ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra kybernetiky

Bakalářská práce

Inteligentní směrování dopravy v městských silničních sítích

Matyáš Švadlenka

Otevřená informatika – Základy umělé inteligence a počítačových věd

Leden 2023

Vedoucí práce: doc. RNDr. Lukáš Chrpa, Ph.D.

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Švadlenka** Jméno: **Matyáš** Osobní číslo: **492315**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Otevřená informatika**

Specializace: **Základy umělé inteligence a počítačových věd**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Inteligentní směrování dopravy v městských silničních sítích

Název bakalářské práce anglicky:

Intelligent Traffic Routing in Urban Road Networks

Pokyny pro vypracování:

1. Seznámit se se simulátorem dopravy SUMO, jazykem pro representaci plánovacích problému PDDL a obecnými doménově-nezávislými plánovači.
2. Navrhnout a implementovat architekturu, která propojí simulátor dopravy SUMO s plánovací komponentou, která ma za úkol směrovat dopravu tak, aby se minimalizovala možnost zácpy v silniční síti. Architektura má mít následující funkcionalitu:
- převod SUMO representace silničních sítí (v XML) do jazyka PDDL. Representace v jazyce PDDL by měla zahrnovat jen relevantní silniční segmenty, které mohou být využity pro směrování dopravy,
- interpretace plánů, které poskytují informace o cestách jednotlivých vozidel, v simulátoru SUMO,
- využití modulu TraCI (součást SUMO) pro kontinuální plánování (směrování) a simulaci dopravy pro delší časové úseky.
3. Vyhodnotit plánovací techniky využité pro směrování dopravy na několika mapách (cca 4) a několika dopravních scénářích (např. dopravní špička) a porovnat je z baseline technikou směrování po nejkratší (či nejrychlejší) cestě.

Seznam doporučené literatury:

[1] Pablo Álvarez López, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, Evamarie WieBner: Microscopic Traffic Simulation using SUMO. ITSC 2018: 2575-2582
[2] Maria Fox, Derek Long: PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. Journal of Artificial Intelligence Research 20:61-124 (2003)
[3] Lukás Chrpa, Mauro Vallati, Simon Parkinson: Exploiting automated planning for efficient centralized vehicle routing and mitigating congestion in urban road networks. SAC 2019: 191-194
[4] Lukás Chrpa, Daniele Magazzeni, Keith McCabe, Thomas Leo McCluskey, Mauro Vallati: Automated planning for Urban traffic control: Strategic vehicle routing to respect air quality limitations. Intelligenza Artificiale 10(2): 113-128 (2016)

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. RNDr. Lukáš Chrpa, Ph.D. optimalizace CIIRC

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **24.01.2022** Termín odevzdání bakalářské práce: **10.01.2023**

Platnost zadání bakalářské práce: **30.09.2023**

doc. RNDr. Lukáš Chrpa, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Poděkování / Prohlášení

Chtěl bych poděkovat své rodině, svým spolužákům a kamarádům za neustálou podporu během mého studia. Na závěr bych chtěl poděkovat svému školiteli doc. RNDr. Lukáš Chrpa, Ph.D. za jeho trpělivost a podporu.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 10. 1. 2023

.....

Abstrakt / Abstract

Tato práce se zabývá problémem inteligentního směrování dopravy. Jednou z možností jak tento problém řešit je pomocí technik automatického plánování, ale kvůli složitosti problému (velikosti silničních sítí) není možné získat řešení dostatečně rychle na použití v reálném světě. Z toho důvodu byly nejprve vyvinuty a vyhodnoceny techniky abstrakce silničních sítí, umožňující použití technik automatického plánování, které byly na závěr vyhodnoceny na několika dopravních scénářích ve známém simulátoru dopravy SUMO. Součástí práce je architektura vyvinuta za cílem spojení různých komponent této práce a poskytnout prostředí umožňující jednoduše testovat techniky a vytvářet dopravní scénáře.

Klíčová slova: Automatické plánování, Teorie grafů, SUMO, OpenStreetMap, Python.

This work focuses on the problem of intelligent vehicle routing. One of the options how to solve this problem is with techniques of automated planning, but because of the problem's complexity (size of road networks), its not possible to generate solution fast enough in real world scenario. For this reason abstraction of road networks was developed, which substantially simplifies them, allowing the use of automated plannings techniques, that were at the end evaluated in several traffic scenarios, simulated in well known simulator SUMO. Included in this work is architecture (code base) which was developed in order to connect different parts of this work and provide easy to use environment for development and testing of techniques, algorithms, traffic scenarios.

Keywords: Automated planning, Graph theory, SUMO, OpenStreetMap, Python.

Title translation: Intelligent traffic routing in urban road networks

/ Obsah

Tabulky / Obrázky

Kapitola 1
Úvod

Problém inteligentního řízení dopravy nabývá postupem času na důležitosti, převážně kvůli narůstajícímu počtu aut, a jeho důsledkům, které můžeme pozorovat jak v ekonomickém sektoru, tak na životním prostředí. Jedná se zejména o dopravní zácpy, vznikající téměř pravidelně ve větších městech během dopravní špičky. Zabránění vzniku dopravních zácp není vždy možné, ale ve většině případech lze distribuovat dopravu tak, aby byl snížen dopad, který mají na provoz silničních sítí. Obecně lze tento problém popsat jako hledání cest, které minimalizují čas jízdy všech vozidel, v závislosti na současné situaci. Hledání cest pro individuální vozidla, jak dělá moderní navigační systém, není dostačující, jelikož se jedná o dynamický systém a změna cesty jednoho vozidla má vliv na všechna ostatní.

Pro řešení tohoto problému lze uplatnit Automatické plánování [1], jedná se o větev umělé inteligence, která se obecně zaměřuje na hledání strategie pro agenty (bezpilotní vozidla) v komplikovaných prostředí. Velmi často se Automatické plánování používá (včetně této práce) pro nalezení sub-optimálních řešení, z důvodu kompromisu mezi kvalitou řešení a výpočetním časem. Jeho uplatnění je například v Hubbleově vesmírném dalekohledu [2], nebo ve vozidlech prozkoumávající Mars [3].

Důležitou roli ve složitosti problému hraje, kromě počtu vozidel, také velikost silniční sítě. Z tohoto důvodu byly vyvinuty a vyhodnoceny techniky zjednodušení map silničních sítí, na kterých bude možné použít Automatické plánování v dostatečně rychlém čase (řádově desítky sekund), aby bylo možné včasně reagovat na aktuální situace v reálném světě.

Následně bylo z OpenStreetMap použito několik silničních map (pro jednoduchost je z map extrahována pouze silniční síť) společně se Simulátorem SUMO (Simulation of Urban Mobility) [4], který umožňuje tyto mapy reprezentovat a simulovat na nich dopravu. Takto vytvořené dopravní scénáře byly na závěr použity pro vyhodnocení technik Automatického plánování za různých dopravních podmínek.

Hlavním přínosem této práce je architektura, která propojuje plánovací komponentu se simulátorem SUMO, dále umožňuje převádět mapy silničních sítí stažených z OpenStreetMap (formátu *.osm*) do simulátoru SUMO (formátu *net.xml*), vytvářet podgrafy silničních sítí, generovat dopravní scénáře a na ně aplikovat techniky automatického plánování. Architektura byla vyvinuta s důrazem na rychlé a jednoduché testování technik, které závisí na předem nastavených parametrech, stejně tak generování dopravních scénářů, jelikož je nutné testovat za mnoha různých dopravních podmínek. Výhodou architektury je její přizpůsobitelnost, jednotlivé komponenty lze použít samostatně a je tedy možné vyvíjet nové algoritmy, nebo měnit stávající. Dále by kladen důraz na dlouhodobý vývoj a poskytnutí prostředí, které je jednoduché použít, a zároveň si ho může uživatel jakkoliv modifikovat a přizpůsobit vlastním potřebám.

Kapitola 2
Simulátor SUMO

Simulátory dopravy jsou používány pro modelování reálných situací příliš složitých na popsání analytickými způsoby za účelem získání informací, které mohou pomoci ke zlepšení dopravní sítě. Obecně se dělí do čtyř skupin, SUMO patří do skupiny mikroskopických simulátorů, tedy je více soustředěn na chování individuálních objektů v simulaci, které modeluje každé zvlášť. Mezi tyto objekty patří například vozidla pohybující se na silničních sítích, ale také vlaky, chodci, semafory atd. Simulace je vytvořená z několika souborů: silniční síť (*net.xml*), soubor definující vozidla a jejich cesty (*rou.xml*) a soubor (*sumocfg.xml*), který simulací spustí a zahájí vizualizaci, tyto soubory dohromady definují dopravní scénář. Vizualizace simulace probíhá v grafickém prostředí sumo-gui, ve kterém je možné běh simulace zastavit, nebo nastavit rychlost jejího průběhu.

Mezi hlavní výhody SUMO patří přenositelnost (je možné použit simulátor i v operačním systému Linux nebo s mnoha různými typy souborů definující silniční síť), open-source zdrojový kód, a především spousta knihoven a programů, které umožňují uživatelům přizpůsobit simulátor vlastním potřebám. Mezi tyto programy patří program *netconvert*, který konvertuje různé typy souborů definující silniční síť, do nového souboru, použitelného pro tvorbu simulace. Program *netedit* umožňuje vizualizovat silniční síť ještě před samotnou simulací, ale především ji v něm lze měnit, či vytvářet. Mezi významné knihovny patří *TraCI*, která dovoluje propojit uživatelem vytvořený program s běžící simulací, což dovoluje objekty během simulace měnit, a také lze vytvářet vlastní objekty, jako například adaptivní semafory a jiné.

2.1 Načítání map silničních sítí

OpenStreetMap poskytuje on-line grafické prostředí, ve kterém je možné navštívit jakékoliv místo na Zemi a zvolit pohled pouze na dopravní mapu. Poté je nutné nastavit obdélníkovou oblast definující zeměpisnou délku a šířku, z které bude stažen soubor typu *osm* obsahující silniční síť. Pro stejný účel poskytuje simulátor SUMO program *OSMWebWizard*, který je schopen z obdobně zvolené oblasti vytvořit simulaci. Oba způsoby v souboru ponechají mnohem více informací než samotnou silniční síť, např.: řeky, parky, geometrii budov, železniční sítě apod. Pro extrahování pouze silniční sítě z map a vytvoření souboru, který je schopen simulátor SUMO spustit, byla vytvořena třída Converter.

