anotherNode
        # vytvarime novou edge
        else:
            edge = Edge(anotherNode,None)
```

```
            … atributy edge
            feature2Edge[adjacentFeature] = edge
            edgesToProcess.append(edge)
        # pokud se feature nachazi v dict, mame informaci o tom, ze na ni je stanicni node
        if adjacentFeature in feature2Stations.keys():
            # muze jich tu byt i vice (max. 2)
            stations = feature2Stations[adjacentFeature]
            connectedStation = None
            # vytvarime
            for stationToConnect in stations:
                # stanice se musi nachazet na danem nodu, tedy sousedit jak z processedEdge tak nyni se sousednim
                    edge
               if (stationToConnect.feature1 == feature and stationToConnect.feature2 == adjacentFeature) or
                   (stationToConnect.feature2 == feature and stationToConnect.feature1 == adjacentFeature):
                       connectedStation = stationToConnect
                       for stationNode in station2Nodes[stationToConnect]:
                           lineEdge = stationNode.edges[0]
                           loadEdge = Edge(stationNode,anotherNode)
                           loadEdge.edgeType = 'LOAD'
                           ...
                           unloadEdge = Edge(anotherNode,stationNode)
                           unloadEdge.edgeType = 'UNLOAD'
                           ...
            if connectedStation:
                feature2Stations[adjacentFeature].remove(stationToConnect)
                feature2Stations[feature].remove(stationToConnect)
```

SEGMENT 4.2.4

```
# prochazeni vsech ulicnich feature, na ktere se bude vazat dopravni sit
for feature in feature2Node.keys():
    # na jednu feature se muze vazat i vice nodu
    nodes = feature2Node.get(feature)
    # prislusny edge reprezentujici streetFeature
    edge = feature2Edge.get(feature)
    # geometrie prislusne feature
    featureGeometry = loads(edge.featureRef.geometry)
    # do tohoto dictionary se budou ukladat postupne rozdelovane geometrie dane feature s vazbami na prislusne
      edge

    geometry2Edge = dict()
    geometry2Edge[featureGeometry] = edge
    for node in nodes:
        # rozdeleni geometrii v bode
        lines,minGeom = splitGeom(geometry2Edge.keys(), node.geometry)
        # edge ktera se bude rozdelovat
        edge = geometry2Edge[minGeom]
        # navazani nodu na spravnou stranu edge
        # pointGeom = list(minGeom.coords)[0]
        if edge.node1.geometry.intersects(lines[0]):
            newEdge1 = Edge(edge.node1,node)
            newEdge2 = Edge(edge.node2,node)
        else:
            newEdge1 = Edge(edge.node2,node)
            newEdge2 = Edge(edge.node1,node)
        # dokonceni inicializace
        newEdge1.geometry = lines[0]
        newEdge2.geometry = lines[1]
        newEdge1.edgeType = 'STREET_SPLIT'
        newEdge2.edgeType = 'STREET_SPLIT'
        newEdge1.time = newEdge1.geometry.length/streetCnst
        newEdge2.time = newEdge2.geometry.length/streetCnst
        newEdge1.featureRef = feature
        newEdge2.featureRef = feature
        geometry2Edge.pop(minGeom)
        geometry2Edge[newEdge1.geometry] = newEdge1
        geometry2Edge[newEdge2.geometry] = newEdge2

        edge.node1.edges.remove(edge)
        edge.node2.edges.remove(edge)
        edges.remove(edge)
```

SEGMENT 5.1.1

```
    # prvotni kontrola vhodnosti
     prevEdge = self.edges[len(self.edges)-1] if len(self.edges)>0 else None
     if prevEdge:
         # z line nelze prejit do load
         if (prevEdge.edgeType == 'LINE' or prevEdge.edgeType=='LINE_SHARED') and edge.edgeType == 'LOAD':
             return None
         # z cekoholiv jineho nez line nelze prejit do unload
         if (prevEdge.edgeType != 'LINE' and prevEdge.edgeType!='LINE_SHARED') and edge.edgeType == 'UNLOAD':
             return None
     # max 5 visitoru
     if len(node.visitors) >= maxVisitors:
         return None
     # nelze vytvaret cyklicke cesty
     if node in self.nodes:
         return None


     visitor = Visitor()
     visitor.lineRefs = list(self.lineRefs)
     visitor.commonSegmentRef = list(self.commonSegmentRef)
     visitor.edge2Time = self.edge2Time.copy()
```

