FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Diskrétní simulace automobilové dopravy se spojitým časem
Student:	Patrik Schweika
Vedoucí:	doc. RNDr. Pavel Surynek, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Teoretická informatika
Katedra:	Katedra teoretické informatiky
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Cílem práce je navrhnout simulátor silniční dopravy, který bude používat diskrétní model silniční sítě a spojitý časový model. Předpokládáme, že silniční síť bude modelována jako neohodnocený konečný graf, přičemž jednotlivé silnice budou reprezentovány podgrafy sestávající cest, jejichž počet a délka se bude odvíjet od vlastností silnice, jako je reálná délka a kapacita. V rámci simulátoru budeme chtít testovat průchodnost dopravního systému. Úkoly řešitele budou následující:

1. Prostuduje techniky diskrétní simulace se spojitým časem a zhodnotí jejich využitelnost v simulaci dopravy.

2. Na základě nastudovaných technik navrhne simulátor dopravy, který bude umožňovat studium dopravní průchodnosti. Předpokládáme využití simulátoru pro porovnání lokálního agentního řízení dopravy a centralizovaného řízení.

3. Simulátor implementuje a opatří jej vhodným grafickým výstupem a uživatelským rozhraním. Systém zdokumentuje po uživatelské a programátorské stránce.

–

[1] Philipp Andelfinger, Jordan Ivanchev, David Eckhoff, Wentong Cai, Alois C. Knoll: From Effects to Causes: Reversible Simulation and Reverse Exploration of Microscopic Traffic Models. SIGSIM 2019, 173-184
[2] Guni Sharon, Michael Albert, Tarun Rambha, Stephen D. Boyles, Peter Stone: Traffic Optimization for a Mixture of Self-Interested and Compliant Agents. AAAI 2018: 1202-1209
[3] Dustin Carlino, Mike Depinet, Piyush Khandelwal, Peter Stone: Approximately Orchestrated Routing and Transportation Analyzer: Large-scale traffic simulation for autonomous vehicles. ITSC 2012: 334-339
[4] Philipp Andelfinger, Yadong Xu, Wentong Cai, David Eckhoff, Alois C. Knoll: Fast-Forwarding Agent States to Accelerate Microscopic Traffic Simulations. SIGSIM 2018, 113-124

Elektronicky schválil/a doc. Ing. Jan Janoušek, Ph.D. dne 6. prosince 2020 v Praze.

Bakalářská práce

DISKRÉTNÍ SIMULACE AUTOMOBILOVÉ DOPRAVY SE SPOJITÝM ČASEM

Patrik Schweika

Fakulta informačních technologií ČVUT v Praze
Katedra teoretické informatiky
Vedoucí: doc. RNDr. Pavel Surynek, Ph.D.
13. května 2021

Odkaz na tuto práci: Patrik Schweika. *Diskrétní simulace automobilové dopravy se spojitým časem.* Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Obsah

Seznam obrázků

Seznam tabulek

Seznam algoritmů

Chtěl bych poděkovat především vedoucímu své práce doc. RNDr. Pavlu Surynkovi, Ph.D., za jeho cenné rady a nápady při tvorbě této bakalářské práce. Dále bych chtěl poděkovat rodině a přátelům za podporu během studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 13. května 2021 .....

Abstrakt

V práci se zabýváme návrhem a realizací simulátoru automobilové dopravy s diskrétním prostorem a spojitým časem, jehož hlavní využití předpokládáme pro porovnání algoritmů na plánování tras. V simulátoru implementujeme dva odlišné algoritmy pro plánovaní tras. Prvním je Dijkstrův algoritmus, který plánuje trasy podle nejkratších cest. A druhým je centralizovaný algoritmus založený na maximálním souběžném toku. Nakonec pomocí simulátoru provádíme experimenty na připravených testovacích scénářích, kde tyto algoritmy porovnáváme. Výsledky experimentů ukázali, že průchodnost silniční sítě je významné lepší pro tokový algoritmus. V příloze práce lze nalézt dokumentaci a uživatelskou příručku k simulátoru.

Klíčová slova diskrétní simulace, simulace dopravy, plánováni tras, maximální souběžný tok, Scala

Abstract

In this thesis we deal with design and implementation of vehicle traffic simulator with discrete space and continuous time. The main intended use of the simulator is for comparison of route planning algorithms. We implement two distinct algorithms for route planning in the simulator. First one is Dijkstra's algorithm, which plan routes according to the shortest paths. Second is centralized algorithm based on maximum concurrent flow. Finally, we perform experiments in the simulator on prepared testing scenarios, where we compare the algorithms. Experiment results shown that the network throughput is significantly higher for flow-based algorithm. Documentation and user manual of the simulator can be found in the appendix.

Keywords discrete simulation, traffic simulation, route planning, maximum concurrent flow, Scala

Seznam zkratek

JSON	JavaScript Object Notation
LP	Lineární Programování
MCF	Maximum Concurrent Flow
XML	Extensible Markup Language

Úvod

Pro většinu z nás je silniční doprava nedílnou součástí našich životů. Ať už se potřebujeme dostat do práce nebo do obchodního střediska, využíváme k tomu sdílenou dopravní infrastrukturu. S přibývajícím počtem vozidel se dopravní síť rychleji zahltí a naše cesty trvají déle a déle. Možný příchod autonomních vozidel nám otevírá nové možnosti v oblasti plánování tras, který by byli schopné tento problém zmírnit. Vozidla by mohla mezi sebou kooperovat tak, aby dopravní síť zahltili co nejméně a tím zvětšili její propustnost. Abychom ale byli schopni určit, který způsob pro plánování tras je pro tento problém lepší, musíme je otestovat a porovnat mezi sebou. Prvním nástrojem, na kterém se dá algoritmus pro řízení dopravy otestovat je dopravní simulátor. V simulátoru si můžeme připravit reálné dopravní scénáře a vyzkoušet, jak si na nich algoritmus povede.

Výsledek práce lze využít pro více účelu. Ať už k otestování algoritmu pro plánování tras, nebo ke zlepšení stávajících dopravních síti či výstavbě nových silničních sítí. Proto si myslíme, že by z toho mohli těžit komunity z obou oborů.

Téma jsem si zvolil, jelikož většina existujících řešení modelují čas diskrétně a prostor spojitě a my chceme vyzkoušet odlišný přístup se spojitým časem a diskrétním prostorem a zhodnotit jeho použitelnost pro simulaci dopravy.

V práci se nejprve zabýváme analýzou na téma diskrétní simulace, simulace dopravy a existujícími řešeními v této oblasti. Poté návrhem a implementací dopravního simulátoru, jehož hlavní využiti bude pro porovnávání algoritmů na plánování tras. A na závěr experimenty, kde otestujeme výkonnost výsledného simulátoru, a porovnáme algoritmy pro plánování tras.

Práci členíme na pět hlavních kapitol a závěr. V další kapitole cíle práce, si zadání blíže popíšeme a specifikujeme si jednotlivé cíle práce. Dále si stanovíme funkční požadavky, které musí výsledný simulátor splňovat a představíme si hypotézu, kterou budeme chtít pomocí simulátoru otestovat na experimentech. V první kapitole si představíme téma počítačové simulace se zaměřením na diskrétní simulace. Rozebereme si způsoby, jak lze čas diskrétní simulace modelovat. Dále si představíme několik náhodných rozdělení pomocí, kterých se v simulacích nejčastěji modelují náhodné proměnné. V rámci druhé kapitoly si uvedeme téma dopravních simulátoru a zavedeme definice z teorie grafů, které budeme používat při návrhu. Na závěr kapitoly zhodnotíme několik existujících řešení pro simulaci dopravy. Následně ve třetí kapitole provedeme návrh simulátoru a algoritmů pro plánování tras, pomocí nastudovaných technik z předchozích kapitol. Ve čtvrté kapitole provedeme výzkum dopravy v simulátoru na několika připravených scénářích. Zde budeme porovnávat implementované algoritmy pro plánování tras. Výsledky experimentů okomentujeme a vyvodíme z nich závěry. V poslední kapitole provedeme výkonnostní testy simulátoru na scénářích z předchozí kapitoly. A v závěru si shrneme východiska práce, probereme možnosti, jak by se dala práce rozšířit a jaký měla táto práce přínos.

Cíle práce

Cílem této práce je návrh a implementace simulátoru dopravy s diskrétním prostorem a spojitým časem, který bude umožňovat studium průchodnosti dopravy na základě použitého algoritmu pro plánování tras. Zejména chceme v naší práci porovnat algoritmy pro lokální plánování tras, které plánují trasy tak, aby byla optimální pro jednotlivá vozidla. A algoritmy pro centralizované plánování tras, které plánují trasy tak, aby byli optimální v rámci celé dopravní sítě. Naše základní hypotéza je taková, že centralizované plánování tras umožní silničním sítím pojmout větší počet vozidel, než se zahltí a zvětšit průchodnost celé sítě. Na závěr chceme použít implementovaný simulátor, abychom tuto hypotézu otestovali na několika připravených experimentech. Pro vyhotovení této práce je nutné splnit několik následujících úkolů.

Prvním úkolem je rešerše na téma počítačová simulace se zaměřením na diskrétní simulace a způsoby modelování času. Dále je nutné provést analýzu na simulátory dopravy. Konkrétně na typy dopravních simulátorů a jejich vstupy. Dále je potřeba nastudovat dva odlišné způsoby řízení dopravy, lokální a centrální, které v simulátoru použijeme. Po dohodě s vedoucím jsme se shodli na následujících variantách. Pro lokální řízení dopravy použijeme Dijkstrův algoritmus. A pro centralizované řízení dopravy použijeme algoritmus založený na více komoditním toku v síti. Na závěr rešeršní části je potřeba zhodnotit již existující řešení pro simulaci dopravy. Zejména jejich návrh z pohledu počítačové simulace a možnosti plánování tras.

Dalším úkolem je návrh samotného dopravního simulátoru na základě nastudovaných technik diskrétní simulace a simulace dopravy. Simulátor musí být navržený dostatečně flexibilně, aby bylo možné přidávat implementace dalších plánovacích algoritmů. Abychom byli schopni provést experimenty je potřeba navržený simulátor a vybrané způsoby plánování tras implementovat. Výsledný simulátor musí splňovat následující funkční požadavky:

- Možnost nahrát vytvořené dopravní situace z konfiguračního souboru, který bude obsahovat dopravní síť, toky vozidel a výběr plánovacího algoritmu.
- Statistický výstup, který bude možné zobrazit v průběhu simulace i po jejím skončení.
- Možnost exportovat statistické výstupy do souboru.
- Vhodný grafický výstup pro vizualizaci probíhající simulace a jejích aktuálních statistik.
- Vhodné uživatelské rozhraní pro ovládání simulátoru.

Implementovaný simulátor je potřeba opatřit programátorskou dokumentaci a uživatelskou příručku.

Posledním úkolem práce je připravit testovací scénáře pro simulátor a provést na nich experimenty, které porovnají algoritmy pro řízení dopravy a otestují výkonnost simulátoru. V experimentech je potřeba otestovat naší hypotézu o tom, že pro centralizované plánování tras bude průchodnost silniční sítě lepší než pro lokální plánování. Výsledky experimentů je nutné řádně okomentovat a zdůvodnit, zda jsou v souladu s naší hypotézou či nikoliv.

Kapitola 1

Počítačová simulace

Pod pojmem počítačová simulace si můžeme představit počítačový program, který slouží pro studování různých systému, ať už reálných nebo abstraktních. Pro vytvoření takové simulaci si musíme zvolit model, který budeme schopni implementovat na počítači. Výstupem simulace jsou nejčastěji statistické výstupy a vizualizace dané simulace. Pokud se model chová dostatečné podobně jako sledovaný systém, můžeme z těchto výstupů vyvozovat určité skutečnosti o simulovaném systému. [1]

Simulátory se podle [2] rozdělují podle několika kritérii:

- **Diskrétní** – stav simulace se mění pouze v určitých časových okamžicích.
- **Spojité** – stav simulace se mění spojitě v čase. Tento typ se nejčastěji používá pro simulace fyzikálních systému. Tento systém je typicky modelován diferenciálními rovnicemi, které definují, jak se simulované veličiny mění v čase.

- **Statické** simulují model v určitém časovém okamžiku. Typickým příkladem je tzv. Monte Carlo simulace, která slouží k odhadování pravděpodobností různých jevů.
- **Dynamické** simulují průběh modelu v čase. Simulace je typicky ukončena po uplynutí předem dané doby nebo nějakou ukončující podmínkou.

- **Deterministické** neobsahují náhodné proměnné. Každý běh simulace pro stejný model bude mít stejné výsledky. Typickým příkladem deterministické simulace je spojitá simulace modelována pomocí diferenciálních rovnic.
- **Stochastické** obsahují náhodné proměnné. Každý běh stochastické simulace se stejným vstupem může mít odlišné výsledky. Tento typ se velmi často používá pro simulace reálných systému, kde spousta jevů nelze předpovídat a tak je považujeme za náhodné. Stochastické simulace používají generátory pseudonáhodných čísel.

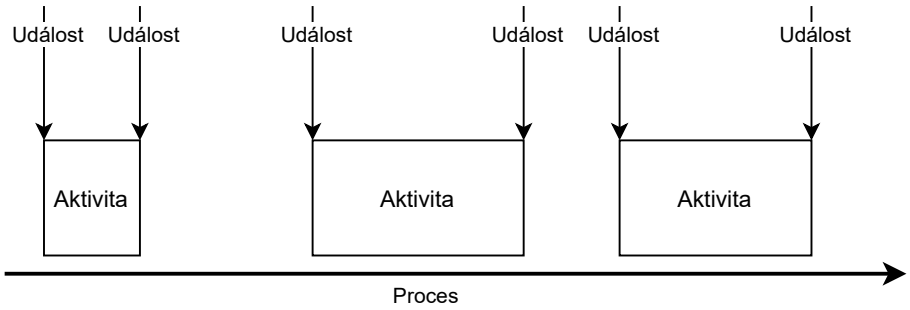
My se v naší práci zaměříme na diskrétní, dynamické a stochastické simulace.

1.1 Diskrétní simulace

Je typ počítačové simulace, kdy se vnitřní stav simulace mění pouze v určitých časových okamžicích. Těmto okamžikům říkáme události. Znakem diskrétních simulací je, že mezi událostmi se stav simulace nemění. Události se typicky vkládají do fronty. Kde na vrcholu fronty je vždy událost, která nastane nejdříve. Simulace skončí pokud je fronta událostí prázdná nebo nastane ukončující podmínka. [3]

Podle [3] jsou základní komponenty diskrétní simulace:

- *Čas* je proměnná, která uchovává aktuální čas simulace.
- *Stav simulace* je množina proměnných, které definují aktuální stav systému.
- *Událost* (angl. Event) je základní jednotkou diskrétní simulace. Nastává v jeden určitý okamžik a mění stav simulace.
- *Fronta událostí* (angl. Future Event List) obsahuje události seřazené podle času, kdy mají nastat. Na vrcholu fronty se nachází událost, která nastane nejdříve. Obvykle je implementována jako prioritní fronta, kde priorita je čas události.
- *Aktivita* (angl. Activity) je déle trvající činnost, která se skládá ze dvou událostí. První událost nastane na začátku činnosti a druhá na jejím konci.
- *Proces* (angl. Process) se nazývá objekt, který se skládá ze sekvence aktivit nebo událostí. Až se všechny jeho události vykonají, tak proces zanikne.
- *Generátory náhodných čísel* (angl. Random Number Generators) ze kterých generujeme pseudonáhodná čísla pro simulaci.



Obrázek 1.1 Ilustrace průběhu procesu v simulaci

▶ **Příklad 1.1.** Zde si uvedeme příklad diskrétní simulace se zadefinovanými pojmy. Vezměme si například pobočku pošty, kde budeme chtít modelovat příchody zákazníků, zařazení do fronty u přepážky a odchody zákazníků. Zde budeme mít 2 typy procesů zákazník a přepážka. Zákazník se bude skládat ze 3 událostí: příchod, zařazení do fronty k přepážce a odchod a přepážka bude mít aktivitu vyřízení požadavku zákazníka. V tabulce 1.1 si ukážeme možný běh simulace pro náš model s 1 přepážkou a 3 příchozími zákazníky v časech 0:10, 1:40 a 8:00. S tím, že zařazení zákazníka do fronty trvá 5 sekund a vyřízeni požadavku zákazníka může trvat od 2 do 5 minut.

Tabulka 1.1 Příklad diskrétní simulace pobočky pošty

čas v min	proces	událost	fronta u přepážky
0:00	-	začátek simulace	∅
0:10	zákazník 1	příchod	∅
0:15	zákazník 1	zařazení do fronty (5 s)	[zák. 1]
1:40	zákazník 2	příchod	[zák. 1]
1:45	zákazník 2	zařazení do fronty (5 s)	[zák. 1, zák. 2]
2:15	přepážka	zpracování požadavku 1 (2 min)	[zák. 1, zák. 2]
2:15	zákazník 1	odchod	[zák. 2]
5:15	přepážka	zpracování požadavku 2 (3 min)	[zák. 2]
5:15	zákazník 2	odchod	∅
8:00	zákazník 3	příchod	∅
8:05	zákazník 3	zařazení do fronty (5 s)	[zák. 3]
13:05	přepážka	zpracování požadavku 3 (5 min)	[zák. 3]
13:05	zákazník 3	odchod	∅
13:05	-	konec simulace	∅

1.1.1 Modely času

Jedna z hlavních funkcí diskrétní simulace je posun času. Samotný simulátor se musí starat o jeho posouvání a vykonávání událostí ve správném sekvenčním pořadí. Podle [2] existují dva způsoby jak lze čas diskrétní simulace modelovat:

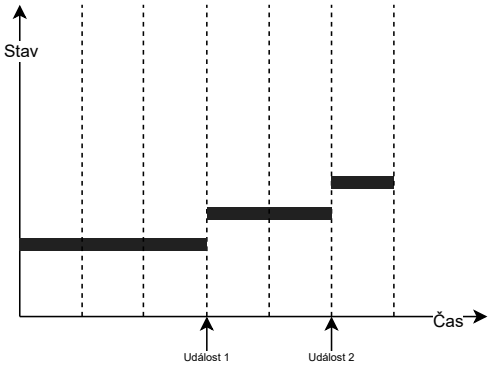
Time slicing posouvá čas simulace po pevně zvoleném kroku, např.: pokaždé posune čas o pět vteřin. Čas simulace se posune bez ohledu na to jestli nastane nějaká událost (změna stavu simulace). Tento způsob je velmi citlivý na vzdálené události a vykoná spoustu zbytečné práce. Pokud další událost nastane za pět minut tak, potom tento způsob bude muset 60-krát posunout čas o pět vteřin, než se změní stav simulace.

Tato metoda se používá, pokud chceme spojitý prostor v simulaci. Velikost časového kroku nám bude určovat dobu za kterou se aktualizují pozice pohybujících se objektů v prostoru a bude mít dopad na jemnost celé simulace. Pokud bude krok příliš velký, simulace nebude přesná a pohyb v prostoru bude skokovitý. Naopak pokud zvolíme krok velmi malý, dostaneme jemnější a detailnější simulaci, ale zaplatíme za to častějšími aktualizacemi a simulace tím pádem poběží mnohem déle. Znázornění posunu času pomocí této techniky lze vidět na obrázku 1.2.

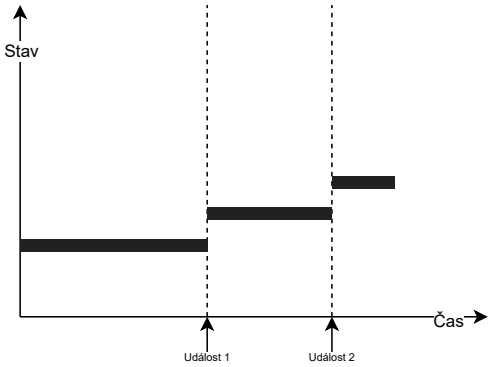
Next event posouvá čas simulace na nejbližší naplánovanou událost, např. pokud je nejbližší naplánovaná událost za 5 minut, tak se čas rovnou posune o 5 minut. Čas simulace se tedy pohybuje pouze po naplánovaných událostech a nemůže nastat případ, kdy bychom posunuli čas a nebyla by žádná událost k provedení. Časový krok je variabilní podle časové vzdálenosti mezi vykonávanými událostmi. Tento způsob je méně výpočetně náročný než předešlí, pokud jsou události velmi vzdálené.

Tento způsob nejčastěji využíváme, pokud máme diskrétní prostor v simulaci a tím pádem můžeme naplánovat události o pohybu na přesný čas, protože víme za jakou dobu se dostaneme na další pozici při dané rychlosti. Znázorněni posunu času této techniky lze vidět na obrázku 1.3.