První částí se zabývá metoda *osm_filter*, používající program osmfilter, který poskytuje OpenStreetMap. Osmfilter umí ponechat chtěné objekty a odstranit ze souboru ostatní, příkaz použit v programu je

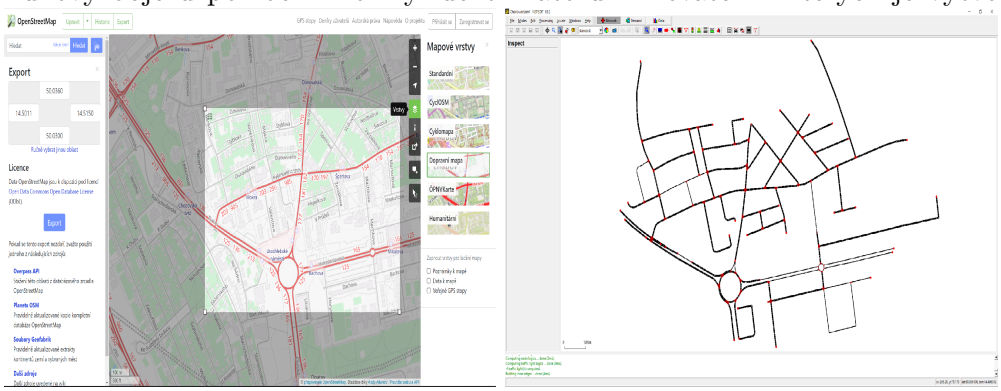
```
--keep-ways="highway=primary =tertiary =residential
=primary_link =secondary =secondary_link =trunk =trunk_link =motorway
=motorway_link" --keep-nodes= --keep-relations
```

Po provedení příkazu v souboru zůstanou pouze objekty, které definují silniční síť, tento soubor pak bude uložen pod stejným jménem se sufixem "_filtered".

Druhou část vykonává metoda *net_convert*, pomocí již zmíněného programu *netconvert*. Pracuje již na vyfiltrovaném souboru z první části, použitím příkazu
`--geometry.remove --ramps.guess --junctions.join --roundabouts.guess`
`--remove-edges.isolated --keep-edges.components 1`
`--numerical-ids.node-start 0 --numerical-ids.edge-start 0`
zajistí odstranění přebytečných objektů, spojení blízkých křižovatek, vytvoření vstupů a výstupů z dálnice, určení kruhových objezdů a ponechání největší komponenty grafu silniční sítě, poslední řádka zaručí, aby silnice a křižovatky měli číselné identifikátory počínaje od 0 (jinak jsou identifikátory dlouhé kombinace čísel a písmen).

2.2 Reprezentace silniční sítě

SUMO reprezentuje silniční síť jako orientovaný graf, kde hrany jsou silnice a vrcholy jsou křižovatky. Tento orientovaný graf je uložen v souboru s příponou *net.xml*, jedná se o klasický formát *xml*, ve kterém mají všechny objekty ”tag”, například pro silnice se jedná o tag ⟨*edge*⟩ a pro křižovatky ⟨*junction*⟩ a unikátní identifikátor. Navíc má každý objekt tvořící silniční síť několik atributů, mezi které patří: atributy ”x” a ”y” udávající pozici, tvar reprezentující tento objekt a další. Spojení mezi křižovatky je definováno pomocí atributů ”from” a ”to”, jejíž hodnota odpovídá identifikátoru propojených křižovatek. Ke konci souboru jsou popsané spojení mezi individuálními silnicemi tagem ⟨*connection*⟩, obdobně jako u křižovatek, navíc jsou ovšem atributy popisující přechod mezi silničními pruhy. Na úplném konci se může nacházet tag ⟨*roundabout*⟩ popisující kruhový objezd pomocí množiny identifikátorů křižovatek z kterých je vytvořen.



Obrázek 2.1. Fotka silniční sítě mapy Chodov v OpenStreetMap (nalevo) s určenou obdélníkovou oblastí, na pravé straně je její reprezentace v programu netedit.

2.3 Vozidla a jejich cesty

Jako v předchozí kapitole jsou vozidla a jejich cesty reprezentované ve formátu *xml*, konkrétně *rou.xml*. Na začátku souboru se může nacházet definice typů vozidel (popřípadě mohou být ve vlastním souboru) pomocí tagu ⟨*vType*⟩ s atributy udávající maximální rychlost, délku vozidla, minimální bezpečnou vzdáleností mezi auty a další, pro jednoduchost byl zvolen průměrný model osobních vozidel: ⟨*vType length=”5.0” minGap=”2.5” maxSpeed=”50.0”*⟩.

Dále se v souboru nachází jednotlivé cesty vozidel, určené tagem ⟨*route*⟩ s parametrem `edges` udávající množinu identifikátorů silnic, po kterých auto pojede a pro větší přehlednost byly přidány parametry `fromJunction`, `toJunction` mapující identifikátory počáteční a koncové křižovatky.

Na konci souboru jsou vozidla definována tagem ⟨*vehicle*⟩ s parametry určující příjezd vozidla do simulace (vozidla musí být seřazena v souboru podle tohoto parametru), identifikátor cesty vozidla, po které pojede, typ vozidla a příchozí pruh při vstupu do silniční sítě.

```
<?xml version='1.0' encoding='utf-8'?>
<routes>
    <vType accel="3.0" decel="6.0" id="CarDefault" length="5.0" minGap="2.5" maxSpeed="50.0" sigma="0.5" />
    <route id="sr460" edges="62 129 61 6 5 161 162 361 163 164 38 134 116 115 114 113 112 74 73 72 71 70 69 68" fromJunction="154" toJunction="42" />
    <route id="sr461" edges="135 56 111 110 335 336 337 338 63 339 95 251 364 365 366 367 368 348 141 431 255" fromJunction="117" toJunction="154" />
    <route id="sr462" edges="226 227 228 116 359 300 301 124 30 29 397 398 399 400 401 237 0" fromJunction="127" toJunction="117" />
    <route id="sr463" edges="308 305 306 307 155 198 199 200 247 366 367 368 348 141 431 255" fromJunction="114" toJunction="154" />
    <route id="sr464" edges="279 280 281 282 283 284 285 394 395 396 28 27 26 25 50" fromJunction="42" toJunction="11" />
    <vehicle id="v0" depart="0.0" route="sr460" type="CarDefault" departLane="random" />
    <vehicle id="v904" depart="0.0" route="sr461" type="CarDefault" departLane="random" />
    <vehicle id="v1700" depart="0.0" route="sr462" type="CarDefault" departLane="random" />
    <vehicle id="v2408" depart="0.0" route="sr463" type="CarDefault" departLane="random" />
    <vehicle id="v3776" depart="0.0" route="sr464" type="CarDefault" departLane="random" />
    <vehicle id="v905" depart="4.53" route="sr461" type="CarDefault" departLane="random" />
    <vehicle id="v2409" depart="5.26" route="sr463" type="CarDefault" departLane="random" />
    <vehicle id="v3777" depart="6.7" route="sr464" type="CarDefault" departLane="random" />
    <vehicle id="v1" depart="8.82" route="sr460" type="CarDefault" departLane="random" />
    <vehicle id="v906" depart="9.06" route="sr461" type="CarDefault" departLane="random" />
    <vehicle id="v1701" depart="10.26" route="sr462" type="CarDefault" departLane="random" />
    <vehicle id="v2410" depart="10.53" route="sr463" type="CarDefault" departLane="random" />
    <vehicle id="v3778" depart="13.41" route="sr464" type="CarDefault" departLane="random" />
    <vehicle id="v907" depart="13.58" route="sr461" type="CarDefault" departLane="random" />
    <vehicle id="v2411" depart="15.79" route="sr463" type="CarDefault" departLane="random" />
    <vehicle id="v2" depart="17.65" route="sr460" type="CarDefault" departLane="random" />
    <vehicle id="v908" depart="18.11" route="sr461" type="CarDefault" departLane="random" />
    ...
</routes>
```

Obrázek 2.2. Fotka (zkráceného) souboru obsahující vozidla a jejich cesty.

Kapitola 3
Automatické Plánování

Automatické plánování se snaží v zadaném problému najít takovou sekvenci akcí, jejíž aplikace transformuje iniciální stav problému na stav cílový. Obecně se dá říci, že se jedná o prohledávání stavového prostorou přechodových grafů, toto lze řešit pomocí Dijkstrova algoritmu v čase $\Theta(|V|\log(|V|+|E|))$, kde V je počet možných stavů a E počet akcí, ale V může i v jednoduchých problémech dosahovat hodnot vyšších než 10^{20}, z toho důvodu se používají algoritmy řízené heuristikami (například A*). Existuje několik typů automatického plánování, mezi které patří: klasické, temporální, pravděpodobností, podmíněné a další. Tato práce používá klasické plánování, které se zaměřuje na efektivní prohledávání prostoru (použití správných heuristik a algoritmů) a na takovou reprezentaci akcí a stavů, aby je nebylo nutné vyčíslit, díky čemuž je zvýšena obecnost problémů. Klasické plánování je obecně myšleno v rámci omezeného a přechodového stavového systému, tedy systému, který je: deterministický, konečný (počet stavů, akcí, objektů, apod.) a plně pozorovatelný (všichni agenti mají přístup k celému stavovému prostoru).

3.1 Akční jazyky

Akční jazyky popisují a formalizují přechodové systémy za účelem definování akcí a jejich efektů v těchto systémech. Existují dvě třídy akčních jazyků, akční popisující jazyky (ADL) a akční dotazující jazyky (AQL). Akční popisující jazyky rozvíjí jazyk STRIPS, dosahují vyšší expresivity a dovolují použití podmínkových operátorů. PDDL patří mezi akční popisující jazyky, jedná se o jeden z nejznámějších a nejpoužívanějších jazyků, byl vyvinut za cílem standardizace akčních jazyků, je založen na predikátové logice (více kompaktní než-li výroková logika) a tím byla umožněna Mezinárodní Plánovací Soutěž (IPC), s kterou se dále vyvíjí (dnes PDDL 3.1). Se zavedením PDDL jazyka také přišlo rozdělení plánovacího modelu na doménové a problémové soubory. Doménové soubory jsou abstraktním zápisem problémových souborů a problémové soubory jsou aplikací doménových na konkretní situaci.

Jak již dříve zmíněno, problémy musí být formálně popsány, k tomuto slouží STRIPS formalismus. Jedná se o nejznámější formalismus plánování, je to zejména z důvodu, že to je nejjednodušší formalismus dosahující dostatečné expresivity pro mnoho různých problémů. Problém popsán ve STRIPS formalismu je čtveřice $\langle P,\ A,\ I,\ G\rangle$, kde: P je konečný set atomů (predikátů), A je konečný set akcí, I a G popisují iniciální a cílový stav. Každá akce a je trojice (pre_a, add_a, del_a), pre_a je předpoklad (precondition) akce a, který definuje podmínky nutné pro umožnění vykonání akce, add_a popisuje změnu stavu a del_a odstraňuje předchozí stav. Všechny akce a mají definovanou cenu pomocí funkce $c(a)$ (je možné mít akce s cenou 0), toto je zavedeno, aby bylo možné hodnotit kvalitu výsledků a zavést možnost penalizace různých akcí.

