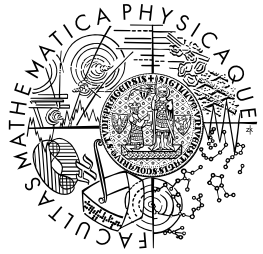


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

David Beinhauer

Optimalizace rozmístění stanic pro nabíjení elektrických vozidel

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Martin Pilát, Ph.D.

Studijní program: Informatika

Studijní obor: Informatika se specializací Umělá inteligence

Praha 2022

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

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

V ..... dne ....

Podpis autora

Rád bych poděkoval panu Mgr. Martinu Pilátovi, Ph.D. za odborné vedení mé práce, cenné rady a vstřícnost při konzultacích a vypracování bakalářské práce. Dále bych rád poděkoval členům své rodiny a přátelům za podporu během psaní bakalářské práce.

Název práce: Optimalizace rozmístění stanic pro nabíjení elektrických vozidel

Autor: David Beinhauer

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Martin Pilát, Ph.D., Katedra teoretické informatiky a matematické logiky

Abstrakt: S rostoucím počtem elektrických vozidel roste i potřeba vytvořit vhodnou infrastrukturu pro jejich nabíjení. K řešení tohoto problému může výrazně napomoci použití vhodných optimalizačních metod. V práci jsme implementovali zjednodušený simulátor dopravy sloužící jako vhodný nástroj pro jejich analýzu. Analyzovali jsme také optimalizační metody tzv. hladovým algoritmem, genetickým algoritmem a algoritmem k-means. Na základě experimentů vykazovala prokazatelně lepší výsledky optimalizace za využití genetického algoritmu a hladová optimalizace. K-means optimalizace nevykazovala známky lepších výsledků oproti náhodnému přístupu.

Klíčová slova: optimalizace, simulátor dopravy, k-means, genetický algoritmus

Title: Optimization of the Placement of Electric Vehicle Charging Stations

Author: David Beinhauer

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Martin Pilát, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: As the number of electric vehicles grows, so does the need to create a suitable network of charging stations. A solution of this problem can be significantly improved by the usage of suitable optimization techniques. We implement a simplified traffic simulator serving as a suitable tool for their analysis. We also analyze optimization techniques using the so-called greedy algorithm, genetic algorithm and k-means algorithm. Based on the experiments, the optimizations using the genetic algorithm and the greedy algorithm showed noticeably better results. The k-means method did not show signs of results better than a random approach.

Keywords: optimization, traffic simulator, k-means, genetic algorithm

Obsah

Úvod **5**

1 Související práce a pojmy **9**
1.1 Zeměpisné souřadnice a sférická vzdálenost 9
1.2 Genetický algoritmus 9
1.3 K-Means 10
1.4 A-star 11
1.5 Související práce 12

2 Příprava mapy silniční sítě **15**
2.1 Extrakce dopravní sítě 15
2.1.1 Extrakce s pomocí nástroje Osmium 15
2.1.2 Příprava dat pomocným skriptem 17
2.2 Dodatek k volbě programovacího jazyka a nutnosti předpřípravy dat 17

3 Popis simulátoru **19**
3.1 Poznámka ke značení a definicím 19
3.2 Předpoklady návrhu modelu 19
3.3 Dopravní síť 20
3.4 Objekty dopravní sítě 22
3.4.1 Segmenty 22
3.4.2 Křižovatky 23
3.4.3 Silnice 23
3.4.4 Města 24
3.4.5 Nabíjecí stanice 24
3.5 Vozidla 25
3.5.1 Generování vozidel 25
3.5.2 Hledání trasy vozidla 26
3.5.3 Zajíždění na nabíjecí stanici 28
3.5.4 Druhy vozidel 30

3.6 Specifika reálné dopravní sítě . . . 31
3.6.1 Poznámky k definici modelu dopravní sítě . . . 31

4 Průběh simulace **35**
4.1 Inicializace simulátoru . . . 35
4.2 Popis simulace . . . 36
4.3 Události simulace . . . 36
4.3.1 Výjezd vozidla . . . 36
4.3.2 Pohyb vozidla . . . 36
4.3.3 Události nabíjení vozidla . . . 37
4.3.4 Konec cesty vozidla . . . 37
4.4 Běh simulace . . . 38

5 Optimalizační algoritmy pro rozmístění nabíjecích stanic **39**
5.1 Ztrátová funkce (loss function) . . . 39
5.1.1 Poznámka k vybitým vozidlům . . . 40
5.2 Optimalizace hladovým (greedy) algoritmem . . . 40
5.3 Optimalizace genetickým algoritmem . . . 41
5.4 Optimalizace inspirovaná k-means . . . 43

6 Analýza výsledků optimalizace **47**
6.1 Poznámka ke značení . . . 47
6.2 Výběr parametrů simulátoru . . . 48
6.2.1 Popis analýzy . . . 48
6.3 Analýza optimalizačních metod . . . 50
6.3.1 Poznámka k výsledkům experimentů . . . 52
6.3.2 Náhodný přístup . . . 52
6.3.3 Optimalizace hladovým algoritmem . . . 53
6.3.4 Optimalizace genetickým algoritmem . . . 53
6.3.5 Optimalizace s pomocí k-means . . . 54
6.3.6 Porovnání optimalizačních metod . . . 55
6.4 Pomocné nástroje pro analýzu výsledků . . . 56

Závěr **59**

Seznam použité literatury **61**

A Uživatelská dokumentace **63**
A.1 Příprava před spuštěním . . . 63
A.1.1 Sestavení programu . . . 63
A.2 Poznámka k příkladům . . . 65
A.3 Formát souborů reprezentujících mapu . . . 65

A.3.1 Soubor reprezentující křižovatky . . . 66
A.3.2 Soubor reprezentující silnice . . . 66
A.3.3 Soubor reprezentující města . . . 67
A.3.4 Výstupní soubory . . . 68
A.4 Ovládání programu . . . 68
A.4.1 Parametry programu . . . 68
A.4.2 Příklad spouštění programu . . . 72
A.5 Výstup programu . . . 73

B Programátorská dokumentace 77
B.1 Struktura programu . . . 77
B.2 Třída Optimizer . . . 78
B.3 Třída TimeTable . . . 78
B.4 Třída TrafficSimulator . . . 79
B.5 Třída Map . . . 79
B.5.1 Operace na mapě . . . 79
B.6 Třídy objektů mapy . . . 80
B.7 Třídy pro načítání a předpřípravu dat . . . 81
B.7.1 Třída MapReader . . . 81
B.7.2 Třída GraphAdjuster . . . 81
B.8 Třída Vehicle . . . 82
B.9 Pomocné třídy . . . 82
B.10 Testy programu . . . 83

C Přílohy práce 85

Úvod

V důsledku zvyšujícího se zájmu společnosti chránit životní prostředí významně roste také snaha mnoha mezinárodních a vládních organizací o nahrazení vozidel poháněných spalovácími motory na fosilní paliva za enviromentálně přívětivější varianty [1]. V posledních letech se zejména díky výraznému technologickému pokroku v této oblasti jeví jako nejlepší alternativa využití elektrických vozidel, jejichž počty v posledních letech násobně rostou [2]. Zmíněný fenomén je ale také příčinou vyšší poptávky po nabíjecích stanicích, které jsou v určitých oblastech špatně dostupné a jejichž kapacita často není dostatečná.

Problém bohužel nemá jednoduché řešení, protože samotný proces naplánování rozmístění, kapacity a počtu nabíjecích stanic je značně komplexní. Během rozmísťování stanic je vyvíjen tlak na minimalizaci počtu stanic a jejich kapacit, protože výstavba nové stanice je logisticky, finančně i časově náročná. Zároveň pokud umístíme stanici do málo frekventované oblasti, pak potenciální užitek nabíjecí stanice nebude plně využit. Naopak nedostatečný počet stanic ve značně vytížených oblastech může vézt k vytváření front a nárustu čekací doby na stanicích. To je v kombinaci s poměrně značnou časovou náročností nabíjení problematické a pro klienty stanic nepraktické. V neposlední řadě je také potřeba brát v úvahu vzdálenost a s ní související časový deficit způsobený cestou do stanice.

S rostoucím výpočetním výkonem se stále více nabízí řešení tohoto problému s využitím celé škály optimalizačních technik, jako jsou například evoluční algoritmy, či různé variace algorimů strojového učení. Zmíněné metody ovšem často potřebují nějakou formu zpětné vazby, s jejíž pomocí ohodnocují různé varianty řešení a volí z nich to nejoptimálnější. K tomuto účelu může dobře posloužit vhodně navržený simulátor dopravy, jenž umožňuje analýzu různých variant rozmístění nabíjecí stanic.

Cíl a popis práce

Cílem této práce je navrhnout simulátor dopravy, jenž umožňuje co nejjednodušším způsobem pracovat s různými variantami rozvržení nabíjecích stanic a

ohodnocovat jejich kvalitu. Dalším významným cílem této práce je s pomocí navrženého simulátoru navrhnout a analyzovat optimalizační algoritmy pro rozmístění nabíjecích stanic v dopravní síti.

Návrh simulátoru je založen na snadné manipulaci s různými variantami rozmístění nabíjecích stanic. Důraz je kladen především na jednoduchou nastavitelnost počtu a kapacity jednotlivých stanic s ohledem na možné využití různých strategií pro jejich rozmísťování. V neposlední řadě je vyžadován také snadný přístup ke statistickým údajům relevantním k řešenému problému jako je například průměrné vytížení stanic, čekací doba na nabití, průměrná doba cestování, průměrná hladina nabití vozidel v daných úsecích dopravní infrastruktury a podobně. Na druhou stranu je samotný model provozu v jistých aspektech výrazně zjednodušen především z důvodu výpočetní efektivity simulátoru.

Hlavní význam simulátoru je popisovat především dálkové trasy, u kterých je nejčastější potřeba nabití vozidla během cesty. Zmíněné zaměření na dálkové cesty je motivováno předpokladem, že převážná část majitelů elektrických vozidel má možnost své vozidlo dobít v cílové destinaci z vlastního zdroje energie na úroveň dostatečnou pro cesty na krátké vzdálenosti, které tak nejsou relevantní pro řešení našeho problému. Při návrhu je upozaděno řešení dopravních situací na úrovni jednotlivých vozidel, jako je například projíždění křižovatek či tvoření dopravních zácp. Samotná simulace je realizována pomocí metody diskrétní simulace.

K simulátoru jsou také dodány pomocné skripty pro předpřípravu volně dostupných informací o reálné dopravní síti ze serveru Geofabric[1], s jejichž pomocí mohou být tato data převedena do vstupního formátu našeho simulátoru. Zmíněné skripty nejsou nutné pro správné fungování simulátoru. Pokud však chceme simulovat reálnou silniční síť je s ohledem na rozsáhlost dat téměř nerealizovatelná jejich příprava bez použití zmíněných skriptů, či nástroje s odpovídající funkcionalitou.

V optimalizační části jsou analyzovány 3 optimalizační algoritmy pro nalezení optimálního řešení. První tzv. hladový algoritmus hledá optimální řešení heuristicky na základě využití nabíjecích stanic v simulaci, další zvolené metody optimalizace je s využitím technik genetického algoritmu a s pomocí algoritmu k-means.

Práce je rozdělena do několika kapitol. První kapitola je zaměřena na popis a definici náročnějších pojmů a metod používaných v práci a jsou zde také zmíněny a popsány související práce. V druhé kapitole popisujeme proces přípravy mapy silniční sítě použité v simulátoru. Třetí kapitola popisuje vlastnosti simulátoru. Ve čtvrté kapitole je pak popsán proces simulace dopravy. V páté kapitole jsou popsány optimalizační metody použité v této práci. V šesté kapitole jsou popsány výsledky analýzy jednotlivých optimalizačních algoritmů. V závěru práce dis-

[1] `https://download.geofabrik.de/`

kutujeme výsledky práce a diskutujeme možnosti pokračování práce. V příloze pak nalezneme uživatelskou, programátorskou dokumentaci simulátoru a popis přílohy programu.

Kapitola 1

Související práce a pojmy

V této části popisujeme komplikovanější teoretické koncepy používané v práci a práce s podobným zaměřením.

1.1 Zeměpisné souřadnice a sférická vzdálenost

Zeměpisné souřadnice slouží k určování polohy na povrchu Země. Nejčastěji jsou udávány jako trojice *zeměpisná šířka* (anglicky latitude), *zeměpisná délka* (anglicky longitude) a nadmořská výška. V naší práci využíváme první dva údaje (tedy zeměpisnou šířku a délku). K určení vzdáleností mezi body na sféře u nichž známe jejich zeměpisné souřadnice se používá takzvaná *haversinová formule*. Podrobněji popisuje problematiku Chang [3].

1.2 Genetický algoritmus

Genetický algoritmus (GA) je heuristický algoritmus, jehož cílem je pomocí aplikace principů evoluční biologie nalézt řešení problémů, pro něž neexistuje použitelný exaktní algoritmus. Existuje mnoho variant GA a popis všech by byl vyčerpávající. Popíšeme tedy pouze variantu používanou v této práci. Podrobnější popis genetického algoritmu popisuje Mitchell [4].

Princip řešení problému je založen na evoluci generací jedinců, kde každý z nich reprezentuje jedno řešení daného problému. První generace je typicky inicializována náhodně. Každý jedinec je ohodnocen *fitness funkcí*, která vyjadřuje kvalitu řešení. Na základě této funkce jsou následně stochasticky vybráni (tzv. selekce) jedinci (rodiče), jež jsou pomocí tzv. *genetických operátorů*, mezi něž patří *křížení* a *mutace*, modifikováni v nové jedince následující generace. Tento postup je iterativně opakován s očekáváním zlepšující se kvality řešení a je ukončen

po dosažení zvolené ukončovací podmínky (typicky dostatečná kvalita řešení, dosažení maximálního počtu iterací, malá změna fitness mezi generacemi apod.).

Individua (rodiče) pro křížení vybíráme tzv. *turnajovou selekcí*, v níž nejprve náhodně zvolíme 2 jedince a z nich vybereme se zvolenou vyšší pravděpodobností jedince s lepší fitness. Takto dostaneme individua, na nichž budeme následně aplikovat genetické operátory. Část nové populace vznikne zkopírováním zvoleného počtu jedinců s nejlepším ohodnocením z aktuální generace do nové generace. Takto je zajištěno, že v nové generaci bude vždy zastoupen jedinec s aktuálně nejlepším ohodnocením. Selekce vybírá tolik jedinců, aby nová generace obsahovala stejný počet jedinců, jako to předešlá.

Po selekci rodičů následuje křížení, jehož výsledkem jsou dva noví jedinci vzniklí kombinací obou rodičů. V naší práci volíme *jednobodové křížení*, kde je zvolen bod v jedinci, jenž určuje část jedince, která je vyměněna mezi rodiči. Předpokladem jednobodového křížení je reprezentace jedince posloupností parametrů (aby bylo možné zvolit bod v posloupnosti, od něhož je zbylá část vyměněna s částí druhého rodiče).

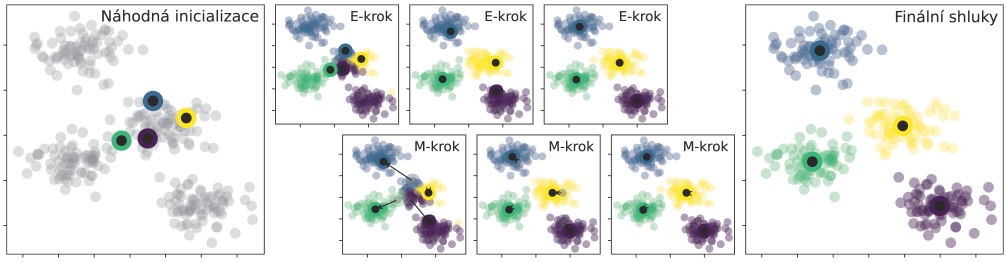
Na nových jedincích vzniklých křížením je následně aplikována mutace. Zde jsou s malou pravděpodobností náhodně změněny jednotlivé části jedince (tzv. geny). Cílem mutace je především zabránit vzniku jednotvárné populace.

Algoritmus 1 Pseudokód genetického algoritmu.

```
function Genetický algoritmus
    Náhodná inicializace populace.
    while ukončovací podmínka není splněna do
        Zvol jedince pro další generaci (selekce).
        Zkombinuj zvolené jedince (křížení).
        Náhodně pozměň nové jedince (mutace).
    end while
end function
```

1.3 K-Means

Algoritmus *k-means* je příkladem *shlukovací metody*. Předpokládáme, že shlukované objekty lze chápat jako body v metrickém prostoru a počet shluků je pevně zvolen na začátku algoritmu (ozn. jako k). Každý shluk je reprezentován *centroidem*. Jedná se o bod v prostoru, jenž typicky reprezentuje střed daného shluku bodů. Pozice centroidů v první iteraci algoritmu jsou typicky voleny náhodně, nebo pomocí vhodně zvolené heuristiky (jako ochrana před konvergencí k nesprávnému lokálnímu optimu). V průběhu algoritmu jsou objekty přiřazovány do



Obrázek 1.1 Příklad průběhu algoritmu k-means. Černé kruhy značí centroidy, barevné lemování slouží k jejich odlišení. Barva jednotlivých bodů značí náležení bodu do shluku centroidu s příslušnou barvou.

shluku, jehož centroid je nejblíže od objektu. Tento krok označujeme jako *E-krok*. Po rozdělení objektů do shluků je aktualizována pozice centroidu daného shluku, tak aby ležel v těžišti shluku (typicky se jedná o průměrnou pozici všech objektů ve shluku). Tento krok se typicky označuje jako *M-krok*. Po aktualizaci centroidů jsou body přiřazeny do nových shluků a postup výše je iterativně opakován do doby, než se poloha centroidů ustálí.

V reálných aplikacích často nevíme, jaké k zvolit. Tento problém se typicky řeší spuštěním algoritmu s různými hodnotami k a na základě požadovaných parametrů je následně zvolena nejlepší hodnota k. Podrobněji popisuje problematiku MacQueen [5].

Algoritmus 2 Pseudokód algoritmu K-Means.

```
function K-Means
    Náhodná inicializace centroidů.
    while změna pozice centroidů do
        Roztřiď objekty do nejbližších shluků.
        Spočítej novou pozici centroidů.
    end while
end function
```

1.4 A-star

A-star (A*) je algoritmus pro vyhledávání optimálních cest v kladně ohodnocených grafech. Jedná se o modifikaci Dijkstrova algoritmu s přidaným heuristickým prvkem.

Výpočet algoritmu je téměř totožný s Dijkstrovým algoritmem, od něhož se odlišuje ve výběru následujícího vrcholu pro otevření, který volí na základě funkce

$f(x) := g(x) + h(x)$. Funkce $g(x)$ zde označuje reálnou (nalezenou) vzdálenost od počátečního bodu do bodu x (Dijkstrův algoritmus pracuje pouze s touto hodnotou). Funkce $h(x)$, pak značí heuristickou funkci odhadující vzdálenost vrcholu x od požadovaného cílového vrcholu. Pokud je heuristická funkce vhodně zvolena, je možné znatelně omezit počet operací pro nalezení nejkratší cesty mezi dvěma vrcholy v porovnání s Dijkstrovým algoritmem.

Typicky je u heuristické funkce h požadována tzv. *přípustnost* (nenahodnocuje vzdálenost do cíle), či *monotónost* (hodnota funkce h roste po přechodu na další hranu), více informací o tomto algoritmu popisuje Russell [6].

1.5 Související práce

V této sekci si popíšeme několik souvisejících prací a stručně popíšeme jejich vlastnosti.

Simulátor SUMO[1] je velmi detailní simulátor dopravy v rozsahu funkcionalit zdaleka přesahující naši implementaci. V programu lze do jisté míry simulovat také elektromobily. Pro naše učely je ale tento program velmi komplexní a práce s optimalizačními algoritmy by tak byla neprakticky komplikovaná.