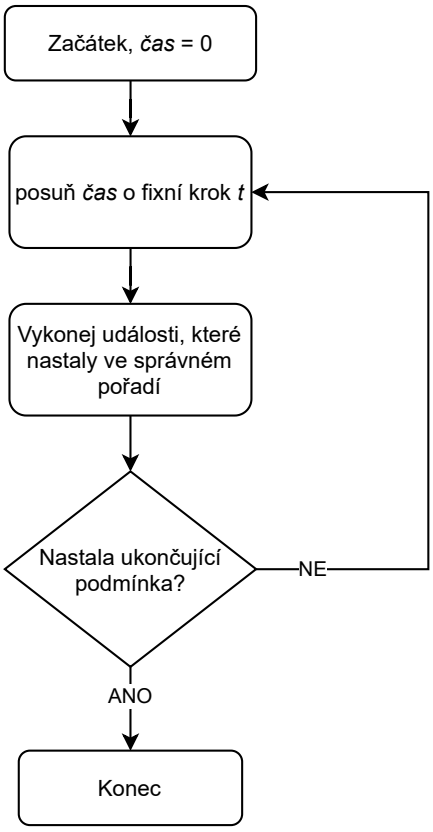
Pokud chceme diskrétní simulaci s posunem času pomocí metody *Next event* vizualizovat v reálném čase. Tak je potřeba myslet na různé velikosti časových kroků. Při použiti metody *Time slicing* tento problém řešit nemusíme.



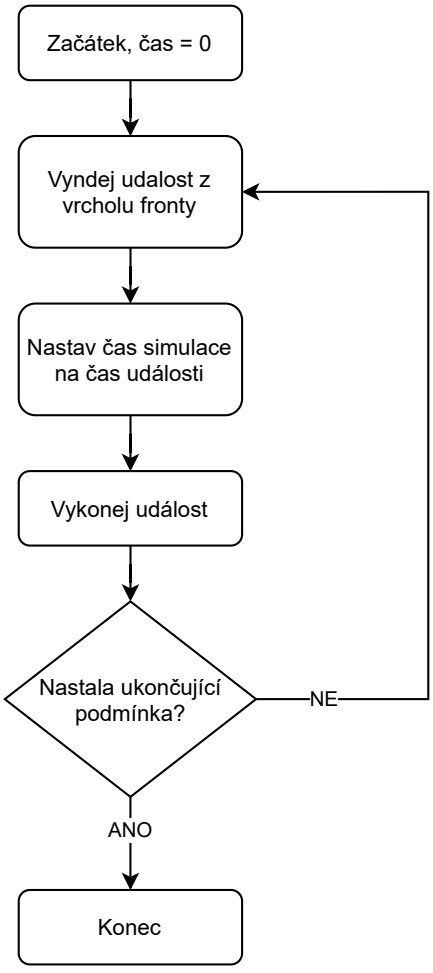
Obrázek 1.2 Ilustrace metody Time slicing. Svislé čerchované čáry značí posun času simulace



Obrázek 1.3 Ilustrace metody Next event



Obrázek 1.4 Vývojový diagram metody Time slicing



Obrázek 1.5 Vývojový diagram metody Next event

1.1.2 Generování náhodných čísel

Mnoho jevů ve skutečnosti nelze předvídat. Může se jednat např.: o příchody zákazníků do obchodu. Nikdy nevíme dopředu v kolik přesně přijde další zákazník. Proto bývají generátory pseudonáhodných čísel nedílnou součástí simulací, abychom byli schopni tyto jevy modelovat. Tato generovaná čísla, ale nemohou být úplně náhodná. Proto často uvažujeme, že pochází z určitého náhodného rozdělení. [3]

Dále si představíme náhodné rozdělení, která se nejčastěji v simulacích používají a u některých si uvedeme příklad, jak by se dala využít. Následující definice jsou čerpány z knihy [4].

Rovnoměrné rozdělení je spojité rozdělení s konstantní hustotou mezi parametry a a b, pro které platí $a < b$. Značí se $U(a, b)$, kde a je spodní a b horní hranice. Hustota uniformního rozdělení:

$$f(x) = \begin{cases} \frac{1}{b-a} & x \in (a, b) \\ 0 & \text{jinde} \end{cases}$$

Poissonovo rozdělení je diskrétní rozdělení a vyjadřuje pravděpodobnost, že nastane x výskytu nezávislých události za určitý časový interval. Značí se $Poisson(\lambda)$, kde λ udává průměrný počet událostí za časový interval. Vzorec pro pravděpodobnost Poissonova rozdělení:

$$P(X = x) = \frac{e^{-\lambda}\lambda^x}{x!},\ x = 0, 1, \ldots \infty$$

Typické použiti tohoto rozdělení je pro modelování jakéhokoli druhu událostí, které nastávají nezávisle na sobě v čase. Na příkladu si uvedeme jak rozdělení použít. Pokud si vezmeme příklad o příchodech zákazníků a máme empiricky změřeno, že každou hodinu přijde 10 zákazníku. Tak použijeme $Poisson(10)$, ze kterého budeme vybírat náhodné hodnoty.

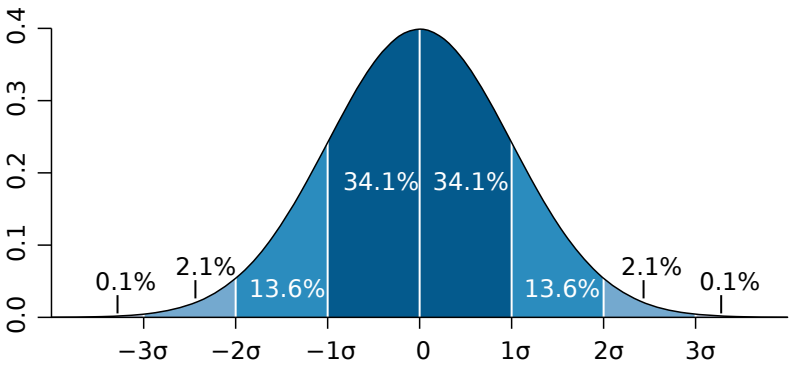
Exponenciální rozdělení je spojité rozdělení a vyjadřuje délku intervalu mezi nezávisle se vyskytujícími událostmi, jejichž pravděpodobnost výskytu má Poissonovo rozdělení. Značí se $Exp(\lambda)$, kde $\lambda > 0$ a převrácená hodnota $\frac{1}{\lambda}$ nám udává střední hodnotu intervalu mezi výskyty událostí. Hustota exponenciálního rozdělení:

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & \text{jinde} \end{cases}$$

Normální rozdělení je spojité rozdělení, jehož hustota je dána Gaussovou křivkou. Značí se $N(\mu, \sigma^2)$, kde μ je střední hodnota a σ^2 rozptyl rozdělení. Většinu hodnot najdeme kolem střední hodnoty, čím dále půjdeme od střední hodnoty, tak bude hodnot ubývat. Přesněji najdeme 68% hodnot v intervalu $(\mu - \sigma, \mu + \sigma)$, 95% hodnot v $(\mu - 2\sigma, \mu + 2\sigma)$ a 99,7% v $(\mu - 3\sigma, \mu + 3\sigma)$. Hustota normálního rozdělení:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu^2)}{2\sigma^2}},\ x \in (-\infty, \infty).$$

Velká část přírodních jevů lze modelovat pomocí tohoto rozdělení. Pokud bychom např. navážili 100 kaprů a z naměřených hodnot nakreslili graf, tak by nám zcela jistě vyšel podobný graf hustoty. Proto se toto rozdělení hodí pro proměnné, kde většina hodnot je blízko průměru, ale můžou se zde objevit i velice odlehlé hodnoty (angl. outliers). Toto rozdělení by se dalo použít např. pro modelování rychlosti vozidel. Většina by měla rychlost blízkou průměru, ale s menší pravděpodobností se zde mohou objevit i jedinci, kteří jedou velmi pomalu nebo velmi rychle.



Obrázek 1.6 Graf hustoty normálního rozdělení. Převzato z [5]

Kapitola 2

Simulátory dopravy

V této kapitole si představíme základní skupiny dopravních simulátorů. Poté si povíme, jaké jsou jejich typické vstupy. A dále provedeme rešerši na způsoby plánování tras, které použijeme v našem simulátoru. Na konci kapitoly zhodnotíme již existující simulátory dopravy.

Dopravní simulátory jsou užitečným nástrojem pro zkoumání a vylepšování dopravních systémů. Například nám umožňují identifikovat takzvaná úzká hrdla (angl. bottleneck) v silniční síti. Jsou to místa, které nejvíce omezují průchodnost silniční sítě. Pro zrychlení celkové dopravy v síti je nutné se na tyto problémové úseky zaměřit. Dále můžeme pomocí simulátorů testovat algoritmy pro plánování tras vozidel a zkoumat jejich výsledné statistiky ze simulace.

Dopravní simulátory se podle [6] dělí do několika skupin, dle detailnosti simulace:

- *Mikroskopický* uvažují vozidla, jako chytré individuální agenty. Objevují se zde detailní interakce vozidel na silnicích jako například předjíždění pomalých vozidel a sledování následujícího vozidla.

- *Submikroskopický* oproti mikroskopickému přístupu navíc modeluje i jednotlivé součásti vozidla jako např.: převodovku, motor a brzdy. Tento přístup poskytuje ještě detailnější simulaci, než mikroskopický přístup. Je, ale méně vhodný pro simulace velkých silničních sítí.

- *Makroskopický* modeluje pouze základní parametry v silniční síti. Tento přístup je založeny na vztazích mezí tokem, rychlostí a hustotou. Vozidla jsou zde uvažována agregovaně. Např. každé vozidlo nemá rychlost, ale je zde modelována pouze průměrná rychlost na silnici.

- *Mesoskopický* je kombinací makroskopického a mikroskopického přístupu. Většinou modelují vozidla jako jednotlivé agenty nebo je seskupují do skupin. Chování jednotlivých vozidel zde není tak detailní a realistické, jako v mikroskopickém přístupu. Např. nenajdeme zde sledování následujícího vozidla. V mesoskopických simulátorech je nejčastěji prostor modelován diskrétně. Výhoda tohoto přístupu oproti detailnějším spočívá v rychlejší a jednodušší simulaci. Díky tomu je možné simulovat mnohem větší scénáře. Nevýhodou je nižší realističnost simulace.

Velká výhoda makroskopických simulátoru je v rychlosti simulace. Nicméně přesnost mikroskopických a submikroskopických simulátorů nám umožňuje sledovat mnohem detailnější jevy, jako jsou emise nebo interakce jednotlivých vozidel. [6]

My se v naší práci zaměříme na mikroskopické a mesoskopické dopravní simulátory, jelikož chceme porovnávat algoritmy pro plánování tras. A k tomu potřebujeme modelovat jednotlivá vozidla na silnicích, která jsou v makroskopickém přístupu agregována.

2.1 Vstupy

V této podkapitole si popíšeme typické vstupy dopravních simulátorů. Pro úplnost si nejdříve zadefinujeme pojmy z teorie grafů, které budeme v textu používat. Definice jsou čerpány z předmětu BI-AG1 [7].

▶ **Definice 2.1.** *Orientovaný graf G je uspořádaná dvojice (V, E), kde*

- *V je neprázdná konečná množina **vrcholů** a*
- *E je množina **orientovaných hran**.*

*Orientovaná hrana $(u, v) \in E$ je **uspořádaná** dvojice vrcholů $u, v \in V$. Jeho množinu vrcholů a hran značíme $V(G)$ a $E(G)$*

▶ **Definice 2.2.** *Graf H je **podgrafem** grafu G, pokud platí*

- *$V(H) \subseteq V(G)$ a*
- *$E(H) \subseteq E(G)$.*

Tento vztah značíme $H \subseteq G$.

Dopravní simulátor potřebuje typicky tři vstupy, které dohromady tvoří dopravní scénář. Prvním vstupem bývá konfigurace, která je specifická pro daný dopravní simulátor. Druhým vstupem je **dopravní síť** (angl. traffic network), která nám definuje mapu na které budeme provoz simulovat. Obvykle je reprezentována orientovaným graf, kde silnice tvoří hrany a křižovatky vrcholy.

Posledním vstupem jsou **dopravní toky** (angl. traffic flow/demand), které nám definují poptávku v dané dopravní síti. Nezbytným údajem pro toky je počátek a cíl, což jsou nejčastěji různé vrcholy z dopravní sítě. V anglické literatuře se tento pojem často nazývá zkráceně OD (origin-destination) pairs, což v překladu znamená páry počátku a cíle.

Další vlastnosti dopravních toků může být rychlost vozidel z toku. Rychlost je obvykle definovaná normálním rozdělením $N(\mu, \sigma^2)$ ze kterého jsou pak jednotlivé rychlosti vozidel generovány. Dále zde můžeme najít celkový počet vozidel a prodleva mezi přijíždějícími vozidly, která bývá generována z exponenciálního rozdělení $Exp(\lambda)$.

2.2 Plánování tras

Trasy pro dopravní toky lze zadefinovat několika způsoby. První možnost je zadat trasu staticky, kdy je na vstupu zadána celá cesta ze startu do cíle, kterou budou všechna vozidla z toku následovat. Další možnost je nechat výběr trasy na simulátoru, kde je nutné specifikovat jaký algoritmus má simulátor použít k výběru tras. Pro úplnost si zavedeme definice cesty, cesty v grafu, ohodnoceného orientovaného grafu, vzdálenosti a nejkratší cesty dle [8].

▶ **Definice 2.3.** *Nechť $m \geq 0$. Cesta P_m délky m je graf:*

$$(\{0, \ldots, m\}, \{\{i, i+1\} \mid i \in \{0, \ldots, m-1\}\}).$$

Podgraf grafu G izomorfní s nějakou cestou P se nazývá cesta v grafu G.

▶ **Definice 2.4.** *Nechť P je cesta v grafu G a koncové vrcholy jsou u a v, potom takové cestě jedná o cestu z u do v, nebo uv-cestu.*

▶ **Definice 2.5.** *Nechť $G = (V, E)$ je orientovaný graf. Každé hraně $e \in E$ přiřadíme číselnou váhu $w(e)$, kde $w : E \to R$. Takový graf se nazývá ohodnocený orientovaný graf.*

V naší práci budeme předpokládat pouze kladná ohodnocení hran. Tedy, že platí

$$\forall e \in E : w(e) > 0$$

Funkci w se také často říká cenová funkce a říká nám jakou cenu bude mít průchod dané hrany. Tento termín zavádíme, protože pokaždé nechceme nejkratší trasu ve smyslu vzdálenosti, ale můžeme chtít nejlevnější trasu podle jiného kritéria. Například chceme nalézt nejrychlejší cestu. Pokud budeme předpokládat, že silnice nejsou ucpané a, že dokážeme jet maximální povolenou rychlostí tak by cenová funkce byla podíl délky a maximální povolené rychlosti na silnici.

▶ **Definice 2.6.** *Vzdálenost $d(u, v)$ je minimum z délek všech uv-cest, případně $+\infty$, pokud žádná taková cesta neexistuje.*

▶ **Definice 2.7.** *Nejkratší cesta z u do v je libovolná uv-cesta, jejíž délka je rovná vzdálenosti $d(u, v)$.*

Dále si zadefinujeme dva způsoby, kterými budeme v simulátoru plánovat trasy. Prvním způsobem bude plánování tras podle nejkratších cest pomocí Dijkstrova algoritmu. A druhý způsob bude centrální plánování tras založené na více komoditním toku, které zohledňuje zadefinovanou dopravní poptávku.

2.2.1 Hledání nejkratších cest

Jako variantu lokálního řízení dopravy chceme v simulátoru použít Dijkstrův algoritmus, který slouží pro hledání nejkratších cest v ohodnoceném orientovaném grafu. Níže je pseudokód Dijkstrova algoritmu s prioritní frontou.

Algoritmus 1: Dijkstrův algoritmus s prioritní frontou

Vstup: $G = (V, E, w)$ - ohodnocený orientovaný graf, *src* - počáteční vrchol z V, *dest* - cílový vrchol z V
Výstup: Nejkratší cesta z počátečního do cílového vrcholu, pokud existuje.

```
1  for v ∈ V(G) do
2      dist[v] ← ∞
3      prev[v] ← undefined
4  queue ← empty priority queue
5  dist[src] ← 0
6  queue.insert(src)
7  while queue not empty do
8      v ← extract vertex with min dist from queue
9      if v = dest then
10         return constructPath(prev, dest)
11     for (v,u) ∈ E do
           // Relaxace hrany
12         if dist[u] > dist[v] + w(v, u) then
13             dist[u] ← dist[v] + w(v,u)
14             prev[u] ← v
15             queue.insert(u)
16 return not found
```

Níže je pseudokód algoritmu pro vytvoření cesty ze seznamu předchůdců. V *Dijkstrově algo-*

ritmu je použit na 10. řádce.

Algoritmus 2: Algoritmus pro vytvoření cesty ze seznamu předchůdců

Vstup: prev - seznam předchůdců, *dest* - cílový vrchol
Výstup: Seznam reprezentující cestu z počátečního do cílového vrcholu.

```
1 path ← []
2 current ← dest
3 while current ≠ undefined do
4 |   path.prepend(current)
5 |   current ← prev[current]
6 return path
```

Pokud použijeme binární haldu pro reprezentaci prioritní fronty v *Dijkstrově algoritmu*, tak bude časová složitost algoritmu $O(|E| \log |V|)$ a paměťová složitost $O(|V|)$. [8]

2.2.2 Více komoditní tok v síti

V naší práci chceme plánovat trasy vozidel centralizovaně, tak aby průchodnost sítě byla co největší. Jedna z možností, jak to lze dělat, je pomocí takzvaného maximálního souběžného toku (angl. maximum concurent flow), který je instancí více komoditního toku. Nejdříve si zadefinujeme tok v sítí pro jednu komoditu, která má jeden zdroj a jeden stok. Poté zobecníme tuto definici na tok více komodit, kde každá komodita má vlastní zdroj a stok. A na konci si představíme optimalizační úlohu pro problém maximálního souběžného toku. Definice toku s jednou komoditou je převzata z předmětu BI-AG2, konkrétně z přednášky o tocích v síti [9].

▶ **Definice 2.8.** *Sítí nazveme čtveřici (G, s, t, c), kde $G = (V, E)$ je orientovaný graf, s a t dva různé vrcholy grafu G (zdroj a stok). Kapacita $c : E \to \mathbb{R}_0^+$ je funkce ohodnocující hrany nezápornými reálnými čísly.*

Tok v síti je každá funkce $f : E \to \mathbb{R}_0^+$ splňující

- *Zachování kapacity hran*

$$\forall (u, v) \in E : 0 \leq f(u, v) \leq c(u, v)$$

- *Kirchhoffův zákon*

$$\forall u \in V \setminus \{s, t\} : \sum_{(x,u)\in E} f(x, u) = \sum_{(u,y)\in E} f(u, y)$$

Velikost toku je

$$w(f) = \sum_{(s,x)\in E} f(s, x) - \sum_{(x,s)\in E} f(x, s)$$

Nyní si zadefinujeme více komoditní tok, který je zobecněním definice toku s jednou komoditou. Následující definice je čerpána z článku [10].