Konkrétně v problému inteligentního řízení dopravy je ve STRIPS formalismu silniční síť přechodovým grafem, agenti, kteří interagují s prostředím, jsou vozidla a akce reprezentují přechod vozidel mezi jednotlivými křižovatkami (uzly) po silnicích (hranách).

Výsledná sekvence těchto akcí dostane vozidla ze vstupní křižovatky (iniciálního stavu) do cílové křižovatky (cílový stav). Cena akcí je zavedena formou penalizace vykonání každé akce, tato penalizace se odvíjí od současné kapacity silnice a má za cíl minimalizovat vznik dopravních zácp, z toho důvodu lze problém chápat jako minimalizace této penalizace.

3.2 Popis problému

Silniční síť je pomocí formalismu STRIPS definována následovně: všechny křižovatky patří do skupiny objektů *junction*, jednotlivá jména křižovatek odpovídají identifikátorům, které mají v souborech definující silniční síť, navíc je přidán prefix "j" (nelze definovat proměnné pouze pomocí čísel). Namísto silnice jsou pro spojení mezi křižovatkami definované cesty (skupina *route*), toto je z důvodu, že silniční síť je zjednodušená metodami popsanými v Kapitole 5.2 a tedy dochází k sjednocení několika silnic. Ze stejného důvodu je přidán predikát *allowed*, který má jako argumenty současnou cestu vozidla a jeho cílovou křižovatku. Predikát *allowed* je pravdivý v případě, že vozidlo může použít cestu pro dosažení cílové křižovatky, je použit, protože graf silniční sítě je vytvořen ze spojení několika podgrafů, a proto je nutné odstranit nově vzniklé cesty při spojení (nemusejí vést do cílové křižovatky vozidla). Samotné spojení mezi cestami a křižovatkami je definované predikátem *connected*, který má jako argumenty počáteční křižovatku, cestu a cílovou křižovatku.

Všechny cesty mají definovanou kapacitu, jejíž maximální hodnota je vypočtena pomocí vzorce:

$$\frac{length(route)}{length(vehicle) + minGap} * lanes,$$

kde čitatel odpovídá délce cesty a ve jmenovateli je sečtena délka vozidla s minimální bezpečnou vzdáleností (definována v Kapitole 2.3), nakonec je celý výraz vynásoben počtem silničních pruhů (minimální počet ze všech jednotlivých silnic tvořící cestu). Dále jsou pro cesty definovány kategorie zaplnění maximální kapacity, jsou to popořadě nízká (pod 35%), střední (35% až 75%) a vysoká (75% až 100%), zaplněná je v případě přesažení maximální kapacity. Jak již bylo v úvodu této kapitoly zmíněno, pro zvýšení současné kapacity cesty při příjezdu vozidla není možné zvyšovat číselné proměnné v klasickém plánování, z toho důvodu jsou zavedeny predikáty *using* a *use*. Predikát *use* je použit pro počítání vozidel na cestě, musí být pro každou cestu definován počínaje od 0 až po maximální kapacitu, rozdělen procentuálně podle kategorií zaplnění (například *use*0 až *use*10 pro nízkou, atd.), a predikát *using* je použit pro zvýšení současné zaplněnosti, například pro zvýšení kapacity z 0 na 1 je použit následovně: (*notusinguse*0) a (*usinguse*1) způsobí, že *use*0 bude nastaven na negativní booleovskou hodnotu (0) a *use*1 na pravdivou (1).

Pro vozidla jsou zavedeny čtyři akce pohybu silnic: *drive-to-light*, *drive-to-medium*, *drive-to-heavy*, *drive-to-congested*, odpovídající kategoriím zaplnění maximální kapacity. Parametry těchto akcí jsou: vozidlo, současná křižovatka, cesta (kterou chce vozidlo použít), následující křižovatka, cílová křižovatka vozidla a dva predikáty *use*. Předpoklady těchto akcí jsou vždy stejné, nejprve se zkontroluje jestliže cílová křižovatka vozidla je správná, poté zda kapacita cesty odpovídá prvnímu predikátu *use*, jestli druhý predikát *use* následuje po prvním, zdali kategorie kapacity po zvýšení (druhý predikát *use*) odpovídá současné akci, vyhodnocení predikátu *allowed*, kontrola pozice auta a zda cesta vozidla spojuje současnou křižovatku a následující (predikát *connected*). Pokud se všechny předpoklady akce vyhodnotí na pravdivé, nastane efekt akce, první predikát

use je nastaven na nepravdivou hodnotu, druhý na pravdivou, vozidlo se posune z originální křižovatky na následující (add_a, del_a) a na závěr je zvýšena hodnota penalizace. Penalizace je vypočtena na základě délky silnice, ta je definována pro všechny kategorie (až na zaplněnou, pro tu je nastavena penalizace na 100000), pro nízkou je nastavena na skutečnou délku cesty, pro střední se jedná o desetinásobek délky a pro vysokou o stonásobek.

Problémový soubor začíná definicí jména problému, poté jména domény (musí odpovídat jménu doménového souboru), pak jsou vypsány všechny objekty a jejich proměnné (vozidla, křižovatky, cesty, predikát *use*). V iniciálním stavu je nastavena počáteční hodnota penalizace na 0, pak následuje definice silniční sítě společně s kapacitou a délkou cest včetně predikátu *use*. Na konci iniciální stavu se nachází počáteční pozice vozidel, po iniciálním stavu následuje cílový stav, v kterém jsou definovány cílové křižovatky vozidel. Na konci souboru se nachází metrika minimalizace penalizace.

Obdobně je popsán doménový soubor, počínaje jménem domény, následují objektové skupiny a následně jsou vypsány všechny predikáty, stejně tak jsou vypsány názvy a argumenty všech funkcí (*total-cost* odpovídající současné penalizaci a funkce délky, která vrací hodnotu penalizace cest). Na konci souboru jsou definované akce a jejich předpoklady a efekty.

```
(define
(problem example)
(:domain utc_allowed)
(:objects
j102 ... j440 - junction
r42 ... r997 - road
use0 ... use100 - use
v93 v1125 v0 v2240 v94 v1126 v2995 v95 v2241 v1127 v2996 v96 v1128 v2242 v97
v2243 v2997 v98 v1130 v2244 v1131 v99 v2998 v1132 v1031 v100 v2245 v2133 v2999 - car
)
(:init
(= (total-cost) 0)
(connected j432 r1002 j333)
...
(allowed r1012 j358)
...
(= (length-light r1004) 5)
(= (length-medium r1004) 53)
(= (length-heavy r1004) 532)
(using r1004 use0)
(light r1004 use0)
...
(medium r1004 use5)
...
(heavy r1004 use10)
(cap r1004 use10)
...
(next use0 use1)
...
(next use99 use100)
(at v93 j87)
(togo v93 j358)
(at v1125 j241)
(togo v1125 j316)
...
(:goal (and
(at v93 j358)
(at v1125 j316)
...
))
(:metric minimize (total-cost))
)
```

Obrázek 3.1. Problémový soubor (výrazně zkrácen pro přehlednost).

```
(define (domain utc_allowed)
    (:requirements :typing)

    (:types road junction car use)

    (:predicates
        (togo ?c - car ?dest - junction)
        (connected ?junction1 - junction ?road - road ?junction2 - junction)
        (at ?car - car ?junction - junction)
        (next ?x ?y - use)
        (light ?r - road ?u - use)
        (medium ?r - road ?u - use)
        (heavy ?r - road ?u - use)
        (cap ?r - road ?u - use)
        (using ?r - road ?u - use)
        (allowed ?r - road ?dest - junction)
    )

    (:functions
        (length-light ?road - road) ; time taking to drive to the road in light traffic
        (length-medium ?road - road) ; time taking to drive to the road in medium traffic
        (length-heavy ?road - road) ; time taking to drive to the road in heavy traffic
        (total-cost)
    )


    (:action DRIVE-TO-light
        :parameters (?c - car ?j1 - junction ?r - road ?j2 - junction ?d - junction ?u1 ?u2 - use)
            :precondition (and (togo ?c ?d)
                (using ?r ?u1)
                (next ?u1 ?u2)
                (light ?r ?u2)
                (allowed ?r ?d)
                (at ?c ?j1)
                (connected ?j1 ?r ?j2)
            )

    :effect (and (not (using ?r ?u1))
                (using ?r ?u2)
                (not (at ?c ?j1))
                (at ?c ?j2)
                (increase (total-cost) (length-light ?r))
        )
    )
```

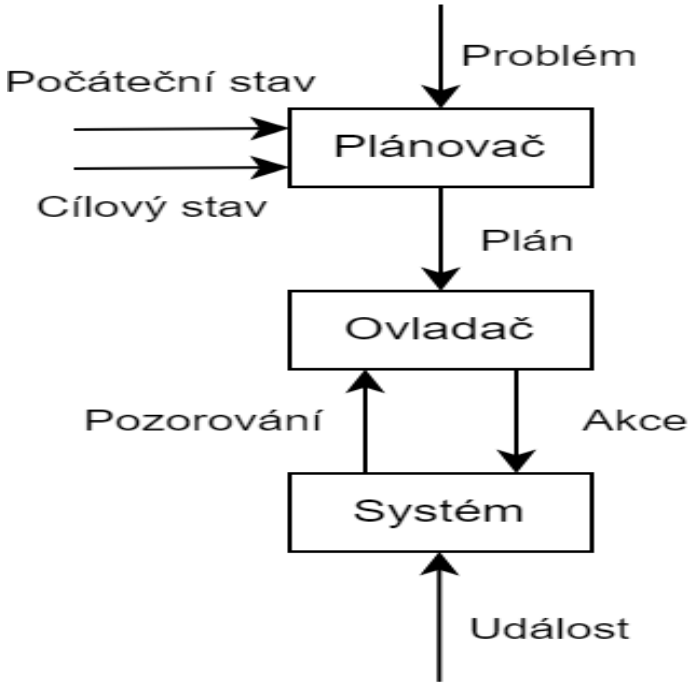
Obrázek 3.2. Doménový soubor použit v této práci (chybějící akce vozidel na obrázku jsou definovány obdobě). Závorky kolem výrazů jsou typickou vlastností PDDL, je to zejména z parsovacích důvodů.

3.3 Práce automatického plánování

V Obrázku 3.3 je ukázána práce automatického plánování, nejprve je potřeba vytvořit problémový soubor reprezentující současnou situaci na silniční síti, ten je pak spolu s doménovým souborem předán plánovači, tedy programu, který umí řešit problémy automatického plánování, v této práci byl zvolen plánovač Mercury 3.4. Jakmile máme tyto dva soubory je možné použít plánovač, který pomocí vhodných algoritmů najde plán (řešení), z tohoto řešení jsou jednotlivé akce pohybů vozidel poté pomocí vyvinuté architektury (ovladač) přeloženy na úplné cesty pro jednotlivá vozidla a zaslány simulátoru SUMO (systém).