Simulátor City Flow[2] se pyšní především výpočetní efektivitou[3]. Bohužel nenabízí jednoduché uživatelské rozhraní pro rozšíření o metody potřebné pro naši optimalizaci.

V práci Hiwatari, Ikeya a Okano [7] je navržen simulátor dopravy pro následnou analýzu nově navrženého optimalizačního algoritmu rozmístění nabíjecích stanic na území Japonska. Optimalizační algoritmus v práci se snaží minimalizovat počet vozidel, kterým se vybila baterie, pomocí posunů nabíjecích stanic směrem k místům, kde se v minulosti vybila baterie vozidla. Návrh simulátoru naší práce a optimalizační algoritmus s použití metody k-means jsou inspirovány zmíněnou prací.

V práci Niccolai, Bettini a Zich [8] je zkoumáno několik variant evolučních algoritmů pro optimalizaci rozmístění nabíjecích stanic elektrických vozidel ve městě Milán. V práci byl porovnáván také hladový přístup rozmísťující iterativně nabíjecí stanice na místa lokálních extrémů specificky zadefinované ztrátové funkce.

Práce Zhu et al. [9] se zabývá optimalizací rozmístění nabíjecích stanic v okolí města Peking užitím technik genetického algoritmu. V práci jsou porovnávány dvě varianty modelů popisující možné pozice nabíjecích stanic a je v ní poukázáno, že správná volba modelu ovlivňuje kvalitu řešení optimalizace.

[1] `https://www.eclipse.org/sumo/`

[2] `https://cityflow.readthedocs.io/en/latest/index.html`

[3] `https://cityflow.readthedocs.io/en/latest/introduction.html`

V práci Kınay, Gzara a Alumur [10] je navžen simulátor pro rozvrhování pozic nabíjecích stanic. Problém je zde matematicky popsán. V práci jsou navrženy dvě varianty optimalizací. Jedna minimalizující celkovou cenu rozmístění nabíjecích stanic s minimalizací odchylek od původní trasy, druhá optimalizuje pozice stanic s ohledem na minimalizaci odchylek.

Kapitola 2

Příprava mapy silniční sítě

Hlavní motivací pro vytvoření simulátoru dopravy je vytvoření uživatelsky přívětivého prostředí pro práci s různými variantami rozmístění nabíjecích stanic na reálné dopravní síti. Je tedy více než žádoucí pracovat s reprezentací reálné silniční sítě. K tomuto účelu používáme volně dostupné mapy formátu `.osm.pbf` ze serveru Geofabric[1].

V této kapitole popisujeme jakým způsobem lze zmíněné mapy upravit do formátu požadovaného naším simulátorem a poukazujeme na důležité poznatky této problematiky. V našich experimentech pracujeme s reprezentací mapy České republiky[2], která je přiložena k práci v již požadovaném formátu simulátoru a pro správné spuštění simulátoru tak postupy popsané v této kapitole nejsou nutné.

2.1 Extrakce dopravní sítě

V této sekci popisujeme proces extrakce dopravní sítě z mapy formátu `.osm.pbf` do formátu požadovaného našim simulátorem a poukazujeme na specifické aspekty námi zvolené mapy pro analýzu optimalizačních metod.

2.1.1 Extrakce s pomocí nástroje Osmium

Soubory formátu `.osm.pbf` obsahují mimo informace o dopravní síti, také velké množství informací irelevantních pro funkci našeho simulátoru. Důsledkem této skutečnosti je několikanásobně vyšší množství dat pro zpracování, což mimo jiné vede k vyšší paměťové i časové náročnosti programu. Je tak žádoucí všechny nepotřebné informace odstranit. K tomuto účelu používáme nástroj `Osmium`[3]. S

[1] `https://download.geofabrik.de/`

[2] `https://download.geofabrik.de/europe/czech-republic.html`

[3] `https://osmcode.org/osmium-tool/`

jehož pomocí jsme schopni z mapy vyextrahovat silniční síť a zároveň vybrat pouze námi zvolené druhy silnic.

Dodatek k výběru druhů silnic

Mezi zvolené typy silnic, které náš simulátor rozeznává, patří dálnice, silnice pro motorová vozidla a silnice první třídy (v kontextu dat se jedná o typy `motorway`, `trunk` a `primary`). Silnice nižších tříd jsme zanedbali, neboť předpokládáme, že výše zmíněné silnice tvoří souvislý graf, případně, že jsou rozděleny do relativně nízkého počtu komponent souvislosti a dostatečně aproximují zkoumanou dopravní síť. S ohledem na skutečnost, že v reálném případě je silniční síť v naprosté většině případů souvislý graf, je žádoucí minimalizovat rozpad dopravní sítě na komponenty souvislosti. V našem případě je graf rozdělen do 75 komponent souvislosti, což se jeví jako poměrně vysoké číslo a vyvolává pochybnosti, zda je naše volba typů silnic vhodná. Vysoký počet komponent si odůvodňujeme výběrem mapy pokrývající pouze výřez reálné oblasti, což může vézt k rozpadu silniční sítě na více komponent souvislosti především v hraničních oblastech.

Významným faktorem podněcujícím k eliminaci silnic nižší třídy je fakt, že s klesající třídou silnic typicky několikanásobně roste jejich počet. Výsledný graf je pak mnohonásobně hustší, což vede k vyšší výpočetní náročnosti celého simulátoru, případně na méně výkonných strojích k nepoužitelnosti simulátoru z časových i paměťových důvodů. V našem případě vzrostl počet hran grafu při zahrnutí silnic 2. třídy asi pětinásobně.

Předpokládáme, že neexistuje mnoho hustě osídleným oblastí v simulované oblasti, které bychom takto vyloučili ze simulace. Jelikož předpokládáme, že hustěji osídlené oblasti mají v blízkém okolí přístup k silnicím vyšších tříd. Nevýhodou tohoto přístupu je zanedbání realistické reprezentace sítě na úrovni jednotlivých měst, či malých oblastí. Nemůžeme tak dostatečně realisticky simulovat například průjezdy vozidel křižovatkou, dopravní zácpy ve městech apod. Zároveň jsme omezeni na cesty vozidel vedoucích pouze po silnicích vyšší třídy. Předpokládáme však že tato omezení nejsou s ohledem na námi řešený problém nijak závažná, neboť nás zajímají především dálkové cesty, během nichž je častější potřeba řidiče použít nabíjecí stanici.

Extrakce měst

Pro dosažení co nejrealističtějších výsledků simulace je potřeba vhodně generovat počáteční a cílové pozice vozidel. K tomuto účelu nám slouží informace o pozicích a počtu obyvatel osídlených oblastí na mapě. Pro získání těchto informací nám znovu poslouží nástroj `Osmium`. S jeho pomocí získáváme informace o městech (oblasti označené jako `city` nebo `town`).

Vynechali jsme menší osídlené oblasti, jelikož předpokládáme, že počet výjezdů a cílů cesty je v této oblastí zanedbatelný. Navíc je počet menších celků vysoký a jejich zahrnutí by vyžadovalo více výpočetního výkonu.

2.1.2 Příprava dat pomocným skriptem

Postupy popsanými výše jsme získané informace zásadně zredukovali. Veškerá vyfiltrovaná data jsou ale stále ve formátu `osm.pbf`, který musíme vhodným způsobem zpracovat a odstranit nepotřebné informace o silnicích a obytných oblastech. K tomuto účelu jsme využili knihovny programovacího jazyka Python `Pyrosm`[4]. A naimplementovali program v jazyce Python, jenž načte data ze souborů ve formátu `osm.pbf` a náležitě je upraví do podoby vstupních souborů našeho simulátoru dopravy. Podrobnější informace k přípravě dat i veškeré programy jsou dodány v příloze v adresáři `map_processing`.

2.2 Dodatek k volbě programovacího jazyka a nutnosti předpřípravy dat

Pro správné fungování simulátoru není práce s nástroji popsanými v sekci (sekce 2.1). nutná (je například možné pracovat s ručně připravenou reprezentací dopravní sítě a nerešit jakoukoli přípravu dat z reálných map).

Reprezentace mapy ve formátu `osm.pbf` je obecně velmi rozšířená, což má za následek existenci početné skupiny knihoven (především v jazyce Python), jež umí s těmito soubory pracovat a vhodným způsobem je zpracovávat. Zároveň existuje řada knihoven programovacího jazyka Python, které nabízejí širokou škálu funkcionalit pro práci s grafy a grafovými algoritmy. Může nám tedy připadat kontraproduktivní pracovat s několika různými nástroji pro předpřípravu dat a pracovat s více programovacími jazyky, když lze většinu funkcionalit pokrýt programem napsaným v jazyce Python s využitím vhodných knihoven. Původním cílem práce bylo zvolit tento přístup. Během práce na simulátoru jsme však narazili na vysokou paměťovou i časovou náročnost tohoto postupu a upustili jsme od něj. Na základě těchto skutečností jsme se nakonec rozhodli v jazyce Python pouze předzpracovat data a zbytek simulátoru naimplementovat v jazyce C++ s využitím knihovny `boost`[5].

[4] `https://pyrosm.readthedocs.io/en/latest/`
[5] `https://www.boost.org/`

Kapitola 3

Popis simulátoru

V této kapitole definujeme řadu důležitých pojmů pro správný popis simulátoru a dále popisujeme veškeré vlastnosti a funkcionality simulátoru.

Hlavní cílem návrhu simulátoru je uživatelská přívětivost v rámci analýzy různých variant optimalizačních algoritmů. S ohledem na námi řešený problém je kladen důraz především na rozšiřitelnost programu o nové optimalizační techniky a snadnou nastavitelnost parametrů úzce souvisejících s vlastnostmi nabíjecích stanic, mezi něž patří například počet nabíjecích stanic, rozmístění stanic, počet nabíjecích slotů na stanicích a podobně.

3.1 Poznámka ke značení a definicím

V následujícím textu definujeme řadu objektů simulace, jenž jsou často reprezentovány uspořádanou množinou veličin. Často budeme potřebovat specifikovat jednotlivé složky těchto objektů. Pro zjednodušení notace, tak budeme zapisovat parametry ve formátu `objekt.jméno_parametru`.

Tedy pokud například definujeme objekt o jako uspořádanou množinu (x, y, z). Pak parametr y objektu o zapisujeme ve formátu $o.y$.

Dále je potřeba zmínit, že některé definice uvedené v následujícím textu, jsou zjednodušeny a jejich zápis nemusí být matematicky naprosto přesný. Jsou například vynechány nějaké předpoklady z definic, které zřejmě vyplývají z textu. K tomuto kroku bylo přistoupeno pod snahou zpřehlednit jednotlivé definice.

3.2 Předpoklady návrhu modelu

S ohledem na výpočetní efektivitu simulátoru a uživatelskou přívětivost je potřeba náležitě zjednodušit simulovaný model.

Dopravní síť reprezentujeme vhodným grafem rozšířeným o potřebné parametry. Simulátor je navržen pro simulaci dopravních sítí pokrývající větší územní celky, které jsou rozlohou a hustotou sítě srovnatelné s dopravní sítí České republiky. Z tohoto důvodu není zaručeno vhodné chování simulace na dopravních sítích s výrazně odlišnými vlastnostmi.

Při volbě velikosti simulované oblasti jsme předpokládali, že většina majitelů vozidel má možnost dobíjet své vozy v počáteční a cílové destinaci (např. mají nabíjecí sloty v místě bydliště, či na pracovišti). Pro cesty na krátké vzdálenosti tak budou schopni dobít své automobily na dostatečnou hladinu baterie a během cesty nebude potřeba použít nabíjecí stanici.

Na základě tohoto předpokladu je kladen důraz především na realistické simulování dálkových tras vozidel. V zájmu výpočetní efektivity simulátoru a zkoumaného problému opomíjíme většinu cest na krátkou vzdálenost a naopak nadhodnocujeme počet dálkových tras vozidel. V simulátoru je nastavitelná střední hodnota vzdálenosti trasy, čímž můžeme míru tohoto předpokladu náležitě regulovat.

Důsledkem zjednodušení je v simulaci zanedbána část vlastností reálné dopravní sítě. Například hustota provozu či skutečnost, že v reálném případě budou nabíjecí stanici využívat také řidiči jedoucí na krátké vzdálenosti tak nebude v simulaci dostatečně zahrnuta.

Protože je v reálném případě dopravní síť značně komplexní (především na úrovni silnic nižší třídy), je potřeba síť řádně zjednodušit. Tato problematika je podrobněji popsána v kapitole o přípravě mapy viz. kapitola 2.

3.3 Dopravní síť

Řada myšlenek použitých pro definici modelu silniční sítě je inspirována návrhem modelu v práci Hiwatari, Ikeya a Okano [7].

Dopravní síť je v zájmu efektivity a jednoduchosti simulace reprezentována vhodným grafem. K popisu grafu potřebujeme nejprve zadefinovat několik pomocných pojmů a objektů simulátoru.

Prvním pojmem je *křižovatka*, jež zastupuje podmnožinu vrcholů v grafu silniční sítě, které reprezentují křižovatky v reálné silniční síti.

Definice 1 (Křižovatka)**.** Křižovatka *je uspořádaná čtveřice* (n, c, m, d)*, kde:*

- $n \in \mathbb{N}_0$ - *identifikátor*
- $c \in \mathbb{R}^2$ - *zeměpisné souřadnice (viz. sekce 1.1)*
- $m \in \mathbb{N}_0$ - *identifikátor nejbližšího města od křižovatky*

- $d \in \mathbb{R}$ - *vzdálenost vrcholu od středu města s identifikátorem* m

Dále definujme *typy silnice* používané simulátorem, s jejichž pomocí můžeme v simulaci dopravy rozlišovat mezi různými kapacitami jednotlivých úseků dopravní sítě (např. dálnice typicky efektivněji pokryjí hustý provoz než silnice 1. třídy).

Definice 2 (Množina typů silnice)**.** Množina typů silnice *je množina* $\{m, t, p, o\}$ *všech typů silnice používaných v simulaci dopravní sítě, kde:*

- m - *dálnice*
- t - *silnice pro motorová vozidla*
- p - *silnice 1. třídy*
- o - *ostatní typy silnic*

Segmenty silnic reprezentují vybrané body v silniční síti. A slouží jako vrcholy grafu.

Definice 3 (Segment silnice)**.** Segment silnice *je uspořádaná dvojice* (n, b)*, kde:*

- $n \in \mathbb{N}_0$ - *identifikátor segmentu (určuje přesnou pozici)*
- $b \in [0, 1]$ - *průměrná hladina baterie vozidel v místě segmentu*

Silnice reprezentuje část silniční sítě mezi 2 křižovatkami, jenž neprochází žádnou křižovatkou. Tento objekt shlukuje všechny segmenty silnice mezi dvojicemi křižovatek.

Definice 4 (Silnice)**.** *Nechť* K *značí množinu všech křižovatek v silniční síti a* T *značí množinu typů silnic. Zároveň předpokládejme, že* $k_1 < k_2$*,*

Pak silnice nad množinou křižovatek K *je uspořádaná pětice* (k_1, k_2, d, t, S)*, kde:*

- $k_1, k_2 \in K$ - *křižovatky na koncích silnice*
- $d \in \mathbb{R}^+$ - *délka segmentu silnice*
- $t \in T$ - *typ*
- S - *množina segmentů silnice, pro kterou je splněna podmínka:*

$$(\forall s, r \in \mathbb{N}; r < s)(s \in S \implies r \in S)$$

Nyní již můžeme zadefinovat *graf dopravní sítě*, s nímž pracujeme v naší simulaci dopravy.

Definice 5 (Graf dopravní sítě)**.** *Mějme množinu křižovatek K, množinu silnic R nad množinou křižovatek K a $d \in \mathbb{R}$, pro které jsou splněny podmínky:*

1. $$(\forall r_1, r_2 \in R; r_1 \neq r_2)\,(|r_1.S \cap r_2.S| \leq 1)$$

2. $$(\forall r_1, r_2 \in R)\,(\{r_1.k_1, r_1.k_2\} \cap \{r_2.k_1, r_2.k_2\} = \emptyset \implies r_1.S \cap r_2.S = \emptyset)$$

3. $$(\forall r \in R)(r.d = d)$$

Definujme množinu segmentů všech silnic z R:

$$V = \bigcup_{r \in R} r.S$$

Následně definujme množinu sousedních dvojic segmentů silnice $r \in R$ jako:

$$E_r = \{(s_1, s_2) \in (r.S)^2; |s_1.n - s_2.n| = 1\}$$

A množinu sousedních dvojic segmentů všech silnic z R jako:

$$E = \bigcup_{r \in R} E_r$$

Pak graf dopravní sítě $G = (V, E)$ *je uniformně ohodnocený souvislý graf s vrcholy V a hranami E délky d.*

3.4 Objekty dopravní sítě

3.4.1 Segmenty

Segmenty silnice z definice 3 jsou nejmenší stavební jednotkou celého simulátoru. Reprezentují vrcholy grafu silniční sítě a slouží k diskretizaci reálné silniční sítě, která je s jejich pomocí rozdělena na uniformně dlouhé úseky. Uchovává informaci o průměrné hladině baterie v jednotlivých usecích dopravní sítě, která je využívána v optimalizaci.

3.4.2 Křižovatky

Křižovatky z definice 1 zastupují podmnožinu vrcholů grafů sítě, stupně vyššího než 2. Společně s městy se jedná o jediné objekty simulátoru, jenž si uchovávají zeměpisnou polohu v reálné dopravní síti.

Za pomocí těchto objektů jsou v simulátoru významným způsobem optimalizovány algoritmy pro grafové hledání optimální cesty. Jsou používány k redukci vrcholů a hran prohledávaného grafu, k heuristickému prohledávání stavového prostoru při hledání nejkratších cest, či vhodných nabíjecích stanic. Jsou také hojně používány v optimalizačních algoritmech pro rozmístění nabíjecích stanic.

3.4.3 Silnice

Silnice z definice 4 sdružují podmnožinu hran grafu ležící mezi 2 křižovatkami a neprocházející jinou křižovatkou. Jsou potřebné pro zadefinování hran grafu simulátoru. Uchovávají informaci o typu silnice, jsou používány pro hledání nejkratší cesty v grafu a jsou používány k simulování hustoty provozu v různých částech dopravní sítě.

V simulátoru předpokládáme, že délky všech silnic dopravní sítě jsou dělitelné délkou segmentu. To však v reálném případě typicky nenastane. V popisu reálné silniční sítě pracujeme pouze s přibližnou délkou jednotlivých silnic. Tato skutečnost však výrazně neovlivňuje námi požadované vlastnosti simulace.

Hustota provozu

Silnice je používána pro simulování hustoty provozu v jednotlivých úsecích dopravní sítě. V simulátoru je zaimplementována jednoduchá logika pracující s hustotou provozu v závislosti na typu (viz. definice 2) a délce silnice.

Princip simulování hustoty provozu je založen myšlence *kapacity silnice.*

Definice 6 (Kapacita silnice)**.** *Mějme d délku hran grafu dopravní sítě G z definice 5. Nechť R je množina silnic G. Dále $f : T \to \mathbb{R}$, kde T je množina všech typů silnic z definice 4 a $\alpha, \beta \in \mathbb{R}$.*

Kapacita silnice *je funkce $c : R \to \mathbb{R}^+$, definována jako:*

$$c(r) := \alpha \cdot f(r.t) + \beta \cdot d \cdot |r.S|$$

V simulátoru byly hodnoty α, β zvoleny na základě empirické volby. Simulování hustoty provozu je pak odvozeno od aktuálního počtu vozidel na silnici. Pomocí jednoduché funkce *relativní délky silnice.*

Definice 7 (Relativní délka silnice)**.** *Mějme d délku hran grafu dopravní sítě G. Silnici r grafu G, jejíž kapacita je $c_r \in \mathbb{R}^+$ a nechť $n_r \in \mathbb{N}_0$ je aktuální počet vozidel na silnici r. Dále funkce zpoždění vozidla $f : \mathbb{R} \to \mathbb{R}^+$.*

Relativní délka silnice r *je definována, jako* $t_r : \mathbb{R}^+ \times \mathbb{N}_0 \to \mathbb{R}^+$, *kde:*

$$t_r(n_r) = \begin{cases} |r.S| & pro\ n_r \leq c_r \\ |r.S| \cdot f(n_r - c_r) & pro\ n_r > c_r \end{cases}$$

Doba průjezdu vozidla po silnici se pak vypočítá jako doba potřebná pro ujetí vzdálenosti d, která je rovna relativní délce silnice, vozidlem s pevně zvolenou průměrnou rychlostí v (danou parametry simulátoru). Simulátor vždy aktualizuje hodnotu relativní délky silnice po změně počtu vozidel na silnici.