▶ **Definice 2.9.** *Více komoditní sítí nazveme trojici* (G, K, c)*, kde* G *je* orientovaný graf*,* c *je kapacita hran a* K *je množina* k *komodit, kde* k_i *je* i*-tá komodita a definuje ji trojice* (s_i, t_i, d_i)*, kde* s_i *(zdroj) a* t_i *(stok) jsou dva různé vrcholy grafu* G *a* d_i *je poptávka komodity* k_i*. Tok komodity* k_i *v síti je každá funkce* $f_i : E \to \mathbb{R}_0^+$*.*

Více komoditním tokem nazveme každou k*-tici toků* $\mathbf{F} = (f_1, f_2, \ldots, f_k)$ *splňující*

- *Zachování kapacity hran*

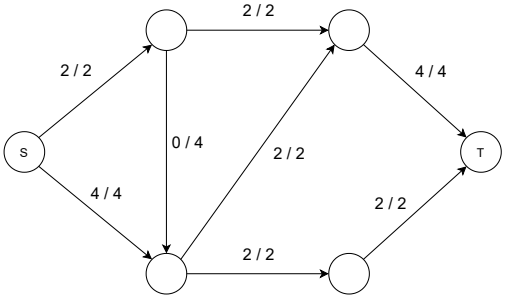
$$\forall (u, v) \in E : \sum_{i=1}^{k} f_i(u, v) \leq c(u, v)$$

- *Kirchhoffův zákon*

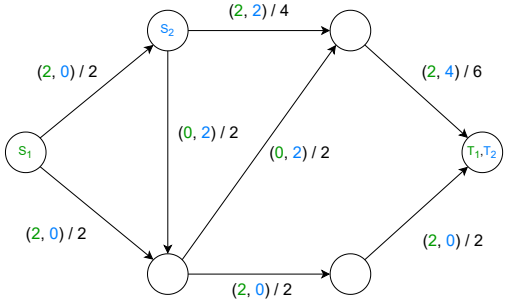
$$\forall k_i \in K, \forall u \in V : \sum_{v \in V} f_i(u, v) - \sum_{v \in V} f_i(v, u) = \begin{cases} 0 & u \neq s_i,\ t_i \\ d_i & u = s_i \\ -d_i & u = t_i \end{cases}$$

Velikost více komoditního toku je

$$w(\mathbf{F}) = \sum_{i=1}^{k} \sum_{(s,x) \in E} f_i(s, x) - \sum_{(x,s) \in E} f_i(x, s) = \sum_{i=1}^{k} d_i$$



Obrázek 2.1 Příklad toku v síti s jednou komoditou. Tok u hrany značí f/c



Obrázek 2.2 Příklad více komoditního toku v síti s dvěma komoditami. Tok u hrany značí $(f_1, f_2)/c$

Pokud chceme najít určitý více komoditního toku, tak se úloha často formuluje jako optimalizační problém. Snažíme se najít takový tok, který maximalizuje nebo minimalizuje danou účelovou funkci a zachovává podmínky více komoditního toku (Kirchhoffovy zákony a zachování kapacity hran). Optimalizační problém se obvykle řeší pomocí lineárního programování (zkráceně LP) za použití nějaké knihovny. Pokud bychom chtěli toky pouze celočíselné tak je tento problém již pro 2 komodity NP-těžký. Když ale připustíme neceločíselné toky, optimální řešení lze najít v polynomiálním čase. [11]

V našem případě chceme najít maximální souběžný tok a nebudeme vyžadovat, aby tok byl celočíselný. Problém maximálního souběžného toku (zkráceně MCFP) se snaží najít maximální násobek poptávek komodit, který daná síť ještě snese. Optimalizační proměnné bude tvořit více komoditní tok $\mathbf{F} = (f_1, f_2, \ldots, f_k)$ a skalár komodit λ. Optimalizační úloha vypadá dle [11]

následovně.

$$\max_{\lambda \geq 0} \quad \lambda \tag{2.1}$$

$$\text{s.t.} \quad \forall (u,v) \in E : \sum_{i=1}^{k} f_i(u,v) \leq c(u,v), \tag{2.2}$$

$$\forall k_i \in K, \forall u \in V : \sum_{v \in V} f_i(u,v) - \sum_{v \in V} f_i(v,u) = \begin{cases} 0 & u \neq s_i,\ t_i \\ \lambda\, d_i & u = s_i \\ -\lambda\, d_i & u = t_i \end{cases}. \tag{2.3}$$

2.3 Existující řešení

V této části si představíme několik již existujících řešení pro simulaci dopravy. Byli vybrány volně dostupné open source řešení. Při výběru byl kladen důraz na kvalitu daných řešení. Každé vybrané řešení si stručně uvedeme a popíšeme si jaké má vstupy. Poté si řekneme jaké obsahuje možnosti pro plánování tras vozidel a jaké jsou jeho výstupní statistiky a na závěr si upřesníme, jak je řešení navrženo z pohledu počítačové simulace.

2.3.1 SUMO (Simulation of Urban MObility)

Prvním zvoleným projektem je populární open source simulátor SUMO. Je dostupný od roku 2001 pod licencí Eclipse Public License V2 a je stále rozvíjen komunitou. Řadí se mezi mikroskopické dopravní simulátory. Umožňuje modelování dopravních situací včetně: osobních automobilů, hromadné dopravy, pohybu civilistů a světelných křižovatek. Nad tímto simulátorem je vybudována velká škála nástrojů, řadí se mezi ně např.: editor dopravních sítí a toků (netedit), prohlížeč simulace s grafickým rozhraním (sumo-gui) nebo také nástroj pro ovládání simulace během běhu (TraCI). [6]

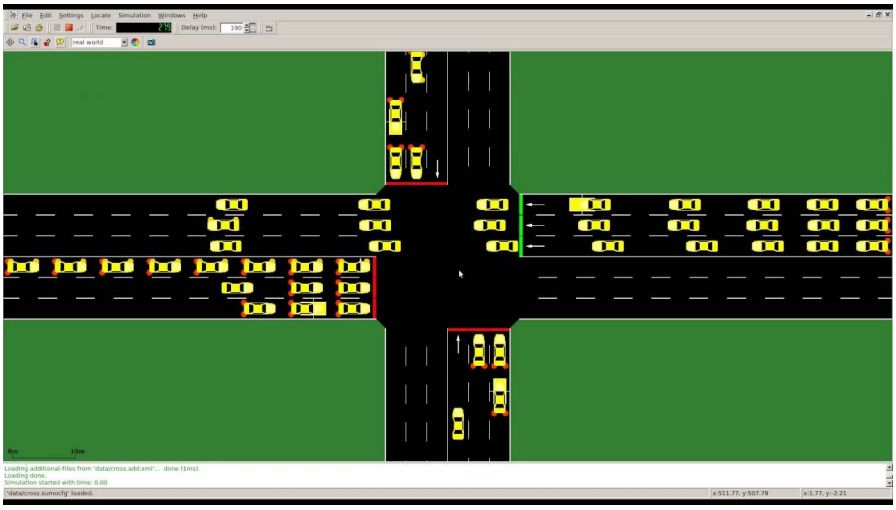
Dopravní síť je v simulátoru SUMO implementovaná jako orientovaný graf, kde silnice jsou hrany a křižovatky jsou vrcholy. Silnice jsou dále tvořeny jednotlivými pruhy, které mají tyto atributy: šířka, nejvyšší povolená rychlost a omezení na typ vozidel (např. pruh pouze pro autobusy). Pokud mají pruhy na jedné silnici odlišné vlastnosti, tak je v dopravní síti spoj namísto jedné hrany pro silnici modelován více hranami pro jednotlivé pruhy.

Plánování tras je v simulátoru SUMO velmi omezené, lze plánovat trasy pouze podle nejkratších cest. Pro jednotlivá vozidla nebo toky vozidel lze vybrat trasu buď staticky a nebo použít Dijkstrův algoritmus nebo A* s různými heuristickými a cenovými funkcemi pro nalezení nejkratší cesty. Je možné zde nastavit, aby se nejkratší cesty hledali po určitém intervalu a tím se vozidla vyhýbala zaplněným silnicím, ale tento způsob stále není optimální v tom smyslu, že nové nalezené cesty jsou nejkratší pouze v daný okamžik a nepočítá s budoucí dopravní poptávkou.

Simulátor SUMO má poměrně mnoho statistických výstupů. Najdeme zde jak se během simulace vyvíjeli následující položky.

- Pozice a rychlosti jednotlivých vozidel.
- Agregovaná dopravní data pro jednotlivé silnice, pruhy, toky vozidel a celou simulaci. V dopravních datech najdeme např: informace o rychlosti a času cesty (max, min, průměr), počet vozidel, velikost fronty, tok vozidel za hodinu. Ale také jsou zde i data o vyloučených emisích (CO, CO_2, hluk, spotřebované palivo).

SUMO je implementovaný jako diskrétní simulátor využívající techniku *Time slicing*, jelikož modeluje prostor spojitě. Velikost časového kroku, lze v simulátoru nastavit.



Obrázek 2.3 Obrázek ze simulátoru SUMO. Obrázek přezván z [12]

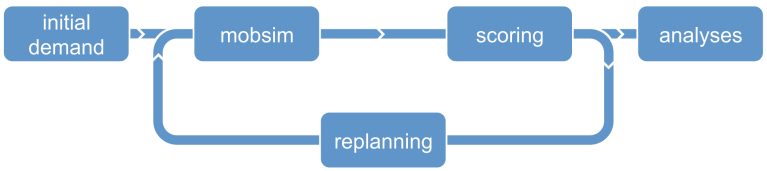
2.3.2 MATSim

Posledním vybraným projektem je MATSim. Jedná se o open source framework pro mesoskopickou simulaci dopravy implementovaný v jazyce Java. Je zaměřený na simulaci dopravy na velkých oblastech s velkým počtem agentů. [13]

Simulátor MATSim vyžaduje před spuštěním simulace 3 soubory ve formátu XML (Extensible Markup Language). Prvním souborem je `config.xml`, který obsahuje konfiguraci pro simulaci. Druhým souborem je `network.xml`, která obsahuje definici dopravní sítě a posledním vstupem je `population.xml`, který obsahuje dopravní poptávku.

Velkou odlišností tohoto simulátoru od ostatních je způsob, kterým plánuje trasy. Snaží se najít takové trasy, aby zachovali rovnováhu pro všechny agenty (angl. user equilibrium). Využívá k tomu evoluční algoritmus, který slouží k hledání řešení, které optimalizuje kriteriální funkci, průměr skóre všech agentů. Pro evoluční algoritmy není konvergence ke globálnímu optimu nijak zaručená, ale přesto dávají pro velkou škálu problémů uspokojivé výsledky.

Evoluční algoritmus v MATSim funguje následovně. Na vstup dostane dopravní scénář a pro vozidla zvolí nejkratší trasy a spustí s nimi první běh simulace. Poté daný běh simulace ohodnotí kriteriální funkcí. Následně upraví trasy, které použil v předchozím běhu a takto se celá smyčka opakuje dokud nedosáhne stanoveného počtu iterací. Na konci vrátí algoritmus trasy pro, které byla hodnota kriteriální funkce nejvyšší. Tato smyčka je znázorněna na obrázku 2.4.

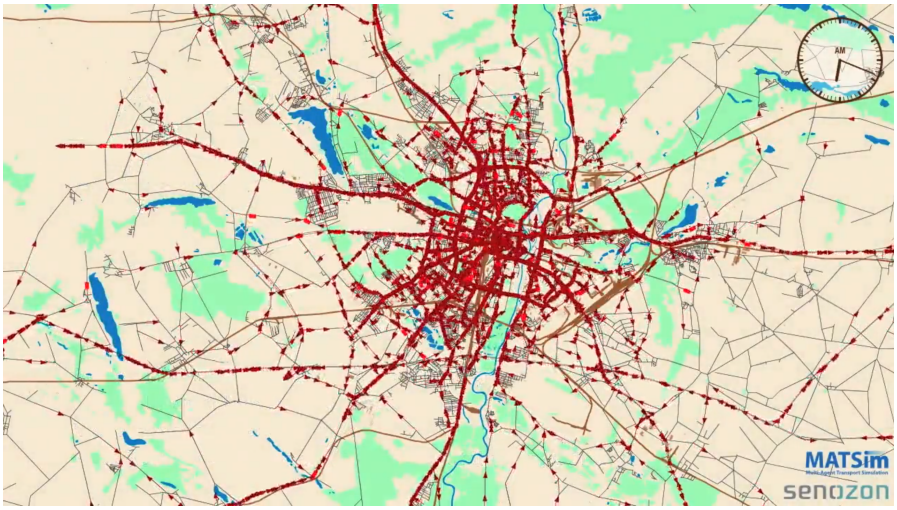


Obrázek 2.4 Znázornění evoluční metody v simulátoru MATSim. Převzato z [13]

Ve statistických výstupech nalezneme tyto údaje.

- Skóre výše zmíněné kriteriální funkce pro každou iteraci evol. algoritmu.
- Ujetá vzdálenost, která je agregovaná pro jednotlivé toky nebou celou simulaci.

MATSim k simulaci využívá knihovnu QSim. Ta umožňuje do jisté míry paralelní aktualizování agentů, ale je založena na diskrétních časových krocích (Time slicing), tedy musí aktualizovat všechny agenty po každém časovém kroku. Prostor je v tomto simulátoru modelován diskrétně.



Obrázek 2.5 Obrázek ze simulace dopravy ve městě Poznaň v Polsku pomocí MATSim

2.4 Shrnutí

V této kapitole jsme si řekli jaké jsou typické vstupy pro dopravní simulátor. Nejčastěji to je konfigurace pro simulátor, dopravní síť a dopravní toky/poptávka. A zavedli jsme si definice orientovaného grafu a podgrafu.

Dále jsme si představili způsoby, kterými chceme plánovat trasy v našem simulátoru a během experimentů tyto způsoby porovnat. Prvním způsobem je plánování tras podle nejkratších cest, které nezohledňuje budoucí poptávku a vždy najde trasu, která je nejlepší pouze v danou chvíli. K tomu použijeme Dijkstrův algoritmus 1. Druhý způsob, který využijeme na plánovaní tras je maximální více komoditního tok. Který bude plánovat trasy centralizovaně a bude využívat znalost budoucí dopravní poptávky. Nevýhoda tohoto způsobu, ale spočívá v tom, že nijak nezohledňuje čas simulace a nalezené trasy jsou nejlepší co se týče průchodnosti sítě až ve chvíli, kdy je plně zahlcena.

Na závěr jsme si popsali některé existující řešení v této oblasti. Konkrétně šlo o dopravní simulátory SUMO a MATSim. Ani jedno řešení nenabízí možnost řízení dopravy podle tokových algoritmů. Obě řešení spadají do skupiny mikroskopických simulátorů a z pohledu počítačové simulace jsou diskrétní, stochastické a dynamické. Čas posouvají pomocí diskrétních časových kroků (Time slicing) a prostor ve kterém se vozidla pohybují modelují spojitě nebo diskrétně.

Kapitola 3
Návrh simulátoru

V této kapitole navrhneme dopravní simulátor. Nejdříve si specifikujeme vstupy simulátoru. Poté navrhneme simulátor z pohledu diskrétní simulace. V další části si upřesníme, jak budeme modelovat dopravní síť a prostor na silnicích. Pote si zadefinujeme dopravní toky a rozebereme, jak budeme řídit vozidla a pohyb vozidel na křižovatkách. Na závěr navrhneme dva algoritmy, kterými budeme plánovat trasy v simulátoru.

Dopravní simulátor navrhneme, jako mesoskopický. Z pohledu počítačové simulace bude diskrétní, dynamický a stochastický. Vozidla budeme uvažovat, jako jednoduché individuální agenty, které sledují přidělenou trasu. Prostor ve kterém se budou vozidla pohybovat budeme modelovat diskrétně. Na silnicích chceme modelovat pouze základní jevy, jako např.: předjížděni pomalejších vozidel rychlejšími a dávaní přednosti na křižovatkách. Ale nechceme modelovat detailní interakce vozidel, jako např. sledování následujícího vozidla. , které jsou typické pro mikroskopické simulátory. Výhoda tohoto přístupu spočívá v možnosti simulovat větší množství vozidel. Nevýhodou je menší realističnost celé simulace. Pro účely porovnání algoritmů pro plánování tras, nám tento přístup bude stačit.

Vstupem do našeho dopravního simulátoru bude soubor s dopravním scénářem a výběrem plánovacího algoritmu. Dopravní scénář je tvořen dopravní sítí a dopravními toky. V následujících částech návrhu si povíme, jak budou modelovány.

3.1 Diskrétní simulace

Většina dopravních simulátorů, dělí čas na předem dané intervaly a tím ho diskretizuje. V našem simulátoru se odlišíme tím, že nebudeme čas posouvat po fixním časovém kroku, ale budeme ho modelovat jako spojitou veličinu. K tomu použijeme metodu *Next event* z kapitoly 1.1.1.

Náš model diskrétní simulace bude obsahovat tři druhy procesů. Prvním procesem je dopravní tok, který bude mít na starosti vkládáni nových vozidel do simulace. Druhým procesem je vozidlo, které bude sledovat přidělenou trasu do té doby, než se dostane do cíle. A posledním je proces, který bude řídit provoz na křižovatkách. V dalších částech návrhu si tyto procesy rozebereme více.

3.1.1 Stav diskrétní simulace

Stav diskrétní simulace musí zachycovat aktuální stav celého systému, který je simulován. Stav dynamické diskrétní simulace musí uchovávat aktuální čas simulace a frontu naplánovaných událostí. V našem případě navíc ještě uchovávat seznam aktivních procesů v simulaci a model dopravního scénáře. Ve stavu tedy budou následující proměnné:

- **Čas simulace** – nezáporná reálná proměnná uchovávající aktuální čas simulace.
- **Fronta událostí** – frontu událostí implementujeme jako prioritní frontu událostí, kde na vrcholu je událost, která nastane nejdříve. Tedy má nejnižší čas na, který je naplánována. V případě, že je více událostí naplánováno na stejný čas, tak potom jsou vyndávaný z fronty v pořadí v libovolném pořadí.
- **Aktivní procesy** – seznam obsahující všechny běžící procesy v simulaci.
- **Dopravní scénář** – obsahuje dopravní síť a dopravní toky. Dopravní síť uchovává informace o rozmístění vozidel na silnicích.

3.1.2 Simulace v reálném čase

Protože chceme simulaci vizualizovat v reálném čase, tak je potřeba počítat s různou velikostí časového kroku mezi nastávajícími událostmi. Simulaci chceme překreslit pokaždé, když nastane nějaká událost a tím se změní stav simulace. Níže je popsán algoritmus pro hlavní smyčku metody *Next event*, který uspí simulaci podle doby, která uplynula od poslední události. To nám zaručí, že čas simulace nebude neproporcionálně skákat mezi událostmi a bude plynout stejnou rychlostí. V následujícím odstavci si algoritmus popíšeme detailněji.

Algoritmus 3: Algoritmus hlavní smyčky simulace v reálném čase

Vstup: state = (time, queue, …) - počáteční stav simulace s časem a frontou událostí, timeFactor - časový faktor udává o kolik se má simulace zpomalit

```
1 while queue not empty) do
2     lastTime ← time
3     event ← queue.pop()
4     time ← event.time
5     event.execute()
6     draw(state, event)
7     stats(state, event)
8     timeToSleep ← (time − lastTime) ∗ timeFactor
9     sleep(timeToSleep)
```

Algoritmus dostane na vstupu počáteční stav simulace, který obsahuje čas simulace, frontu událostí, aktivní procesy a dopravní scénář (dopravní síť a toky). Dále dostane faktor, který nám udává o kolik se má simulace zpomalit. Hlavní částí algoritmu je smyčka na 1. řádku, která se opakuje dokud fronta událostí není prázdná. Uvnitř této smyčky si uložíme předchozí čas a vyndáme událost z vrcholu fronty. Poté nastavíme čas simulace na čas události a vykonáme událost. Po vykonání události se nám změní stav simulace a je potřeba ho znovu překreslit a přepočítat statistiky, to nám zařídí metody `draw` a `stats` na 6. a 7, řádku, které dostanou jako argument stav simulace a poslední událost. A na konci vypočteme dobu na, kterou máme simulaci uspat jako rozdíl mezi aktuálním a předchozím časem simulace. Tento rozdíl vynásobíme faktorem ze vstupu a dostaneme výsledný čas na uspání.

3.2 Model dopravní sítě

Dopravní síť na úrovni spojů (silnice a křižovatky) budeme reprezentovat podobně jako ostatní simulátory orientovaným grafem $G = (V, E)$. Kde vrcholy V budou tvořit křižovatky a hrany E budou tvořit silnice. Každá křižovatka bude obsahovat souřadnice, které udávají její umístění na mapě. Silnice budou mít tyto vlastnosti: délka, počet pruhů a prioritu. Prioritu silnic využijeme při pohybu vozidel na přednostních křižovatkách, o kterých si povíme v podkapitole 3.3.2. Pokud bychom chtěli modelovat obousměrnou silnici stačí zadat mezi křižovatky u a v dvě orientované hrany (u, v) a (v, u).

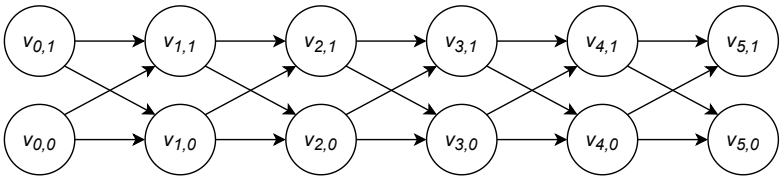
3.2.1 Diskrétní prostor na silnicích

V našem simulátoru chceme modelovat prostor na silnicích diskrétně podle vlastností dané silnice. První vlastností, která bude využívána při modelování prostoru je délka silnice, která bude vypočtena jako Euklidovská vzdálenost mezi jejími koncovými vrcholy a zaokrouhlena na celá čísla nahoru. Další vlastností bude počet pruhů, který bude zadán při definici silnice.