Tento postup je opakován periodicky až do konce simulace, v této práci byla rozdělena doprava do 30-sekundových intervalů, pro které pak byly jednotlivě vytvořeny problémové soubory. Předpokládá se, že vozidla používají navigační systém, který zasílá polohu vozidla a jeho destinace je dopředu známa, stejně tak je pomocí tohoto systému vozidlo schopno přijmout novou trasu. V problémovém soubory jsou všechna vozidla v periodě chápána jako vstupující do sítě v jeden moment (stejný čas), silniční síť je ze začátku pro jednoduchost prázdná v každém problémovém souboru (nutnost simulace budoucích poloh vozidel). Plánovači je poté nastaven časový limit na 25 sekund (časová rezerva simulující prodlevu komunikace mezi vozidly a systémem), v případě že nebyla nalezena nová cesta pro některé vozidlo, bude použita původní (nejkratší).

Rozlišujeme mezi dvěma způsoby plánování: on-line a off-line, v případě on-line plánování je simulace vždy pozastavena během běhu plánovače a jakmile je vygenerováno řešení jsou vozidlům předány trasy pomocí knihovny *TraCI* a simulace pokračuje (toto by odpovídalo použití v reálném světě). V druhém způsobu, jsou z původního souboru vozidel a jejich tras extrahovány informace a pomocí nich je vytvořena nová simulace spolu se souborem obsahující nové trasy vozidel, není tedy použit simulátor SUMO a simulaci je možné spustit bez přerušování (zde je výhoda použití multiprocessingu, tedy dosáhnutí paralelního běhu několika plánovačů).



Obrázek 3.3. Jednoduchý koncepční model ukazující práci automatického plánování [5].

3.4 Mercury

V automatickém plánování existují dva typy plánovačů: doménově závislé, které potřebují k řešení problémů doménový soubor, oproti ním doménově nezávislé plánovače nepotřebují doménové soubory a jsou schopny řešit větší skupinu problémů (kompromis mezi expresivitou a obecností za výkon). Plánovač Mercury [6] je doménově závislý, sequential satisficing (postupně zlepšuje řešení, tedy nehledá pouze optimální), implementován v systému Fast Dawnward (systém založený na heuristickém prohledávání), v této práci je použita verze *sequential agile* (soustředěna na nalezení výsledku v co nejkratším čase).

Práce Mercury je rozdělena do tří kroků, nejprve začne problémový soubor parsovat (kontrola správnosti souboru, zjednodušování axiomů, ..), v druhém kroku je postaven graf (stavového prostoru) a alokována paměť, v posledním kroku dojde k aplikací prohledávacích algoritmů, jakmile je nalezeno řešení, vytvoří se soubor (s příponou *1*, jelikož se jedná o první řešení), do kterého je řešení uloženo, jedná se o seřazena sekvence akcí vozidel. Po nalezení prvního řešení prohledávání pokračuje (pokud je dostatek času), tentokrát jsou v algoritmu upraveny parametry (hloubka prohledávání a další) a plánovač se snaží zlepšit původní řešení, pokud se mu tak povede, vytvoří se další výsledkový soubor s inkrementovanou příponou. Po dokončení běhu jsou výsledkové soubory (plány) interpretovány a sekvence akcí jsou přeloženy na plné cesty vozidel, v případě existence více výsledkových souborů dostane vozidlo cestu ze souboru s nejvyšší příponou (nejlepšího výsledku). Kromě výsledkových souborů generuje Mercury další, jedná se o loggovací soubory, soubory měřící čas a další, tyto jsou standardně smazány (je možné změnit nastavení a soubory nemazat).

```
1  (drive-to-light v125 j87 r226 j407 j358 use0 use1)
2  (drive-to-light v125 j407 r413 j410 j358 use0 use1)
3  (drive-to-light v125 j410 r432 j104 j358 use0 use1)
4  (drive-to-light v125 j104 r115 j92 j358 use0 use1)
5  (drive-to-light v125 j92 r114 j109 j358 use0 use1)
6  (drive-to-light v125 j109 r457 j102 j358 use0 use1)
7  (drive-to-light v125 j102 r323 j371 j358 use0 use1)
8  (drive-to-light v125 j371 r774 j423 j358 use0 use1)
9  (drive-to-light v125 j423 r892 j271 j358 use0 use1)
10 (drive-to-light v125 j271 r893 j424 j358 use0 use1)
11 (drive-to-light v125 j424 r13 j166 j358 use0 use1)
12 (drive-to-light v125 j166 r229 j171 j358 use0 use1)
13 (drive-to-light v125 j171 r394 j436 j358 use0 use1)
14 (drive-to-light v125 j436 r401 j358 j358 use0 use1)
15 (drive-to-light v126 j87 r226 j407 j358 use1 use2)
16 (drive-to-light v126 j407 r413 j410 j358 use1 use2)
17 (drive-to-light v126 j410 r432 j104 j358 use1 use2)
18 (drive-to-light v126 j104 r115 j92 j358 use1 use2)
19 (drive-to-light v126 j92 r114 j109 j358 use1 use2)
20 ...
```

Obrázek 3.4. Výsledkový soubor plánovače (zkrácen pro přehlednost), sekvence akcí jsou seřazeny podle jejich času vykonání, pro získání celé cesty je nutné spojit všechny akce pro dané vozidlo.

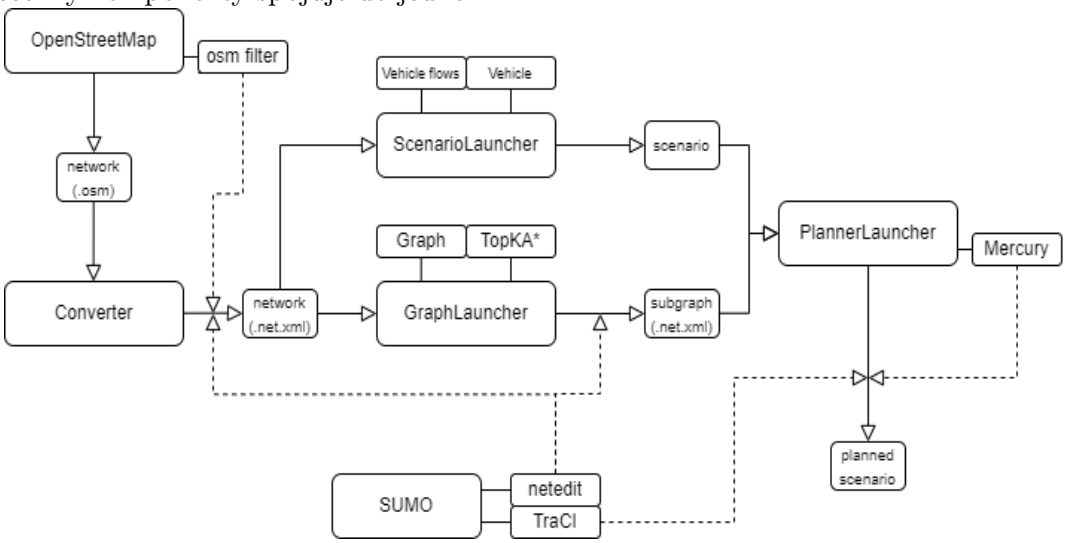
Kapitola 4
Implementace

Pro architekturu vyvinutou v této práci byl zvolen programovací jazyk Python. Toto je především z důvodu, že SUMO je také z velké části vyvinut v tomto jazyku, včetně knihovny TraCI, dále Python poskytuje mnoho knihoven, mezi ty známější které byly použity patří například: numpy, prompt_toolkit, pip, matplotlib a další. Navíc se jedná o objektově orientovaný jazyk, tedy jednotlivé komponenty architektury je možné reprezentovat jako třídy s vlastními parametry a metodami (třída Graf s parametry: hrana, uzel a metodou načtení ze souboru), další vlastnost těchto jazyků je dědičnost dovolující vytvoření hierarchií tříd (třída Junction a Edge dědí vlastnosti od třídy XmlObject, která implementuje ukládání do souboru formátu xml).

Cílem architektury je umožnit spojení různých komponent této práce, mezi které patří:

1. **Konverze** - popsána v kapitole 2 pomocí třídy Converter.
2. **Vytváření podgrafů** - obstaráno třídou GraphLauncher, umožnující aplikovat a vizualizovat jednotlivé techniky abstrakce silničních sítí.
3. **Vytváření dopravních scénářů** - přidávání vozidel (jednotlivě, nebo pomocí dopravních toků), generování adresáře obsahující soubory asociované se simulací (třída ScenarioLauncher).
4. **Plánování** - třída PlannerLauncher, která umožňuje zvolit a použít plánovač, po zadání parametrů (délka běhu, počet procesorových vláken, ..) a vytvořit tak plánovanou simulaci.

a poskytnout jednoduché a interaktivní použití. Z tohoto důvodu byla použita knihovna *prompt_toolkit*, která dovoluje přizpůsobit příkazovou řádku a rozšířit její funkcionalitu. Jednotlivé komponenty byly rozděleny do vlastních adresářů a je tak možné je použít jako samostatné knihovny, nebo lze spustit program ”main.py”, který všechny komponenty spojuje do jedné.



Obrázek 4.1. Vývojový diagram architektury ukazující průběh práce od začátku (stažení mapy silniční sítě) až po použití plánovací komponenty nad dopravním scénářem. Čárkované šipky představují použití externích programů.

4.1 Uživatelské rozhraní

Jak již dříve zmíněno, uživatel zadává vstup do příkazové řádky, která byla rozšířena o funkci našeptávání (tedy stačí zadat pouze prvních pár písmen a objeví se tabulka se všemi možnými vstupy), dále byla přidána historie předešlých vstupů (pomocí klávesových šipek ji lze procházet) a na konec byla přidána funkce automatického dokončení slova (opět po zadání prvních pár písmen) pomocí klávesy TAB. Všechny tyto funkce byly implementovány v rámci třídy UserInterface, která slouží jako rodičovská třída pro ostatní třídy pracující se vstupem od uživatele.

Při spuštění komponenty (programu) je uživatel nejprve požádán o zadání jména funkce, neplatné či neexistující jména nejsou přijata (objeví se varování), mezi výchozí funkce všech komponent patří `exit` pro ukončení programu a `help` pro vytisknutí všech funkcí a jejich dokumentací. Po zadání jména funkce je uživatel požádán o vyplnění jejích argumentů (pokud nějaké má, v opačném případě je spuštěna), argumenty jsou ve formátu: `jméno_argumentu="hodnota"`, kde `hodnota` je buď výchozí hodnota funkčního parametru, nebo je prázdná, a v tom případě musí být uživatelem vyplněna. Pokud nějaký z argumentů není vyplněn, nebo je špatného typu (například je místo celého čísla zadán text) vyskočí okénko s chybovou hláškou.