3.4.4 Města

Důležitým prvkem simulátoru je zakomponování vlivu zalidnění simulovaných oblastí. K tomuto účelu nám slouží objekt *město*.

Definice 8 (Město)**.** Město *je uspořádaná trojice* (n, c, p), *kde:*

- $n \in \mathbb{N}_0$ - *identifikátor*
- $c \in \mathbb{R}^2$ - *zeměpisné souřadnice centra města (viz. sekce 1.1)*
- $p \in \mathbb{N}_0$ - *počet obyvatel města*

Každému městu náleží příslušná množina křižovatek definována jako:

Definice 9 (Množina křižovatek města)**.** *Mějme město* m *v simulátoru a množinu všech křižovatek* K *grafu* G, *pak* množina křižovatek města *je:*

$$K_m = \{k \in K | k.m = m.n\}$$

Koncept měst je důležitý pro realističtější generování počátečních a koncových pozic vozidel v simulaci. Slouží také pro heuristický výběr vhodné nabíjecí stanice pro nabití vozu.

3.4.5 Nabíjecí stanice

Nabíjecí stanice jsou vzhledem ke zkoumanému problému důležitým prvkem simulátoru.

Definice 10 (Nabíjecí stanice)**.** Nabíjecí stanice *v grafu dopravní sítě* $G = (V, E)$ *je uspořádaná čtveřice* (n, m, v, c), *kde:*

- $n \in \mathbb{N}_0$ - *identifikátor*
- $m \in \mathbb{N}_0$ - *identifikátor nejbližšího města od stanice*

- $v \in V$ - *pozice stanice (vrchol v grafu dopravní sítě)*
- $c \in \mathbb{N}$ - *kapacita stanice (počet zákazníků, jež stanice pokryje najednou)*

Informace o nejbližším městě je důležitá především pro heuristickou volbu nabíjecí stanice vozidlem v simulaci. V simulátoru je uchovávána informace o aktuálním vytížení stanice a o vozidlech čekajících na volný nabíjecí slot pro simulování čekání automobilů ve frontě. Simulátor si pro každou nabíjecí stanici pamatuje průměrnou hladinu baterie vozidel při příjezdu do stanice, pro použití v optimalizačních algoritmech. V návrhu simulátoru předpokládáme, že zákazníci stanice dobíjejí své vozidlo na plnou kapacitu baterie. Návrh simulátoru však umožňuje rozšíření o nabíjení pouze na zvolenou hladinu baterie bez významnějších zásahů do původní implementace.

3.5 Vozidla

Vozidla jsou nejdůležitějším prvkem simulace. Jednotlivá vozidla jsou simulátorem spravována samostatně. Každému vozidlu je při generování přiřazena počáteční a cílová pozice a počáteční hladina baterie.

3.5.1 Generování vozidel

Vozidla jsou v simulaci generována, aby co nejlépe aproximovala reálný provoz vozidel. Doba čekání mezi výjezdy vozidel je náhodně generována z exponenciální distribuce s nastavitelnými parametry. Počáteční hladina baterie vozidla je vygenerována z normálního rozdělení s pevně zvolenými parametry. Definujeme také horní a dolní hranici hladiny počáteční baterie. Pokud vygenerovaná hladina překračuje jednu z těchto hranic, pak je nahrazena hodnotou překročené hranice. Tento krok slouží jako ochrana pro vygenerování hladiny baterie pouze v rozmezí $[0, 1]$.

Počáteční i cílová pozice vozidla je generována několikafázově z kombinace více náhodných distribucí.

Při generování pozice výjezdu je nejprve generováno město m_1 výjezdu s pravděpodobností úměrnou jeho počtu obyvatel (čím více obyvatel, tím je výjezd vozidla pravděpodobnější). Tedy:

Definice 11 (Pravděpodobnost výjezdu z města)**.** *Zvolme množinu M všech měst v simulátoru. Pak* pravděpodobnost výjezdu *vozidla z města $m \in M$, je rovna:*

$$p_m = \frac{m.p}{\sum_{m_i \in M} m_i.p}$$

Po vygenerování počátečního města m_1 je uniformně náhodně zvolena křižovatka k_1 z množiny křižovatek města m_1 (viz. definice 9). Následně je uniformně náhodně vybrána silnice r_1 napojená na křižovatku k_1. Neboli silnice r_1, pro kterou platí $k_1 \in \{r_1.k_1, r_1.k_2\}$. Nakonec je uniformně náhodně zvolen segment v_1 (vrchol grafu silniční sítě) silnice r_1, který již jednoznačně popisuje pozici výjezdu vozidla.

Generování cílové pozice vozidel je realizováno podobně s rozdílem generování koncového města. Při náhodném výběru koncového města je pravděpodobnost výběru města ovlivněna počtem obyvatel města a vzdáleností od počátečního města.

Proces generování cílového města probíhá následovně. Nejprve je spočítána pravděpodobnost výjezdu vozidla pro všechna města v simulátoru (viz. definice 11). Dále je pro počáteční město m_1 a každé město $m_i \in M$ simulátoru vypočítána jejich geografická vzdálenost (viz. sekce 1.1), kterou označíme jako $d_{1,i}$. Následně pro všechna města $m_i \in M$ vypočítáme pravděpodobnost vzdálenosti $d_{1,i}$, kterou si označíme $p_{1,i}$. Pravděpodobnost vzdálenosti $p_{1,i}$ vypočítáme s pomocí nastavitelné exponenciální distribuce popisující vzdálenost cílové pozice od počáteční pozice. Pravděpodobnost cílového města $m_2 \in M$ vůči počátečnímu městu $m_1 \in M$ spočítáme následovně:

Definice 12 (Pravděpodobnost cílového města)**.** *Mějme $\alpha \in \mathbb{R}^+$ a počáteční město $m_1 \in M$. Označme vzdálenost města m_1 od města $m_2 \in M$ jako $d_{m_1,m_2} \in \mathbb{R}$. Označme pravděpodobnost výjezdu z města m_2 jako p_{m_2} a pravděpodobnost vzdálenosti d_{m_1,m_2} jako p_{m_1,m_2}, kterou dostaneme z námi zvolené exponenciální distribuce (stejná notace platí pro libovolné město $m_i \in M$). Pak pravděpodobnost $p_{m_1,m_2{}_{final}}$, že město m_2 je cílovým městem vozidla s počátkem ve městě m_1 je rovna:*

$$p_{m_1,m_2{}_{final}} = \frac{p_{m_2} + \alpha \cdot p_{m_1,m_2}}{\sum_{m_i \in M} p_{m_i} + \alpha \cdot p_{m_1,m_i}}$$

Cílové město $m_2 \in M$ vygenerujeme z distribuce měst dané pravděpodobnostmi cílového města vůči počátečními městu m_1 (viz. definice 12). Proces generování přesné cílové pozice ve městě m_2 je totožný s procesem generování přesné počáteční pozice ve zvoleném městě $m \in M$.

3.5.2 Hledání trasy vozidla

Po vygenerování vozidla musíme v simulátoru vyřešit také otázku naplánování trasy vozidla, které je realizováno algoritmem A-star (viz. sekce 1.4), který jako heuristickou funkci volí zeměpisnou vzdálenost (viz. sekce 1.1) mezi křižovatkami.

S ohledem na efektivitu výpočtu pracujeme v simulaci při hledání cesty se *zjednodušeným grafem dopravní sítě* vůči počátečnímu vrcholu v_1 a cílovému vrcholu v_2.

Algoritmus 3 Postup při náhodném generování pozice ve městě (vrcholu v grafu).

function GENEROVÁNÍ NÁHODNÉ POZICE(m - město)
Uniformně náhodně vygeneruj křižovatku k ve městě m.
Uniformně náhodně vygeneruj silnici r napojenou na křižovatku k.
Uniformně náhodně vygeneruj segment v na silnici r.
return v
end function

Algoritmus 4 Pseudokód procesu generování nového vozidla v simulaci.

function GENEROVÁNÍ VOZIDLA
Z normálního rozdělení vygeneruj počáteční hladinu baterie b.
V závislosti na počtu obyvatel náhodně vygeneruj počáteční město m_1.
$v_1 \leftarrow$ GENEROVÁNÍ NÁHODNÉ POZICE(m_1)
V závislosti na počtu obyvatel a vzdálenosti of města m_1 náhodně vygeneruj koncové město m_2
$v_2 \leftarrow$ GENEROVÁNÍ NÁHODNÉ POZICE(m_2)
Z exponenciální distribuce vygeneruj čas t čekání na výjezd vozidla.
$vehicle \leftarrow$ vozidlo s počáteční hladinou baterie b, počátečním vrcholem v_1 a cílovým vrcholem v_2.
return $vehicle, t$ ▷ vygenerované vozidlo a čas čekání na jeho výjezd.
end function

Definice 13 (Zjednodušený graf dopravní sítě)**.** *Mějme graf dopravní sítě* $G = (V, E)$ *(viz. definice 5) s množinou křižovatek* K *a množinou silnic v síti* R*. Zvolme počáteční* $v_1 \in V$ *a koncový vrchol* $v_2 \in V$ *a silnice* $r_1 \in R$*, resp.* $r_2 \in R$*, t.ž.* $v_1 \in r_1.S$*, resp.* $v_2 \in r_2.S$*.*

Zjednodušený graf dopravní sítě *vůči počátečnímu vrcholu* v_1 *a koncovému vrcholu* v_2 *je graf* $H = (V', E')$*, kde:*

- $V' = K \cup r_1.S \cup r_2.S$
- $E' = (R \setminus \{r_1, r_2\}) \cup \{$*hrany na silnicích* r_1 *a* $r_2\}$

Zjednodušený graf dopravní sítě vůči počátečnímu vrcholu v_1 a cílovému vrcholu v_2 (viz. definice 13), je tedy graf, jehož vrcholy jsou křižovatky a hrany jsou silnice, s vyjímkou počáteční a koncové silnice, jejichž reprezentace je ponechána z původní definice 5 grafu dopravní sítě.

Postup plánování trasy vozidla je pak následující. Zvolíme vhodný zjednodušený graf dopravní sítě a hrany grafu ohodnotíme relativní délkou silnice (viz. definice 7), v případě počáteční a koncové silnice tuto hodnotu náležitě rozdělíme mezi jednotlivé hrany na silnici. Následně na tomto grafu spustíme algoritmus A-star pro nalezení nejkratší cesty mezi vrcholy v_1 a v_2 s ohledem na aktuální provoz v dopravní síti.

Algoritmus A-star používá jako heuristickou funkci zeměpisnou vzdálenost křižovatek. Neumíme však efektivně vypočítat tuto vzdálenost pro segmenty (viz. definice 3). Při hledání nejkratší cesty se tedy omezujeme pouze na hledání cest mezi nejbližšími křižovatkami počátečního a koncového vrcholu. Výslednou cestu následně dostaneme řádným rozšířením cesty z počátečního a koncového vrcholu do jejich nejbližších křižovatek. Vyjímkou je cesta uvnitř jedné silnice, kterou nalezneme triviálně.

Vzhledem k výpočetní náročnosti hledání optimální cesty vozidla, hledáme tuto cestu pouze při vygenerování vozidla, či ve speciálních případech, jako je například plánování cesty na nabíjecí stanici, nebo cesty z nabíjecí stanice.

3.5.3 Zajíždění na nabíjecí stanici

Speciální situace nastává v moment, kdy vozidlo potřebuje navštívit nabíjecí stanici. V tomto případě je naplánována cesta z počáteční pozice do heuristicky zvolené nabíjecí stanice postupem popsaným v části sekce 3.5.2. Po řádném nabití vozidla je pak stejným způsobem nalezena cesta ze stanice do cílové pozice.

Rozhodnutí o zajetí na nabíjecí stanici

Každé vozidlo má možnost na základě aktuální hladiny baterie aktualizovat svou trasu tak, aby směřovala k nejbližší nabíjecí stanici. Z důvodu výpočetní náročnosti

přepočítání trasy, může ale každé vozidlo využít této možnosti pouze jednou. Po zajetí vozidla na nabíjecí stanici je mu však tato možnost znovu přidělena.

Jelikož každé vozidlo může pozměnit svou trasu pouze jednou, je důležité zvolit vhodný okamžik přeplánování trasy. Tento okamžik naplánujeme pomocí vhodně nastavitelného normálního rozdělení, ze kterého vygenerujeme hladinu baterie, při které se v budoucnu rozhodneme zda zamíříme na nabíjecí stanici, či ne.

Volby okamžiku rozhodování s pomocí normálního rozdělení je motivován následujícími myšlenkami. Nechceme vyrazit ani plánovat cestu na nabíjecí stanici moc brzy, neboť se může v průběhu cesty podstatně změnit hustota provozu a naplánovaná cesta tedy nebude optimální. Zároveň je cesta na nabíjecí stanici při dostatečně vysoké hladině baterie zbytečná z důvodu malé kapacity baterie, kterou je možno doplnit. Naopak v případě velmi nízké hladiny baterie, při níž aktualizujeme cestu, můžeme narazit na problém, že vozidlo nedojede na nabíjecí stanici před úplným vybitím baterie.

Zda vozidlo v moment aktualizace cesty vyrazí na nabíjecí stanici, rozhodneme na základě očekávané hladiny baterie vozidla v cílové destinaci, kterou vypočítáme z aktuální hustoty provozu a spotřeby vozidla. V simulátoru je nastavitelná nejnižší očekávaná cílová hladina baterie, pro kterou vozidlo ještě nemění svou trasu, tak aby směřovalo na nabíjecí stanici.

Výběr nabíjecí stanice

Vozidlo se musí v okamžiku rozhodnutí vyrazit na nabíjecí stanici rozhodnout, kterou stanici zvolit. Toto rozhodnutí je realizováno heuristicky.

Postup výběru stanice probíhá následovně. Pro každé město (viz. definice 8) je na začátku simulace nalezen námi zvolený počet očekávaných nejbližších nabíjecích stanic. Množinu těchto stanic pro libovolné město m označíme jako C_m.

Postup heuristického výběru nejbližších stanic města je následující. Pro každou stanici c zvolíme její nejbližší křižovatku k (viz. definice 1) a město m_k, do nějž křižovatka k náleží (viz. definice 9). Vzdálenost stanice c od libovolného města m pak spočítáme jako součet $k.d$ a zeměpisné vzdálenosti (viz. sekce 1.1) měst m a m_k.

Následně pak v závislosti na městu m, kterému náleží křižovatka k, která je nejblíže od aktuální pozice vozidla, zvolíme kandidáty na vhodné nabíjecí stanice pro zvolené vozidlo. Tito kandidáti jsou právě stanice C_m. Pro každého kandidáta $c \in C_m$ následně vypočítáme s pomocí relativních délek silnic (viz. definice 7) očekávanou dobu cesty (délka cesty v grafu relativních délek) d_c vedoucí z aktuální pozice přes nabíjecí stanici c do cílové pozice. Zvolená nabíjecí stanice $c_{min} \in C_m$, je taková, že $d_{c_{min}}$ je nejmenší ze všech stanic z C_m.

Algoritmus 5 Proces výběru nabíjecí stanice vozidlem.

function VÝBĚR NABÍJECÍ STANICE(
v_{start} - aktuální pozice vozidla,
v_{end} - cílová pozice vozidla,
$C_{closest}$ - množina množin nejbližších stanic každého města)
    $k \leftarrow$ Nejbližší křižovatka od v_{start}.
    $C_{closest_m} \leftarrow$ Množina z $C_{closest}$ náležící městu s ID $k.m$
    $d_{min} \leftarrow 0$ ▷ Nejkratší očekávaná doba cesty.
    c_{min} ▷ Aktuálně nejlepší stanice pro výběr.
    for all $c \in C_{closest_m}$ **do**
        $d_{in} \leftarrow$ A-STAR(v_{start}, $c.v$) ▷ Očekávaná doba cesty do stanice.
        $d_{out} \leftarrow$ A-STAR($c.v$, v_{end}) ▷ Očekávaná doba cesty ze stanice do cíle.
        if $d_{in} + d_{out} < d_{min}$ nebo c_{min} není definováno **then**
            $c_{min} \leftarrow c$ ▷ Aktualizace nejlepší stanice.
            $d_{min} \leftarrow d_{in} + d_{out}$
        end if
    end for
    return c ▷ Předej vybranou nabíjecí stanici.
end function

Tento zjednodušený přístup hledání vhodné nabíjecí stanice podstatně zefektivňuje simulaci. Neboť výpočetně nejnáročnější operací celé simulace je hledání nejkratších cest v grafu. Omezením výběru stanic rychlost simulace výrazně vzroste.

Nakládání s vybitými vozidly

Simulátor musí řešit také případy, kdy se vozidlům vybije baterie a nemůžou dojet do cíle. V takovém případě je vozidlo vyřazeno ze simulace a dále již nijak neovlivňuje simulaci. Veškeré informace o vozidlech s vybitými bateriemi jsou však zaznamenávány a uchovávány pro algoritmy optimalizující pozice nabíjecích stanic.

3.5.4 Druhy vozidel

Rozhraní simulátoru je přizpůsobeno možnosti rozlišovat mezi různými druhy vozidel, jejichž vlastnosti a chování na silnici se může lišit. V aktuální implementaci je používán jediný typ vozidla pojmenovaný *auto*. Veškeré vlastnosti vozidel ze sekce 3.5, platí pro tento typ vozidla. Tento typ vozidla je specifický svým chováním po dosažení cílové destinace. Poté co auto dorazí do cíle čeká

náhodně dlouhou dobu, jež je dána exponenciální distribucí a následně vyrazí zpět do počáteční pozice.

Očekáváme, že tento typ vozidla vhodně popisuje reprezentativní vzorek vozidel, protože předpokládáme, že většina řidičů se typicky po čase vrací do místa výjezdu.

3.6 Specifika reálné dopravní sítě

V této sekci poukážeme na specifické případy při simulování reálné dopravní sítě, které se odlišují od návrhu simulátoru, a odlišnosti v reálné implementaci simulátoru.

3.6.1 Poznámky k definici modelu dopravní sítě

V obsahu sekce 3.3 popisujeme zjednodušený model dopravní sítě. Část předpokladů, či vlastností zde popisovaných však plně neodpovídá realitě. Tyto nesrovnalosti popisujeme v následujícím textu.

Při popisu křižovatek tvrdíme, že se jedná o vrcholy stupně více než 2. Tento předpoklad je běžně splněn i v reálné silniční síti. V implementaci simulátoru jsou reprezentovány reálnými křižovatkami. Ze sítě simulátoru jsou ale typicky odstraněny silnice nižších tříd, čímž se také v grafu zmenšují stupně vrcholů a může docházet k degeneraci křižovatek na vrcholy stupně 2 (viz. sekce 2.1). Během přípravy mapy se snažíme tyto křižovatky z implementace eliminovat. Nezaručujeme však, že se v síti simulátoru nevyskytují žádné degenerované křižovatky stupně 2.

Tato skutečnost ale zásadně neovlivňuje vlastnosti simulátoru. Hlavní motivací pro zadefinování objektů křižovatek je zefektivnění hledání optimální trasy vozidel (viz. sekce 3.5.2), jež je s přehledem výpočetně nejnáročnější operací v simulátoru. Na reálné dopravní síti je redukce vrcholů stupně 2 nepostradatelná s ohledem na praktickou použitelnost simulátoru na sítích větších územních celků.