```
        visitor.containsShared = self.containsShared
        # pouze pricitame cas
        if edge.edgeType == 'STREET' or edge.edgeType == 'STREET_SPLIT' or edge.edgeType == 'LINE' or edge.edgeType
          =='LINE_SHARED' or edge.edgeType == 'UNLOAD':
            visitor.time = self.time + (edge.time if edge.time else 0)
            visitor.edge2Time[edge] = edge.time
            visitor.headway = self.headway
        elif edge.edgeType == 'LOAD':
            line = edge.lineSegmentRef.line
            visitor.lineRefs.append(line)
            time = min([edge.time,self.headway])/2 + loadTime
            visitor.time = self.time + time
            visitor.edge2Time[edge] = time
            if (edge.commonSegmentRef):
                visitor.containsShared = True
                visitor.commonSegmentRef.append(edge.commonSegmentRef)
                visitor.time -= 0.00001*len(edge.commonSegmentRef)
            # nejvyssi dosazeny interval
            visitor.headway = edge.time if edge.time>self.headway else self.headway
```

SEGMENT 5.1.2

```
        # musime vyloucit navstevu v pripade, ze jiz zde je navstevnik s nizsim casem a jeho lineRefs jsou totozne ci
                  podmnozinou stavajicich,nebo casovy rozdil je vetsi nez interval stavajiciho visitora
        # take pokud jiz zde je visitor bez pouzitizadne linky, tak vylucujeme navstevu
        for recentVisitor in node.visitors:
            if recentVisitor.time<=visitor.time:
                if visitor.time - recentVisitor.time >= recentVisitor.headway:
                    return None
                if set(recentVisitor.lineRefs)<=set(visitor.lineRefs) and visitor.containsShared == False:
                    return None
                if len(recentVisitor.lineRefs) == 0 and recentVisitor.containsShared == False:
                    return None
                if (recentVisitor.containsShared) and len(visitor.lineRefs)>0:
                    for commonSegment in recentVisitor.commonSegmentRef:
                        if visitor.lineRefs[-1] in set(commonSegment) and visitor.containsShared==False:
                            return None
                        for cs in visitor.commonSegmentRef:
                            if (set(cs)<=set(commonSegment)):
                                return None
```

SEGMENT 5.2.1

```
    for edge in edgesClone:
        if edge.edgeType != 'STREET' and edge.edgeType != 'STREET_SPLIT': continue
        if edge.featureRef == feature1:
            geom = edge.geometry if edge.geometry!=None else loads(edge.featureRef.geometry)
            lines,minGeom = splitGeom([geom], geom1)
            startNode = replaceSplittedEdge(edge, lines)
        if edge.featureRef == feature2:
            geom = edge.geometry if edge.geometry!=None else loads(edge.featureRef.geometry)
            lines,minGeom = splitGeom([geom], geom2)
            finishNode = replaceSplittedEdge(edge, lines)
```

SEGMENT 5.2.2

```
    initialVisitor = Visitor()
    initialVisitor.nodes.add(startNode)
    initialVisitor.currentNode = startNode
    initialVisitor.currentNode.visitors.append(initialVisitor)
    visitors = [initialVisitor]
    while len(visitors)>0:
        # serazeni navstevniku a vybrani toho, s nejnizsim casem
        visitors = sorted(visitors, key=operator.attrgetter('time'))
        visitor = visitors[0]
        for edge in visitor.currentNode.edges:
            node = edge.node1 if visitor.currentNode == edge.node2 else edge.node2
            # overeni vhodnosti a pripadne vytvoreni noveho navstevnika
            newVisitor = visitor.nextVisitor(edge,node)
            if newVisitor==None: continue
            visitors.append(newVisitor)
        visitors.remove(visitor)
```

SEGMENT 5.2.3