Další možnost jak předat programu vstup je pomocí přesměrování souborů na standardní vstup, tyto soubory je možné vytvořit manuálně, nebo pouze přepsat již existující, které jsou automaticky vytvořeny po použití specifických funkcí programu (ukládání grafu) v rámci loggování.

```
Launching UserInterface for 'graph', initializing objects..
Starting program, for help type: 'help'
Input command arguments between "quotes"
separated by space (arguments with values are optional).
Enabling new command: 'help'
Enabling new command: 'exit'
Enabling new command: 'load_graph'
Enabling new command: 'plot_graph'
Enabling new command: 'simplify'
Enabling new command: 'subgraph'
Enabling new command: 'merge'
Enabling new command: 'save_graph'
Enabling new command: 'print_graphs'
Enabling new command: 'delete_graph'
Input command: p
               plot_graph
               print_graphs
```

```
1 load_graph map_name="Sydney"
2 subgraph subgraph_name="sg0" graph_name="Sydney" from_junction="292"
  to_junction="248" c="1.3" plot="False"
3 subgraph subgraph_name="sg1" graph_name="Sydney" from_junction="87"
  to_junction="358" c="1.3" plot="False"
4 subgraph subgraph_name="sg2" graph_name="Sydney" from_junction="357"
  to_junction="248" c="1.3" plot="False"
5 subgraph subgraph_name="sg3" graph_name="Sydney" from_junction="241"
  to_junction="316" c="1.3" plot="False"
6 merge graph_name="sg0" graph_a="sg0" graph_b="sg1" plot="False"
7 merge graph_name="sg0" graph_a="sg0" graph_b="sg2" plot="False"
8 merge graph_name="sg0" graph_a="sg0" graph_b="sg3" plot="False"
9 save_graph graph_name="sg0" file_name="sydney_static_sg"
  scenario_name="sydney_static"
```

Obrázek 4.2. Na levé straně je zobrazen dynamický vstup uživatele v příkazové řádce spolu s funkcí našeptávání při zadávání jména příkazu. Na straně pravé je ukázán loggovací soubor vygenerován po uložení grafu (pro lepší přehlednost byly řádky zalomeny).

4.2 Dopravní Scénáře

Dopravní scénáře jsou chápány jako adresáře obsahující soubory asociované se simulací, např.: soubory spouštící simulaci, podgrafy silniční sítě vytvořené pro konkrétní scénář, soubory s vozidly a jejich cestami, atd. Třída ScenarioLauncher je zodpovědná za vytváření a spouštění scénářů, nejprve uživatel zadá jméno scénáře a poté je umožněno přidávat do simulace vozidla. Vozidla je možné přidávat individuálně, nebo je na výběr několik toků:

1. **Náhodný tok** - který `periodicky` vytvoří náhodný počet vozidel z `intervalu`
2. **Uniformní tok** - rovnoměrně rozpoloží `počet aut` v `časovém intervalu`
3. **Zvyšující tok** - kombinace uniformních a náhodných toků, dojde k rozdělení `časovém intervalu` na třetiny, poté jsou pro každou třetinu přidány uniformní toky, od nižší intenzity dopravy (1 vozidlo každých 7 sekund) až po vyšší intenzitu (1 vozidlo každé 3 sekundy), náhodné toky jsou použity jako přechody mezi jednotlivými uniformními toky, za cílem zmírnění skoku mezi zvyšující se intenzitou dopravy. [1]

Jelikož musí být vozidla v souborech seřazena podle jejich příjezdu (do simulace), jsou řazena pomocí binárního vyhledávacího stromu, který poskytuje vložení prvků s průměrnou asymptotickou složitostí $\Theta(\log(N))$ a poté pomocí inorder průchodu lze v lineárním čase získat seřazená vozidla.

Jméno souboru, který spouští simulaci (*sumocfg*) je použito jako name-space (jmenný prostor), tedy bude použito i na soubor obsahující vozidla a jejich cesty, statistické a loggovací soubory. Tato konvence je zachována i pro simulace vytvořené pomocí plánovací komponenty.

Kromě generování dopravních scénářů pomocí příkazové řádky byla vyvinuta třída Session, které používá na vstupu soubory formátu *json*, ve kterém jsou definované parametry plánovací komponenty (timeout, délka dopravního úseku, jméno plánovače a domény) a dalších parametrů, například: jméno silniční sítě, jméno vytvořených dopravních scénářů (jako prefix je k němu přidán čas vytvoření scénáře), parametry algoritmu Top K A*, počet toků, atd. Navíc je vytvořen soubor s pravděpodobnostním rozdělením toků, jedná se o maticový formát, kde první řádka reprezentuje identifikátory počátečních křižovatek a první sloupce slouží pro koncové křižovatky. Ve zbylých sloupcích a řádcích jsou čísla použity jako váhy šance vytvoření toků (čím vyšší číslo, tím vyšší šance), v případě, že je šance rovná nule není tok vytvořen. Po vytvořením prvního toku jsou ostatní vytvořeny tak, aby měli alespoň jednu společnou křižovatku na cestě s předchozími toky, tedy aby došlo ke konfliktu. Navíc takto vygenerované scénáře jsou automaticky vyhodnoceny v souboru formátu *csv*, ve kterém je porovnána velikost originální mapy oproti podgrafu, dále jsou vypsány informace o tocích (typ toku, počínající a koncová křižovatka, délka toku, průměrný počet vozidel za minutu a celkový počet) a na konci souboru jsou statistické informace simulátoru SUMO popsány v Kapitole 6.

```
[scenario_name]
  |
  └── maps* (folder related to subgraphs)
  |     └── information (".info" files logging how subgraph was generated)
  |     └── networks    (".net.xml" subgraphs)
  └── problems* (folder containing pddl problems)
  |     └── [scenario] (folder of pddl problems generated from "[scenario].sumocfg")
  └── results* (folder containing result of planner)
  |     └── [scenario] (folder of pddl results generated from "[scenario].sumocfg")
  └── routes ("[scenario].rou.xml" files, corresponding to "[scenario].sumocfg")
  └── simulation (folder )
        └── config ("[scenario].sumocfg" files, launching simulation)
        └── statistics* ("[scenario].stat.xml" files containing statistics of simulation)
        └── information ("[scenario].info" files logging how simulation was generated)
```

Obrázek 4.3. Adresářová struktura dopravních scénářů, složky označené symbolem "*" jsou vytvořeny pouze až když je použita korespondující komponenta (plánovací na složky problems, results).

[1] Slova vyznačena `šedě` jsou argumenty zadané uživatelem

```
name,junctions,edges,length
Dejvice,95,184,15141
default_sg,73,129,10439
flow_name,from,to,duration,per_minute,total
exponential_flow,69,81,3600,13.7,821
exponential_flow,83,70,3600,20.1,1208
exponential_flow,78,19,3600,18.0,1081
uniform_flow,15,73,3600,16.4,984
name,total_results,initial_results,problem_count
14_36_31_dejvice_default_planned,120,120,120
name,routeLength,speed,duration,waitingTime,timeLoss,departDelay,departDelayWaiting,totalTravelTime,totalDepartDelay
14_36_31_dejvice,1296.84,2.45,1250.68,974.99,1144.01,8922.36,-1.00,5120277.00,36528122.00
default_planned_1,1333.48,4.20,1147.09,910.32,1051.91,6286.07,-1.00,4696190.00,25735191.00
```

Obrázek 4.4. Výsledkový soubor jednoho z testovaných scénářů na mapě Dejvice.

	117	42	154	127
117	0	0	0	1
11	0	1	0	0
42	0	0	1	0
154	1	0	0	0

```
{
  "pddl_params": {
    "timeout": 25,
    "window": 30,
    "planner": "Mercury",
    "domain": "utc_allowed"
  },
  "scenario_name": "new_york_static",
  "duration": 3600,
  "seed": 650,
  "num_scenarios": 2,
  "probability_file": "New_York_static",
  "network": "New_York",
  "num_cpus": 1,
  "num_processes": 4,
  "num_threads": 3,
  "time_limit": null,
  "k": 3000,
  "flows": ["exponential_flow", "uniform_flow"],
  "flow_count": 3,
  "c": 1.3
}
```

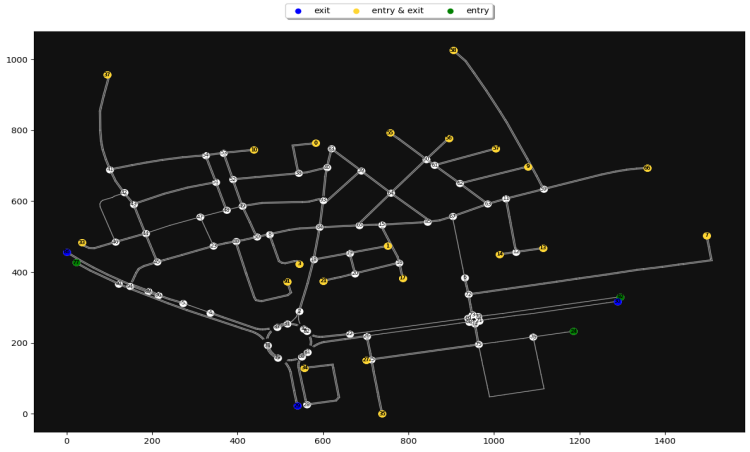
Obrázek 4.5. Na levé straně je ukázána pravděpodobností matice toků mapy New York a na pravé *json* soubor použitý pro automatické generování scénářů.

Kapitola 5
Techniky abstrakce silničních sítí

Pro techniky byla vytvořena třída GraphLauncher, před jejím použití je nutné konvertovat mapy z OpenStreetMap jak již bylo popsáno v Kapitole 2.1 a poté po spuštění programu na příkazové řádce použít příkaz ”load” a jako argument vyplnit jméno souboru silniční sítě. Jakmile je silniční síť načtena, bude uložena pod stejným jménem a je možné s ní dále pracovat. Mezi možné operace nad silniční sítí patří: vizualizace, zjednodušení sítě, vytváření, spojování a ukládání podgrafů.