Problematika souvislosti grafu

Z definice 5 vyplývá, že graf dopravní sítě musí být souvislý graf. Vlivem výběru pouze části dopravní sítě (více viz. sekce 2.1) však typicky dochází k rozpadu grafu sítě na více komponent souvislosti. Ty musíme před začátkem simulace dopravy sloučit s co nejmenším zásahem do reprezentace dopravní sítě.

Komponenty souvislosti slučujeme jednoduchou heuristickou metodou, založenou na hledání křižovatek nejbližších měst (viz. definice 9) z dvojice různých komponent. Tyto křižovatky následně propojíme silnicí typu s nejnižší kapacitou (viz. definice 2 a definice 6) a délkou danou zeměpisnou vzdáleností (viz. sekce 1.1).

Předpokládáme, že většina komponent souvislosti vznikla vlivem výřezu nebo úpravy mapy v hraničních či v málo osídlených oblastech. Proto předpokládáme, že přidání nové uměle vytvořené hrany zásadně neovlivňuje realističnost simulace reálné dopravní sítě.

Poznámka ke generování vozidel

Při generování vozidel (viz. sekce 3.5.1) pracujeme pouze s hustotou zalidnění oblasti, což ale nutně nemusí realisticky popisovat reálný provoz. V simulaci například nebereme v úvahu, že některé málo zalidněné oblasti mohou být frekventovanější, než ty s vyšší hustotou zalidnění. Může se například jednat o pracoviště mnoha zaměstnanců mimo obytnou oblast.

Při generování cílových měst zároveň předpokládáme, že přesun vozidel mezi velmi vzdálenými městy je málo pravděpodobný, ale zároveň se snažíme eliminovat cesty na krátkou vzdálenost (typicky na území jednoho města) z důvodů zefektivnění simulace a získání relevantních dat potřebných pro náš optimalizační problém.

V rámci realistické simulace je v oblasti generování vozidel vysoký potenciál pro budoucí zlepšení našeho simulátoru. Například můžeme použít metodu využitou v práci autorů Hiwatari, Ikeya a Okano [7], pracující s mírou zaměstnanosti v různých částech simulované oblasti.

Simulátor je však navržen tak, aby bylo možno v budoucnu doimplementovat různé varianty generování vozidel s co nejmenším zásahem do již naimplementovaných variant.

Poznámka k hledání cest

Popis hledání optimální cesty vozidla ze sekce 3.5.2 umožňuje pouze jednu korekci během trasy. K tomuto kroku bylo přistoupeno vzhledem ke snaze simulovat co nejhustší provoz.

Je možné, že se hustota provozu v průběhu cesty vozidla výrazně změní a nalezená trasa vozidla se bude výrazně lišit od reálně nejoptimálnější trasy. Vzhledem k malé hustotě simulovaného provozu, který zásadním způsobem omezujeme pouze na dálkové trasy (viz. sekce 3.2). Předpokládáme, že tato situace nebude nastávat velmi často.

Dalším odůvodněním přístupu plánování cesty je i skutečnost, že většina řidičů plánuje své cesty před výjezdem a jen v omezené míře zásadně upravují svou trasu v průběhu jízdu, aby byla co nejoptimálnější. Jelikož většina řidičů není schopna tento velice komplexní problém efektivně vyřešit v průběhu cesty. Jsme si vědomi možnosti používání široké škály navigačních programů, jež jsou schopny informovat řidiče o dopravní situaci a aktualizovat optimální cestu.

Předpokládáme však, že tyto zmíněné systémy nejsou neomylné a často není navržená cesta nejoptimálnější a náš zjednodušený přístup plánování cesty není výrazně horší ani rozdílný od reálného chování řidičů.

Kapitola 4

Průběh simulace

V této kapitole popisujeme průběh a princip simulace dopravy v simulátoru, jenž popisuje kapitola 3.

Pro simulování dopravy v připravené dopravní síti používáme *diskrétní simulaci*. Jedná se o typ simulace, jenž pracuje v tzv. diskrétním čase (skokově). Slouží pro zjednodušení simulace komplexního systému pomocí vhodné abstrakce.

Simulace typicky používá parametry jako je čas (mění se skokově a určuje pořadí událostí v simulaci), událost (změny v systému), fronta (čekání entit na provedení události) a podobně. Příkladem diskrétní simulace může být popis obchodního střediska, kde simulujeme příchody a odchody zákazníků, jejich čekání ve frontě u pokladen atd. Více informací o diskrétní simulaci popisuje Matloff [11].

4.1 Inicializace simulátoru

Před prvním spuštěním simulátoru je nejprve potřeba řádně načíst a inicializovat mapu simulace a veškeré její parametry. Kapitola 2 popisuje podrobněji proces předpřípravy mapy a kapitola 3 definuje podobu simulátoru.

Následně je pak možné opakovaně spouštět simulaci dopravy. Před začátkem každé simulace jsou vždy vynulovány veškeré statistické údaje popisující vlastosti simulovaného prostoru, například informace o průměrném stavu baterie v jednotlivých segmentech (z definice 3), o průměrné době čekání na nabíjecích stanicích, počtu vozidel, kterým se vybila baterie a podobně.

Dále je před každým spuštěním simulace nutné rozmístit nabíjecí stanice do dopravní sítě. Ty mohou být rozmístěny buď náhodně, nebo za pomoci zvolené optimalizační metody.

4.2 Popis simulace

Dopravní síť simulujeme pomocí diskrétní simulace. Pro správnou funkcionalitu musí simulátor obsahovat časový plán, jenž chronologicky rozvrhuje vykonávání akcí v čase. Především je potřeba definovat všechny možné události a jejich důsledky v simulovaném prostoru. V aktuální implementaci pracujeme pouze s jedním typem vozidla, kterým je typ auto (viz. sekce 3.5.4). Všechny popisované vlastnosti simulace se tedy vztahují pouze na tento objekt.

4.3 Události simulace

Pro správný popis diskrétní simulace musíme zadefinovat vhodné události v simulaci. Události v simulaci jsou dány typem události, časem vykonávání a objektem zapojeným do události. S jejich pomocí jsme schopni popsat veškeré jevy v simulovaném prostoru. Veškeré události simulace se vážou na konkrétní vozidlo, neboť se jedná o jediný prvek simulace, jenž svým chováním přímo ovlivňuje průběh simulace v čase.

Zvolili jsme 6 typů událostí, které vhodně popisují simulované prostředí s ohledem na námi řešený optimalizační problém. Mezi tyto události patří výjezd vozidla, přesun vozidla po hraně grafu, čekání ve frontě na nabíjení, nabíjení, konec nabíjení a čekání auta na návrat.

4.3.1 Výjezd vozidla

Událost *výjezd vozidla* (v simulátoru označován jako `START`) je jedna ze stěžejních událostí simulace. Během této události je naplánován první pohyb vozidla po silnici v grafu simulátoru (viz. sekce 3.3) a zároveň je zde vygenerováno nové vozidlo a naplánován jeho čas výjezdu, dle pravidel ze sekce 3.5.1.

4.3.2 Pohyb vozidla

Událost *pohyb vozidla* (`RUNNING`) po silnici (viz. definice 4) je s přehledem nejčastější událost v simulaci. V této události je simulován přejezd vozidla po silnici v grafu dopravní sítě simulátoru.

Tato událost reprezentuje přejezd vozidla po části silnice (nikdy nepřesáhne hranice silnice), jenž je spravován samotným vozidlem. S její pomocí je naplánován čas dojezdu vozidla na konec vybraného úseku hran grafu. Na tento čas je také naplánována následující událost vozidla. Dále je zde nasimulován přejezd vozidla po zvolené posloupnosti hran, vybíjení baterie při přesunu a je aktualizována informace o hustotě provozu na silnicích. Tato událost je také jedinou

příležitostí, kdy se může vozidlo rozhodnout změnit svou trasu a vydat se směrem k nejoptimálnější nabíjecí stanici (viz. sekce 3.5.3).

Důvod volby pohybu po souvislé části silnice místo po jednotlivých hranách grafu je motivován myšlenkou zjednodušené dopravní sítě z definice 13. Víme totiž, že většina naplánované cesty je složena ze silnic (viz. sekce 3.5.2) a s ohledem na výpočetní výkon je zbytečně náročné simulovat přejezdy po jednotlivých hranách. Nevýhodou této implementace je komplikovanější simulace vybíjení baterie a nemožnost simulovat hustotou provozu na jednotlivých hranách odděleně, čímž simulace ztrácí na realističnosti.

4.3.3 Události nabíjení vozidla

Proces nabíjení vozidla vyžaduje nejvíce událostí pro jeho popsání ze všech procesů v simulaci.

První událostí je *čekání vozidla ve frontě* na nabíjení (`WAITING_LINE`), jež vždy nastane při dojezdu vozidla na nabíjecí stanici i v případě, že ve stanici je volná nabíječka a není tak potřeba čekat v řadě. Během této události je vozidlo zařazeno do fronty příslušné nabíjecí stanice a čeká zde na dobu přiřazení k nějakému nabíjecímu slotu stanice.

V momentě přiřazení slotu se přesune do události *nabíjení vozidla* (`CHARGING`). Ta pokrývá většinu interakce vozidla na nabíjecí stanici. Je zde zvoleno na jakou hladinu baterie bude vozidlo nabito. Dále je zde naplánován čas ukončení nabíjení vozidla a je simulováno nabíjení vozidla.

Po uplynutí času potřebného pro nabíjení vozidla následuje událost *konce nabíjení* (ozn. `END_CHARGING`). V této události je naplánována nová optimální trasa vozidla, kterému je vrácena možnost se v budoucnu rozhodnout směřovat k nabíjecí stanici. Je zde také naplánováno nabíjení dalšího vozidla čekajícího v řadě.

4.3.4 Konec cesty vozidla

Poslední událostí simulace je čekání vozidla na návrat (`WAITING_RETURN`). V této událost vozidlo pouze čeká náhodnou dobu z exponenciální distribuce. Na dobu ukončení čekání naplánuje následující pohyb vozidla (viz. sekce 3.5.4).

Pokud se již vozidlo vracelo zpět do místa výjezdu, pak je vozidlo řádně odstraněno ze simulace.

Proces návratu je typický pouze pro druh vozidla s názvem auto (viz. sekce 3.5.4), protože je ale v naší implementaci zahrnut pouze jeden druh vozidla, můžeme tuto vlastnost zobecnit na všechny vozidla.

4.4 Běh simulace

Průběh simulace je poměrně přímočaný. Při spuštění je zadána doba simulace a všechny další potřebné parametry a je vygenerováno první vozidlo simulace. Následně jsou již pouze chronologicky vykonávány naplánované události do doby než je překročena doba simulace, která následně končí. Po skončení lze ze simulace získat relevantní informace pro optimalizaci. Případně lze veškeré informace vynulovat a spustit novou simulaci.

Kapitola 5

Optimalizační algoritmy pro rozmístění nabíjecích stanic

Náš simulátor poskytuje celkem 3 odlišné strategie pro optimalizaci rozmístění nabíjecích stanic, z nichž 2 také umožňují optimalizovat počet použitých stanic. V této kapitole zvolené algoritmy popíšeme a zadefinujeme také ztrátovou funkci optimalizace.

5.1 Ztrátová funkce (loss function)

Abychom mohli optimalizovat řešení problému musíme zvolit způsob měření kvality jednotlivých modelů rozmístění nabíjecích stanic. K tomuto účelu slouží tzv. *ztrátová funkce* (loss), kterou volíme podle řešeného problému. Typicky je ztrátová funkce definována tak, aby s klesající funkční hodnotou rostla kvalita řešení.

Zadefinujme ztrátovou funkci, která popisuje kvalitu modelu M rozmístění nabíjecích stanic na dopravní síti simulátoru.

Definice 14 (Ztrátová funkce modelu)**.** Ztrátová funkce modelu M *s parametry* $\{p_1, p_2, p_3, p_4, p_5\} \in \mathbb{R}_0^+$*, je definována jako* $L : \mathbb{N}_0^2 \times \mathbb{R}^3 \to \mathbb{R}$*, kde:*

$$L(v_1, v_2, v_3, v_4, v_5) := \sum_{i=1}^{5} p_i \cdot v_i$$

Pro hodnoty:

- $v_1 \in \mathbb{N}_0$ *- počet nabíjecích stanic v modelu* M
- $v_2 \in \mathbb{N}_0$ *- počet vozidel, jimž se v modelu* M *během simulace vybila baterie*

- $v_3 \in \mathbb{R}$ - *průměrné trvání cesty vozidel v minutách*
- $v_4 \in \mathbb{R}$ - *průměrný rozdíl hladin baterie vozidel na počátku a konci cesty*
- $v_5 \in \mathbb{R}$ - *průměrná doba čekání ve frontě na nabíjecí stanici v minutách*

S ohledem na volbu optimalizačních algoritmů, dává smysl zaměřit se při volbě parametrů ztrátové funkce pouze na analýzu počtu nabíjecích stanic a počtu vozidel, kterým se během simulace vybila baterie. Zbylé parametry nejsme schopni s námi zvolenými optimalizačními algoritmy cíleně optimalizovat a mají spíše informativní funkci, pro detailnější analýzu řešení. Z tohoto důvodu volíme pro tyto vlastnosti hodnoty parametrů, tak aby výrazně nezasahovaly do hodnoty ztrátové funkce.

5.1.1 Poznámka k vybitým vozidlům

Je potřeba poukázat na skutečnost, že v reálném případě téměř nikdy nenastává situace, kdy se baterie vozidla úplně vybije a vozidlo nemůže pokračovat dále v cestě. Pokud by k tomuto jevu docházelo často, pak by bylo využití elektrických vozidel prakticky nepoužitelné pro běžné uživatele. V našem simulátoru však k tomuto jevu dochází nerealisticky často.

Důvodem volby návrhu takového simulátoru je jednoduchá práce s údaji o vybitých vozidlech v optimalizačních algoritmech. Jedná se totiž o parametr, který vždy jasně popisuje kvalitu řešení. To se například nedá říct o průměrném rozdílu hladin baterie na začátku a v cíli cesty, či o průměrné době cestování vozidel. Bylo by podstatně obtížnější odhadnout kvalitu řešení, vzhledem k těmto parametrům.

Oproti tomu počet aut s vybitou baterií dobře popisuje počet aut, které potřebují navštívit nabíjecí stanici, která je ovšem velmi vzdálená od jejich trasy. Zjednodušeně řečeno, čím méně vybitých aut v simulaci, tím optimálnější je rozmístění stanic v modelu.

Myšlenku optimalizace řešení pomocí vybitých vozidel jsme převzali z práce autorů Hiwatari, Ikeya a Okano [7].

5.2 Optimalizace hladovým (greedy) algoritmem

Nejjednodušší implementovaný optimalizační algoritmus jsme pojmenovali jako tzv. *Hladový algoritmus*. Jedná se o heuristický postup, jehož cílem je hladově na základě aktuálních výsledků simulace optimalizovat pozice nabíjecích stanic a jejich počet.

Princip řešení je založen na opakovaném simulování dopravy na modelech, ve kterých jsou nabíjecí stanice rozmísťovány pomocí jednoduché heuristiky

založené na výsledcích předchozí simulace. K jeho popisu se hodí zadefinovat pojem tzv. *užitečné nabíjecí stanice.*

Definice 15 (Užitečnost nabíjecí stanice)**.** *Nabíjecí stanici nazveme* užitečnou*, pokud byla v nějakém okamžiku simulace použita alespoň jedním vozidlem. V opačném případě nazveme stanici* zbytečnou.

Dále pracujeme s parametrem $r \in \mathbb{N}_0$, pro který musí platit $|M_0| - |M_1| \leq r$, kde $|M|$ značí počet nabíjecích stanic v modelu M. Parametr r tedy popisuje, o kolik se může zmenšit celkový počet nabíjecích stanic v modelu M_1 oproti modelu M_0.

Algoritmus nejprve spustí simulaci na aktuálním modelu M_0. Následně nalezne všechny užitečné nabíjecí stanice U_0 a vloží je do novém modelu M_1. Zbytečné nabíjecí stanice v novém modelu nepoužívá. Po nalezení všech užitečných stanic spočítá rozdíl $r_0 = |M_0| - |U_0|$. Pokud $r_0 \leq r$, pak nabíjecí stanice modelu M_1 jsou právě všechny užitečné stanice. Pokud je rozdíl počtu stanic vyšší než r, pak do M_1 náhodně dogenerujeme stanice, aby $|M_0| - |M_1| = r$.

Tento postup opakujeme iterativně do doby než je buď dosaženo maximálního povoleného počtu iterací algoritmu, nebo pokud je rozdíl mezi hodnotami chybových funkcí dvou posledních modelů menší než zvolená hranice (algoritmus dokonvergoval k řešení).

5.3 Optimalizace genetickým algoritmem

Další optimalizační metodou je optimalizace genetickým algoritmem (viz. sekce 1.2). Použití genetického algoritmu pro zvolený optimalizační problém je inspirováno prací autorů Niccolai, Bettini a Zich [8]. Metoda umožňuje podobně jako optimalizace hladovým algoritmem optimalizovat také počet nabíjecích stanic.

Jedinci v optimalizaci reprezentují jedno rozmístění nabíjecích stanic. Pro snadnou manipulaci kódujeme jedince jako uspořádanou n-tici nabíjecích stanic (kde n značí celkový počet stanic). U jedinců nepožadujeme stejnou velikost a je tak možné optimalizovat i počet nabíjecích stanic. Jako fitness jedince volíme loss modelu po simulaci (definice 14). Musíme brát v potaz, že v našem případě mají lepší jedinci nižší hodnotu fitness, oproti typické variantě GA, kde vyšší fitness typicky značí lepší kvalitu řešení.

Pracujeme s variantou GA, kde je počet jedinců v různých generacích stejný. Optimalizace GA pak probíhá následovně. Nejprve zkopírujeme zvolený počet nejlepších jedinců aktuální generace do nové generace. Zbylé jedince dostaneme křížením. Individua pro křížení volíme turnajovou selekcí. Dále aplikujeme jednobodové křížení, v němž využijeme zakódování jedince jako uspořádanou n-tici

Algoritmus 6 Algoritmus optimalizující pozice a počet nabíjecích stanic hladovým způsobem.

function HLADOVÁ OPTIMALIZACE(
s_0 - počáteční počet stanic,
r - nejvyšší povolený rozdíl mezi iteracemi,
$iter_{max}$ - maximální počet iterací,
$loss_{diff}$ - hranice konvergence (rozdíl chybové funkce mezi iteracemi)
)
    Náhodně inicializuj model M s celkovým počtem stanic s_0.
    $iter \leftarrow 0$ ▷ Aktuální číslo iterace.
    $loss_{prev} \leftarrow 0$ ▷ Hodnota chybové funkce minulého modelu.
    while $iter < iter_{max}$ **do**
        Simuluj dopravu na modelu M.
        $loss_{curr} \leftarrow$ hodnota chybové funkce modelu M
        if $|loss_{prev} - loss_{curr}| < loss_{diff}$ **then**
            if $Iter > 0$ **then** ▷ Přeskoč pokud se jedná o první iteraci.
                Ukonči optimalizaci.
            end if
        end if
        $s \leftarrow$ počet stanic modelu M
        $U \leftarrow$ užitečné stanice modelu M
        if $s - |U| \leq r$ **then**
            $M \leftarrow$ model s nabíjecími stanicemi U
        else
            $r_{curr} \leftarrow s - |U|$ ▷ Počet stanic pro náhodné vygenerování.
            $S_{rand} \leftarrow$ Náhodně vygeneruj r_{curr} stanic.
            $M \leftarrow$ model s nabíjecími stanicemi U a S_{rand}
            $loss_{prev} \leftarrow loss_{curr}$
        end if
        $iter \leftarrow iter + 1$
    end while
end function

stanic. Náhodně zvolíme bod křížení jako pozici v reprezentaci jedince s menším počtem stanic. Reprezentaci jedince rozdělíme pomocí tohoto bodu na dvě části (první část je před zvoleným bodem, druhá je za ním). Následně mezi jedinci určenými pro křížení vyměníme druhou část reprezentace a dostaneme tak jedince nové generace.