```
    # pocet useku v ramci intervalu
    granularity = 20.0

    # iterace vsemi navstevniky
    for visitor in node.visitors:
        time = visitor.time
        # casovy krok
        step = visitor.headway/granularity if visitor.headway>0 else 0.2
        headway = visitor.headway if visitor.headway>0 else step
        headwayTime = step/2

        probTotal = 0
```

```
    # postupne nacitani casu
    while headwayTime<headway:
        # pravdepodobnost, ze vyuzijeme danou variantu
        probForTime = countProb(headwayTime, node, visitor)
        if probForTime <= -1: break
        # postupne nacitani pravdepodobnosti, tedy vlastne samotna integrace
        probTotal += probForTime
        headwayTime += step
    if probTotal == 0:
        continue
    # samotne spocteni pravdepodobnosti
    visitorProbMap[visitor] = probTotal*step/headway
```

SEGMENT 5.3

```
# prochazeni všech uzlu site
for node in nodes:
    eligible = False
    for edge in node.edges:
        # vraci se pouze uzly na ulicni siti
        if edge.edgeType == 'STREET' or edge.edgeType == 'STREET_SPLIT':
            eligible = True
            break
    if eligible == False: continue
    # spocteni pravdepodobnosti pro visitory v uzlu
    probMap = getVisitorsProbabilities(node)
    result['geom'] = dumps(node.geometry)
    time = 0
    probTotal = 0
    # vazeny prumer casu, vaha=pravdepodobnost
    for visitor in probMap.keys():
        time = time+visitor.time*probMap[visitor]
        probTotal += probMap[visitor]

    result['time'] = (time/probTotal) if probTotal>0 else 0
```

SEGMENT 5.4

```
# ziskani pravdepodobnosti variant na vystupnim uzlu
probMap = getVisitorsProbabilities(finishNode)
# jejich prochazeni
for visitor in probMap.keys():
    result = dict()
    result['time']=visitor.time
    result['prob']=probMap[visitor]
    geoms = []
    segments = []
    time = 0
    # prochazeni všech hran daneho navstevnika a vytvareni segmentu
    for edge in visitor.edges:

        if edge.edgeType == 'LINE' or edge.edgeType == 'LINE_SHARED':
            lineText = ''
            for ls in edge.geometry:
                geoms.append(ls.coords)
            if edge.commonSegmentRef==None:
                lineText = edge.lineSegmentRef.line.name
            else:
                for line in edge.commonSegmentRef:
                    lineText += line.name+','
                lineText = lineText[:-1]
            time = time + edge.time

        elif edge.edgeType == 'LOAD':
            # pouze prestup
            if time!=0:
                segment = dict()
                segment['name'] = 'CHUZE'
                segment['time'] = time
                segments.append(segment)
            segment = dict()
            segment['name'] = 'CEKANI'
            segment['time'] = visitor.edge2Time[edge]
            segments.append(segment)
            time = 0
            station1 = edge.node1.station.name if edge.node1.station else edge.node2.station.name
        elif edge.edgeType == 'UNLOAD':
            station2 = edge.node1.station.name if edge.node1.station else edge.node2.station.name
            segment = dict()
            segment['name'] = lineText
            lineText = ''
            segment['time'] = time
            segment['station1'] = station1
            segment['station2'] = station2
            segments.append(segment)
            time = 0
        else:
            geoms.append(edge.geometry.coords)
            time = time + edge.time
    # vytvoreni posledniho segmentu
    segment = dict()
    segment['name'] = 'CHUZE'
    segment['time'] = time
    segments.append(segment)
```

```
        result['segments'] = segments
        result['geoms'] = dumps(MultiLineString(geoms))
        results.append(result)
```