5.1 Vizualizace

Vizualizace je založena na známé knihovně matplotlib, pomocí které jsou silnice vykresleny po částech jako přímky a křižovatky reprezentují vyplněné kruhy (spolu s číslem reprezentující identifikátor křižovatky v souboru silniční sítě). Pozice a tvar silnic a křižovatek jsou extrahovány ze souboru definující silniční síť pro SUMO. Díky tomuto lze měnit silniční síť v programu netedit a zároveň se tyto změny projeví i v tomto programu. Reprezentace silniční sítě je obdobná jako v simulátoru SUMO, jedná se o orientovaný graf, kde uzly (křižovatky) reprezentuje třída Junction, hrany (silnice) reprezentuje třída Edge. Navíc je zde třída Route, reprezentující sled složený z několika na sebe navazujících silnic. Cestovaní mezi křižovatkami je definováno ve třídě Junction, která obsahuje datovou strukturu hash table mapující identifikátory příchozích cest do množiny identifikátorů odchozích cest.



Obrázek 5.1. Ukázka silniční sítě vykreslené knihovnou matplotlib. Modře jsou vyznačeny koncové křižovatky, zeleně počáteční a žlutě křižovatky, které jsou jak koncové tak počáteční. Čísla zobrazená u souřadnicových os jsou v jednotkách metrů.

5.2 Zjednodušování silničních sítí

Pro zjednodušování silniční sítě slouží příkaz *simplify*, který jako argumenty očekává jméno grafu silniční sítě a booleovskou hodnotu (True, False), která, v případě že je

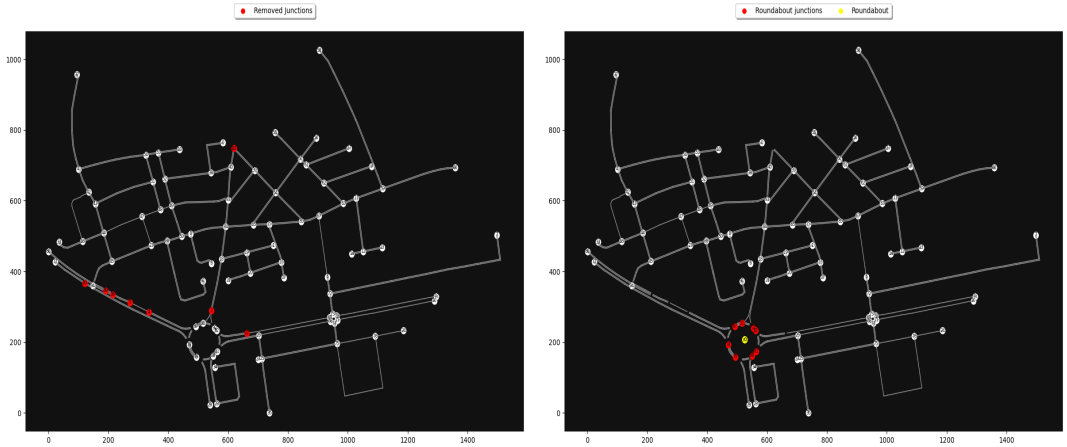
pravdivá, zahájí vizualizaci jednotlivých kroků zjednodušení grafu (výchozí hodnota je False). Zjednodušení grafu je nevratná operace, ale všechny informace v grafu se zachovají, tedy výsledná cesta jakéhokoliv průchodu grafu má stále stejné identifikátory a počet silnic.

5.2.1 Odstranění nadbytečných křižovatek

Simulátor SUMO mnohdy přidá křižovatky, které ale v reálné silniční síti nejsou, je to zejména z vykreslovacích důvodů, nebo kvůli nedostatku informací z OpenStreetMap. Tyto křižovatky typicky mají pouze jednu vstupní a výstupní silnici, obecně se za nadbytečné považují právě tehdy když mají ze všech příchozích cest pouze jednu možnou odchozí cestu, a zároveň je tato odchozí cesta pro každou příchozí unikátní (výjimkou jsou počáteční křižovatky).

5.2.2 Transformování kruhových objezdů

Nejprve dojde k odstranění všech křižovatek, které formují kruhový objezd a následně je nahradí jediná křižovatka, jejíž pozice bude v těžišti odstraněných křižovatek. Vstupní cesty do této nové křižovatky budou všechny vstupní cesty do kruhového objezdu a jako výstupní budou všechny možné kombinace cest (včetně cesty po kruhovém objezdu) vedoucích ven z kruhového objezdu.



Obrázek 5.2. Vizualizace zjednodušení silniční sítě, vlevo zjednodušení křižovatek a vpravo zjednodušení kruhového objezdu. Červeně jsou vyznačeny křižovatky, které budou odstraněny, nová křižovatka reprezentující kruhový objezd je vyznačena žlutou barvou.

Místo (město)	#křižovatek	zjednoduš.	#segmentů	zjednoduš.
Chodov (Praha)	95	75	190	160
Dejvice (Praha)	95	86	184	171
Re di Roma (Řím)	107	89	170	149
Kreuzberg (Berlín)	131	85	291	214
South Bronx (New York)	177	175	403	400
Clerkenwell (Londýn)	308	230	650	507
Clovelly (Sydney)	441	416	1061	1018
Ballsbridge (Dublin)	575	515	978	917

Tabulka 5.1. Výsledky zjednodušení křižovatek a segmentů silničních sítí.

5.3 Top K A*

Standardní algoritmy, které se snaží najít nejlepších (Top) K cest, se zaměřují na nalezení přesně těchto K cest, patří mezi ně např. Yenův algoritmus nebo K* [7]. Takovéto řešení je pro tento problém nepraktické, protože je velmi těžké určit, jaký počet cest by měl být zvolen pro dosažení dobrého výsledku. Z tohoto důvodu byla tato podmínka nahrazena parametrem c, který limituje jak délku cesty, která musí být maximálně dlouhá jako $c \times$ nejkratší délka cesty, tak počet možný cest. Limitování délky cest dává lepší popis situace, protože tímto výrazně limitujeme prohledávaný prostor a zároveň tak maximalizujeme využití silniční sítě v dané oblasti.

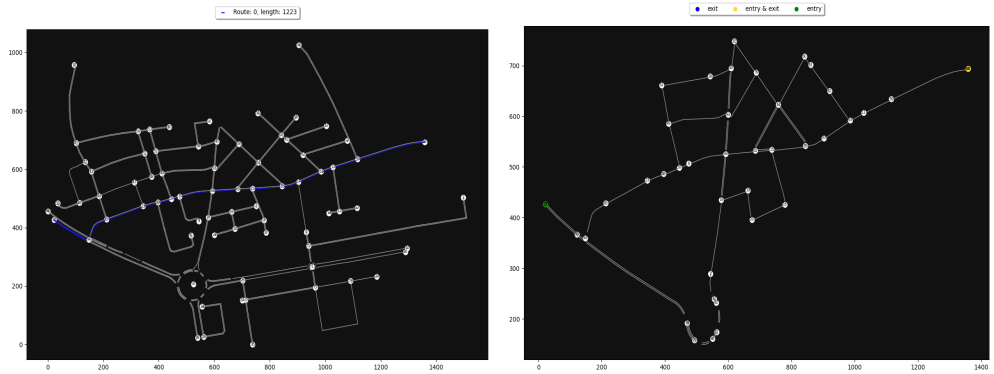
Algorithm 1: Variace A* pro nejlepších K cest

```
   Data: Starting junction id start, target junction id target,
         suboptimality bound c, max number of routes K
   Result: The set of found routes
 1 Routes ← ∅
 2 queue, shortest_path ← A*(start, target)
 3 if shortest_path is undefined then
 4 |   return ∅
 5 end
 6 Routes ← Routes ∪ {shortest_path}
 7 limit ← c × length(shortest_path)
 8 while queue not empty do
 9 |   priority, route ← pop(queue)
10 |   if priority > limit or |Routes| == K then
11 |   |   break
12 |   else if destination(route) == target then
13 |   |   Routes ← Routes ∪ { route}
14 |   |   continue
15 |   else
16 |   |   foreach unvisited neighbour of route do
17 |   |   |   route' ← add(route, neighbour)
18 |   |   |   priority' ← length(route') + distance(neighbour, target)
19 |   |   |   push(queue, (priority', route'))
20 |   |   end
21 |   end
22 end
23 return Routes
```

Algoritmus 1 obsahuje pseudokód variace A*. Oproti K*, kde dochází ke střídání A* a Dijsktrova algoritmu (A* prohledáváním grafu průběžně staví podgraf, na kterém Dijkstrův algoritmus hledá alternativní cesty), je v tomto algoritmu použit pouze A* pro nalezení nejkratší cesty, kterou spolu s frontou neprohledaných křižovatek vrátí (řádek 2). Poté je zkontrolována existence nejkratší cesty (řádek 3), v případě, že by neexistovala, je vrácen prázdný seznam, jinak je nejkratší cesta přidána do seznamu cest (řádek 6), následně je na řádku 7 vypočtena maximální povolená délka alternativních cest. Řádky 8 až 21 tvoří hlavní část algoritmu, kde se obdobně jako v A* ze začátku odebírají nalezené cesty z prioritní fronty (řádek 9). V případě překročení stanové délky nebo maximálního počtu cest (udávaného parametrem K, jehož výchozí hodnota je nastavena na nekonečno), je hledání cest ukončeno (řádek 10-11), pokud je poslední křižovatka cesty shodná s cílovou, je přidána do seznamu nalezených cest a ukončuje se její prohledávání (řádek 12-14). Na řádcích 16-21 se do fronty přidají nové cesty, vytvořené z původní cesty přidáním nenavštívené sousední křižovatky (řádek 17) a

do fronty jsou přidány s prioritou rovnou jejich délce plus Euklidovské vzdálenosti od cílové křižovatky (řádek 18-19). Euklidovská vzdálenost je konzistentní a přípustnou heuristika, jelikož se pohybujeme v Euklidovském prostoru, a proto nejkratší silnice mezi dvěma křižovatkami bude vždy tvaru úsečky a zároveň platí trojúhelníková nerovnost. Algoritmus končí, jakmile najde všechny přípustné cesty a vrátí jejich seznam (řádek 23).

Jeho použití je pomocí příkazu *subgraph*, který jako argumenty očekává: nové jméno vytvořeného podgrafu (pod kterým bude uložen během běhu programu), jméno původní sítě, z které bude podgraf vytvořen, všechny argumenty algoritmu 1 a nakonec booleovský argument pro ukázání animace průběhu.



Obrázek 5.3. Snímek animace algoritmu (vlevo), která postupně ukazuje nalezené cesty vyznačené modře, zároveň je v legendě ukázána jejich délka, v tomto případě je ukázána první (nejkratší) cesta. Vpravo je výsledný podgraf postaven z nalezených cest.