Nové jedince ještě následně mutujeme. Mutaci dělíme na dvě podčásti. V první části procházíme každou nabíjecí stanici jedince a se zvolenou (malou) pravděpodobností ji nahradíme novou náhodně vygenerovanou. Ve druhé části náhodně odebíráme, či přidáváme nové náhodně vygenerované stanice. Tato část mutace nám umožňuje získávat jedince odlišných velikostí (tedy i zkoumat varianty řešení s proměnlivým počtem nabíjecích stanic).

Algoritmus 7 Algoritmus optimalizující pozice a počet nabíjecích stanic s pomocí genetického algorimu.

function OPTIMALIZACE GA(
num_{gen} - počet generací,
pop_{size} - velikost populace,
k - počet nejlepších jedinců pro přímou volbu do nové populace
)
    Nahodně inicializuj první generaci jedinců a na každém spusť simulaci.
    $iter \leftarrow 0$ ▷ Aktuální číslo iterace.
    while $iter < num_{gen}$ **do**
        Zkopíruj k nejlepších jedinců do nové populace.
        $pairs \leftarrow$ Turnajovou selekcí zvol $(pop_{size} - k)/2$ dvojic pro křížení.
        for all pairs **do**
            Proveď jednobodové křížení na dvojici jedinců.
            Proveď mutaci na zkřížených jedincích.
            Přidej zmutované jedince do nové populace.
        end for
    end while
end function

5.4 Optimalizace inspirovaná k-means

Poslední optimalizace použita v práci je inspirována algoritmem k-means (viz. sekce 1.3). Návrh algoritmu byl také inspirován optimalizačním algoritmem z Hiwatari, Ikeya a Okano [7]. Jako jediná z optimalizačních metod optimalizuje tato metoda pouze pozice nabíjecích stanic.

Algoritmus je založen na myšlence, že centroidy z algoritmu k-means jsou pozice nabíjecích stanic. Zvolené k z algoritmu tedy udává počet nabíjecích stanic. V prvním kroce algoritmu jsou centroidy generovány náhodně. Následně probíhá krok podobný hledání shluků. Pro každou křižovatku v grafu (viz. definice 1) spočítáme vzdálenost od každého centroidu pomocí Dijkstrova algoritmu. Křižovatka pak náleží shluku, který odpovídá nejbližšímu centroidu od křižovatky.

Komplikovanější část algoritmu je metoda přepočítávání nové pozice centroidu. Předpokladem pro správnou funkci je přístup k informacím o průměrné hladině baterie vozidel ve vrcholech grafu (viz. definice 3).

Algoritmus nejprve spočítá tzv. *váhu křižovatky*. Ta popisuje "míru vybitosti" vozidel v okolí křižovatky. Při výpočtu váhy libovolné křižovatky k procházíme všechny silnice r napojeny na křižovatku k. Každá silnice poskytuje také informaci o průměrné hladině nabití vozidel, jež silnicí projížděly. Informace o průměrné hladině nabití vozidel je uložena v segmentech silnice. Každý segment s silnice r přispěje k váze křižovatky k hodnotou závislou na vzdálenosti segmentu od křižovatky v a hodnotě $s.b$ (průměrné hladině baterie v segmentu s). Takto projdeme všechny křižovatky grafu a všechny silnice, které jim náleží a dostaneme tak váhu pro každou křižovatku grafu.

Na váhu křižovatky se dá z pohledu algoritmu k-means nahlížet tak, že váha udává počet bodů v prostoru, jež jsou umístěny na pozici křižovatky k. Hlavní motivací pro zadefinování této veličiny je vytvoření vhodné abstrakce pro aplikaci algoritmu k-means.

Algoritmus 8 Funkce pro výpočet váhy křižovatky.

function VÝPOČET VÁHY KŘIŽOVATKY(
k - křižovatky, jejíž váhu chceme spočítat)
  $w \leftarrow 0$ ▷ Počáteční váha křižovatky k je 0.
  for all silnice r náležící vrcholu k **do**
    $w_r \leftarrow 0$ ▷ Váha silnice r.
    $n \leftarrow$ |r.S| ▷ Počet segmentů silnice.
    for all segment s silnice r **do**
      $i \leftarrow$ vzdálenost segmentu s od křižovatky k ▷ V počtech segmentů.
      $avg \leftarrow$ průměrná hladina baterie v segmentu s ▷ V rozmezí (0-1).
      $w_r \leftarrow w_r + 1 - i/n$ ▷ Přičti k váze silnice r, váhu segmentu s.
    end for
    $w \leftarrow w + w_r$ ▷ Přičti k váze křižovatky k váhu silnice r.
  end for
  return w
end function

Po spočítání vah křižovatek je aplikován klasický postup aktualizace nové

pozice centroidu algoritmu k-means. Spočítá se vážený průměr souřadnic všech křižovatek náležících do zvoleného shluku s pomocí vah křižovatek. Souřadnice z nichž počítáme průměr jsou reálné zeměpisné souřadnice (viz. sekce 1.1) křižovatek.

Vypočtená průměrná pozice P ale téměř nikdy nebude odpovídat pozici nějaké křižovatky v grafu, jež jsou jediné objekty v grafu simulace, u nichž známe zeměpisné souřadnice. Pozice P tak nemůže být novým centroidem, jelikož náš optimalizační algoritmus přiřazuje vrcholy do shluků na základě vzdáleností v grafu. Je tedy potřeba spočtenou pozici aproximovat s nějakou pozicí v grafu.

Vzhledem k paměťovým a výpočetním omezením simulátoru jsme si schopni pamatovat pouze informace o zeměpisných souřadnicích křižovatek. Musíme tedy vypočtenou pozici řádně aproximovat pouze za pomoci souřadnic křižovatek. Mějme tedy centroid $centr$. Nejprve nalezneme nejbližší křižovatku k od $centr$ (ze zeměpisných souřadnic). Následně nalezneme nejbližší sousední křižovatku k_{neigh} křižovatky k od hledaného centroidu $centr$. Tímto postupem odhadujeme, že nová pozice centroidu leží na silnici r mezi křižovatkami k a k_{neigh}. Zároveň protože je křižovatka k blíže hledané pozici než k_{neigh}, předpokládáme, že hledaná pozice bude náležet segmentu s silnice r, jež je blíže ke křižovatce k než ke křižovatce k_{neigh}. Segment s, jenž je vhodnou aproximací hledané pozice, volíme uniformně náhodně ze všech segmentů z $r.S$, které jsou blíže vrcholu v. Takto dostaneme novou pozici centroidu a v některých případech také nabíjecí stanice, která je dána segmentem s.

Algoritmus 9 Funkce pro nalezení nejbližší pozice v grafu od zeměpisné souřadnice.

function NEJBLIŽŠÍ POZICE GRAFU OD ZEMĚPISNÉ POZICE(
$coord$ - zeměpisné souřadnice hledané pozice)
    $k \leftarrow$ Nejbližší křižovatka od pozice se souřadnicemi $coord$.
    $k_{neigh} \leftarrow$ Soused křižovatky k, který je nejblíže od pozice $coord$.
    $r \leftarrow$ - Silnice dána křižovatkami k a k_{neigh}.
    $best_{pos} \leftarrow$ Uniformně náhodně vygenerovaný segment silnice r, který je blíže od křižovatky k.
end function

Po nalezení nového centroidu následně nalezneme nové shluky a postup výše opakujeme ve zvoleném počtu iterací. Po uplynutí všech iterací vytvoříme nový model v němž budou nabíjecí stanice umístěny na pozicích centroidů. Spustíme simulaci na novém modelu a následně znovu aplikujeme postup podobný algoritmu k-means. Tento postup taktéž opakujeme zvolený počet iterací.

Postup hledání nejbližšího pozice v grafu od geografické souřadnice je pouze aproximační a nezaručuje nalezení reálně nejbližšího bodu. Předpokládáme však, že tento postup odhadne pozici nejbližšího bodu s dostačující přesností.

Algoritmus 10 Algoritmus optimalizující pozice nabíjecích stanic s pomocí algorimu k-means.

function OPTIMALIZACE POMOCÍ K-MEANS(
$means_{iter}$ - počet iterací cyklu algoritmu k-means,
sim_{iter} - počet simulací modelu (počet spuštěných algoritmů k-means)
)
    Náhodně vygeneruj centroidy.
    $iter_1 \leftarrow 0$ ▷ Aktuální číslo iterace.
    while $iter_1 < sim_{iter}$ **do**
        Vytvoř model s nabíjecími stanicemi na pozicích centroidů.
        Spusť simulaci.
        Spočítej váhy vrcholů.
        $iter_2 \leftarrow 0$
        while $iter_2 < means_{iter}$ **do**
            Nalezni shluky centroidů.
            Nalezni optimální geografické pozice nových centroidů.
            Aproximačně nalezni pozice nových centroidů.
        end while
    end while
end function

Kapitola 6

Analýza výsledků optimalizace

V této kapitole popisujeme a porovnáváme výsledky optimalizací s pomocí optimalizačních algoritmů, které popisuje kapitola 5.

Pro analýzu výsledků optimalizace, která vhodně popisuje kvalitu řešení je vhodné zvolit parametry simulátoru tak, aby co nejrealističtěji popisovaly simulované prostředí. Tato volba je bohužel v mnoha případech značně komplikovaná, neboť řada parametrů je definována pro snadnou práci se simulátorem a jejich hodnota není jednoduše dohledatelná z reálných zdrojů. Z tohoto důvodu řadu parametrů nastavujeme empiricky, nebo pomocí porovnání výsledků simulace.

6.1 Poznámka ke značení

S ohledem na množství parametrů simulátoru a na přehlednost textu, budeme popisovat parametry zvolené pro analýzu ve formátu podobném `JSON`[1], vždy s klíčem udávajícím jméno parametru v našem simulátoru a polem zkoumaných hodnot. Appendix A.4.1 obsahuje popis všech parametrů v simulátoru. Tedy například jako:

```
"carConsumption" : [0.001],
"carVelocity" : [1, 2],
"numStations" : [300]
```

V příkladu je specifikováno, že analyzujeme všechny kombinace hodnot parametrů simulátoru takové, že počet nabíjecích stanic je vždy 300, průměrná spotřeba je vždy 0.001 a průměrná rychlost všech vozidel je buď 1 nebo 2. Celkem tedy v tomto příkladě analyzujeme 2 různé varianty nastavení simulátoru.

[1] `https://www.json.org/json-en.html`

6.2 Výběr parametrů simulátoru

Simulátor navržen v naší práci nabízí širokou škálu nastavitelných parametrů. Z tohoto důvodu před analýzou jednotlivých optimalizačních metod nejprve analyzujeme nastavení parametrů simulátoru a u části z nich také analyzujeme správnost implementace simulátoru s ohledem na zvolený parametr. Vzhledem k výpočetní náročnosti simulace dopravy je pro nás prakticky nemožné podrobně analyzovat všechny parametry. Analyzujeme proto jen část námi vybraných parametrů.

6.2.1 Popis analýzy

Pro zvolené kombinace parametrů vždy simulujeme 10 náhodných modelů splňujících požadované parametry. Jejich výsledky následně průměrujeme a analyzujeme.

Pouze na našem rozhodnutí bez jakékoli analýzy jsme pevně zvolili následující parametry:

```
"simulationTime" : [1000],
"segmentLength" : [1],
"stationCapacity" : [1],
"batteryTresholdLambda" : [20],
"carBatteryMean" : [0.7],
"carBatteryDeviation" : [0.2],
"carStartBatteryBottomLimit" : [0.5],
"notChargingTreshold" : [0.7],
"chargingWaitingTime" : [5],
"meanChargingLevel" : [0.5]
```

Většina těchto parametrů ovlivňuje simulaci pouze minimálně, pokud jsou zvoleny v rozumném rozmezí hodnot. Za povšimnutí stojí nahlédnout na parametr `segmentLength`, značící délku hran v grafu v kilometrech a určuje přesnost, s jakou jsme schopni v simulované dopravní síti pracovat (v našem případě je to 1 kilometr). Dále je vhodné zmínit, že volíme pevnou dobu úplného nabití vozidla na 5 minut, která je optimistickým předpokladem s ohledem na reálné hodnoty[2] a každá nabíjecí stanice má pouze 1 nabíjecí slot. Tyto hodnoty volíme, protože s pomocí našich optimalizačních algoritmů neumíme efektivně optimalizovat čas čekání na stanici a nemusíme se jimi výrazně zabývat.

Asi nejdůležitějším zvoleným pevným parametrem je doba simulace. Tu nastavujeme na 1000 minut, tedy 16 hodin a 40 minut. Pro tuto hodnotu se rozhodujeme,

[2] `https://pod-point.com/guides/driver/how-long-to-charge-an-electric-car`

neboť se jedná o dostatečně dlouhý časový úsek a zároveň je simulace upočítatelná v rozumném čase.

Parametry simulátoru, u nichž zkoumáme více variant jsou:

```
"numStations" : [300],
"numClosestStations" : [1, 3, 5, 10],
"carConsumption" : [0.0001, 0.001],
"exponentialLambdaCities" : [0.001, 0.01, 0.1],
"exponentialLambdaDepartures" : [1000, 10000],
"endCityRatio" : [0.01, 0.1, 1, 10, 100],
"chargingTreshold" : [0.3, 0.5, 0.7],
"batteryTolerance" : [0.0, 0.05, 0.1],
"carVelocity" : [1, 1.5, 2, 3, 5]
```

Mezi parametry výše zařazujeme také celkový počet nabíjecích stanic, který nastavujeme na hodnotu 300. Důvodem této volby je potřeba co nejefektivnější simulace, protože počet zkoumaných kombinací parametrů je velmi vysoký. Počet nabíjecích stanic podrobněji analyzujeme až v analýze optimalizačních algoritmů.

Ze zkoumaných parametrů nás zajímá především vliv parametru udávající počet nabíjecích stanic pro výběr vhodné nabíjecí stanice vozidlem (viz. sekce 3.5.3). Zajímá nás především do jakých hodnot má rostoucí počet vybíraných nabíjecích stanic zásadní vliv na počtu vozidel, kterým se vybila baterie.

Výsledky experimentů bohužel nejsou moc vypovídající a paradoxně vycházely nejhůře výsledky pro hodnotu 10 parametru `numClosestStations`. To si odůvodňujeme volbou pouze 300 nabíjecích stanic v modelu s celkový počet měst 521. Vzhledem ke strategii zajíždění vozidel na nabíječku, jež významně závisí na počtu měst, lze význam parametru pozorovat spíše až u rostoucího počtu nabíjecích stanic překračujícího počet měst na mapě. Při dalších experimentech, v nichž jsme pracovali s vyšším počtem nabíjecích stanic jsme již pozorovali výraznější zlepšení kvality řešení s rostoucím počtem stanic pro výběr. Z těchto experimentů bohužel nemáme dostatek dat pro ilustraci.

Pro ilustraci přikládáme dvě tabulky popisující závislost počtu vybitých vozidel v simulaci, z nichž vozidlo vybírá (viz. tabulka 6.1), a na průměrné době cestování (viz. tabulka 6.2). Tabulky pracují s výsledky simulací, v nichž jsme volili pouze parametry s následujícími hodnotami:

```
"numStations" : [300],
"numClosestStations" : [1, 5, 10],
"carConsumption" : [0.001],
"exponentialLambdaCities" : [0.001, 0.01, 0.1],
"exponentialLambdaDepartures" : [1000],
```

Počet stanic pro výběr	`endCityRatio`	Počet vybitých vozidel
1	1	626296
1	0.01	626699
5	0.01	632876
5	1	638826
5	100	642185
10	0.01	654256
1	100	655669
10	100	657132
10	1	661079

Tabulka 6.1 Porovnání volby variant počtu stanic z nichž vozidlo vybírá a parametru `endCityRatio`, jenž určuje důležitost vzdálenosti při generování koncového města (viz. appendix A.4.1). Tabulka je setříděna podle počtu vozidel, jimž se vybila baterie během simulace. V simulaci bylo rozmístěno celkem 300 nabíjecích stanic.

```
"endCityRatio" : [0.01, 1, 100],
"chargingTreshold" : [0.5],
"batteryTolerance" : [0.05],
"carVelocity" : [1.5]
```

Většina dalších parametrů ovlivňovala simulaci očekávaným způsobem a vzhledem k jejich vysokém počtu je v tomto textu podrobněji neanalyzujeme. K práci jsou v příloze dodány veškeré výsledky simulací ve formě tabulky, jež popisuje volbu parametrů a obsahuje podrobnější informace o výsledcích simulace (viz. appendix C).

6.3 Analýza optimalizačních metod

Na základě výsledků experimentů ze sekce 6.2 volíme pro další experimenty následující parametry simulátoru:

```
"simulationTime" : [1000],
"segmentLength" : [1],
"numClosestStations" : [10],
"carConsumption" : [0.001],
"stationCapacity" : [1],
"exponentialLambdaCities" : [0.005],
"exponentialLambdaDepartures" : [1000],
"endCityRatio" : [1],
"batteryTresholdLambda" : [20],
"carBatteryMean" : [0.7],
```

Počet stanic pro výběr	endCityRatio	Průměrná doba cestování (v minutách)
1	100	97.6049
10	100	111.366
1	0.01	114.14
10	1	114.739
1	1	114.947
5	0.01	115.651
5	100	115.877
5	1	115.981
10	0.01	117.251

Tabulka 6.2 Porovnání volby variant počtu stanic z nichž vozidlo vybírá a parametru endCityRatio, jenž určuje důležitost vzdálenosti při generování koncového města (viz. appendix A.4.1). Tabulka je setříděna podle průměrné doby cestování vozidel v simulaci. V simulaci bylo rozmístěno celkem 300 nabíjecích stanic.

```
"carBatteryDeviation" : [0.2],
"carStartBatteryBottomLimit" : [0.5],
"chargingTreshold" : [0.5],
"notChargingTreshold" : [0.8],
"batteryTolerance" : [0.05],
"carVelocity" : [1.5],
"chargingWaitingTime" : [5],
"meanChargingLevel" : [0.5]
```

Pro správné chování optimalizačních metod potřebujeme vhodně zvolit parametry pro výpočet ztrátové funkce. Ty volíme tak, aby hodnota ztrátové funkce odrážela především počet vybitých vozidel v simulaci. Část hodnoty přidělujeme také počtu nabíjecích stanic. Parametry dalších složek ztrátové funkce volíme tak, aby měly na hodnotu ztrátové funkce minimální vliv. Tento postup je motivován myšlenkami, které popisuje sekce 5.1. Parametry ztrátové funkce volíme následovně:

```
"stationNumberParameter" : [100],
"runDownParameter" : [100],
"durationParameter" : [1],
"batteryDifferenceParameter" : [1],
"waitingTimesParameter" : [1]
```

Nabízí se zde také možnost volby většího zapojení počtu nabíjecích stanic v hodnotě ztrátové funkce pro optimalizaci počtu stanic. Z experimentů však pozorujeme, že tato změna nemá výraznější vliv na výsledcích optimalizace

Počet nabíjecích stanic	Počet vybitých vozidel
4000	586865
2000	590083
1000	631222
500	687167
300	753241

Tabulka 6.3 Počet vybitých vozidel v simulaci v závislosti na počtu nabíjecích stanic při náhodném rozdělení stanic.

(pokud nevolíme výrazně vyšší hodnoty parametru `stationNumberParameter`). Zároveň v následující analýze téměř nikdy nezmiňujeme další parametry pro výpočet chybové funkce, protože tyto hodnoty byly často velmi podobné, nebo z nich nebyla patrná závislost na ostatních parametrech optimalizace či simulátoru.