Diskrétní prostor na silnici ve kterém se budou vozidla pohybovat budeme reprezentovat orientovaným grafem $G = (V, E)$. Vrcholy V budou tvořeny pozicemi pro vozidla a hrany budou možné pohyby vozidel na další pozice na silnici. Hrany v tomto grafu povedou vždy na dopředné pozice na silnici. A to buď na následující pozici ve stejném nebo v sousedním pruhu. Dále si tento graf popíšeme formálněji.

Nechť $G = (V, E)$ je graf silnice s délkou n a m pruhy. Počet vrcholů v grafu bude roven $|V| = nm$. Dále si formálněji popíšeme, jak budou tvořeny hrany. Označení $v_{i,j}$ značí vrchol z V na i-té pozici v j-tém pruhu, kde $0 \leq i < n \wedge 0 \leq j < m$. Z vrcholu $v_{i,j}$ povede hrana na následující vrcholy:

- $v_{i+1,j}$, pokud $i < n$
- $v_{i+1,j+1}$, pokud $i < n \wedge j < m$
- $v_{i+1,j-1}$, pokud $i < n \wedge j > 0$



Obrázek 3.1 Příklad orientovaného grafu silnice délky 6 s 2 pruhy

3.3 Dopravní toky

Dopravní poptávku v silniční síti budeme definovat pomocí dopravních toků. Definují nám odkud a kam se potřebují vozidla v síti dostat. Každý dopravní tok bude mít následující vlastnosti:

- **Zdroj** – počáteční vrchol z dopravní sítě.
- **Stok** – cílový vrchol z dopravní sítě, musí být rozdílný od zdroje.
- **Počet vozidel** – nezáporné celé číslo, udávající celkový počet vozidel v toku.
- **Rychlost vozidel** – náhodné rozdělení ze kterého bude generována rychlost jednotlivých vozidel.

- **Interval mezi příjezdy** – náhodné rozdělení ze kterého bude generován interval mezi příjezdy vozidel.
- **Trasy** – seznam tras ze zdroje do stoku. Tyto trasy budou přiděleny algoritmem pro plánování tras. Více si o nich povíme v kapitole 3.4.

Pro generování rychlostí vozidel se nám nejvíce hodí normální rozdělení $N(\mu, \sigma^2)$, které je vhodné použít pro modelování přírodních jevů. Pokud budeme brát rychlosti z tohoto rozdělení, tak bude mít většina vozidel rychlost blízkou zadefinované střední hodnotě μ. A vozidel se vzdálenějšími rychlosti od střední hodnoty bude rychle ubývat. Tedy vozidla, které jsou extrémně rychlá nebo pomalá bude velmi málo. Proto si myslíme, že toto rozdělení odpovídá realitě nejvíce.

Pro mezi příjezdové intervaly použijeme exponenciální rozdělení $Exp(\lambda)$, které se nejčastěji používá ke generování intervalů mezi událostmi, které mají Poissonovo rozdělení a tedy se vyskytují nezávisle na sobě. Což je přesně případ intervalů mezi příjezdy vozidel. Pokud zvolíme např. $\lambda = 1/5$, tak přijede v průměru 1 vozidlo za 5 jednotek simulačního času.

Podle těchto zadefinovaných vlastností bude proces dopravního toku vkládat vozidla do simulace. Nyní si dále popíšeme jak bude vkládání probíhat. Proces dopravního toku nejprve přiřadí nějakou trasu z jeho seznamu tras. Poté mu vygeneruje rychlost z jeho rychlostního rozdělení. A vloží toto vozidla na začátek silnice, kterou trasa začíná. Dále si proces vygeneruje číslo z rozdělení pro příjezdy, které udává za jak dlouho přijede následující vozidlo. a Na tento čas si naplánuje událost pro přidání vozidla a celý tento proces se opakuje dokud nepřidá do simulace všechna vozidla z toku.

3.3.1 Řízení vozidel

Vozidla jsou do simulace přidávána procesem dopravního toku a je odebráno ze simulace, když dojede do svého cíle. Každé vozidlo v simulaci bude obsahovat následující vlastnosti:

- **Aktuální silnice** – umístění v dopravní síti.
- **Pozice na silnici** – umístění na aktuální silnici.
- **Rychlost** – nezáporné a nenulové reálné číslo, které udává kolik pozic na silnici vozidlo ujede za jednotku simulačního času. Pokud tedy bude rychlost jedna, vozidlo se po jednotce simulačního času posune o právě jednu pozici.
- **Trasa** – přidělena trasa, kterou vozidlo následuje do cíle.

Předpokládáme, že rychlost vozidla bude statická a během simulace se již nebude měnit. Zároveň ještě předpokládáme, že vozidla ze své rychlosti dokážou ihned zastavit anebo se okamžitě rozjet s touto rychlostí. Důsledkem tohoto předpokladu je, že vozidla budou mít v našem simulátoru nekonečné zrychlení. Tyto předpoklady neodpovídají realitě, ale pro účely našeho simulátoru to nebude ubírat na jeho použitelnosti.

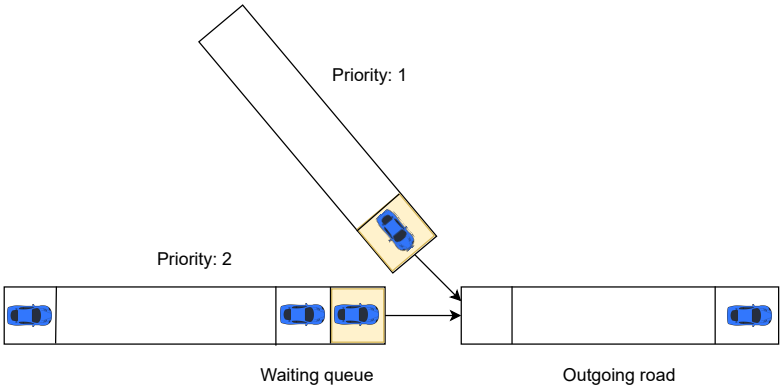
Proces vozidla bude do simulace plánovat tři typy událostí:

- **Pohyb vozidla** – je událost, která po provedení posune vozidlo na další volnou pozici na silnici nebo na začátek další silnice z trasy, v případě průjezdu křižovatkou. Událost o pohybu je naplánována na čas, který je určen z rychlosti vozidla. Čas je vypočten jako $\frac{1}{rychlost}$.
- **Zastavení vozidla** – proces vozidla naplánuje tuto událost ve chvíli, kdy se nemůže pohnout na další pozici, jelikož jsou obsazená nebo se nachází na konci křižovatky přes, kterou nemůže zatím projet. Až se další pozice uvolní nebo je vozidlu povolen průjezd přes křižovatku, tak vozidlo naplánuje událost o pohybu.
- **Dokončení cesty** – vozidlo naplánuje tuto událost ve chvíli, kdy dorazí do cíle. Po vykonání události je vozidlo odebráno ze simulace.

3.3.2 Pohyb na křižovatkách

Křižovatky slouží v dopravní síti jako spojení mezi silnicemi a umožňují vozidlům přejet z jedné silnice na druhou. Každá křižovatka má příchozí a odchozí silnice. V našem simulátoru předpokládáme, že se lze z každé příchozí silnice dostat na jakoukoliv odchozí silnici na křižovatce.

Populární dopravní simulátory obsahují velké množství typů křižovatek. Patří mezi ně např.: světelné křižovatky, které jsou ale poměrně náročné na implementaci a vyžadují složitější konfiguraci při modelování dopravní sítě. Proto v simulátoru použijeme pouze prioritní křižovatky, které jsou jednoduché na implementaci a při modelování dopravních scénářů vyžadují minimální konfiguraci. Požadují jen, aby silnice měly nastavenou číselnou prioritu. Prioritní křižovatky fungují tak, že nejdříve po nich přejedou vozidla, která jsou na silnici s nejvyšší prioritou. Tento typ křižovatky si lze představit, jako křižovatku na hlavní a vedlejší silnici. Kde mají přednost vozidla na hlavní silnici. Pokud bychom chtěli modelovat tento případ, tak akorát nastavíme na hlavní silnici větší číselnou prioritu než na vedlejší silnici.



Obrázek 3.2 Příklad prioritní křižovatky se dvěma příchozími silnicemi a jednou odchozí. Žlutou barvou jsou zvýrazněna vozidla čekající na přejezd křižovatky. Po této křižovatce nejdříve přejede vozidlo ze spodní silnice

Pohyb na každé křižovatce bude řízen *procesem*, který má na starosti výběr vozidla pro přejezd dané křižovatky. Až vybrané vozidlo přejede na další silnici, tak proces vybírá další vozidlo. Dále si popíšeme detailněji, jak výběr probíhá.

K výběru vozidla je využíván seznam čekajících vozidel, který je tvořen vozidly na konci silnic u dané křižovatky. Z tohoto seznamu vybereme vozidlo, které je na silnici s nejvyšší prioritou a zároveň může přejet na jeho další silnici, kterou má v trase. Vozidlo může přejet na další silnici pouze pokud začátek dané silnice je volný. Pokud existuje vozidel se stejnou prioritou více, vybereme jedno z nich náhodně.

3.4 Plánování tras

V simulátoru budeme chtít porovnat dva odlišné způsoby řízení dopravy – lokální a centrální. Lokální řízení dopravy nebere v úvahu budoucí poptávku a plánuje trasy pouze ze současného stavu v silniční síti. Pro lokální řízení dopravy použijeme Dijkstrův algoritmus. Centrální řízení dopravy počítá s budoucí poptávkou na silnicích a plánuje trasy tak, aby byli optimální v rámci celého dopravního systému. Jako variantu centrálního řízení dopravy použijeme algoritmus založený na více komoditním toku v síti.

Plánování tras proběhne před začátkem simulace, kde se každému dopravnímu toku naplánují trasy, které budou vozidla z toku následovat do cíle. Každá trasa bude zároveň obsahovat pravděpodobnost výběru dané trasy, s tím, že součet pravděpodobností všech tras pro dopravní

tok musí být jedna. Výsledné přiřazení tras jednotlivým vozidlům probíhá pomocí váženého náhodného výběru z tras.

Algoritmus 4: Algoritmus pro vážený náhodný výběr trasy

Vstup: paths - seznam cest spolu s pravděpodobností výběru dané cesty (součet všech pst. musí být 1)
Výstup: Náhodně vybraná trasa ze seznamu paths

```
1 rand ← random(0,1) // Náhodný výběr z uniformního rozd. U(0,1)
2 i ← 0
3 sum ← 0
4 for (path,chance) ∈ paths do
5     sum ← sum + chance
6     if rand ≤ sum then
7         return path
```

Algoritmus pro výběr trasy dostane na vstupu seznam tras pro daný tok, spolu s jejich pravděpodobnostmi výběru. Na 1. řádce si vygeneruje číslo z uniformního rozdělení $U(0, 1)$. A poté prochází všechny trasy, do té doby, dokud součet pravděpodobností nebude větší nebo roven vygenerovanému číslu. Až se tato podmínka splní, algoritmus vrátí trasu z aktuální iterace.

3.4.1 Lokální řízení dopravy

Lokální řízení dopravy nebere při plánování tras v potaz budoucí poptávku na silnicích. A trasy plánuje vždy pouze ze současného stavu v silniční síti. Stav silniční sítě se, ale během simulace neustále mění. Pokud tedy pro vozidla budeme hledat takové trasy, kterými se do cíle v současném stavu dostanou nejrychleji, může nastat situace, kdy tato vozidla velmi zatíží určité silnice. A tedy tyto naplánované trasy přestanou být nejrychlejšími a velmi zatíží silniční síť.

Jako způsob lokálního řízení dopravy použijeme Dijkstrův algoritmus, který plánuje trasy podle nejkratších cest. Dijkstrův algoritmus jsme si popsali v podkapitole 2.2.1. Naším vstupem do Dijkstrova algoritmu bude dopravní síť, kterou reprezentujeme orientovaným grafem G. A jako cenovou funkci w zvolíme délku silnic, což je v našem případě Euklidovská vzdálenost mezi její počáteční a koncovou křižovatkou. Pomocí Dijkstrova algoritmu najdeme před začátkem simulace nejkratší cestu pro všechny dopravní toky z jejich zdroje do stoku. Této nejkratší cestě nastavíme pravděpodobnost výběru 1, tedy se přiřadí každému vozidlu z toku.

3.4.2 Centralizované řízení dopravy

Centralizované řízení dopravy při plánování tras využívá znalost budoucí poptávky. A plánuje trasy pro vozidla tak, aby byli optimální v rámci celé silniční sítě. V našem případě chceme optimalizovat průchodnost silniční sítě. Tedy, aby průtok vozidel sítí byl co největší. K tomu použijeme algoritmus založený na maximálním souběžném toku (zkráceně MCF) z kapitoly 2.2.2.

Komodity ve více komoditním toku budou tvořeny dopravními toky, které obsahují zdroj a stok. Prvním krokem k získání těchto tras je zformulovat optimalizační problém MCF pomocí lineárního programování a použít vhodný řešič lineární optimalizace pro nalezení optimálního řešení. Poté si určíme, jakou budeme používat kapacitní funkci c pro silnice. A povíme si, jak budeme komoditám přidělovat jejich poptávku d. A na závěr si vysvětlíme způsob, kterým získáme

trasy pro jednotlivé dopravní toky z nalezeného MCF.

Algoritmus 5: Algoritmus pro nalezení maximálního souběžného toku pomocí LP

Vstup: $S = (G, K, d, c)$ – G je graf dopravní sítě, K jsou komodity tvořeny dopravními toky, d je poptávka komodit a c je kapacita hran.
Výstup: Více komoditní tok F a skalár komodit λ

```
1  f[(u,v),k] ← [] // Seznam pro proměnné toků komodit
2  for k ∈ K do
3      for (u,v) ∈ E do
4          f[(u,v),k] ← realVariable(interval = ⟨0, c((u, v))⟩)
5  λ ← realVariable() // Proměnná pro skalár komodit
   // Zachování kapacity hrany
6  for (u,v) ∈ E do
7      flowSum ← Σ_{k∈K} f[(u,v),k]
8      constraint(flowSum ≤ c(u, v))
   // Kirchhoffův zákon
9  for u ∈ V, k ∈ K do
10     incomingFlows ← Σ_{v∈V} f[(v,u),k]
11     outgoingFlows ← Σ_{v∈V} f[(u,v),k]
12     if u = k.source then
13         constraint(outgoingFlows − incomingFlows = λ d(k))
14     else if u = k.sink then
15         constraint(incomingFlows − outgoingFlows = λ d(k))
16     else
17         constraint(outgoingFlows − incomingFlows = 0)
18 maximize(λ)
19 if solution was found then
20     return (f, λ)
21 else
22     return undefined
```

Algoritmus dostane na vstupu graf dopravní sítě G, dopravní toky (komodity) K, poptávku komodit d a kapacity hran c. Na 1. řádku algoritmu je vytvořen seznam, který bude obsahovat proměnné toků komodit. Proměnné jsou v seznamu indexovány pomocí hrany a komodity. Poté je pro každou hranu (u, v) a komoditu k do tohoto seznamu přidána optimalizační proměnná, která nabývá hodnot od 0 do kapacity hrany $c(u, v)$. Poté vytvoříme proměnnou λ, která bude skalárem poptávek komodit. Tuto proměnnou budeme chtít maximalizovat se zachováním dvou následujících podmínek. První podmínkou je zachování kapacity hrany, to zajistíme na řádcích 6-8, kde pro všechny hrany (u, v) vytváříme omezení, že součet toků na hraně nesmí převýšit její kapacitu c. Druhou podmínkou, kterou musíme zachovat je Kirchhoffův zákon. O to se stará smyčka na řádcích 9-17, která pro všechny vrcholy u a komodity k, vytváří podmínky na odchozí a příchozí toky podle typu vrcholu. Na řádce 18 je zahájena optimalizace skaláru komodit λ s ohledem na vytvořené omezení. Pokud řešení existuje tak algoritmus vrátí nalezené hodnoty více komoditního toku a skaláru komodit.

Nyní si povíme, jak budeme přiřazovat kapacitu silnicím. Kapacita silnic musí reflektovat, jaký průtok ma dané silnice neboli, jak rychle budou vozidla schopny tuto silnici opouštět. Jelikož vozidla opouští silnice přes prioritní křižovatky, které nejdříve překročí vozidla na silnicích s vyšší prioritou. Tak dává smysl přiřazovat kapacitu silnicím na základě toho jakou mají prioritu v cílové

křižovatce. Tuto kapacitu lze vystihnout jako podíl priority silnice ku prioritám všem příchozím silnicím na křižovatce.

Poptávka dopravních toků musí reflektovat, jak rychle přicházejí vozidla z daného toku. To nám v našem případě definuje mezi příjezdový interval, který je za definován náhodným rozdělením. Jako poptávku dopravního toku můžeme použít převrácenou střední hodnotu rozdělení, pokud existuje.

Po získání maximálního souběžného toku F pomocí lineárního programování ještě potřebujeme nalézt trasy z tohoto toku pro všechny komodity. Pro jednu komoditu k můžeme trasy z jejího toku F_k můžeme najít opakovaným spouštěním Dijkstrova algoritmu na grafu G dopravní sítě s hranami, kde tok komodity $F_k > 0$. Jako ohodnocení grafu w pro Dijkstrův algoritmus použijeme délku silnice, abychom našli nejkratší cesty tvořící tento tok. Po nalezení cesty pomocí Dijkstrova algoritmu najdeme nejnižší hodnotu toku F_k na této cestě. Poté tuto hodnotu odečteme od F_k na hranách, které leží na nalezené cestě.

Algoritmus 6: Algoritmus pro nalezení tras tvořících tok komodity k

Vstup: $S = (G, F_k, k)$ – G dopravní síť, k dopravní tok (komodita), F_k více komoditní tok komodity k

Výstup: Seznam nejkratších tras tvořící tok komodity k spolu s pravděpodobností výběru dané trasy

```
   // Seznam pro cesty s hodnotou nejnižšího toku
 1 paths ← []
 2 flowSum ← 0
 3 while path exists do
 4 |   flowGraph ← G with edges (u, v) where F_k(u, v) > 0
 5 |   path ← dijkstra(flowGraph, k.source, k.sink)
 6 |   minFlow ← find min value of F_k on edges (u, v) from path
 7 |   flowSum ← flowSum + minFlow
 8 |   F_k ← subtract minFlow from F_k, along the edges of path
 9 |   paths.add(shortestPath, minFlow)
   // Seznam pro výsledné trasy s pravděpodobností výběru
10 result ← []
11 for (path, flow) ∈ result do
12 |   chance ← flow/flowSum
13 |   result.add(path, chance)
14 return result
```

Vstupem do algoritmu je graf G dopravní sítě, k dopravní tok a F_k tok komodity k v G. Výstupem je seznam cest, které tvoří tok zadané komodity k. Na řádcích 1-2 si vytvoříme seznam pro nalezené trasy a inicializujeme sumu toků na 0. Cyklus na 3. řádku opakujeme, dokud existuje cesta ze zdroje do stoku. Na 4. řádku si vytvoříme graf z dopravní sítě G, kde budou všechny vrcholy a pouze takové hrany (u, v) pro které je tok komodity $F_k > 0$. Poté v tomto grafu nalezneme nejkratší cestu ze zdroje do stoku. Pokud nebyla cesta nalezena tak opustíme cyklus. Na řádku 6 nalezneme hodnotu nejmenšího toku F_k, který se vyskytuje na nejkratší trase a tuto hodnotu přičteme k sumě toků. Na řádku 8 odečteme hodnotu nejmenšího toku pro všech hrany v nejkratší cestě od F_k. A přidáme cestu do seznamu *paths* spolu s jejím tokem. Až nalezneme všechny cesty tak vypočteme pravděpodobnost jejich výběru jako podíl toku na cestě ku celkovému toku. A tyto trasy vrátíme.

3.5 Shrnutí

Vstup do našeho simulátoru bude tvořit dopravní scénář, který je tvořen silniční sítí, dopravními toky a výběrem algoritmu pro plánování tras. Simulátor budeme modelovat jako diskrétní simulaci spadající do skupiny mikroskopických simulátorů dopravy. Pro posun času simulace použijeme metodu Next event. Náš model diskrétní simulace bude obsahovat tři druhý procesů. Prvním je proces vozidla, který bude řídit jeho pohyb. Dále proces křižovatky, který bude řídit provoz na křižovatkách. A poslední je proces dopravního toku, který bude vkládat vozidla do simulace. A na závěr jsme si představili algoritmus pro simulaci v reálném čase.