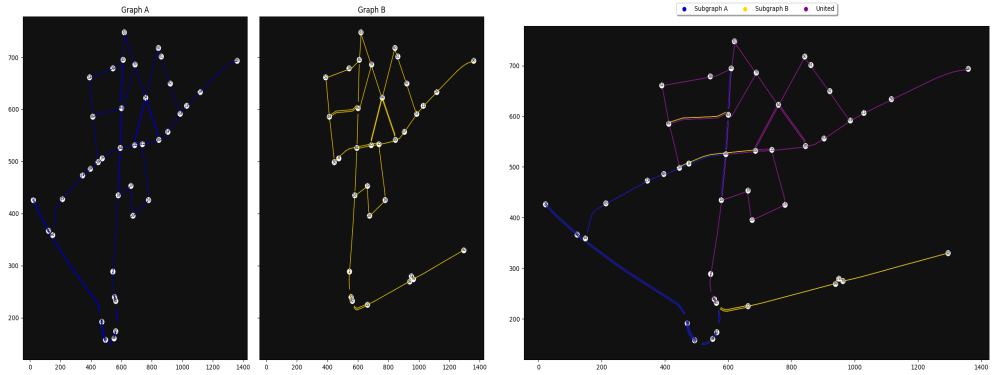
Místo (město)	parametr c		
	1.15	1.25	1.5
Chodov (Praha)	0.01 ± 0.01	0.01 ± 0.01	0.02 ± 0.01
Dejvice (Praha)	0.01 ± 0.01	0.01 ± 0.01	0.02 ± 0.02
Re di Roma (Řím)	0.01 ± 0.01	0.01 ± 0.01	0.01 ± 0.01
Kreuzberg (Berlín)	0.02 ± 0.01	0.04 ± 0.03	0.92 ± 0.42
South Bronx (New York)	0.04 ± 0.04	0.33 ± 0.12	16.11 ± 7.23
Clerkenwell (Londýn)	0.03 ± 0.01	0.20 ± 0.09	1.23 ± 0.33
Clovelly (Sydney)	0.57 ± 0.16	5.12 ± 2.18	*
Ballsbridge (Dublin)	0.35 ± 0.21	4.67 ± 2.53	*

Tabulka 5.2. Měření pro různé hodnoty parametru c reprezentuje čas vytvoření podgrafu v sekundách pro tisíc pokusů ve formátu střední hodnota ± směrodatná odchylka. Symbol "*" je použit z důvodu velkého rozdílu (v řádu statisíců) mezi počtem nalezených cest, závisejícího na volbě startovací a koncové křižovatce.

5.4 Spojování podgrafů

Z původního grafu silniční sítě je vytvořen podgraf (Obrázek 5.3), obsahující pouze silnice a křižovatky, které jsou v některých z cest nalezených algoritmem Top K A*, takto vytvořené podgrafy lze mezi sebou spojovat. Důvodem k vytváření podgrafů je možnost využití znalosti o dopravě v dané lokaci a snížení velikosti silniční sítě. Jelikož ve většině případů není nutné použít celou silniční síť pro hledání alternativních cest,

především kvůli vzdálenosti nebo nízké propustnosti silnic nižších tříd. Pokud víme odkud a kam vozidla nejčastěji jezdí, a kde tedy dochází k dopravním zácpám, lze zjednodušit prohledávaný prostor v závislosti na nastaveném parametru c v algoritmu Top K A*. Jakmile vytvoříme a pospojujeme všechny podgrafy, získáme finální podgraf, na kterém se budou hledat alternativní cesty pro vozidla (Obrázek 5.4).



Obrázek 5.4. Fotka spojení dvou podgrafů (vlevo), výsledný podgraf po spojení (vpravo), fialová barva vyznačuje společnou část podgrafů.

5.5 Vyhodnocení technik

V průměru bylo dosaženo přibližně 10% procentní snížení počtu segmentů (vstupů a výstupů z křižovatek) a křižovatek oproti původní reprezentaci silniční sítě v SUMO (Tabulka 5.1). Je zde několik outlierů: mapa South Bronx, kde téměř nedošlo ke snížení počtu křižovatek a segmentů, protože již byla v SUMO blízko minimalistické reprezentaci, mapa Clerkenwell, kde došlo o více než 20% snížení počtu křižovatek a segmentů.

Velikost a rychlost vytvoření podgrafů vytvořené algoritmem Top K A* (Tabulka 5.2) výrazně závisí na nastavení parametrech c a K, ale i na volbě počáteční a koncové křižovatky, za normálních okolností lze očekávat výrazně menší podgraf, příkladem je Obrázek 5.4.

Výhodou těchto technik je, že se jedná o pre-proccessing, tedy stačí je použít pouze jednou před samotným hledáním alternativních cest. Další výhodou je, že čím více se sníží velikost silničních sítí, tím více vozidel můžeme přidat do simulace. Pro použití na větších mapách je možné algoritmus Top K A* zlepšit, například použitím rychlejšího algoritmu pro nalezení nejkratší cesty, paralelizace hledání přípustných cest, nebo uplatněním složitějších heuristik, které hledají cesty na základě několika vlastností silnic (kategorie, propustnost a jiné), popřípadě shlukových algoritmů.

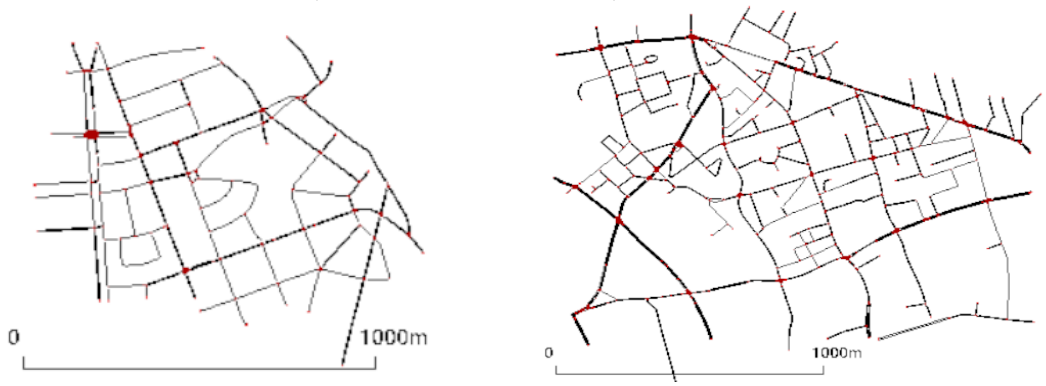
Kapitola 6
Výsledky

Dopravní scénáře byly hodnoceny pomocí statistických souborů, které SUMO generuje (po přidání příkazu "–statistic-output"). Tyto soubory jsou formátu *xml* (přípona *stat.xml*), obsahující informace ohledně průběhu simulace. Mezi tyto informace patří počet načtených a vložených aut do simulace, počet kolizí, statistiky ohledně cestujících (osob v simulaci) a další. Hlavní údaj určující kvalitu simulace (z pohledu vozidel) je označen tagem ⟨*vehicleTripStatistics*⟩, který má jako parametry: délku cesty aut, rychlost, délku simulace (čas kdy poslední auto opustilo simulaci), čas čekání (kdy se auta nebyla v pohybu) a časovou ztrátu (udávající celkový čas, kdy vozidla jela pomaleji než požadováno), všechny tyto parametry jsou udávány jako průměr za celou simulaci a měřeny v základních jednotkách.

Pro testování byly zvoleny čtyři mapy z každé kategorie velikosti: Dejvice reprezentující malé mapy, South Bronx a Clerkenwell reprezentující střední a na závěr Clovelly pro velké mapy. Scénáře na těchto mapách byly nastaveny na délku jedné hodiny a počáteční a cílové křižovatky byly zvoleny dle hlavních cest v dané lokaci (vyznačeny červeně na OpenStreetMap). Pro každou mapu bylo vygenerováno deset scénářů, z kterých byly všechny výsledky zprůměrovány (*csv* soubory v Obrázku 4.4).

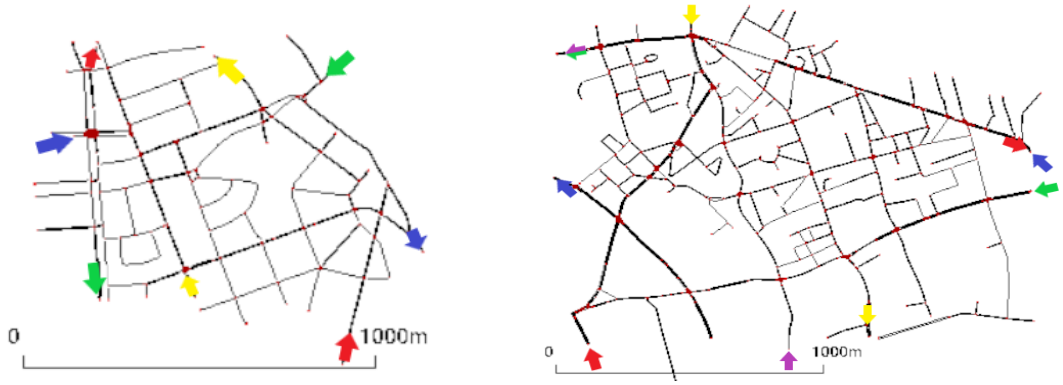
6.1 Parametry

Parametry použité pro generování scénářů jsou následující, pro algoritmus Top K A* bylo použito $c = 1.3$ a $K = 2999$ (pro mapu Dejvice $c = 1.4$). Pro plánovač Mercury byl nastaven limit 25 sekund na vygenerování výsledků a každý problémový soubor pokrýval 30 sekundový dopravní úsek. Toky byly náhodně vybírány mezi uniformním a zvyšujícím, toto slouží k simulování dopravní špičky, kde obvykle existují toky, které již mají vyšší intenzitu (tomuto odpovídají uniformní toky) a zároveň se začínají formovat toky, které postupem času nabývají na intenzitě (zvyšující toky). Pro mapu Dejvice byl počet toků nastaven na 4, jelikož se jedná o menší silniční síť, pro ostatní je počet toků roven 3. Scénáře byly testovány na procesoru Intel i7-8700 (3.2 GHz) a použití paměti RAM nepřesahovalo 4GB (pro 4 paralelní procesy).



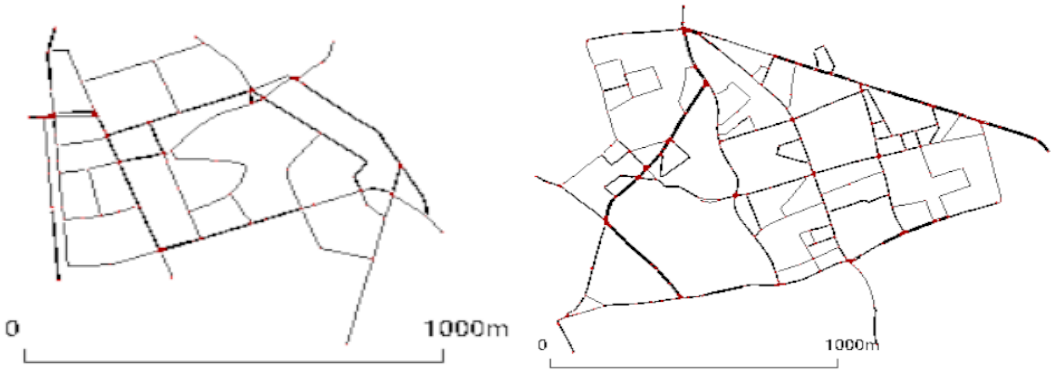
Obrázek 6.1. Ukázka originálních silničních sítí použitých pro testování automatického plánování. Postupně na levé straně je mapa Dejvice a na pravé Clerkenwell.