6.3.1 Poznámka k výsledkům experimentů

V rozboru jednotlivých optimalizačních metod často v tabulkách vypisujeme jen vybranou podmnožinu výsledků. Postup při výběru je popsán u každé tabulky. K tomuto kroku přistupujeme, neboť vetšina experimentů pracuje s mnoha variantami parametrů a výpis všech by byl zbytečně nepřehledný. Kompletní výsledky se nacházení v příloze práce (viz. appendix C). Výsledky některých kombinací parametrů nejsou jsou ve výsledcích zahrnuty, protože experimenty nebylo možné dokončit z důvodu nedostatečné výpočetní výkonnosti stroje, na němž jsme experimenty pouštěli.

6.3.2 Náhodný přístup

O optimalizačních algoritmech musíme vhodným způsobem rozhodnout, zda zlepšují kvalitu řešení problému. K tomuto rozhodnutí potřebujeme referenční hodnotu, kterou získáváme z dat získaných opakovanými simulacemi dopravy s náhodně rozmístěnými stanicemi.

Stanice generuje stejným způsobem jako počáteční pozici vozidla (sekce 3.5.1). Tedy nejprve vygenerujeme město, kterému bude stanice náležet, a následně v daném městě uniformně náhodně generujeme vrchol grafu, jenž určuje pozici stanice.

Tabulka 6.3 znázorňuje výsledky náhodného přístupu.

s_0	$iter_{max}$	r	Počet stanic	Počet vybitých vozidel
4000	20	5	4000	515166
4000	20	1	4000	516282
2000	20	2	2000	516668
2000	20	1	2000	517458
1000	20	2	1000	571645
1000	20	5	1000	572448
500	20	5	495	645667
500	20	2	498	646647
300	20	2	284	678159
300	20	5	285	687528

Tabulka 6.4 Počet vybitých vozidel v simulaci a výsledný počet nabíjecích stanic v závislosti na počátečním počtu nabíjecích stanic při optimalizaci hladovým algoritmem. V tabulce zahrnujeme vybrané hodnoty pro různé parametry optimalizace. Mezi vybrané hodnoty patří ty v níž byl nejnižší a nejvyšší počet vybitých vozidel a ty v níž bylo dosaženo nejnižšího výsledného počtu stanic. Výsledky jsou seřazeny podle počtu vybitých vozidel v simulaci. Hodnoty s_0 značí počáteční počet nabíjecích stanic, $iter_{max}$ maximální počet iterací algoritmu a r maximální rozdíl v počtu stanic mezi iteracemi.

6.3.3 Optimalizace hladovým algoritmem

Při analýze hladového optimalizačního (sekce 5.2) přístupu zvažujeme varianty parametrů:

```
"lossDifferenceTreshold" : [100],
"greedyMaxIterations" : [20],
"greedyNumThrowAway" : [1, 5, 10]
```

Výsledky vykazují prokazatelně menší hodnoty počtu vybitých vozidel než při náhodném přístupu a v případě malého počtu počátečních nabíjecích stanic, také poměrně zásadní redukci výsledného počtu stanic, při níž nedošlo k výraznému nárustu počtu vybitých vozidel. Tabulka 6.4 popisuje vybrané hodnoty počtu vybitých vozidel a výsledného počtu nabíjecích stanic v závislosti na počtu nabíjecích stanic na začátku optimalizace.

6.3.4 Optimalizace genetickým algoritmem

Při analýze optimalizace genetickým algoritmem (sekce 5.3) zvažujeme parametry:

```
"geneticPopulationSize" : [10],
"geneticNumGenerations" : [10, 20],
"geneticNumBestSelection" : [0, 4],
"geneticTournamentSelectionTreshold" : [0.7, 0.9],
```

s_0	pop_{size}	num_{gen}	k	p_{sel}	p_{mut}	Počet stanic	Počet vybitých vozidel
2000	10	10	4	0.9	0.01	2003	495100
4000	10	10	0	0.9	0.01	3997	496395
4000	10	10	0	0.7	0.0001	4001	498511
2000	10	10	0	0.9	0.0001	1987	500384
4000	10	20	4	0.9	0.01	3997	506130
2000	10	20	4	0.9	0.0001	2001	506735
1000	10	10	4	0.7	0.01	1001	526682
1000	10	10	4	0.9	0.0001	997	537434
1000	10	10	0	0.7	0.0001	1002	539771
500	10	10	0	0.7	0.01	501	594346
500	10	10	4	0.9	0.01	503	594541
500	10	10	0	0.7	0.0001	497	595689
500	10	20	4	0.9	0.01	503	606166
300	10	10	4	0.7	0.01	301	640300
300	10	10	4	0.7	0.0001	300	643191
300	10	10	0	0.7	0.0001	301	652122

Tabulka 6.5 Počet vybitých vozidel v simulaci a výsledný počet nabíjecích stanic v závislosti na počátečním počtu nabíjecích stanic při optimalizaci genetickým algoritmem. V tabulce zahrnujeme vybrané hodnoty pro různé parametry optimalizace. Mezi vybrané hodnoty patří ty v níž byl nejnižší a nejvyšší počet vybitých vozidel a ty v níž bylo dosaženo nejnižšího a nejvyššího výsledného počtu stanic. Výsledky jsou seřazeny podle počtu vybitých vozidel v simulaci. Hodnoty s_0 značí počáteční počet nabíjecích stanic, pop_{size} velikost populace v GA, num_{gen} počet generací GA, k počet nejlepších výsledků pro zkopírování do nové populace GA, p_{sel} pravděpodobnost výběru lepšího jedince v turnajové selekci a p_{mut} značí pravděpodobnost mutace jedné stanice.

```
"geneticMutationTreshold" : [0.0001, 0.001, 0.01],
"geneticMemberSizeVariance" : [10]
```

Výsledky prokazují, že optimalizace výrazným způsobem snižuje počet vybitých vozidel v simulaci. Tabulka 6.5 popisuje vybrané hodnoty počtu vybitých vozidel a výsledného počtu nabíjecích stanic v závislosti na počtu nabíjecích stanic na začátku optimalizace.

Je potřeba zmínit, že výsledky optimalizace by pravděpodobně mohly dosahovat lepších výsledků pro větší počet jedinců v populaci a větší počet generací. Vzhledem k výpočetní náročnosti, jsme bohužel experimenty na těchto parametrech nebyli schopni realizovat.

6.3.5 Optimalizace s pomocí k-means

Poslední zkoumaný přístup je optimalizace pomocí algoritmu k-means (sekce 5.4) zvažujeme parametry:

s_0	$means_{iter}$	sim_{iter}	Počet vybitých vozidel
4000	20	50	594054
4000	10	10	595606
2000	20	50	601595
2000	10	2	603219
1000	50	10	645418
1000	20	2	647815
500	50	20	714368
500	50	2	716731

Tabulka 6.6 Počet vybitých vozidel v simulaci v závislosti na počátečním počtu nabíjecích stanic při optimalizaci algoritmem k-means. V tabulce zahrnujeme vybrané hodnoty pro různé parametry optimalizace. Mezi vybrané hodnoty patří ty v níž byl nejnižší a nejvyšší počet vybitých vozidel. Výsledky jsou seřazeny podle počtu vybitých vozidel v simulaci. Hodnoty s_0 značí počáteční počet nabíjecích stanic, $means_{iter}$ počet iterací jednoho cyklu algoritmu k-means a sim_{iter} počet proběhlých simulací modelu (mezi spuštěním algoritmů k-means).

```
"kMeansNumIterationsOneRun" : [10, 20, 50, 100],
"kMeansNumGenerations" : [2, 10, 20, 50]
```

Výsledky tohoto optimalizačního přístupu neprokazují zlepšení. Dokonce v našich experimentech dosahuje tento algorimus horších výsledků než náhodný přístup. Tabulka 6.6 popisuje vybrané hodnoty počtu vybitých vozidel v závislosti na počtu nabíjecích stanic na začátku optimalizace. Z výsledků můžeme nahlédnout, že kvalita řešení roste s počtem iterací algoritmu k-means i s počtem generací modelů (viz. sekce 5.4). Je tak tedy možné, že v případě vyššího počtu opakování by optimalizace dosahovala lepších výsledků.

6.3.6 Porovnání optimalizačních metod

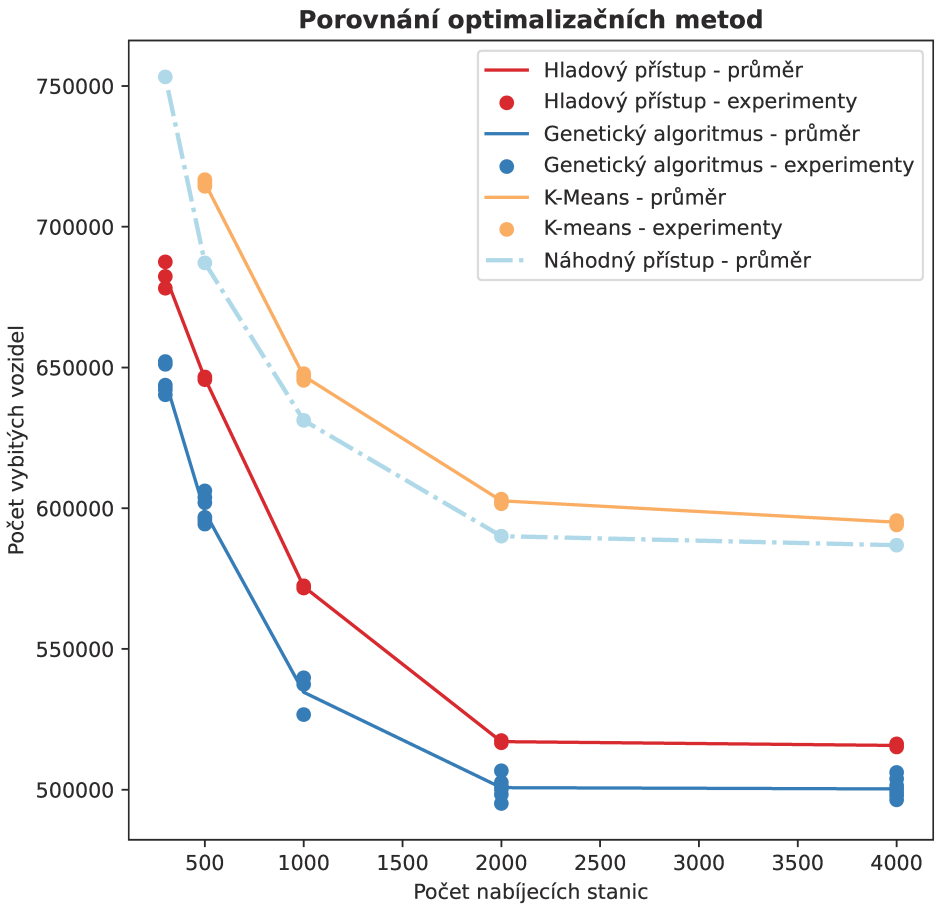
Na základě výsledků experimentů docházíme k závěru, že optimalizace za pomoci genetického algoritmu a optimalizace hladovým přístupem dosahují lepších výsledků, než náhodný přístup. Na druhou stranu optimalizace za pomoci algoritmu k-means vykazuje dokonce horší výsledky než náhodný přístup. Tuto skutečnost si odůvodňujeme velkou mírou aproximace, s níž algoritmus pracuje, a malým počtem opakování algoritmu k-means. Dále je také potřeba zmínit, že například optimalizace pomocí genetického algoritmu, by pravděpodobně dosahovala lepších řešení pro větší velikost populace a počet generací. Tento případ, jsme ale s ohledem na výpočetní náročnost simulace dopravy nebyli schopni realizovat. Dále můžeme nahlédnout, že kolem počtu 2000 stanic se počet vybitých vozidel ustálil a není tak potřeba přidávat do mapy více stanic.

obrázek 6.1 znázorňujeme porovnání výsledků jednotlivých metod pomocí počtu použitých nabíjecích stanic a počtu vybitých vozů.

6.4 Pomocné nástroje pro analýzu výsledků

S ohledem na velké množství parametrů simulátoru a výpočetní náročnosti simulace jsme se rozhodli využít služeb organizace Metacentrum[3]. Za pomoci této organizace jsme schopni otestovat stovky variant parametrů. Vzhledem k množšství zkoumaných variant, je ale pro praktické použití potřeba zautomatizovat proces nastavování zvolených parametrů a vytváření úloh pro spuštění na Metacentru simulací se zvolenými parametry a následnou analýzu výsledků simulací. K tomuto účelu jsme vytvořili pomocný program v jazyce Python (viz. appendix C). Tento program připravuje potřebné skripty pro spouštění jednotlivých úloh v Metacentru a umožňuje jednoduché zobrazení výsledků simulace s možností zkoumání námi zvolených parametrů simulace a setřídění výsledků dle zvoleného parametru.

[3] `https://metavo.metacentrum.cz/`



Obrázek 6.1 Porovnání různých optimalizačních metod. Optimalizační metody barevně rozlišujeme. Body v grafu znázorňují výsledky jednotlivých experimentů. Čárové grafy propojují průměrné hodnoty počtu vybitých vozidel při stejném počtu nabíjecích stanic. Čerchovaný čárový graf značí výsledky náhodného přístupu.

Závěr

V práci jsme navrhli simulátor dopravy speciálně navržen pro snadnou optimalizaci rozmístění nabíjecí stanic v dopravní síti. A následně zkoumali několik vybraných variant optimalizace na mapě dopravní sítě České republiky.

Z důvodu vysoké výpočetní náročnosti zkoumaného problému jsme byli nuceni některé aspekty simulace značně zjednodušit, což částečně vedlo k nerealističnosti výsledků. Ve značné míře jsme například museli zredukovat simulovanou silniční síť a simulování cest vozidel na krátké vzdálenosti. Dalším ze závažných nedostatků simulace je nerealistický počet vozidel, kterým se vybije baterie během cesty. Hlavním důvodem tohoto nedostatku je potřeba vhodně zadefinovat chybovou funkci pro optimalizaci problému (viz. sekce 5.1). Částečně tuto skutečnost způsobuje také velmi zjednodušený proces volby nabíjecí stanice vozidlem (viz. sekce 3.5.3). Tato skutečnost je částečně způsobena nevhodným algoritmem pro volbu nabíjecí stanice, jenž musel být podstatně zjednodušen v důsledku potřeby výpočetně efektivního simulátoru. Na základě experimentů, které podrobně popisuje kapitola kapitola 6, jsme však došli k názoru, že námi navržený simulátor popisuje dostatečně kvalitně aspekty dopravní sítě, jenž jsou potřebné pro optimalizaci rozmístění nabíjecích stanic.

Po provedení experimentů jsme došli k závěru, že hladový optimalizační algoritmus (viz. sekce 5.2) a především optimalizace pomocí genetického algoritmu (viz. sekce 5.3) vykazují lepší výsledky než náhodný přístup, založený na rozmísťování stanic podle počtu obyvatel v jednotlivých částech dopravní sítě (např. v genetickém algoritmu se pro 2000 nabíjecích stanic snížil počet vozidel s vybitou baterií v simulaci o $\frac{1}{6}$). Především u optimalizace genetickým algoritmem, pozorujeme, že kvalita řešení by se mohla ještě vzrůst při zvýšením počtu generací a velikosti populace. Tyto výsledky, jsme však vzhledem k výpočetní náročnosti optimalizace nebyli schopni ověřit. Na druhou stranu optimalizační algoritmus inspirovaný algoritmem k-means (sekce 5.4) dosahoval dokonce mírně horších výsledků jako náhodný přístup. Tuto skutečnost si odůvodňujeme velkou mírou aproximace při hledání vhodné nabíjecí stanice a malým počtem iterací algoritmu v experimentech.

Budoucí práce

Do budoucna se nabízí doimplementovat do simulátoru vhodnější algoritmus pro výběr nabíjecí stanice, čímž by se mohla zlepšit realističnost chování vozidel při jízdě na nabíjecí stanice a výrazně snížit počet vybitých vozidel v simulaci.

Dále se nachází potenciál ve zlepšení optimalizační algoritmu pomocí algoritmu k-means, který pravděpodobně dosahuje špatných výsledků z důvodu značné aproximace, s níž algoritmus pracuje. O potenciálu zlepšení jsme přesvědčeni, protože Hiwatari, Ikeya a Okano [7] dosáhli podobným způsobem kvalitních výsledků.

Dalším potenciálním rozšířením simulátoru je doimplementovat potřebné funkcionality pro optimalizaci počtu nabíjecích slotů nabíjecích stanic a navržení vhodných optimalizačních algoritmů řešící tuto problematiku.

Dále je v simulátoru částečně naimplementována funkcionalita pro načítaní a zápis reprezentace nabíjecích stanic do souboru. Dokončení implementace by nám umožnilo například optimalizovat již předpřipravené řešení a zkvalitnit tak optimalizaci, neboť ve všech akuálně používaných optimalizační algoritmech inicializujeme optimalizaci na náhodně rozmístěných stanicích.

Také se nabízí potenciál v generování výjezdů a tras cest vozidel založených na reálných datech. Čímž bychom mohli zásadně zlepšit realističnost simulovaného provozu.

Seznam použité literatury

[1] UK Government. *COP26 declaration on accelerating the transition to 100emission cars and Vans*. 2022. URL: https://www.gov.uk/government/publications/cop26-declaration-zero-emission-cars-and-vans/cop26-declaration-on-accelerating-the-transition-to-100-zero-emission-cars-and-vans#signatories.

[2] Iea. *Electric cars fend off supply challenges to more than double global sales – analysis*. 2022. URL: https://www.iea.org/commentaries/electric-cars-fend-off-supply-challenges-to-more-than-double-global-sales.

[3] Kang-Tsung Chang. *Introduction to geographic information systems*. Sv. 4. McGraw-Hill Boston, 2008.

[4] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[5] J MacQueen. "Classification and analysis of multivariate observations". In: *5th Berkeley Symp. Math. Statist. Probability*. 1967, s. 281–297.

[6] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[7] Ryoji Hiwatari, Tomohiko Ikeya a Kunihiko Okano. "A road traffic simulator to analyze layout and effectiveness of rapid charging infrastructure for electric vehicle". In: (zář. 2011), s. 1–6. DOI: 10.1109/VPPC.2011.6043186.

[8] Alessandro Niccolai, Leonardo Bettini a Riccardo Zich. "Optimization of electric vehicles charging station deployment by means of evolutionary algorithms". In: *International Journal of Intelligent Systems* 36.9 (2021), s. 5359–5383.

[9] Zhi-Hong Zhu et al. "Charging station location problem of plug-in electric vehicles". In: *Journal of Transport Geography* 52 (2016), s. 11–22.

[10] Ömer Burak Kınay, Fatma Gzara a Sibel A Alumur. "Full cover charging station location problem with routing". In: *Transportation Research Part B: Methodological* 144 (2021), s. 1–22.

[11] Norm Matloff. “Introduction to discrete-event simulation and the simpy language”. In: *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August* 2.2009 (2008), s. 1–33.

[12] © Microsoft 2022 Corob-Msft. *Use the Microsoft Unit Testing Framework for C++ - Visual Studio (windows)*. 2022. URL: https://docs.microsoft.com/en-us/visualstudio/test/how-to-use-microsoft-test-framework-for-cpp?view=vs-2019.

Příloha A

Uživatelská dokumentace

Hlavním cílem programu je na zvolené reprezentaci mapy opakovaně simulovat dopravní síť a na základě této simulace optimalizovat rozmístění nabíjecích stanic s pomocí různých optimalizačních algoritmů. Po ukončení optimalizace vypíše zvolené relevantní informace o jednotlivých simulacích pro následnou analýzu optimalizačních metod. Program také nabízí nastavitelnost široké škály parametrů simulátoru dopravy i optimalizačních metod pro detailnější analýzu.

A.1 Příprava před spuštěním

Rozlišujeme více variant spuštění programu podle OS, který používáme.