Silniční síť budeme reprezentovat orientovaným grafem, kde silnice tvoří hrany a křižovatky vrcholy grafu. Prostor na silnicích budeme také modelovat konečným orientovaným grafem, který je vytvořen podle délky a počtu pruhů na silnici. Vrcholy v silničním grafu reprezentují pozice pro vozidla a hrany tvoří možné přesuny na další pozice na silnici. Křižovatky navrhujeme, jako prioritní křižovatky, po kterých nejdříve přejíždí vozidla na silnicích s vyšší číselnou prioritou.

Poptávku v silniční síti budeme reprezentovat dopravními toky. Každý tok bude obsahovat následující vlastností: počátku a cíle, počtu vozidel, rychlostí vozidel, která bude generována z normálního rozdělení. A mezi příjezdové intervaly, které budou generovány ze zadaného exponenciálního rozdělení. Proces dopravního toku bude vkládat jednotlivá vozidla do simulace. U vozidel předpokládáme, že mají statickou rychlost a dokáží ihned zastavit.

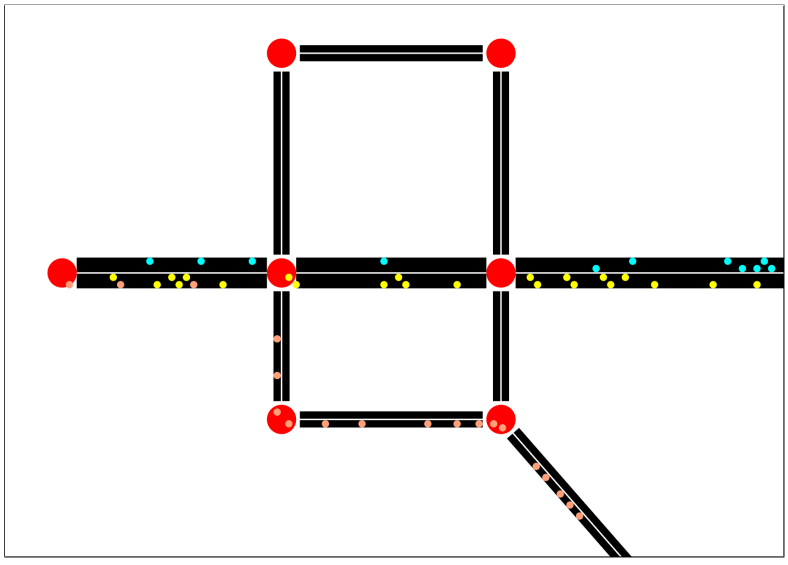
Trasy budeme v simulátoru plánovat dvěma odlišnými způsoby. Na lokální plánování tras použijeme Dijkstrův algoritmus na silniční síť, kde cenovou funkci bude tvořit délka silnic. Centrální plánování tras založíme na maximálním souběžném toku. Tento více komoditní tok nalezneme pomocí lineárního programování a z výsledného toku nalezneme trasy pro jednotlivé dopravní toky opakovaným spouštěním Dijkstrova algoritmu.

Kapitola 4
Výzkum dopravy

V této kapitole budeme chtít zkoumat zda-li je naše základní hypotéza pravdivá. Hypotéza pojednávala o tom, že centralizované řízení dopravy umožní dopravní síti pojmout mnohem větší počet vozidel a tím zvětšit průchodnost sítě než řízení lokální. Nejdříve si představíme experimenty na kterých budeme chtít hypotézu otestovat. Poté odsimulujeme experimenty pomocí implementovaného simulátoru a porovnáme statistické výstupy centralizovaného plánování tras podle maximálního souběžného toku (dále jen zkráceně MCF) s lokálním plánováním tras podle Dijkstrova algoritmu.

4.1 Vizualizace

V této kapitole se budou vyskytovat vizualizace probíhající simulace. V této části si vizualizace pro úplnost vysvětlíme.



Obrázek 4.1 Příklad vizualizace simulace

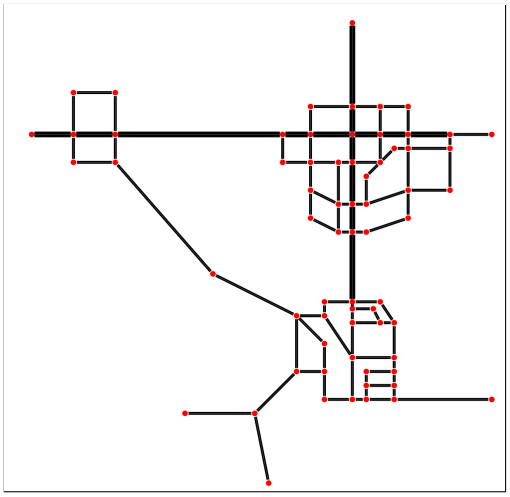
Vizualizace na obrázku 4.1 obsahuje tři typy objektů.

- **Křižovatky** mezi silnicemi jsou značeny velkými červenými kruhy.

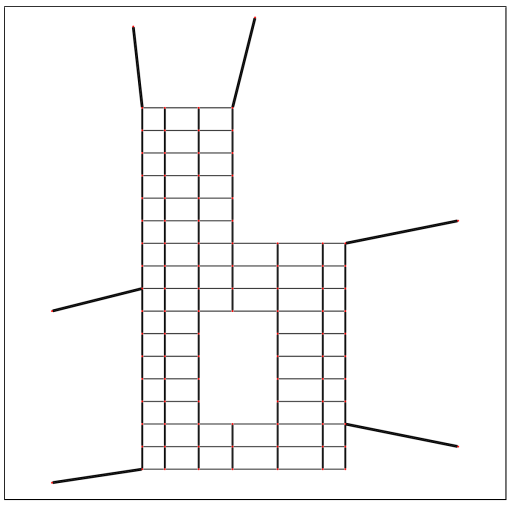
- **Silnice** jsou vyznačeny černými obdélníky mezi křižovatkami. Čím je silnice širší tím více má pruhů. Jednotlivé směry na silnici jsou odděleny bílou čárou.
- **Vozidla** jsou reprezentovány malými barevnými kruhy na silnicích. S tím, že stejně barevná vozidla jsou ze stejného dopravního toku.

4.2 Představení experimentů

V experimentech chceme modelovat situace, kdy je silniční síť velmi vytížená. Protože chceme zjistit, jaká je maximální průchodnost sítě pro jednotlivé algoritmy. Proto budou ve scénářích dopravní toky tvořeny velkým počtem vozidel a interval mezi příjezdy bude malý. Algoritmy porovnáme na dvou odlišných dopravních scénářích.



Obrázek 4.2 Rezidenční síť



Obrázek 4.3 Městská síť

- **Rezidenční síť** na obrázku 4.2 je inspirovaná městskou částí Prahy – Koloděje. Ve scénáři je 8 dopravních toků s různými zdroji a stoky. Každý z toků sestává ze 3 000 vozidel. Celkem musí sítí projet 24 000 vozidel.
- **Městská síť** na obrázku 4.3 je inspirovaná new-yorským ostrovem Manhattan. V této síti je 10 dopravních toků, z toho každý obsahuje 4 000 vozidel. Celkem musí sítí projet 40 000 vozidel.

Tabulka 4.1 Velikost dopravních scénářů

Scénář	# Křižovatky	# Silnice	# Dopravní toky
Rezidence	67	188	8
Město	103	342	10

Pro všechny toky byli rychlosti vozidel generovány z normálního rozdělení $N(\mu, \sigma^2)$, kde $\mu = 1$ a $\sigma^2 = 0.1$. Distribuce rychlosti vozidel je znázorněna v tabulce 4.2

Intervaly příjezdů mezi vozidly jsou generovány z exponenciálního rozdělení $Exp(\lambda)$, kde $\lambda = \frac{1}{5}$, tedy interval mezi vozidly bude v průměru 5 jednotek simulačního času.

Tabulka 4.2 Rozdělení rychlosti vozidel

Procento vozidel	Rychlostní interval
64 %	(0.9; 1.1)
95 %	(0.8; 1.2)
99 %	(0.7; 1.3)

4.3 Výsledky experimentů

V této podkapitole pojednáme o výsledcích experimentů pro Dijkstrův algoritmus a algoritmus MCF. Popíšeme rozdíly v průběhu simulace a provedeme diskuzi nad výsledky.

4.3.1 Vysvětlení statistických výstupů

Pro oba algoritmy jsme spustili stejný dopravní scénář a po skončení simulace jsme exportovali statistické výstupy do souboru. Z exportovaných dat jsme vytvořili tabulku a grafy k danému experimentu. Nyní si vysvětlíme význam jednotlivých položek v tabulkách 4.3 a 4.4.

1. Použitý algoritmus pro plánování tras.

2. Průměrný tok vozidel se počítá z toku vozidel během simulace. Tok vozidel v určitý časový okamžik t nám říká, kolik dojede vozidel do cíle za 1 000 časových jednotek simulace. Můžeme si to představit, jako kdybychom měřili průtok vody v potrubí a zajímalo by nás, kolik průměrně proteče vody potrubím za 60 vteřin. Tok vozidel v časový okamžik t se počítá následujícím vzorcem.

$$flow(t) = finished\_vehicles * \frac{1000}{t}$$

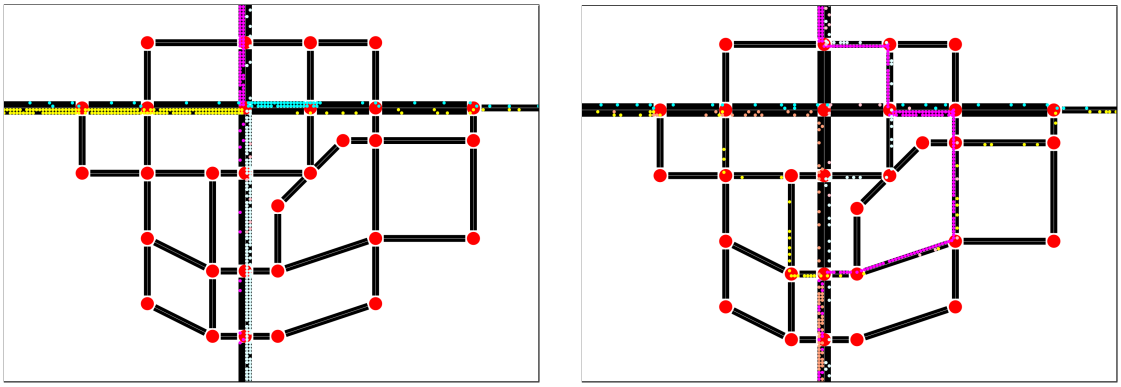
3. Délka simulace je čas simulace, kdy dojelo poslední vozidlo do cíle a simulace skončila.

4. Celková vzdálenost je celkový počet pozic na silnici, která vozidla během simulace překročila.

5. Max vozidel je maximální počet vozidel, která byla v simulaci najednou.

4.3.2 Rezidenční sít

Rezidenční síť byla inspirována nejmenovanou městskou částí Prahy. Silniční síť zde není tak hustá a je zde jen pár hlavních silnic se 2 pruhy. Přesto se zde vyskytuje spousta postranních uliček, které může tokový algoritmus MCF využít pro zvýšení průchodnosti této sítě. Proto si myslíme, že by zde pro algoritmus MCF měla být průchodnost sítě větší než pro Dijkstrův algoritmus.



(a) Dijkstrův algoritmus **(b)** MCF

Obrázek 4.4 Průběh simulace rezidenčního scénáře v čase 5 000

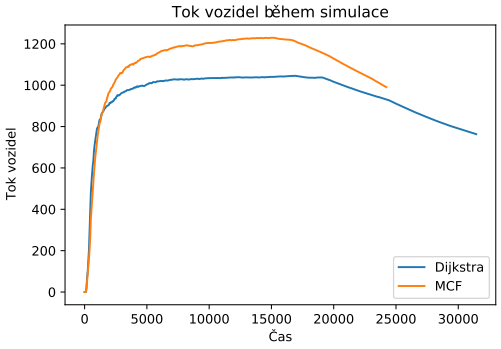
Během provádění experimentu se nám potvrdila domněnka, že při použití algoritmu MCF budou vozidla více využívat postranní silnice v dopravní síti. Tento jev můžeme vidět na obrázku 4.4, který zachycuje situaci kolem hlavní křižovatky v čase 5 000 pro oba algoritmy. Na centrální křižovatce mají všechny silnice stejnou prioritu a tím pádem se zde vozidla o přejezd křižovatky střídají. Na obrázku 4.4a si můžeme všimnout, že vozidla již tuto křižovatku velmi zahltili a utvořili se zde dlouhé fronty. Postranní silnice kolem křižovatky zůstaly nevyužité. Zatímco v průběhu experimentu s MCF (obrázek 4.4b) lze vidět, že se zde neutvořili tak velké fronty a jsou více využívány postranní silnice.

Z výsledných statistických výstupů z tohoto experimentu (tabulka 4.3 a obrázek 4.5), se potvrdila naše základní hypotéza o tom, že průchodnost sítě bude lepší pro algoritmus MCF. To lze poznat z průměrného toku vozidel v tabulce 4.3 a grafu 4.5a, kde vidíme, že algoritmus MCF má tok vozidel ostře vyšší, až na počáteční fázi experimentu. Tok vozidel se u Dijkstrova algoritmu zastavil na menší hodnotě, jelikož se síť dříve ucpala velkými frontami u vytížených křižovatek. Díky většímu toku vozidel skončila simulace pro MCF o 23 % dříve, než pro Dijkstrův algoritmus.

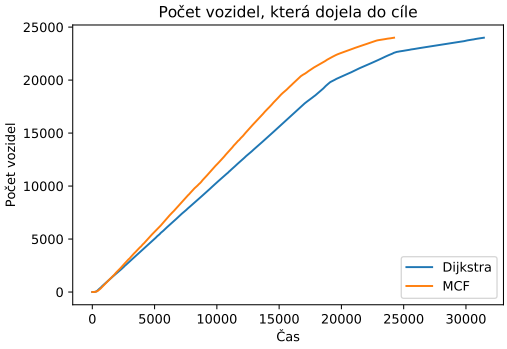
Ještě si můžeme povšimnout rozdílu v celkové ujeté vzdálenosti. Ujetá vzdálenost bude při použití Dijkstrova algoritmu vždy nejmenší možná, jelikož plánuje trasy podle nejkratší cesty v dopravní síti. Co se týká počtu vozidel v simulaci, tak ten dosáhl na vyšší čísla v případě Dijkstrova algoritmu. Při použití algoritmu MCF stíhala vozidla rychleji odjíždět a neutvořili tak velké fronty.

Tabulka 4.3 Statistické výstupy z rezidenčního scénáře

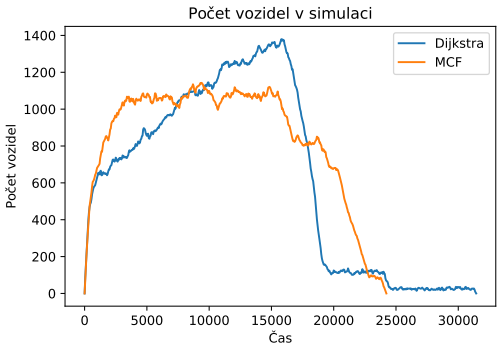
Algoritmus	Průměrný tok vozidel	Délka simulace	Celková vzdálenost	Max vozidel
Dijkstra	948	31 439	7,2 mil	1 380
MCF	1 107	24 228	8,9 mil	1140



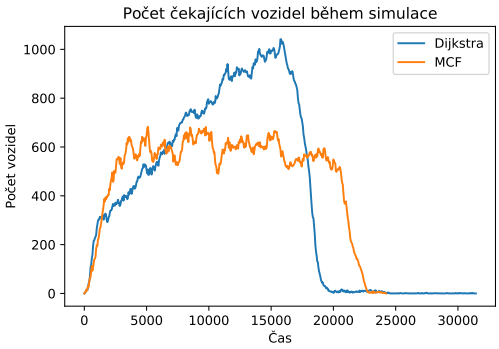
(a) Tok vozidel



(b) Počet vozidel v cíli



(c) Počet vozidel v silniční síti

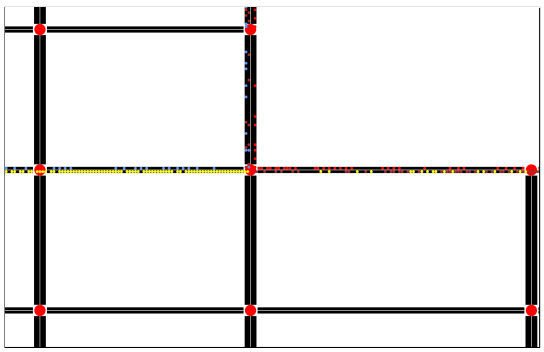


(d) Počet stojících vozidel

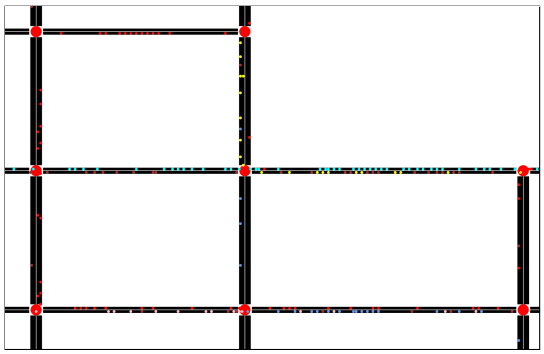
Obrázek 4.5 Grafy statistických výstupů z rezidenčního scénáře

4.3.3 Městská síť

Městská síť je inspirována Manhattanem – městskou částí New Yorku. Silniční síť je zde velmi hustá. Každý dopravní tok v tomto scénáři je vždy směrován z jednoho mostu na jiný. Proto zde budou úzká hrdla tvořit hlavně křižovatky, které spojují mosty s ostrovem. Domníváme se, že algoritmus MCF dokáže urychlit dopravu přes Manhattan díky husté síti silnic.



(a) Dijkstrův algoritmus



(b) MCF

Obrázek 4.6 Průběh simulace městského scénáře v čase 2 500

Na obrázku 4.6, který zachycuje malou část ze simulace scénáře, lze vidět, že v případě

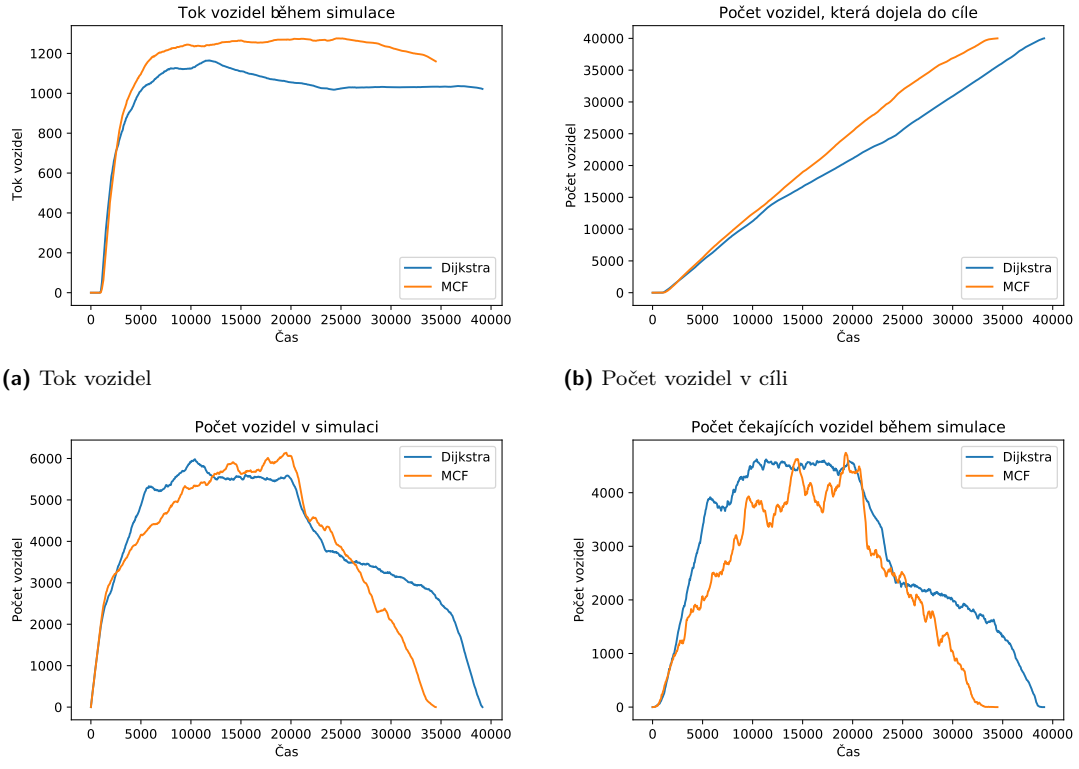
Dijkstrova algoritmu se zde zahltila jedna silnice, přes kterou pro mnohá vozidla vede nejkratší cesta do cíle. Zatímco při použití MCF je využívána i okolní silniční síť. Tedy naše domněnka, že MCF využije hustou síť Manhattanu pro urychlení dopravy byla pravdivá.