Obrázek 6.2. Další silniční sítě, vlevo mapa South Bronx a vpravo Clovelly.

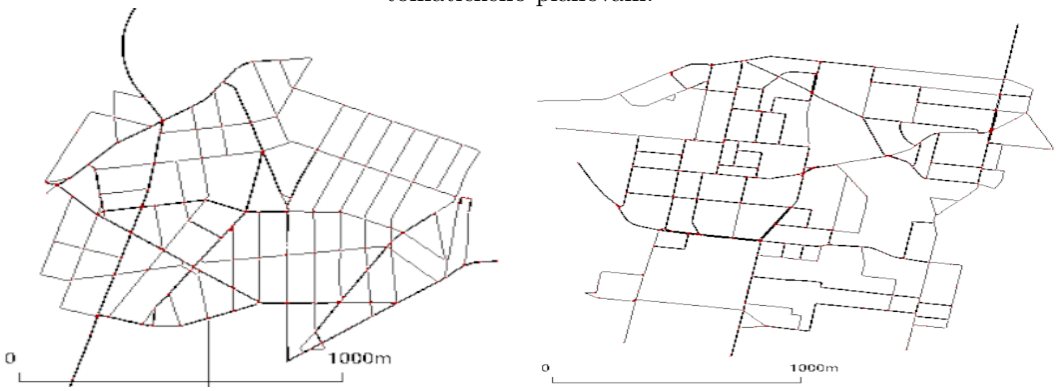


Obrázek 6.3. Ukázka počínajících a koncových křižovatek toků vozidel vyznačených barevnými šipkami map z Obrázků 6.1.

Obrázek 6.4. Toky map Obrázků 6.2.



Obrázek 6.5. Podgrafy silničních sítích obrázků 6.1, které byly použity v problémech automatického plánování.



Obrázek 6.6. Zbylé podgrafy obrázků 6.2.

Místo (město)	#křižovatek	#segmentů	vozidlo/minuta	#vozidel
Dejvice (Praha)	73	126	62	3744
South Bronx (New York)	134	277	47	2873
Clerkenwell (Londýn)	205	378	45	2733
Clovelly (Sydney)	177	327	51	3089

Tabulka 6.1. Průměrné velikosti podgrafů silničních sítí z Obrázku 6.3 v prvních dvou sloupcí, třetí sloupec počítá průměrný počet vozidel všech toků za minutu a poslední sloupec je průměrný počet všech vozidel ve scénářích.

Místo (město)	Plánů (%)	Vzdálenost	Rychlost	Délka běhu	Čekání
		1282.83	2.68	1140.28	873.46
Dejvice (Praha)	100	1329.01	4.12	989.26	767.51
		2280.52	1.09	4077.99	3447.53
South Bronx (New York)	74	2469.07	1.9	2384.96	1887.96
		2088.74	2.12	1549.7	1076.7
Clerkenwell (Londýn)	55	2221.15	2.1	1718.83	1220.76
		2593.68	3.02	1610.0	1215.98
Clovelly (Sydney)	25	2671.83	3.17	1573.53	1173.32

Tabulka 6.2. Tabulka průměrů statistik vozidel, první řádek každé mapy reprezentuje hodnoty originální simulace a pod ní jsou hodnoty simulace vytvořené pomocí automatického plánování. První sloupec označuje počet vytvořených výsledkových souborů (prvních výsledků) ze 120 period. Ostatní hodnoty v tabulce jsou udávány v základních jednotkách, poslední sloupec je celkový čas, kdy vozidla nebyla v pohybu.

6.2 Diskuze výsledků

Jak můžeme pozorovat výsledky zjednodušení silničních map v Tabulce 6.1, zjednodušené silniční sítě jsou výrazně menší oproti původním sítím. Výrazné snížení počtu křižovatek a segmentů bylo dosaženo pro mapu Clovelly (okolo 70 % snížení velikosti oproti původní) a Clerkenwell. Důležitým faktorem ve výsledcích je topologie silničních sítí, můžeme to pozorovat především na výsledcích mapách Clovelly, Clerkenwell a South Bronx. I přes to, že mapa Clerkenwell má více křižovatek a segmentů nežli

Clovelly, tak pro ní plánovačem bylo vygenerováno 55 % řešení pro Clovelly pouze 25 %. Toto je zejména z důvodu velké propojenosti mapy Clovelly, zatímco v mapě Clerkenwell je nejvíce propojená pouze pravá strana silniční sítě, tak v mapách Clovelly a South Bronx existuje mnoho alternativních cest ve většině křižovatkách. Ze stejného důvodu je algoritmus Top K A* výrazně rychlejší (Tabulka 5.2) pro mapu Clerkenwell, než je v mapě South Bronx, která je asi o 49 % menší.

Vysoká propojenost silničních sítí je ale v důsledku pozitivní vlastnost, jelikož jak můžeme pozorovat v mapě South Bronx, kde je použit manhattanský model silniční sítě, plánovač je schopen najít mnoho alternativních cest a je tedy schopen výrazně zlepšit nalezené řešení, téměř byla zdvojnásobena průměrná rychlost vozidel, čas čekání vozidel a běhu simulace byl taktéž snížen téměř o polovinu. Míra zvýšení průměrné rychlosti není přímo mapována do času běhu simulace, jak je zobrazeno pro mapu Dejvice, k tomuto také přispívá delší průměrná cesta vozidel, s kterou se počítá. Stejně tak bylo dosaženo mírného zlepšení v mapě Clovelly a dá se předpokládat, že by bylo vyšší, kdyby bylo vygenerováno více řešení plánovačem, opakem je mapa Clerkenwell, kde plánovač nebyl schopen nalézt lepší řešení nežli nejrychlejší (nejkratší) cesty z předešle zmíněných důvodů.

Kapitola 7
Závěr

V této práci bylo cílem použití technik automatického plánování na problém inteligentního směrování dopravy. Nejprve kvůli složitosti problému byly vyvinuty techniky, které výrazně zjednodušili grafy silničních sítí a snížily tak prohledávaný prostor technik automatického plánování. Mezi tyto techniky patří odstranění přebyteční křižovatek, zjednodušení kruhových objezdů, vytváření a spojování podgrafů. Poté byla vytvořena formální reprezentace problému v plánovacím jazyce PDDL, problém byl definován jako minimalizace počtu aut na jednotlivých cestách, pak byl problém předán plánovači Mercury pro vytvoření nových cest vozidel. Simulátor SUMO byl použit pro simulování dopravy v grafickém prostředí a pro testování technik automatického plánování v porovnání s vozidly používající nejkratší (nejrychlejší) cesty. Silniční sítě v této práci byly staženy ze známého projektu OpenStreetMap.

Aby bylo možné techniky a algoritmy vyvinuté v této práci jednoduše modifikovat, testovat a propojit se simulátorem SUMO, byla vyvinuta architektura v programovacím jazyce Python. Architektura taktéž umožňuje vytvářet dopravní scénáře s proměnlivými dopravními podmínkami, čímž byly dopravní scénáře přiblíženy dopravním situacím v reálném světě (dopravní špička). Interakce uživatele s architekturou je poskytnuta pomocí příkazové řádky, jejíž vlastnosti byly rozšířeny, dále architektura poskytuje vizualizaci silničních sítí a práce algoritmů. Jednotlivé komponenty práce (plánovací, vytváření dopravních scénářů a grafů) byly rozděleny do vlastní knihoven a je tedy možné použít každou samostatně.

Techniky zjednodušení silničních sítí byly testovány na několika silničních sítí, počínaje malými mapami (okolo 100 křižovatek a segmentů) až po velké mapy (přes 300 křižovatek a segmentů). Tyto techniky výrazně snížily prohledávaný prostor (v některých příkladech i o více než 50 %) a umožnily tak použití technik automatického plánování ve stanoveném časovém limitu. Výsledky ukazují výrazné zlepšení oproti nejkratším cestám, za předpokladu, že topologie silniční sítě má dostatečně mnoho alternativních cest a parametry algoritmů byly vhodně nastaveny.

Cíl, který byl v této práci stanoven byl splněn, pro architekturu vytvořenou v této práci byl kladen důraz na možnost rozšíření a navázání na tuto práci. Jedná se zejména o techniky, které by více zjednodušili silniční sítě (zmíněny ke konci Kapitoly 5.5), jak experimentálně ukázáno v mapě Clovelly, samotný algoritmus Top K A* nestačí pro takto velké silniční sítě. Dále by se v budoucnu mělo uvažovat o dopravních srážkách, jak na ně reagovat z pohledu plánování, a také o vozidlech, které nemusí nutně použít nově nalezenou cestu (například kvůli selhání komunikace).

Literatura

[1] Lukáš Chrpa, Mauro Vallati a Simon Parkinson. *Exploiting automated planning for efficient centralized vehicle routing and mitigating congestion in urban road networks.* In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. 2019. 191–194.

[2] Mark D Johnston. *Spike: AI scheduling for Hubble Space Telescope after 18 months of orbital operations.* In: *Working Notes AAAI Spring Symposium on Practical Approaches to Scheduling and Planning*. 1992.

[3] John L Bresina, Ari K Jónsson, Paul H Morris a Kanna Rajan. *Activity Planning for the Mars Exploration Rovers..* In: *ICAPS*. 2005. 40–49.

[4] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner a Evamarie Wieÿner. *Microscopic traffic simulation using sumo.* In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. 2018. 2575–2582.

[5] Malik Ghallab, Dana Nau a Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

[6] Michael Katz a Joerg Hoffmann. Mercury planner: Pushing the limits of partial delete relaxation. *IPC 2014 planner abstracts*. 2014, 43–47.

[7] Husain Aljazzar a Stefan Leue. K : A heuristic search algorithm for finding the k shortest paths. *Artificial Intelligence*. 2011, 175 (18), 2129–2154.

Příloha A
Obsah elektronické přílohy

V příloze se nacházejí dva adresáře, první obsahuje výsledky experimentů, neobsahuje celé dopravní scénáře, pouze *csv* soubory. V druhém adresáři se nachází kód architektury s jedním (ilustrativním) dopravním scénářem, v souboru README je odkaz na veřejný github projektu spolu s instalačním návodem a zbylými dopravními scénáři.