Pokud pracujeme v OS **Windows 10**, pak můžeme použít již předkompilovaný program, jenž je součástí přílohy. V tomto případě můžeme program rovnou spustit. Předkompilovaný spustitelný se nachází v adresáři:

```
Win_executable/
```

V tomto adresáři se nachází spustitelný program, adresář s předpřipravenou reprezentací mapy a další pomocné soubory pro správné fungování programu. Název spustitelného programu je:

```
Traffic_Simulator.exe
```

A.1.1 Sestavení programu

Pokud chceme program sestavit, pak je potřeba postupovat v následujících krocích.

Sestavení pro OS Linux

Pro sestavení programu v OS Linux potřebujeme `CMake 3.11`[1], nebo vyšší (doporučujeme verzi `3.14+`), překladač podporující `C++11`, knihovnu `Boost`[2] a Git[3].

Nejprve se přesuneme do adresáře s požadovanými zdrojovými kódy.

```
cd source_code/Traffic_Simulator_cmake
```

Pro konfiguraci spustíme příkaz (přidáme ještě možnost `-GNinja`, pokud používáme Ninja[4]).

```
cmake -S . -B build
```

Pro sestavení programu spustíme příkaz:

```
cmake --build build
```

Výsledný spustitelný program pak nalezneme:

```
build/apps/Traffic_Simulator
```

Sestavení pro OS Windows 10

Pro sestavení programu v OS Windows 10 potřebujeme překladač podporující `C++11`, Visual Studio 2019[5], Git a `vcpkg`[6].

Nejprve se přesuneme do adresáře, kde je uložen nástroj `vcpkg` a nainstalujeme knihovnu Boost.

```
cd $(Absolute_Path_To_vcpkg_Directory)

.\vcpkg\vcpkg install boost
.\vcpkg\vcpkg integrate install
```

Následně se přesuneme do adresáře naší práce a zde do adresáře:

```
cd source_code\Traffic_Simulator_VS
```

Zde nalezenem soubor:

```
Traffic_Simulator.sln
```

Ten otevřeme v programu Visual Studio 2019, ve kterém projekt sestavíme.

[1] `https://cmake.org/`
[2] `https://www.boost.org/`
[3] `https://git-scm.com/`
[4] `https://ninja-build.org/`
[5] https://visualstudio.microsoft.com/cs/
[6] `https://github.com/microsoft/vcpkg`

A.2 Poznámka k příkladům

V následujícím textu popisujeme příklady použití programu v OS Linux. Pokud pracujeme v OS Windows, pak program spouštíme následujícím příkazem (se zvolenými parametry (viz. appendix A.4.1)):

```
.\Traffic_Simulator.exe
```

Pokud pracujeme s předpřipravenou mapou z adresáře:

```
preprocesed\_maps
```

Pak musíme pro fungování příkazů v příkladech, zkopírovat tento adresář do adresáře se spustitelným souborem programu. Pokud pracujeme s již předkompilovaným programem v OS Windows, pak tato operace není nutná.

A.3 Formát souborů reprezentujících mapu

Pro správné fungování programu je potřeba při každém spuštění načíst reprezentaci mapy (sekce 3.3). Ta je rozdělena do 3 souborů a informace v nich zapsané si nesmějí navzájem odporovat. Tyto 3 soubory jsou načítany v pořadí: soubor s reprezentací křižovatek, soubor s reprezentací silnic a soubor s reprezentací měst. Vzhledem k obvykle velké objemnosti dat reprezentující dopravní síť je doporučeno při přípravě vstupních souborů postupovat podle kroků, jež popisuje kapitola 2, s pomocí přiložených programů v adresáři `map_preprocessing` práce (viz. appendix C).

Pokud chce uživatel pracovat pouze s námi připravenou mapou České republiky, jenž podrobněji popisuje kapitola 2, je v adresáři `preprocesed_maps` dodána reprezentace již připravené mapy. Struktura adresáře je:

```
./prepared_graph/
    cities.txt
    combined_nodes.txt
    edges.txt
    prepared_combined_nodes.txt
    prepared_edges.txt
```

V souborech výše jsou zapsány po řadě, reprezentace měst, reprezentace křižovatek, reprezentace hran a v souborech s předponou `prepared_` je uložena předpřipravená reprezentace mapy (bez křižovatek stupně 2) se stejnou reprezentací, jako v neupravené variantě (reprezentace měst se předpřípravou nemění).

Formát všech vstupních souborů je strukturován po řádcích. V každém souboru je vždy ignorován první řádek, jehož účelem je popis formátu vstupních dat jednotlivých řádků pro každý vstupní soubor. S ohledem na přehlednost vstupních dat je doporučeno tento formát vždy dodržovat. Ve zbytku souboru je postupně na každém řádku reprezentován specifický objekt mapy, který je určen daným typem souboru. Jakékoli nedodržení formátu vstupních souborů vede s vysokou pravděpodobností k selhání načítání dat a není tak umožněna další práce s programem.

A.3.1 Soubor reprezentující křižovatky

Předpokládaná hlavička (1. řádek) souboru je:

```
node_id latitude longitude city_id distance_km
```

Každý řádek, kromě prvního řádku, reprezentuje právě jednu kžižovatku (definice 1). Požadujeme, aby na každém řádku bylo uloženo pět hodnot oddělených mezerou, které reprezentují po řadě: identifikační číslo (ozn. ID) křižovatky, zeměpisnou šířku, v níž se křižovatka nachází, zeměpisnou délku, v níž se křižovatka nachází, ID města, do něhož křižovatka náleží, a vzdálenost křižovatky od centra tohoto města.

Pro správnou funkci programu je potřeba, aby ID křižovatek byla celá nezáporná čísla a aby křižovatky byly v souboru seřazeny vzestupně bez vynechaných hodnot ID. Tedy pokud např. v souboru existuje vrchol s ID 6, musí být předem definovány vrcholy s ID v rozmezí od 1 do 5. Dále požadujeme, aby ID křižovatek byly unikátní. Zeměpisná šířka, zeměpisná délka i vzdálenost města jsou reprezentovány desetinným číslem s desetinnou tečkou. ID města je reprezentováno nezáporným celým číslem, jež musí být zadefinováno v souboru reprezentující města (appendix A.3.3).

Příklad formátu vstupního souboru, jenž reprezentuje křižovatky:

```
node_id latitude longitude city_id distance_km
0 50.0354962 14.4080276 1 5.855519633308941
1 50.0352373 14.4080465 2 5.883717272182215
2 50.0350876 14.4080758 0 5.8998150943334124
3 50.035876 14.4080352 0 5.8898140943334124
```

A.3.2 Soubor reprezentující silnice

Předpokládaná hlavička (1. řádek) souboru je:

```
new_node_id_1 new_node_id_2 length road_type
```

Reprezentace	Popis
`m`	dálnice
`t`	silnice pro motorová vozidla
`p`	silnice 1. třídy nebo
`o`	jiný druh silnice

Tabulka A.1 Výpis všech variant druhů silnice.

Každý řádek, kromě prvního řádku, reprezentuje právě jednu silnici (definice 4) dopravní sítě. Požadujeme, aby na každém řádku byly uloženy čtyři hodnoty oddělené mezerou reprezentující: ID první křižovatky definující silnici, ID druhé křižovatky definující silnici, délka silnice a typ silnice. Předpokládáme, že ID křižovatek jsou v odpovídajícím formátu, který je popsán v části pro reprezentaci křižovatek (appendix A.3.1) a zároveň, že křižovatka s ID ze vstupu byla definována a načtena ze souboru pro reprezentaci křižovatek. Délka silnice je reprezentována desetinným číslem s desetinnou tečkou. Tabulka A.1 vypisuje všechny typy silnice a jejich reprezentaci pro vstupní soubor.

Příklad formátu vstupního souboru, jenž reprezentuje silnice:

```
new_node_id_1 new_node_id_2 length road_type
0 1 28.82 t
1 2 16.777 p
2 3 17.303 m
```

A.3.3 Soubor reprezentující města

Předpokládaná hlavička (1. řádek) souboru je:

```
city_id lat lon population
```

Každý řádek, kromě 1. řádku, reprezentuje právě 1 město (definice 8). Požadujeme, aby na každém řádku byly uloženy 4 hodnoty oddělené mezerou reprezentující: ID města, zeměpisnou šířku, v níž se nachází střed města, zeměpisnou délku, v níž se nachází střed města, a počet obyvatel města. Požadujeme, aby ID města bylo nezáporné celé číslo. Pro zajištění správného fungování programu je doporučeno reprezenovat města v souboru v rostoucím pořadí podle ID města bez opakování a bez vynechaných ID. Tato podmínka však není na rozdíl od reprezentace křižovatek nutná. Zeměpisná šířka a délka jsou reprezentovány desetinným číslem s desetinnou tečkou. Populace města je reprezentována nezáporným celým číslem.

Příklad formátu vstupního souboru, jenž reprezentuje města:

```
city_id lat lon population
0 49.39606475830078 15.590306282043457 51216
```

Příkaz	Popis funkce
`help`	Vypíše všechny možnosti parametrů programu.
`logs`	Program bude vypisovat podrobné informace o průběhu simulace (pro ladění progamu).

Tabulka A.2 Popis pomocných přepínačů v programu.

```
1 49.70147705078125 17.07569122314453 7516
2 51.02574920654297 15.061005592346191 4259
```

A.3.4 Výstupní soubory

Program také umožňuje zápis aktuálně načtené mapy do souborů ve vstupním formátu simulátoru. Tato funkcionalita je především užitečná, pokud je předpříprava původní mapy výpočetně náročná (odstraňování vrcholů stupně 2 nebo sjednocování komponent souvislosti). V takovém případě můžeme předpřipravenou mapu uložit do formátu určeného pro načítání mapy a při opakovaném výpočtu přeskočit předpřípravu dat a rovnou spustit požadovaný výpočet.

A.4 Ovládání programu

Uživatel interaguje s programem pouze při jeho spuštění, kdy je potřeba zvolit příslušnou optimalizační metodu, její parametry a parametry simulace. Před spuštěním programu je potřeba připravit soubory s korektní reprezentací mapy (appendix A.3) a specifikovat cestu k těmto souborům.

A.4.1 Parametry programu

Při spuštění programu zadáváme do terminálu parametry, kterými specifikujeme požadované chování simulátoru. Tyto parametry zapisujeme ve formě přepínačů za příkaz spouštějící program. Tyto přepínače popisujeme v tabulkách rozdělených podle jejich funkce. Tabulka A.2 popisuje pomocné přepínače programu, tabulka A.3 popisuje přepínače pro specifikování vstupních a výstupních souborů, tabulka A.4 popisuje přepínače pro volbu parametrů simulace, tabulka A.5 popisuje přepínače pro volbu obecných vlastností typických pro všechny optimalizační metody, tabulka A.6 popisuje přepínače pro volbu parametrů hladové optimalizace, tabulka A.7, popisuje přepínače pro volbu parametrů optimalizace genetickým algoritmem, tabulka A.8 popisuje přepínače pro volbu parametrů optimalizace pomocí algoritmu k-means a tabulka A.9 popisuje přepínače pro volbu optimalizační metody.

Příkaz	Popis funkce
`nodeCityFile arg`	Cesta k vstupnímu souboru s reprezentací křižovatek. V případě nespecifikované hodnoty je zvolena cesta: `prepared_graph/combined_nodes.txt`
`edgesFile arg`	Cesta k vstupnímu souboru s reprezentací silnic. V případě nespecifikované hodnoty je zvolena cesta: `prepared_graph/edges.txt`
`citiesFile arg`	Cesta k vstupnímu souboru s reprezentací měst. V případě nespecifikované hodnoty je zvolena cesta: `prepared_graph/cities.txt`
`addedEdgesFile arg`	Cesta k výstupnímu souboru pro uložení reprezentace přidaných hran pro vytvoření souvislého grafu. V případě nespecikované hodnoty je zvolena cesta: `prepared_graph/added_edges.txt`
`preparedNodesFile arg`	Cesta k výstupnímu souboru pro uložení reprezentace vrcholů upraveného grafu po eliminaci vrcholů stupně 2. V případě nespecikované hodnoty je zvolena cesta: `prepared_graph/prepared_combined_nodes.txt`
`preparedEdgesFile arg`	Cesta k výstměupnímu souboru pro uložení reprezentace hran upraveného grafu po eliminaci vrcholů stupně 2 V případě nespecikované hodnoty je zvolena cesta: `prepared_graph/prepared_edges.txt`
`savePrepared`	Parametr specifikující, zda uložit předpřipravenou reprezentaci mapy.

Tabulka A.3 Popis přepínačů pro specifikování vstupních a výstupních souborů.

Příkaz	Popis funkce
`simulationTime arg`	Doba simulace v minutách.
`segmentLength arg`	Délka hrany grafu v kilometrech.
`numClosestStations arg`	Počet nabíjecích stanic, které jsou zvažovány při plánování cesty vozidla na nabíjecí stanici.
`carConsumption`	Spotřeba baterie vozidla v procentech kapacity za minutu.
`numStations arg`	Celkový počet nabíjecích stanic v simulaci.
`stationCapacity arg`	Počet nabíjecích slotů na každé stanici.
`exponentialLambdaCities arg`	Parametr pro generování cílového města vozidla. Hodnota je rovna $\frac{1}{d}$, kde d je střední hodnota vzdálenosti cílového města od startu v kilometrech.
`exponentialLambdaDepartures arg`	Parametr pro generování času výjezdu následujícího vozidla. Hodnota je rovna $\frac{1}{t}$, kde t je střední hodnota času následujícího výjezdu vozidla v minutách.
`endCityRatio arg`	Parametr specifikující důležitost vzdálenosti cílového města vůči počtu obyvatel při generování cílového města (viz. definice 12). Pokud < 1, pak je počet obyvatel důležitější. Pokud $= 1$, pak jsou oba parametry stejně důležité. Pokud > 1, pak je důležitější vzdálenost města.
`batteryTresholdLambda arg`	Hodnota $\frac{1}{b}$, kde b je střední hodnota hladiny baterie specifikující, o kolik průměrně převyšuje hladina baterie v době rozhodnování nejnižší možnou hranici pro rozhodnutí, zda jet nabíjet, v moment rozhodnutí (viz. sekce 3.5.3).
`carBatteryMean arg`	Střední hodnota počáteční hladiny baterie.
`carBatteryDeviation arg`	Směrodatná odchylka počáteční hladiny baterie vozidla.
`carStartBatteryBottomLimit arg`	Minimální hladina baterie vozidla při výjezdu.
`chargingTreshold arg`	Nejnižší možná hladina baterie pro rozhodnutí, zda vyrazit na nabíjecí stanici (viz. sekce 3.5.3).
`notChargingTreshold arg`	Nejvyšší hladina pro rozhodnutí, zda vyrazit na nabíjecí stanici (viz. sekce 3.5.3).
`batteryTolerance arg`	Nejnižší možná očekávaná hladina baterie v cíli, kdy se při rozhodování, rozhodnout nezajet na nabíjecí stanici (viz. sekce 3.5.3).
`carVelocity arg`	Průměrná rychlost všech vozidel v kilometrech za minutu.
`chargingWaitingTime arg`	Doba čekání na úplné nabití baterie v minutách.
`meanChargingLevel arg`	Očekávaná střední hodnota hladiny baterie, jež je nabíjena na nabíjecích stanicích (pro odhad doby čekání nabíjení ve frontě).

Tabulka A.4 Popis přepínačů pro specifikování parametrů simulátoru.

Příkaz	Popis funkce
`lossDifferenceTreshold arg`	Minimální rozdíl ztrátové funkce pro pokračování v optimalizaci (pro vybrané optimalizační metody).
`stationNumberParameter arg`	Parametr počtu stanic při výpočtu ztrátové funkce (viz. sekce 5.1).
`runDownParameter arg`	Parametr počtu vozidel, kterým se vybila baterie, pro výpočet ztrátové funkce (viz. sekce 5.1).
`durationParameter arg`	Parametr průměrné doby cestování v minutách pro výpočet ztrátové funkce (viz. sekce 5.1).
`batteryDifferenceParameter arg`	Parametr průměrného rozdílu hladin baterie pro výpočet ztrátové funkce (viz. sekce 5.1).
`waitingTimesParameter arg`	Parametr průměrné čekací doby na nabíjecích stanicích pro výpočet ztrátové funkce (viz. sekce 5.1).

Tabulka A.5 Popis přepínačů pro specifikování obecných vlastností optimalizace.

Příkaz	Popis funkce
`greedyMaxIterations arg`	Maximální počet iterací algoritmu.
`greedyNumThrowAway arg`	Maximální rozdíl počtu stanic mezi iteracemi algoritmu.

Tabulka A.6 Popis přepínačů pro specifikování parametrů hladové optimalizace (viz. sekce 5.2).

Příkaz	Popis funkce
`geneticPopulatioSize arg`	Velikost populace.
`geneticNumGenerations arg`	Počet generací.
`geneticNumBestSelection arg`	Počet nejlepších jedinců pro zkopírování do následující generace.
`geneticTournamentSelectionTreshold arg`	Pravděpodobnost výběru lepšího jedince v turnajové selekci.
`geneticMutationTreshold arg`	Pravděpodobnost mutace nabíjecí stanice.
`geneticMemberSizeVariance arg`	Směrodatná odchylka rozdílu velikosti nového jedince a jeho rodiče pro mutaci velikosti jedince.

Tabulka A.7 Popis přepínačů pro specifikování parametrů optimalizace genetickým algorimem (viz. sekce 5.3).

Příkaz	Popis funkce
`kMeansNumIterationsOneRun arg`	Počet iterací jednoho běhu algorimu k-means.
`kMeansNumGenerations arg`	Počet generací modelů.

Tabulka A.8 Popis přepínačů pro specifikování parametrů optimalizace algoritmem k-means (viz. sekce 5.4).

Příkaz	Popis funkce
`randomModels`	Spustí pevný počet simulací na modelech s náhodně rozmístěnými nabíjecími stanicemi.
`greedy`	Spustí optimalizaci hladovým algoritmem.
`genetic`	Spustí optimalizaci genetickým algorimem.
`kMeans`	Spustí optimalizaci algoritmem K-Means.

Tabulka A.9 Popis přepínačů pro volbu optimalizační metody.