Nyní zhodnotíme statistické výstupy z městského scénáře. Podobně jako u rezidenčního experimentu lze vidět z tabulky 4.4 a obrázku 4.7, že průtok vozidel je pro algoritmus MCF vyšší. Simulace skončila pro MCF o 12 % dříve, než pro Dijkstrův algoritmus. A tedy tento experiment také potvrzuje naši základní hypotézu.

Průběh tohoto scénáře se liší oproti minulému v počtu vozidel během simulace. V rezidenčním experimentu byl pro Dijkstrův algoritmus maximální počet vozidel vyšší. Ale v tomto scénáři je vyšší pro algoritmus MCF. V tomto případě bylo příčinou to, že trasy naplánované Dijkstrovým algoritmem se při velké poptávce zahltili a již nedokázali pojmout více vozidel. Zatímco trasy z algoritmu MCF využívali větší plochu silniční sítě, a tím umožnil síti pojmout větší počet vozidel.

Tabulka 4.4 Statistické výstupy z městského scénáře

Algoritmus	Průměrný tok vozidel	Délka simulace	Celková vzdálenost	Max vozidel
Dijkstra	1 000	39 143	46 mil	5 980
MCF	1 144	34 476	53 mil	6 140



(a) Tok vozidel

(b) Počet vozidel v cíli

(c) Počet vozidel v silniční síti

(d) Počet stojících vozidel

Obrázek 4.7 Grafy statistických výstupů z městského scénáře

4.4 Výstupy z experimentů

Provedené experimenty potvrdili naši hypotézu o tom, že centralizované plánování vylepší průchodnost silniční sítě oproti lokálnímu plánování. V obou případech byl průtok vozidel v silniční síti pro centralizovaný algoritmus MCF významně vyšší než pro Dijkstrův algoritmus. A díky tomu byl i celkový simulační čas pro oba scénáře s MCF nižší. Dále se ukázalo, že celková vzdálenost, kterou vozidla během simulace ujela, byla vždy nižší pro Dijkstrův algoritmus, jelikož plánuje trasy podle nejkratší cesty v síti. V městském scénáři umožnil algoritmus MCF pojmout síti větší počet vozidel. V případě Dijkstrova algoritmu se síť dříve zahltila.

Kapitola 5
Výkonnostní testy

V této kapitole otestujeme výkonost implementovaného simulátoru. Zaměříme se na to, jaký má vliv počet vozidel v simulaci na jeho výkonnost. A porovnáme délku běhu algoritmů pro plánování tras.

5.1 Představení testů

Pro otestování výkonnosti použijeme testovací scénáře z kapitoly 4 o výzkumu dopravy. Nejdříve na scénářích změříme a porovnáme délky běhu algoritmů pro plánování tras. Dále změříme výkonnost simulátoru, kterou vztáhneme vůči 60 jednotkám simulačního času (dále jen virtuální minuta). Zejména se chceme zaměřit na to, kolik reálného času je potřeba k odsimulování virtuální minuty podle počtu vozidel.

5.2 Rychlost algoritmů

Algoritmy jsme opakovaně spouštěli na testovacích scénářích. Poté jsme dobu běhu z těchto opakování zprůměrovali a zapsali do tabulky 5.1

Tabulka 5.1 Průměr délky běhu algoritmů na jednotlivých scénářích

Scénář	Dijkstra	MCF
Rezidence	95 ms	374 ms
Město	116 ms	579 ms

Z dat je vidět, že algoritmus MCF má mnohem delší dobu běhu než Dijkstrův algoritmus. V případě rezidenčního scénáře je téměř 4x pomalejší než Dijkstrův algoritmus. A pro městský scénář je až 5x pomalejší.

Počet proměnných a omezení je v lineárním programu, maximálního souběžného toku, závislý na počtu vrcholů (křižovatek), hran (silnic) a komodit (dopravních toků). Městský scénář je podstatně větší než rezidenční scénář, proto se určité prodloužení běhu algoritmu dalo očekávat. Pro rezidenční scénář bylo vytvořeno proměnných 1 505 a 724 omezujících podmínek na proměnné. U městského scénáře je proměnných 3 421 a omezení 1 372.

5.3 Rychlost simulace

Na rezidenčním a městském scénáři s algoritmem MCF jsme spustili výkonnostní testy simulátoru. Kde jsme měřili, jak dlouho trvá od simulovat virtuální minuty na základě počtu vozidel v simulaci. S tím, že celkový počet vozidel rozdělíme na aktivní (pohybující se) a čekající (stojící) vozidla. Čekající vozidla simulaci nijak nezatěžují, jelikož neplánují žádné události, dokud se neuvolní jejich následující pozice na silnici. Proto se domníváme, že hlavním faktorem, který ovlivňuje výkonnost simulátoru jsou právě aktivní vozidla. Která neustále plánují události o svém pohybu.

Výsledky výkonnostních testů z rezidenčního scénáře jsme zapsali do tabulky 5.2 a z městského scénáře do tabulky 5.3. Ve sloupcích tabulek jsou popořadě zapsány údaje: počet všech, čekajících a aktivních vozidel. V posledním sloupci je reálný čas v milisekundách, který byl potřeba na od simulování virtuální minuty.

Celkový běh simulace rezidenčního scénáře byl 24 tisíc jednotek simulačního času a trval odsimulovat 26 vteřin reálného času. Pro městsky scénář byl celkový běh simulace 35 tisíc jednotek simulačního času a simulace trvala 210 vteřin (3,5 minuty) reálného času. Vidíme, že odsimulování městského scénáře trvalo téměř 8x více reálného času než v případě rezidenčního scénáře. A přitom virtuální běh městského scénáře byl delší jen o 50 % oproti rezidenčnímu scénáři. Toto je způsobeno tím, že městský scénář obsahoval mnohem větší počet vozidel během simulace, než rezidenční scénář. V městském scénáři byl největší počet vozidel v simulaci okolo 6 000 a v rezidenčním 1 000.

Na datech z tabulek 5.2 a 5.3 můžeme vidět, že výkon simulátoru ovlivňují převážně aktivní vozidla. Když se podíváme do tabulky 5.2 na poslední řádek, kde bylo v simulaci 1 000 vozidel a 320 bylo z nich aktivní, tak reálný čas byl 75 ms. A když se podíváme na odlišnou situaci do tabulky 5.3 na 3. řádek, kde bylo v simulaci 350 vozidel, ale z toho byla skoro všechna aktivní. Tak vyšel reálný čas na od simulování podobně i přes to, že v prvním případě bylo 3x více vozidel.

Tabulka 5.2 Čas na odsimulování virtuální minuty v rezidenčním scénáři

# Vozidla	# Čekající vozidla	# Aktivní vozidla	Čas
14	0	14	1 ms
32	2	30	5 ms
75	4	71	11 ms
150	25	125	25 ms
300	150	150	35 ms
600	400	200	40 ms
1000	680	320	75 ms

Tabulka 5.3 Čas na odsimulování virtuální minuty v městském scénáři

# Vozidla	# Čekající vozidla	# Aktivní vozidla	Čas
20	0	20	2 ms
70	0	70	12 ms
350	12	338	65 ms
1 000	130	870	160 ms
2 000	950	1150	270 ms
3 500	2100	1400	310 ms
5 000	3500	1500	360 ms
6 100	4500	1600	450 ms

5.4 Výsledky testů

Z výkonnostních testů se ukázalo, že rychlost simulace ovlivňují především pohybující se vozidla, jelikož neustále plánují do simulace události o pohybu. Čekající vozidla, která stojí ve frontě, simulaci nijak nezatěžují. Dále jsme porovnali délky běhů algoritmů pro plánování tras. Algoritmus MCF běžel vždy déle než Dijkstrův algoritmus. Pro městský scénář byla délka jeho běhu až 5x delší.

Závěr

Cílem této práce bylo na základě rešerše na téma diskrétní simulace, provést návrh a implementaci dopravního simulátoru s diskrétním prostorem a spojitým časem, jehož hlavní využití bude pro porovnání algoritmů na plánování tras. Dále bylo potřeba vybrat a implementovat dva odlišné způsoby na plánování tras, jeden lokální a jeden centralizovaný způsob, a implementovat je. A na závěr tyto algoritmy v simulátoru porovnat na několika připravených experimentech a posoudit zda potvrzují naši hypotézu o tom, že použití centralizovaného řízení dopravy zlepší průchodnost dopravní sítě oproti lokálnímu řízení dopravy.

Nejdříve jsme provedli rešerši na témata diskrétní simulace, simulace dopravy a více komoditní tok. Po rešerši jsme zhodnotili existující řešení v oblasti simulace dopravy. Na základě nabitých poznatků jsme navrhli simulátor dopravy jako diskrétní simulaci, která pro posun času používá techniku Next event. Silniční síť a prostor na silnicích je modelován konečným orientovaným grafem. Pro plánování tras jsme v simulátoru implementovali dva odlišné algoritmy. Prvním z nich je plánování tras podle nejkratší cesty, na který jsme použili Dijkstrův algoritmus. Druhým způsobem je plánování tras podle maximálního souběžného toku, který hledáme pomocí lineárního programování. Tyto algoritmy jsme pomocí simulátoru porovnali na dvou testovacích scénářích, kde se ze statistických výstupů potvrdila naše základní hypotéza. Na závěr jsme provedli výkonnostní testy simulátoru, kde jsme pozorovali, jak počet vozidel ovlivňuje výkonnost simulátoru. Z výsledků testů jsme zjistili, že hlavním faktorem, který ovlivňuje výkonnost jsou pohybující se vozidla. Vozidla, která čekají ve frontách nemají na rychlost simulace příliš velký vliv. V příloze práce lze najít dokumentaci a uživatelskou příručku k simulátoru.

Výsledný simulátor je možné do budoucna rozšířit o následující funkce:

- Paralelní zpracování událostí, které by vedlo ke znatelnému zrychlení simulace.
- Podpora světelných křižovatek, které by v některých dopravních scénářích lépe reflektovali reálnou situaci.
- Grafický editor dopravních scénářů. Díky kterému by se zjednodušilo modelování dopravních scénářů pro simulátor.
- Možnost importovat reálnou silniční síť z OpenStreetMap. Tato funkce by velmi usnadnila přípravu reálných dopravních scénářů, jelikož by se dala část dopravní sítě importovat přímo z aktuálních map.

Hlavním přínosem této práce je implementovaný simulátor, díky kterému je možné rychle porovnávat různé algoritmy pro plánování tras. Dalším přínosem je centralizovaný algoritmus pro plánování tras založený na maximálním souběžném toku, který maximalizuje průchodnost silniční sítě při znalosti budoucí poptávky. Tento způsob plánování tras nelze běžně najít v existujících dopravních simulátorech.

Příloha A

Dokumentace

A.1 Vybrané technologie

K implementaci simulátoru jsem použil programovací jazyk Scala [14] s kterým jsem se seznámil v předmětu BI-OOP a zaujal mě přívětivou syntaxí a kombinací mezi funkcionálním a objektově orientovaným programováním. Jazyk je kompilován na java class soubory, které jsou dále kompilovány na bytecode pomocí platformy JVM (Java Virtual Machine). Proto je možné v jazyku Scala používat jakýkoliv kód z Javy. Díky tomu je možné využívat obrovskou škálu knihoven, která pro Javu v průběhu let vznikla. Podle mého názoru je velká výhoda jazyku Scala oproti jazyku Java především v kompaktnější syntaxi, absenci null referencí a lepší podpoře funkcionálního programování.

Pro vytvoření uživatelského rozhraní jsem si zvolil knihovnu ScalaFX [15], která poskytuje API knihovny JavaFX pro jazyk Scala. Tuto knihovnu jsem si vybral, protože JavaFX obsahuje již v základu velké množství komponent a funkcionalit. Najdeme zde např.: komponenty pro grafy, vykreslování geometrických tvarů a formulářové komponenty (tlačítka, textové pole atd.). Také je tato knihovna velice dobře zdokumentovaná a populární.

k serializaci a deserializaci jsem si vybral knihovnu Play JSON [16], která umožňuje pomocí funkcionální syntaxe nastavit pravidla pro ukládání a načítání jednotlivých objektů ve formátu JSON (JavaScript Object Notation).

Pro lineární programování v jazyku Scala jsem použil knihovnu Optimus [17], která poskytuje jednotné API pro velkou škálu knihoven, které řeší lineární optimalizaci. Jako řešič lineární optimalizace jsem použil proprietární knihovnu Gurobi [18], která se řadí mezi nejrychlejší knihovny v této oblasti a pro akademické účely je volně dostupná. Nejdříve jsem zkoušel použít i volně dostupné open source knihovny pro řešení lineární optimalizace, ale pro větší počty proměnných buď, řešení nenašli anebo ho hledali, až příliš dlouho.

A.2 Struktura projektu

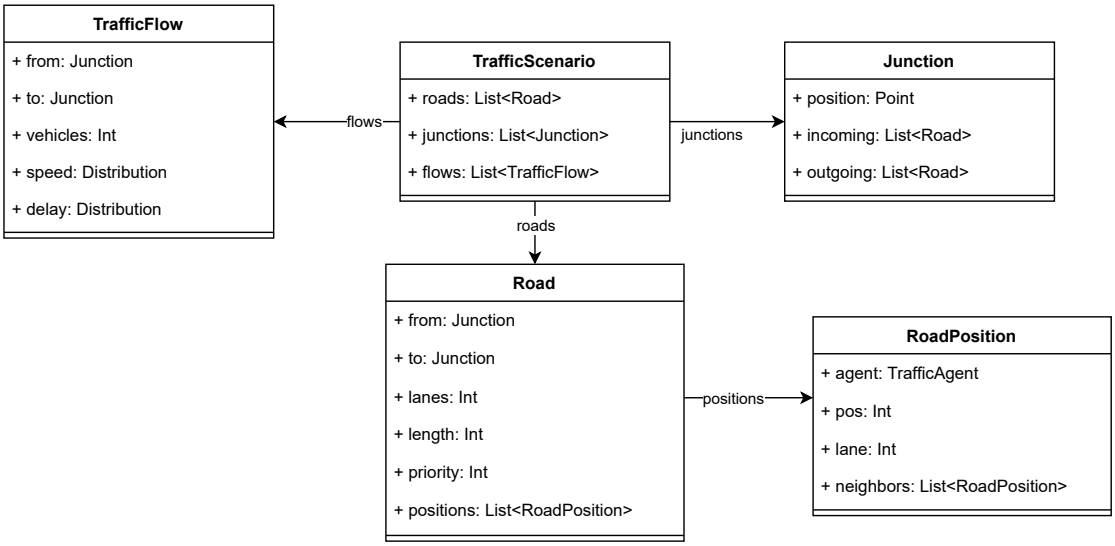
Jelikož simulátor musí splňovat velkou škálu funkčních požadavků, tak je vhodné dělit třídy do různých balíčků a příp. pod-balíčků podle jejich účelu. Celý projekt je členěn do tří hlavních balíčků:

- `Core` – obsahuje třídy, které tvoří jádro simulace. Najdeme zde model dopravní sítě, logiku a modely diskrétní simulace (engine, stav, procesy, události, frontu událostí), generátory náhodných čísel, statistické třídy a algoritmy pro plánování tras.
- `Serialization` – tento balíček obsahuje třídy, které se starají o načítání a ukládaní dopravního scénáře ve formátu JSON. Tento balíček využívá balíček `Core`.
- `UI` – obsahuje třídy, které tvoří uživatelské rozhraní simulátoru. Tento balíček využívá balíčky `Core` a `Serialization`.

A.3 Model dopravního scénáře

Dopravní scénář je modelován třídou `TrafficScenario`, která obsahuje nadefinované silnice (`roads`), křižovatky (`junctions`) a dopravní toky (`flows`). Třída `Junction` obsahuje pozici, která je unikátní v rámci cele silniční sítě a příchozí a odchozí silnice. Třída `Road` je tvořena z počáteční a cílové křižovatky. Dále obsahuje počet pruhů, délku, prioritu a seznam jednotlivých pozic pro vozidla.

Třída `TrafficFlow`, reprezentuje dopravní tok. Obsahuje počáteční a cílovou křižovatku – počet vozidel, které se v toku vyskytují – rychlost a interval mezi vozidly, které jsou definovány třídou `Distribution`, reprezentující náhodné rozdělení.



Obrázek A.1 Diagram tříd dopravního scénáře

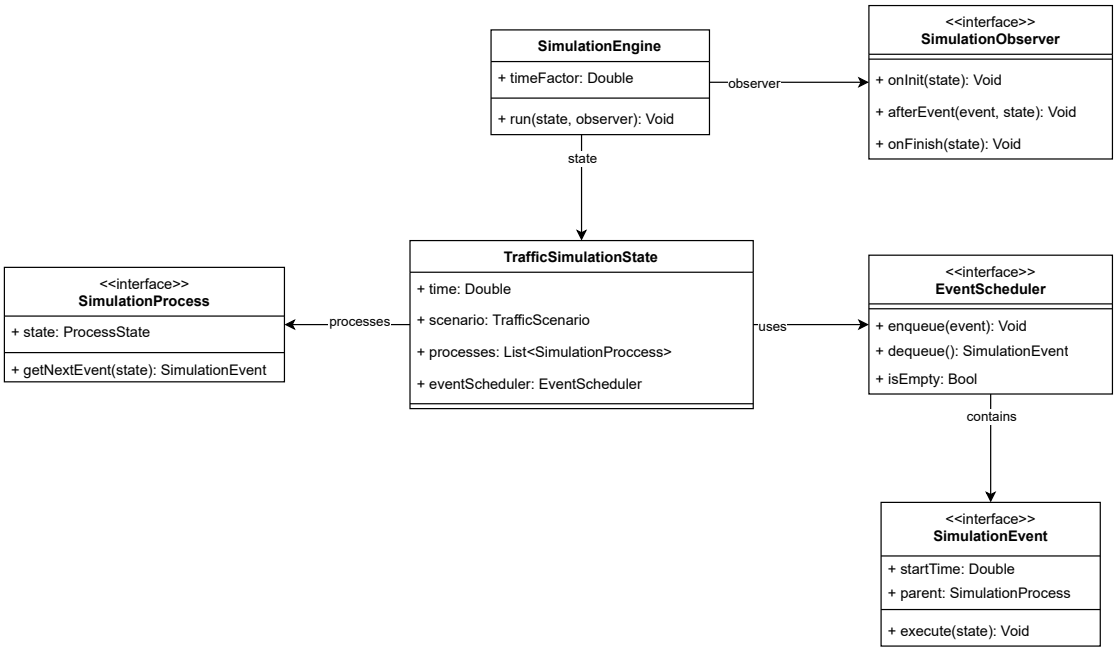
A.4 Jádro diskrétní simulace

Ve třídě `SimulationEngine`, je implementovaná hlavní smyčka diskrétní simulace. Posun času je ve smyčce realizován metodou *Next event*. Pomocí proměnné `timeFactor`, lze simulaci zpomalit. Pokud bude hodnota proměnné 0, tak nedojde k žádnému zpomalení. Hlavní smyčku, lze spustit pomocí metody `run`, která jako první parametr přijímá počáteční stav simulace a jako druhý parametr přijímá rozhraní `SimulationObserver`, které je voláno ve hlavní smyčce a to na začátku, po vykonání události a na konci simulace.

Stav simulace dopravy je modelován třídou `TrafficSimulationState`. Tato třída obsahuje veškeré proměnné, které zachycují aktuální stav simulovaného systému.

- `time` – uchovává aktuální čas simulace. Je reprezentován číslem s plovoucí čárkou.
- `scenario` – simulovaný model dopravního scénáře. Uchovává rozmístění vozidel.
- `processes` – seznam aktivních procesů, které běží v simulaci.
- `eventScheduler` – třída, která uchovává naplánované události.

Událost v simulaci je reprezentována rozhraním `SimulationEvent`. Proměnná `startTime` je čas na který je událost naplánována. V proměnné `parent` je rodičovský proces, který naplánoval událost do simulace. Pomocí metody `execute` je naplánovaná událost vykonána.



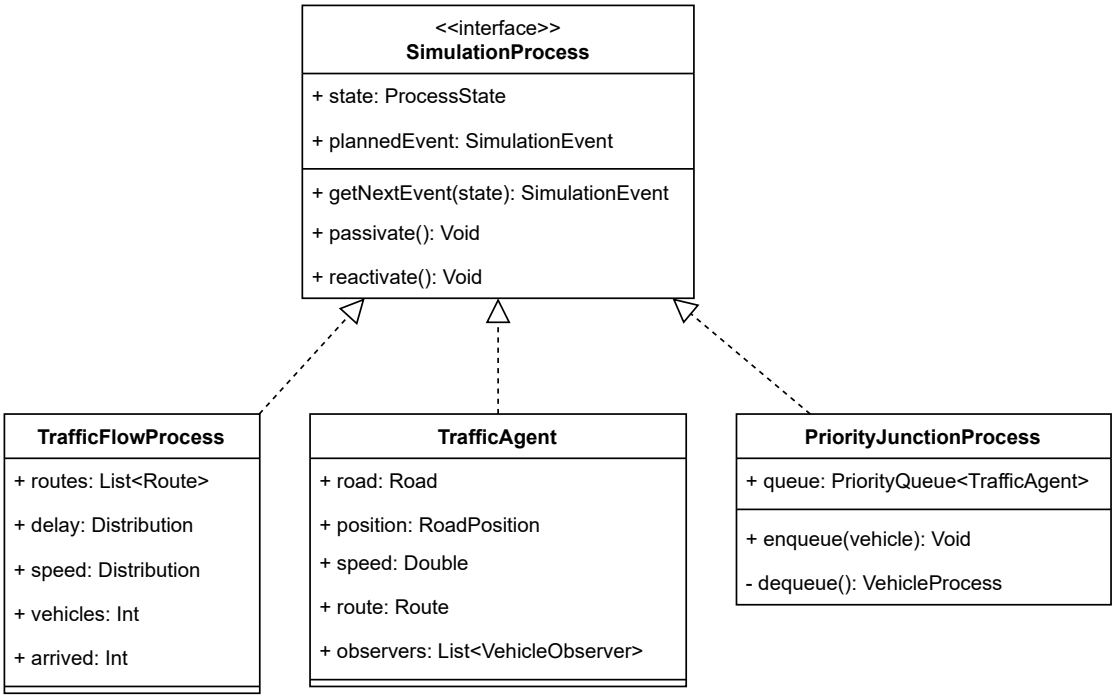
Obrázek A.2 Diagram tříd diskrétní simulace

Rozhraní `SimulationProcess` reprezentuje proces v diskrétní simulaci, který během svého životního cyklu plánuje do simulace události. Získání další události k naplánování je realizováno metodou `getNextEvent`, která dostane na vstupu aktuální stav simulace a vrátí událost k naplánování. Tato metoda je volána pokaždé, když se vykoná předchozí naplánovaná událost procesu. Proměnná `state`, zachycuje stav procesu. Může nabývat následujících hodnot:

- `active` – stav, kdy má proces naplánovanou událost.

- `passive` – stav, kdy je proces uspán a čeká na událost událost, která ho probudí. Např.: se může jednat o proces vozidla, které se nemůže pohnout na další pozici na silnici a uspí se než se některá z pozic uvolní.
- `finished` – stav, kdy proces vykonal všechny své události a může být odstraněn ze simulace.

V simulátoru jsou implementovány tři třídy simulačních procesů. Všechny tyto třídy implementují rozhraní `SimulationProcess`. Třída `TraffiFlowProcess` slouží pro vkládání vozidel podle zadaných vlastností do simulace. Třída `TrafficAgent` řídí pohyb vozidel. A poslední třída `PriorityJunctionProcess` řídí pohyb na křižovatkách.

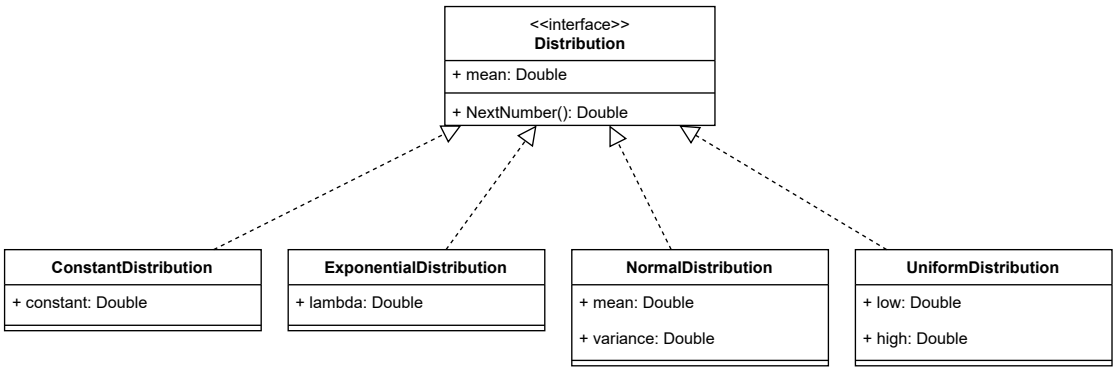


Obrázek A.3 Diagram tříd procesů v simulaci

A.4.1 Náhodná rozdělení

Generátory náhodných čísel jsou v simulátoru zastřešeny pod rozhraním `Distribution`, které definuje metodu `nextNumber`, která vygeneruje další číslo a metodu `mean`, která vrátí střední hodnotu náhodného rozdělení, pokud existuje. V simulátoru jsou implementovány čtyři náhodná rozdělení.

- `ConstantDistribution` – rozdělení, které pokaždé vrátí hodnotu uloženou v `constant`.
- `UniformDistribution` – reprezentuje uniformní rozdělení $U(a, b)$. V konstruktoru přijímá spodní hranici `low` a horní hranici `high`. Čísla z intervalu mezi spodní a horní hranicí jsou generována se stejnou pravděpodobností.
- `ExponentialDistribution` – reprezentuje generátor náhodných čísel z exponenciálního rozdělení $Exp(\lambda)$. V konstruktoru přijímá hodnotu parametrů `lambda`.
- `NormalDistribution` – reprezentuje generátor náhodných čísel z normálního rozdělení $N(\mu, \sigma^2)$. V konstruktoru přijímá hodnoty parametrů `mean` a `variance`.



Obrázek A.4 Diagram tříd pro náhodná rozdělení

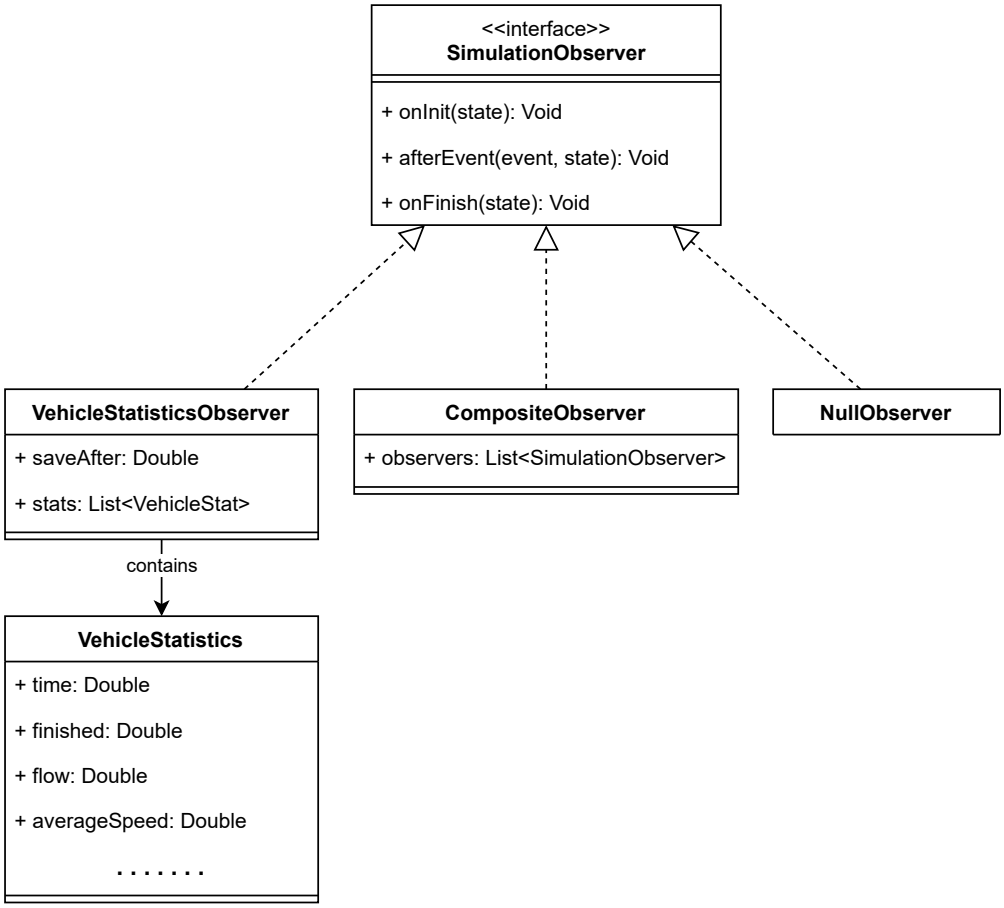
A.4.2 Sledování událostí během simulace

Sledování události během simulace je velice užitečný vzor pomocí, kterého lze realizovat mnoho funkcionalit. V simulátoru je tato metoda využívána pro sběr statistik a vykreslování simulace. Simulaci tak lze překreslit přesně ve chvíli, když nastane nějaká změna stavu a není potřeba ji neustále překreslovat po zvoleném časovém intervalu.

Pro sledování simulace je potřeba implementovat rozhraní `SimulationObserver`, které předáme jako parametr při spouštění simulace do metody `run` ve třídě `SimulationEngine`. Toto rozhraní definuje metody: `onInit`, `afterEvent` a `onFinish`. Všechny tyto metody přijímají v parametru aktuální stav simulace. Metoda `onInit` je zavolána na začátku simulace po inicializační fázi. Metoda `afterEvent` je volána po každé vykonané události a v parametru přijímá tuto událost. A na konci simulace je zavolána metoda `onFinish`.

Pokud chceme sledovat simulaci z více tříd, tak použijeme třídu `CompositeObserver`, která v konstruktoru bere seznam tříd `SimulationObserver` a rozesílá jim událostí z průběhu simulace. A pokud nechceme sledovat simulaci tak stačí použít třídu `NullObserver`, která na události nereaguje.

Pomocí třídy `VehicleStatisticsObserver` je realizován sběr statistik ze simulace. Po každé události, které ovlivňují sledované statistiky definované ve třídě `VehicleStatistics`, je tato třída ihned přepočítá. Po uplynuté simulační době určené parametrem `saveAfter`, uloží aktuální statistiky do seznamu `stats`.



Obrázek A.5 Diagram tříd pro sledování událostí

A.5 Třídy pro plánování tras

Algoritmus pro plánování tras musí implementovat rozhraní `RoutePlanner`, které vyžaduje realizovat metodu `findRoutes` s dvěma parametry. Prvním parametrem je dopravní scénář a druhým je dopravní tok ze scénáře pro, který bude trasy plánovat. Výsledkem této metody je seznam tras, které vedou od počátku do cíle zadaného dopravního toku. U každé trasy je pravděpodobnost jejího výběru (`chance`), s tím, že součet pravděpodobností přes všechny trasy musí být 1.

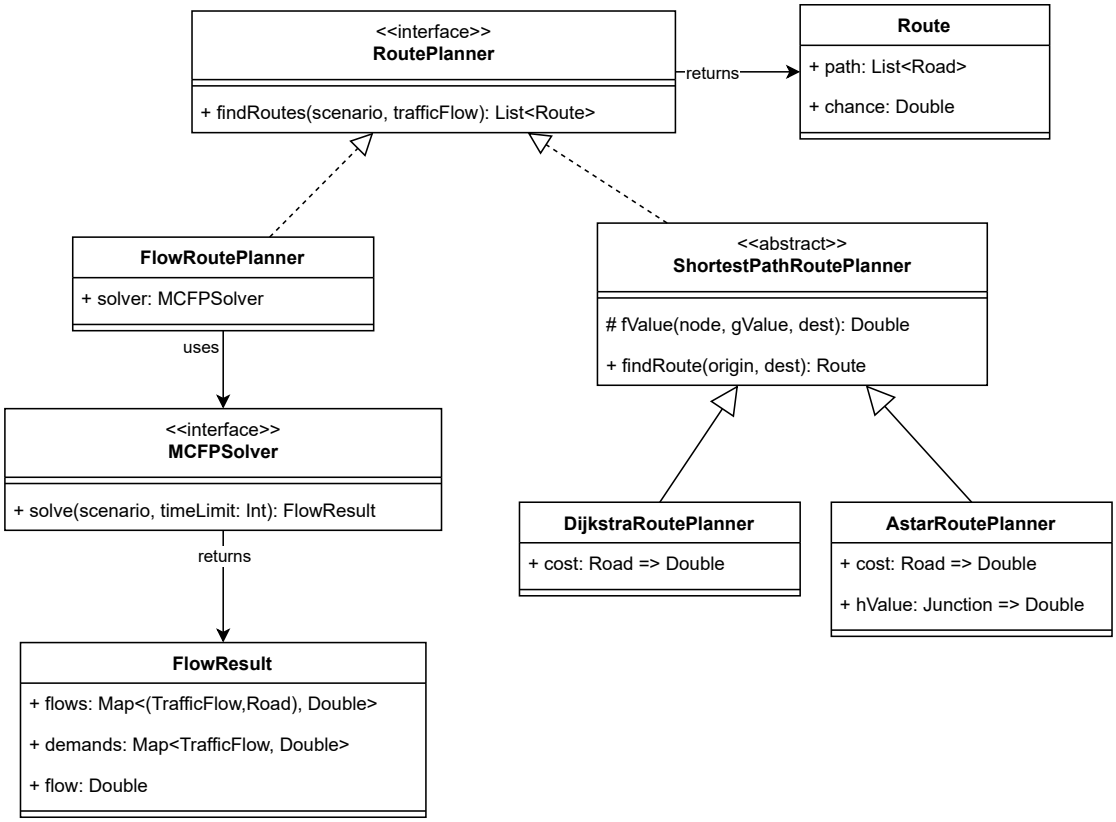
Pro přidání nového algoritmu pro plánování stačí vytvořit v projektu novou třídu, která bude implementovat rozhraní `RoutePlanner`. A do metody `apply` v objektu `RoutePlanner` stačí přidat řádek, kterým určíme název přidaného algoritmu. Tento název je použit při deserializaci dopravního scénáře ze souboru.

V abstraktní třídě `ShortestPathRoutePlanner` je implementována logika Dijkstrova algoritmu s prioritní frontou. Tato třída vyžaduje implementovat funkci `fValue`, která přiřazuje cenu vrcholu pro prioritní frontu. Každému dopravnímu tok vrátí nejlevnější trasu podle zadané funkce `fValue`. Z této třídy dědí následující podtřídy:

- `DijkstraRoutePlanner`, který v konstruktoru přijímá cenovou funkci `cost` pro ohodnocení hran (silnic). Funkce `fValue` vrací hodnotu `gValue` z parametru. V hodnotě `gValue` je aktuální vzdálenost z počátečního vrcholu.
- `AstarRoutePlanner`, který v konstruktoru přijímá ohodnocení hran `cost` a heuristickou funkci

`hValue`. Funkce `fValue` je zde implementována jako `gValue` + `fValue`.

Třída `FlowRoutePlanner` slouží pro plánování tras z více komoditního toku. K nalezení více komoditního toku používá rozhraní `MCFPSolver` (multi-commodity flow problem solver). Toto rozhraní vyžaduje implementovat metodu `solve` se 2 parametry. Prvním parametrem je dopravní scénář a druhým je časový limit typu Int na nalezení dopravního toku. Tato metoda v případě úspěchu vrací objekt `FlowResult`, který obsahuje hodnoty toků pro všechny silnice a komodity (dopravní toky) ve slovníku `flows`, kde klíčem je silnice a komodita. Dále obsahuje hodnoty poptávek pro jednotlivých komodit (`demands`) a velikost více komoditního toku (`flow`). Rozhraní `MCFPSolver` implementuje třída `LinearMCFPSolver`, která tento tok hledá pomocí lineárního programování za použiti knihoven Optimus a Gurobi.



Obrázek A.6 Diagram tříd pro plánování tras

A.6 Načítání vstupního souboru

Dopravní scénář je načítán ze souboru ve formátu JSON pomocí knihovny PlayJSON. Logika načítání je obsažena v balíčku `Serialization`. Jednotlivá pravidla pro načítání tříd jsou definována v objektu `JsonReadsImplicits`. Dopravní scénář lze načíst ze souboru pomocí třídy `TrafficScenarioJsonImport`, která využívá nadefinovaných pravidel z `JsonReadsImplicits`. Vstupní soubor JSON s dopravním scénářem má následující strukturu.

- `roads` – seznam silnic
  - `from` – počáteční pozice silnice.
  - `to` – koncová pozice silnice.
  - `lanes` – počet pruhů. Kladné nenulové číslo typu int.
  - `priority` – priorita silnice. Kladné nenulové číslo typu int.
  - `type` – typ silnice. Může nabývat hodnot `"oneWay"` a `"twoWay"`.
- `flows` – seznam dopravních toků
  - `from` – počáteční pozice dopravního toku.
  - `to` – koncová pozice dopravního toku.
  - `delay` – náhodné rozdělení pro interval mezi příjezdy vozidel.
  - `speed` – náhodné rozdělení pro rychlosti vozidel. Nepovinný údaj.
  - `departure` – čas, kdy začnou z toku přijíždět vozidla. Kladné číslo typu double.
  - `vehicles` – celkový počet vozidel v dopravním toku. Může nabývat kladných celočíselných hodnot.
  - `color` – barva vozidel z dopravního toku. Může být zadána anglickým názvem (`"red"`) nebo hexadecimálním zápisem ve formátu `"#FFFFFF"`.
  - `planner` – algoritmus pro plánování tras. Nepovinný údaj.
- `defaults` – výchozí hodnoty simulace
  - `planner` – výchozí algoritmus pro plánování tras dopravních toků. Může nabývat hodnot `"dijkstra"`, `"astar"` a `"flow"`.
  - `speed` – výchozí rozdělení rychlosti vozidel pro dopravní toky.

Pozice mají následující strukturu:

- `x` – číslo typu double.
- `y` – číslo typu double.

Náhodná rozdělení mají v souboru strukturu:

- `type` – udává typ náhodného rozdělení. Možné hodnoty jsou: `"constant"`, `"uniform"`, `"normal"` a `"exponential"`.
- `constant` – hodnota konstanty pro konstantní rozdělení. Číslo typu double.
- `low` – spodní hranice pro uniformní rozdělení. Číslo typu double.
- `high` – horní hranice pro uniformní rozdělení. Číslo typu double.
- `mean` – střední hodnota pro normální rozdělení. Číslo typu double.
- `variance` – rozptyl pro normální rozdělení. Číslo typu double.
- `lambda` – parametr lambda pro exponenciální rozdělení. Kladné číslo typu double.