A.4.2 Příklad spouštění programu

V této sekci uvádíme příklad spuštění programu se vstupními soubory. Po řadě pro křižovatky, silnice a města s názvy:

```
prepared_graph/combined_nodes.txt
prepared_graph/edges.txt
prepared_graph/cities.txt
```

Dále uvádíme výstupní soubory. Po řadě pro soubor s nově přidanými hranami po sloučení komponent souvislosti, pro reprezentaci vrcholů z předpřipraveného grafu a pro reprezentaci hran z předpřipraveného grafu:

```
prepared_graph/added_edges.txt,
prepared_graph/prepared_nodes_new.txt
prepared_graph/prepared_edges_new.txt
```

Dále v tomto příkladě volíme čas simulace 100 minut, celkový počet stanic 300 a počet generací genetického algoritmu 3. Nakonec specifikujeme, že chceme spustit optimalizaci genetickým algoritmem a uložit předpřipravenou mapu. Další parametry jsou zvoleny automaticky, podle výchozí hodnoty nastavené v programu, kterou lze dohledat příkazem:

```
./Traffic_Simulator --help
```

Optimalizaci na parametrech popsaných výše spustíme příkazem:

```
./Traffic_Simulator \
--nodeCityFile="prepared_graph/combined_nodes.txt" \
--edgesFile="prepared_graph/edges.txt" \
--citiesFile="prepared_graph/cities.txt" \
--addedEdgesFile="prepared_graph/added_edges.txt" \
--preparedNodesFile="prepared_graph/prepared_combined_nodes_new.txt" \
--preparedEdgesFile="prepared_graph/prepared_edges_new.txt" \
--simulationTime=100 \
--numStations=300 \
--geneticNumGenerations=3 \
--genetic \
--savePrepared
```

Pokud zvolíme vstupní soubory (křižovatek, silnic a města):

```
prepared_graph/prepared_combined_nodes.txt
prepared_graph/prepared_edges.txt
prepared_graph/cities.txt
```

Pokud nechceme ukládat předpřipravenou podobu grafu a parametry volíme stejně jako v minulém případě, pak příkaz pro spuštění vypadá následovně:

```
./Traffic_Simulator \
--nodeCityFile="prepared_graph/prepared_combined_nodes.txt" \
--edgesFile="prepared_graph/prepared_edges.txt" \
--citiesFile="prepared_graph/cities.txt" \
--simulationTime=100 \
--numStations=300 \
--geneticNumGenerations=3 \
--genetic
```

A.5 Výstup programu

První část výstupu programu je vždy výpis použitých parametrů. Část tohoto výpisu může vypadat například následovně:

```
Save prepared is: 0
Edges: prepared_graph/prepared_edges.txt
Node City: prepared_graph/prepared_combined_nodes.txt
Cities: prepared_graph/cities.txt
Simulation time: 1000
```

```
Length of the edge segment: 1
Number of closest stations: 10
Car consumtion: 0.001
Number of charging stations: 500
Capacity of the charging station: 1
...
```

Následuje část kontrolních zpráv o procesu eliminace křižovatek stupně 2. Program vždy po eliminaci 10000 křižovatek stupně 2 vypíše informaci o úspěšném průběhu procesu eliminace křižovatek. Příklad tohoto výpisu může vypadat následovně:

```
Iteration: 0 Vertices of degree 2 merged and deleted.
Iteration: 10000 Vertices of degree 2 merged and deleted.
Iteration: 20000 Vertices of degree 2 merged and deleted.
Iteration: 30000 Vertices of degree 2 merged and deleted.
Iteration: 40000 Vertices of degree 2 merged and deleted.
Iteration: 50000 Vertices of degree 2 merged and deleted.
Iteration: 60000 Vertices of degree 2 merged and deleted.
Iteration: 70000 Vertices of degree 2 merged and deleted.
Iteration: 80000 Vertices of degree 2 merged and deleted.
Iteration: 90000 Vertices of degree 2 merged and deleted.
Iteration DONE
```

Dále následuje posloupnost kontrolních zpráv o slučování komponent souvislosti, která je zakončena informací o spuštění zvolené optimalizační metody. Tato posloupnost může vypadat následovně (pro spuštění hladové optimalizační metody):

```
Total number of components: 16
Total number of components: 15
Total number of components: 14
Total number of components: 13
Total number of components: 12
Total number of components: 11
Total number of components: 10
Total number of components: 9
Total number of components: 8
Total number of components: 7
Total number of components: 6
Total number of components: 5
```

```
Total number of components: 4
Total number of components: 3
Total number of components: 2
Total number of components: 1
Running greedy algorithm.
```

Dále následuje výpis kontrolních zpráv o běhu simulace po každých 10 odsimulovaných minutách, který je zakončen výsledky simulace a informací o hodnotě ztrátové funkce. Tento výpis se periodicky opakuje pro každou spuštěnou simulaci. Níže uvádíme příklad výpisu jednoho běhu simulace pro dobu 100 minut.

```
Time 10 elapsed!
Time 20 elapsed!
Time 30 elapsed!
Time 40 elapsed!
Time 50 elapsed!
Time 60 elapsed!
Time 70 elapsed!
Time 80 elapsed!
Time 90 elapsed!
Time 100 elapsed!
Number of stations: 500
Number of run down batteries: 654393
Average traveling duration: 98.0642
Average battery difference between start and end: -0.345312
Average waiting times in charging station: 165.868
------------------------------------------------------
Model loss: 6.54896e+07
```

Po skončení všech simulací je program zakončen finálním výpisem informací o modelu s nejmenší hodnotou ztrátové funkce, nebo v případě volby `randomModels` výpisem aritmetického průměru hodnot popisující vlastnosti simulace.

Speciálním případem je optimalizace za pomoci genetického algoritmu, kde jsou před finálním výpisem vypsány hodnoty ztrátové funkce všech modelů použitých v průběhu optimalizace. Část finálního výpisu genetického algoritmu před výpisem informací o nejlepším modelu může pro 3 generace a velikost populace 3 vypadat následovně:

```
Algorithm finished
Losses for the generation: 0
```

```
Loss: 7.58681e+07
Loss: 7.5309e+07
Loss: 7.55552e+07
Losses for the generation: 1
Loss: 7.44727e+07
Loss: 7.56497e+07
Loss: 7.46101e+07
Losses for the generation: 2
Loss: 7.55472e+07
Loss: 7.50069e+07
Loss: 7.56775e+07

Total best loss is: 7.44727e+07
```

Finální výpis nejlepších výsledků všech optimalizačních metod může vypadat následovně:

```
Best model info:
-----------------------------------------------------
Number of stations: 101
Number of run down batteries: 733511
Average traveling duration: 100.09
Average battery difference between start and end: -0.34632
Average waiting times in charging station: 203.302
-----------------------------------------------------
Model loss: 7.33615e+07
```

Tento výpis nám říká, že v nejlepším modelu bylo použito 101 nabíjecích stanic, počet vozidel, kterým došla baterie je 733511, průměrná doba cestování vozidla v minutách je 100.09, průměrný rozdíl počáteční a koncové hladiny baterie je -0.34632 (záporná hodnota znamená, že vozidlo dojelo do cíle s vyšší hladinou baterie, než s jakou vyjíždělo), průměrný čas čekání vozidel na nabíjecích stanicí než se dostanou na řadu je 203.302 a nakonec hodnota ztrátové funkce modelu je $7.33615e + 07$.

Příloha B

Programátorská dokumentace

Účelem této části je především vysvětlit vzájemnou propojenost jednotlivých tříd programu a vysvětlit jeho fungování v rámci celku. Podrobné informace o konkrétních metodách jednotlivých tříd jsou popsány v podrobné dokumentaci programu (viz. appendix C).

B.1 Struktura programu

Program je strukturován do vrstev tak, aby byla zajištěna jednoduchá práci s programem na různých úrovni abstrakce.

Uživatel pracuje pouze s nejabstraktnější vrstvou, kterou je třída `Optimizer`, jež spravuje veškerou logiku spojenou s optimalizací. Tato třída obaluje třídu `TimeTable`, jež je zodpovědná za správné fungování diskrétní simulace a vytváří abstrakci pro třídu `TrafficSimulator`. Třída `TrafficSimulator` je zodpovědná především za generování událostí simulace (např. následující výjezd vozidla, odkud vozidlo vyjede a kam směřuje apod.) a propojení všech objektů simulace. Důležitým aspektem této vrsty je správa načítání mapy. Třída spravuje objekty třídy `MapReader` a `GraphAdjuster` zodpovědné za správné předzpracování mapy. Dále je zodpovědná za veškeré objekty abstraktní třídy `Vehicle`, které reprezentují vozidla v simulaci. V neposlední řadě také vytváří abstrakci pro třídy `Map`, která reprezentuje nejnižší vrstvu programu. Třída `Map` je zodpovědná za veškeré grafové operace a veškeré operace pracující s reálnou zeměpisnou pozicí. Její součástí je objekt knihovny `boost` s názvem `graph_` reprezentující graf silniční sítě. Dále třída shlukuje informace o nabíjecích stanicích (reprezentované pomocí třídy `ChargingStation`) a městech náležících do simulované oblasti.

Součástí programu je také řada dalších tříd pokrývající pomocné operace programu. Některé z nich jsme již zmínili (`MapReader`, `GraphAdjuster`, `Vehicle`, `ChargingStation`). Za zmínku stojí ještě třídy `Car`, jež je potomkem třídy

`Vehicle` reprezentující typ vozidla "auto" (viz. sekce 3.5.4), `Edge`, která pokrývá funkcionalitu a vlastnosti silnice v mapě (viz. definice 4) a `Node`, jež plní funkci křižovatek na mapě (viz. definice 1). Zbylé třídy slouží především pro shlukování souvisejících informací (parametry simulátoru, optimalizací, vlastnosti nabíjecích stanic, modelu apod.).

Jedinou vyjímkou, jež není součástí struktury výše je část pracující s parametry programu, jež je realizována pomocí knihovny `boost`. Přesněji pomocí funkcionalit z `program_options`.

B.2 Třída Optimizer

Třída `Optimizer` je hlavní třídou celého programu, která na základě uživatelských vstupů spouští a realizuje zvolenou optimalizaci. Vytváří abstrakci nad třídou `TimeTable`, jež používáme pro spouštění simulací. Do rozhraní třídy náleží metody spouštějící optimalizační algoritmy:

```
GreedyAlgorithm
GeneticAlgorithm
KMeansAlgorithm
```

Metoda, jež počítá ztrátovou funkci modelu po uběhnutí simulace:

```
ModelLoss
```

A metoda, která opakovaně simuluje provoz na náhodně vygenerovaném modelu:

```
RunMultipleSimulations
```

Tato metoda slouží pro analýzu optimalizačních metod pro porovnání metod s náhodným přístupem. Kapitola 5 nabízí podrobnější popis jednotlivých optimalizačních metod a ztrátové funkce. Dále třída obsahuje řadu privátních metod zajišťujících správnou funkčnost hlavních metod popsaných výše. Více informacím je specifikováno v podrobné dokumentaci (viz. appendix C).

B.3 Třída TimeTable

Třída `TimeTable` je třída zodpovědná za správné fungování diskrétní simulace. Vytváří abstrakci třídy `TrafficSimulator` a slouží jako rozhraní pro zisk informací o vlastnostech simulace (pro analýzu optimalizačních metod). Do jejího rozhraní náleží především funkce spravující diskrétní simulaci. Další skupinou

metod této třídy jsou metody s předponou `Get`, s jejíž pomocí jsme schopni získat relevantní informace o simulaci. Jediná metoda, jež nenáleží do zmíněných skupin je `LoadMap`. Ta slouží jako zprostředkovatel žádosti o načtení mapy a spouští příslušné metody ve třídě `TrafficSimulator`.

Kapitola 4 nabízí podrobnější popis fungování diskrétní simulace.

B.4 Třída TrafficSimulator

Třídu `TrafficSimulator` lze považovat za jádro celého simulátoru, jejíž funkce je spravovat objekty simulace. V rámci simulace je zodpovědná především za generování náhodných jevů (pozice nabíjecích stanic, generování vozidel, doby následující akce apod.) (viz. sekce 3.5.1), jež je zastoupeno metodami s předponou `Generate`. Pomocí této třídy jsou v simulaci spravovány objekty vozidel (potomci třídy `Vehicle`). Zásadní vlastností této třídy je vytváření rozhraní pro práci s třídou `Map`, jež pokrývá veškeré grafové operace programu. S třídou `Map` také úzce souvisí třídy `MapReader` a `GraphAdjuster`, sloužící pro načtení a předpřípravu uživatelské reprezentace mapy do formátu, jenž je uložen ve třídě `Map`. Appendix A.3 podrobně popisuje formát vstupních souborů.

B.5 Třída Map

Třída `Map` je technicky nejkomplexnější část programu. Slouží jako rozhraní pro veškeré grafové operace a operace, jež pracují s reálnou geografickou pozicí na mapě. Graf dopravní sítě je reprezentován pomocí příslušných objektů grafové části knihovny `boost`. Dále jednotlivé křižovatky (definice 1) a silnice (definice 4) jsou navíc reprezentovány vlastními objekty třídy `Node` (křižovatky) a `Edge` (silnice). Třída také sdružuje všechny nabíjecí stanice (definice 10) vyskytující se v mapě (reprezentovány třídou `ChargingStation`) a informace o rozmístění a populaci měst (definice 8) v simulované oblasti.

Třída obsahuje metody pro spravování podoby grafu, informací o nabíjecích stanicích a městech. Tyto funkcionality pokrývají metody s předponou `Add` a `Set`. Dále jsou zde metody pro získávání informací o objektech mapy začínající příponou `Get` a `Exist` a metody přímo související s diskrétní simulací a optimalizací (`ResetSimulation` a `SimulateVehiclePassedThroughEdge`). Poslední skupinou metod jsou nejkomplexnější operace třídy pracující s grafem, či mapou.

B.5.1 Operace na mapě

S ohledem na řešený problém je velmi důležitým aspektem návrhu hledání nejkratších cest. Program umí pracovat s 2 variantami algoritmu pro hledání nejkratších

cest v grafu. Těmi jsou Dijkstrův algoritmus, jenž je využívám především pro hledání vzdáleností v grafu (např. v K-Means), a A-Star (sekce 1.4), jenž je používám v diskrétní simulaci pro hledání nejvýhodnější cesty vozidla (viz. sekce 3.5.2). Jako heuristika algoritmu A-Star je zvolena reálná geografická vzdálenost mezi křižovatkami, která je spočítána pomocí příslušných vztahů pro výpočet sférické vzdálenosti (viz. sekce 1.1).

Dijkstrův algoritmus, algoritmus A-Star a výpočet sférické vzdálenosti je realizován pomocí metod (ve stejném pořadí, jak jsou vypsány):

```
DijkstraFindDistances
FindShortestPath
ComputeSphericalDistance
```

Speciální metodou související s výpočtem nejkratší cesty v grafu je:

```
FindVehiclePath
```

Ta hledá nejkratší cestu mezi vrcholy, ale bere také v úvahu potřebu návštěvy nabíjecí stanice, z níž vybere heuristicky tu nejvhodnější (viz. sekce 3.5.3).

Další metodou pracující s grafy je metoda pro hledání komponent souvislosti, využívaná v předpřípravě mapy pro simulaci.

```
TestComponents
```

Třída také obsahuje metodu hledající nejbližšího města od křižovatek a nejbližší nabíjecí stanice od středů měst:

```
FindCityNodes
FindCitiesNearestChargingStations
```

B.6 Třídy objektů mapy

Mezi třídy reprezentující objekty na mapě patří reprezentace křižovatek (`Node`), silnic (`Edge`) a nabíjecích stanic (`ChargingStation`). Tyto třídy slouží především pro udržování informací o vlastnostech objektů, obsahují však také jednoduchou logickou vrstvu pokrývající především správnou inicializaci, či aktualizaci informací v průběhu simulace. Třída `ChargingStation` také pokrývá logickou vrstvu související s nabíjením vozidel (spravuje čekání vozidel v řadě, doby nabíjení apod.).

B.7 Třídy pro načítání a předpřípravu dat

Reprezentaci mapy je potřeba na začátku programu načíst z předpřipraveného formátu (viz. appendix A.3) a řádně zpracovat do podoby s níž bude simulátor dále pracovat (definice 5). K tomuto účelu slouží třídy `MapReader` a `GraphAdjuster`.

B.7.1 Třída MapReader

Třída `MapReader` převádí vstupní reprezentaci dat do požadovaného formátu pomocí metod s předponou `Load`. Nejdůležitější metodou tohoto typu je metoda `LoadMap`, jež načte veškeré potřebné informace a provede také předpřípravu mapy pro správné fungování simulátoru a zlepšení efektivity. Zmíněné úpravy provádí s pomocí třídy `GraphAdjuster`. Dalším charakteristickou vlastností třídy `MapReader` je možnost uložení aktuální reprezentace mapy (po předpřípravě mapy) do souboru v požadovaném vstupním formátu pro efektivnější opakované načítání mapy. Mezi tyto metody patří takové, jejichž přípona je `Write`.

Třída nabízí možnost načítat a zapisovat také reprezentaci všech nabíjecích stanic modelu metodami:

```
LoadChargingStations
WriteStations
```

B.7.2 Třída GraphAdjuster

Vzhledem k typicky velkému rozměru vstupních dat je pravděpodobné, že výsledná reprezentace mapy nebude splňovat veškeré potřebné vlastnosti nutné k správnému chování programu, či bude jejich reprezentace zbytečně neefektivní. Typickými příklady tohoto problému je rozpad grafu na více komponent souvislosti či vznik vrcholů stupně 2. Podrobněji tuto problematiku popisuje kapitola 2.

Řešení těchto potíží zajisťuje třída `GraphAdjuster` úzce pracující s třídou `MapReader`. Ta po načtení grafu zkontroluje zda je graf souvislý a pokud není, pak jej převede do souvislé podoby (viz. sekce 3.6.1). Dále se také zbavuje veškerých vrcholů stupně 2, čímž výrazně zefektivňuje budoucí simulaci (viz. sekce 3.6.1). Operace jsou realizovány metodami:

```
MergeTwoComponents
MergeDegreeTwoVertices
```

B.8 Třída Vehicle

Třída `Vehicle` je třída zahrnující veškeré informace o daném vozidle (sekce 3.5), s jejíž pomocí je možné simulovat chování vozidel a příslušně tak upravovat simulátor. Při návrhu této třídy byla snaha co nejobecněji vyjádřit chování vozidel různých druhů a jednotlivé druhy vozidel simulace následně reprezentovat třídou, jež je potomkem `Vehicle`. V naší implementaci je zvolen pouze jeden typ vozidla a to typ `Car`, jehož specifickou vlastností je, že po dosažení cílové destinace vrací zpět do místa výjezdu (viz. sekce 3.5.4).

Metody třídy `Vehicle` slouží převážně k získávání informací o vozidle jinými objekty (např. aktuální stav baterie, aktuální pozice, zda míří na nabíjecí stanici apod.), případně k nastavování stavových parametrů vozidel (míří na nabíjecí stanici, začalo čekat v řadě na nabíjení apod.) Objekty třídy jsou také schopny spočítat čas a hladinu baterie po průjezdu silnicí, dokonce i posloupnosti silnic (celé cesty). Tyto informace jsou důležité především pro diskrétní simulaci (kapitola 4) a při rozhodování, zda je potřeba jet na nabíjecí stanici (sekce 3.5.3). Zbylé metody slouží pro simulaci akcí vozidel. Je zde metoda pro aktualizaci trasy vozidla a nabití vozidla:

```
UpdateVehiclePath
ChargeBattery
```

Nakonec obsahuje třída i metody, jež řádně simulují průjezd vozidla po silnici. V tomto případě jsou naimplementovány 3 varianty, jež řeší všechny možné varianty průjezdu vozidla (sekce 4.3.2) po silnici (průjezd silnice, výjez z vnířku ven, cesta uvnitř silnice).

B.9 Pomocné třídy

Zbylé třídy implementace slouží především k zapouzdření souvisejících informací, jako jsou například parametry simulátoru, optimalizací, informace o pozicích na mapě, shrnující popis modelu a mnoho dalších. Názvy zmíněných tříd jsou:

```
SimulationParameters
OptimizerParameters
MapPosition
ModelRepresentation
```

Tyto třídy slouží především k uchovávání informací o specifických objektech a zjednodušenou manipulaci s daty.

B.10 Testy programu

K programu jsou také dodány testy zvolených problematických funkcionalit a především pak testy načítání uživatelských dat, která jsou často i s ohledem na objem vstupních dat zdrojem problémů. Jedná se o testy rozhraní Microsoft Unit Testing Framework pro C++. Návod na spuštění testů je k dohledaní v [12].

Příloha C

Přílohy práce

K práci jsou přiloženy následující soubory.

Adresář se zdrojovými kódy a unit testy simulátoru dopravy:

```
./source_code/
    Traffic_Simulator_cmake/
    Traffic_Simulator_VS/
    README.md
```

Podrobná programátorská dokumentace:

```
./Traffic_Simulator_Manual.pdf
```

Předpřipravené vstupní soubory simulátoru:

```
./prepared_graph/
    cities.txt
    combined_nodes.txt
    edges.txt
    prepared_combined_nodes.txt
    prepared_edges.txt
```

Adresář s programem pro předpřípravu mapy:

```
./map_preprocessing/
    map_parser.py
    map_reader.py
    README.md
```

Adresář s podrobnými výsledky simulací a výpis jejich parametrů:

```
./analysis_results/
    greedy_results/
    genetic_results/
    kmeans_results/
    random_results/
    simulation_parameters_results/
    README.md
```

Adresář s programem pro přípravu úloh do Metacentra a analýzu výsledků experimentů:

```
./experiment_analysis/
    metacentrum_grid_search.py
```