Výpis kódu A.1 Ukázka vstupního souboru s dopravním scénářem

```
{
    "roads" : [{
        "from" : {
            "x" : 0,
            "y" : 0
        },
        "to" : {
            "x" : 10,
            "y" : 0
        },
        "lanes" : 2,
        "priority" : 1,
        "type" : "twoWay"
    }],
    "flows" : [{
        "from" : {
      "x" : 0,
      "y" : 0
    },
    "to" : {
      "x" : 10,
      "y" : 0
    },
    "delay" : {
      "type" : "exponential",
      "lambda" : 0.2
    },
    "speed" : {
      "type" : "constant",
      "constant" : 1
    },
    "departure" : 0,
    "vehicles" : 1000,
    "color" : "orange"
    }],
    "defaults" : {
        "speed" : {
            "type" : "normal",
            "mean" : 1,
            "variance" : 0.1
        },
        "planner" : "flow"
    }
}
```

Příloha B

Uživatelská příručka

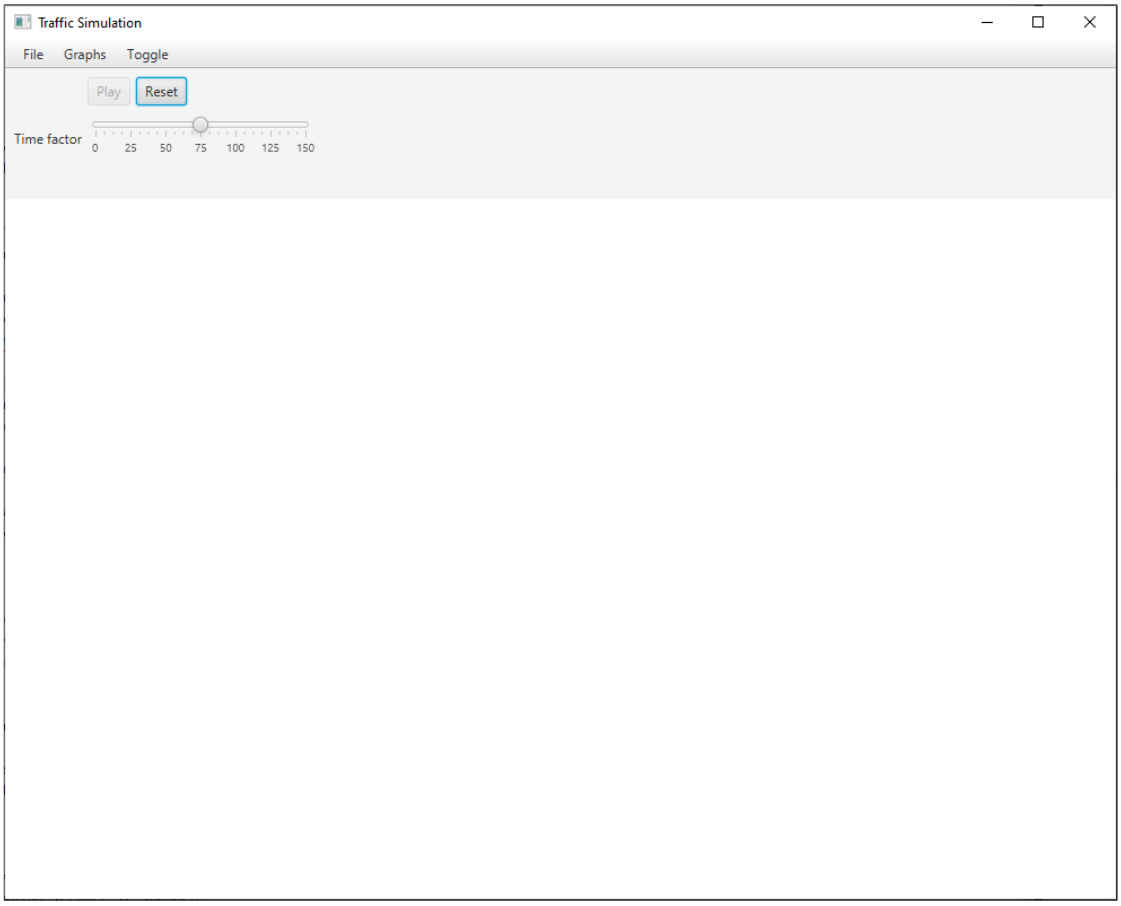
B.1 Minimální požadavky

Pro spuštění simulátoru je potřeba mít nainstalovaný Java SE verzi 1.8 nebo 11. Nainstalovanou verzi javy lze zjistit zadáním příkazu `java -version` do příkazové řádky. Java SE 11 je volně dostupná z odkazu `https://www.oracle.com/cz/java/technologies/javase-jdk11-downloads.html`.

Použití tokového algoritmu vyžaduje proprietární knihovnu Gurobi, která slouží pro řešení lineární optimalizace. Pro akademické účely volně dostupná a lze ji stáhnout na stránce stránce `https://www.gurobi.com/`.

B.2 Spuštění

Pokud jsou minimální požadavky splněny, tak lze simulátor spustit ze souboru `TrafficSimulation.jar` na přiloženém médiu. Po spuštění se zobrazí prázdné hlavní okno simulátoru.



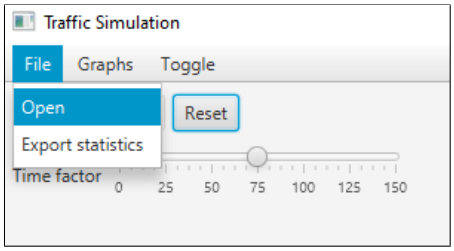
Obrázek B.1 Hlavní okno simulátoru

B.3 Simulátor

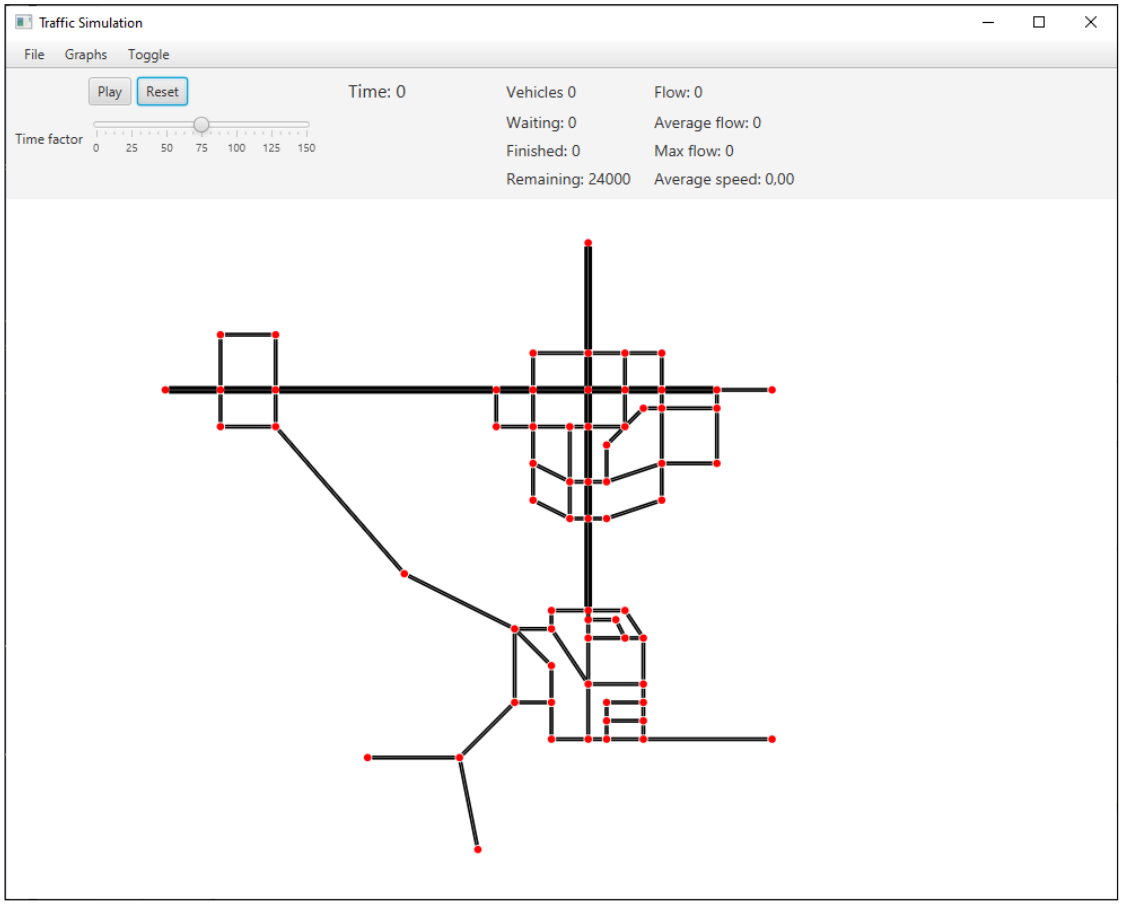
V této sekci popíšeme celkové ovládání simulátoru. Nejprve si řekneme, jak lze načíst dopravní scénář ze souboru do simulátoru. Poté si ukážeme, jak simulaci spustit. A na závěr předvedeme, jak lze zobrazit a exportovat statistické výstupy ze simulace.

B.3.1 Načtení dopravního scénáře

Prvním krokem ke spuštění simulace je načtení dopravního scénáře ze souboru. Podporovaný soubor s dopravním scénáři je ve formátu JSON. Soubor k načtení lze vybrat volbou v horní liště `File > Open`. Po vybrání souboru se v simulátoru zobrazit načtená silniční síť. V případě chyby při načítání, se objeví dialogové okno s chybovou hláškou.



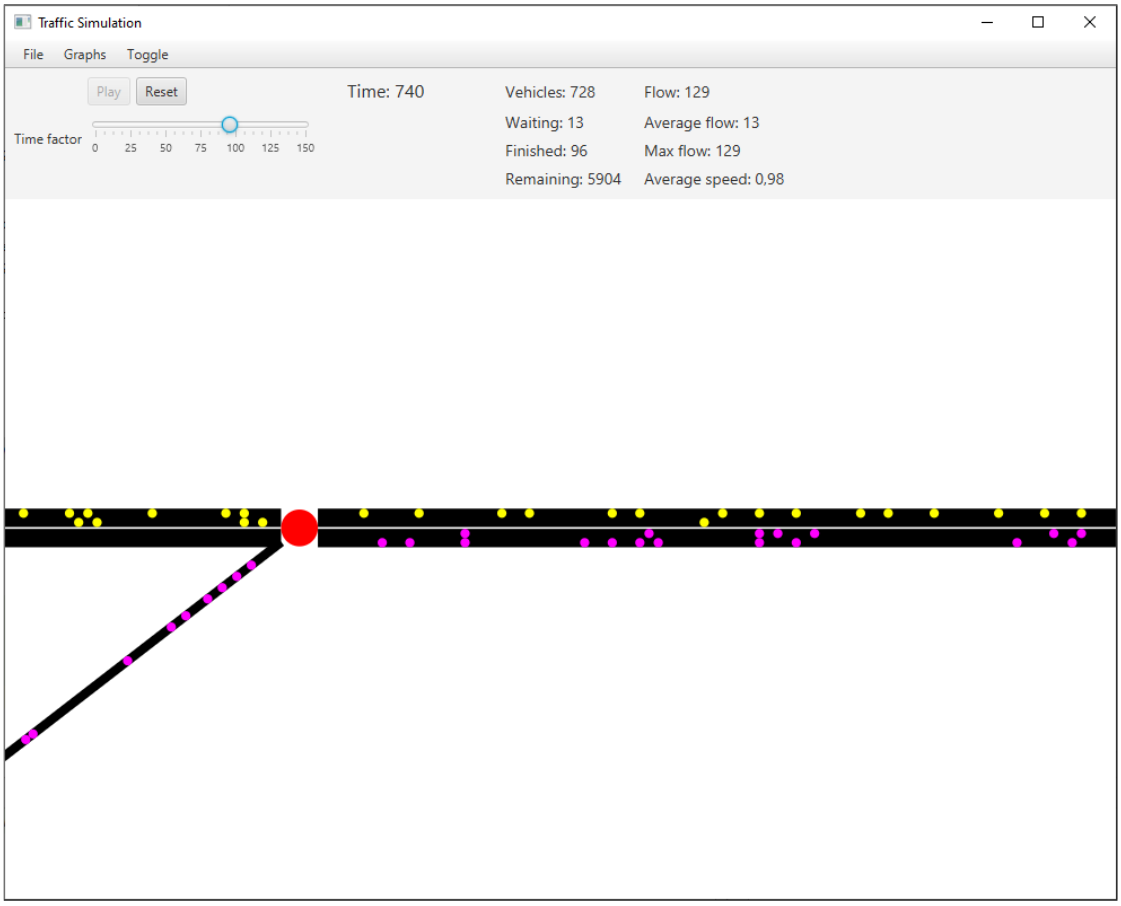
Obrázek B.2 Načtení dopravního scénáře ze souboru



Obrázek B.3 Simulátor s načteným dopravním scénářem

B.3.2 Zapnutí simulace

Po načtení dopravního scénáře lze simulaci spustit kliknutím na tlačítko `Play`. Posuvníkem `Time factor`, lze volit rychlost simulace. Přetažením doleva se simulace zrychlí a přetažením doprava zpomalí. Tlačítko `Reset` vrátí simulaci do počátečního stavu.



Obrázek B.4 Probíhající simulace v simulátoru

Načtenou silniční síť je možné přiblížit a oddálit kolečkem na myši, pokud je kurzor myši nad hlavním oknem s mapou. Po stisknutí levého tlačítka na myši, lze myší mapu posouvat.

B.3.3 Statistické výstupy

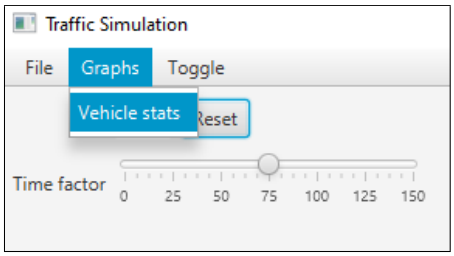
Napravo od ovládacích prvků lze vidět aktuální statistiky během simulace. Najdeme zde:

1. Time – aktuální čas simulace.
2. Vehicles – počet vozidel v simulaci.
3. Waiting – počet stojících vozidel v simulaci.
4. Finished – počet vozidel, která dojela do cíle.
5. Remaining – celkový počet vozidel, která ještě musí dojet do cíle.
6. Flow – velikost toku vozidel.
7. Average flow – průměrná velikost toku vozidel během simulace.
8. Average speed – aktuální průměrná rychlost vozidel v simulaci.

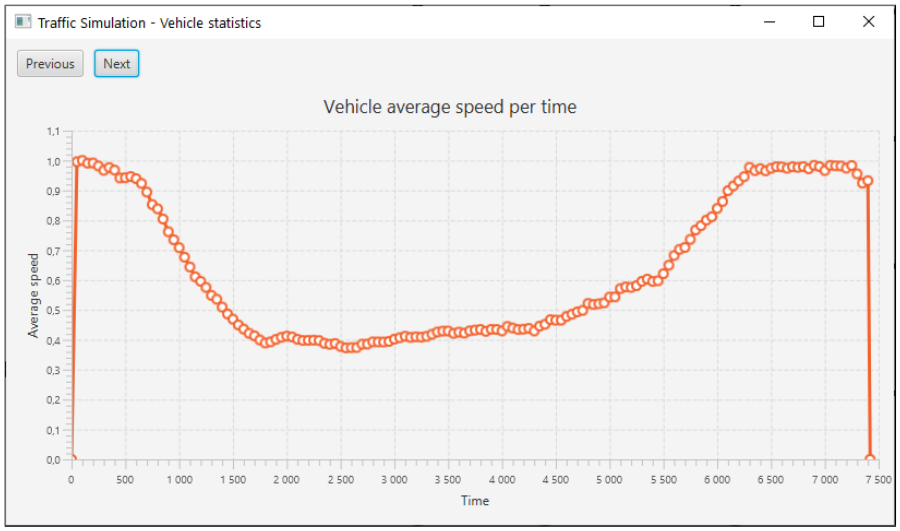
Statistické grafy lze zobrazit volbou v horní liště `Graphs > Vehicle stats`. Po kliknutí se zobrazí dialogové okno s grafy. Mezi následujícími grafy lze přepínat pomocí tlačítek `Previous` a `Next`.

- Velikost toku vozidel.
- Průměrná rychlost vozidel.
- Celkový počet zbývajících vozidel.
- Počet čekajících vozidel.
- Počet vozidel v simulaci.
- Ujetá vzdálenost.

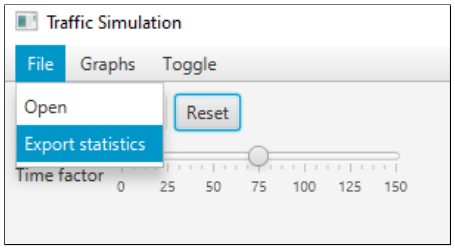
Statistické výstupy lze exportovat do souboru ve formátu JSON pomocí volby v horní liště `File > Export stats`. Po kliknutí se zobrazí dialogové okno pro výběr umístění souboru.



Obrázek B.5 Otevření okna se statistickými grafy



Obrázek B.6 Graf průměrné rychlosti vozidel



Obrázek B.7 Exportovaní statistik do souboru

Bibliografie

1. WINSBERG, Eric. Computer Simulations in Science. In: ZALTA, Edward N. (ed.). *The Stanford Encyclopedia of Philosophy*. Winter 2019. Metaphysics Research Lab, Stanford University, 2019. Dostupné také z: `https://plato.stanford.edu/archives/win2019/entries/simulations-science`.
2. BANKS, J.; CARSON, J.; NELSON, B.; NICOL, D. *Discrete-event System Simulation*. Prentice Hall, 2010. ISBN 9780136062127. Dostupné také z: `https://books.google.cz/books?id=cqSNnmrqqbQC`.
3. ULLRICH, Oliver; LÜCKERATH, Daniel. An Introduction to Discrete-Event Modeling and Simulation. *Simulation Notes Europe*. 2017, roč. 27. Dostupné z DOI: `10.11128/sne.27.on.10362`.
4. JOHNSON, James L. *Probability and statistics for computer science*. John Wiley & Sons, 2011. ISBN 9781118165836.
5. WIKIMEDIA COMMONS. *Standard deviation diagram* [online]. 2007 [cit. 2021-03-22]. Dostupné z: `https://commons.wikimedia.org/wiki/File:Standard_deviation_diagram.svg`.
6. LOPEZ, Pablo Alvarez; BEHRISCH, Michael; BIEKER-WALZ, Laura; ERDMANN, Jakob; FLÖTTERÖD, Yun-Pang; HILBRICH, Robert; LÜCKEN, Leonhard; RUMMEL, Johannes; WAGNER, Peter; WIEßNER, Evamarie. Microscopic Traffic Simulation using SUMO. In: *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. Dostupné také z: `https://elib.dlr.de/124092/`.
7. MALÍK, Josef; SUCHÝ, Ondřej; TVRDÍK, Pavel; VALLA, Tomáš. *BI-AG1 přednáška: Základní definice a pojmy teorie grafů I.* [Online]. 2021 [cit. 2021-03-22]. Dostupné z: `https://courses.fit.cvut.cz/BI-AG1/media/lectures/bi-ag1-p2-handout.pdf`.
8. MALÍK, Josef; SUCHÝ, Ondřej; TVRDÍK, Pavel; VALLA, Tomáš. *BI-AG1 přednáška: Nejkratší cesty v grafech.* [Online]. 2021 [cit. 2021-03-22]. Dostupné z: `https://courses.fit.cvut.cz/BI-AG1/media/lectures/bi-ag1-p12-handout.pdf`.
9. SUCHÝ, Ondřej; VALLA, Tomáš. *BI-AG2 přednáška: Sítě, toky v sítích, Fordův-Fulkersonův algoritmus.* [Online]. 2021 [cit. 2021-04-16]. Dostupné z: `https://courses.fit.cvut.cz/BI-AG2/media/lectures/bi-ag2-p03.pdf`.
10. KLEIN, Philip; PLOTKIN, Serge A.; RAO, Satish. Excluded Minors, Network Decomposition, and Multicommodity Flow. In: *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*. San Diego, California, USA: Association for Computing Machinery, 1993, s. 682–690. STOC '93. ISBN 0897915917. Dostupné z DOI: `10.1145/167088.167261`.

11. SHAHROKHI, Farhad; MATULA, D. W. The Maximum Concurrent Flow Problem. *J. ACM*. 1990, roč. 37, č. 2, s. 318–334. ISSN 0004-5411. Dostupné z DOI: 10.1145/77600.77620.
12. *Obrázek ze simulátoru SUMO* [online]. 2021 [cit. 2021-04-16]. Dostupné z: https://sites.google.com/site/codedwise/sumo-tutorials/controlled-intersection-in-sumo.
13. HORNI, Andreas; NAGEL, Kai; AXHAUSEN, Kay (ed.). *Multi-Agent Transport Simulation MATSim*. London: Ubiquity Press, 2016. Dostupné z DOI: 10.5334/baw.
14. ODERSKY, Martin; SPOON, Lex; VENNERS, Bill. *Programming in Scala: Updated for Scala 2.12*. 3rd. Sunnyvale, CA, USA: Artima Incorporation, 2016. ISBN 0981531687.
15. CHIN, Stephen; REIMERS, Sven. *ScalaFX* [software]. 2021 [cit. 2021-03-15]. Dostupné z: https://www.scalafx.org/.
16. LIGHTBEND. *Play Framework Json library* [software]. 2021 [cit. 2021-03-20]. Dostupné z: https://www.playframework.com/.
17. MICHELIOUDAKIS, Evangelos; SKARLATIDIS, Anastasios. *Optimus: an open-source mathematical optimization library* [software]. 2021 [cit. 2021-04-05]. Dostupné z: https://github.com/vagmcs/Optimus.
18. GUROBI OPTIMIZATION, LLC. *Gurobi Optimizer Reference Manual* [software]. 2021 [cit. 2021-04-03]. Dostupné z: http://www.gurobi.com.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `thesis` ..... adresář se zdrojovou formou práce ve formátu LaTeX
- `simulator`
  - `project` ..... adresář se zdrojovými kódy implementace
  - `exe` ..... adresář se spustitelnou formou implementace
  - `scenarios` ..... adresář s testovacími scénáři pro simulátor
- `BP_Schweika_Patrik_2021.pdf` ..... text práce ve formátu PDF