České vysoké učení technické v Praze

Fakulta jaderná a fyzikálně inženýrská

Katedra matematiky

Decentralizované řízení dopravní signalizace: optimalizace zelené vlny

BAKALÁŘSKÁ PRÁCE

Vypracoval: Kamil Ondrák
Vedoucí práce: Ing. Václav Šmídl, Ph.D.
Konzultant: Dr. Ing. Jan Přikryl, Ph.D.
Rok: 2010

Zadání práce s podpisem děkana.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškerou použitou literaturu.

Praha, 9. července 2010 Kamil Ondrák

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Václavu Šmídlovi, Ph.D. a konzultantovi Dr. Ing. Janu Přikrylovi, Ph.D. za hodnotné rady, neustálou ochotu a veškerý věnovaný čas, který mi v průběhu přípravy mé práce věnovali.

Kamil Ondrák

Název práce:

Decentralizované řízení dopravní signalizace: optimalizace zelené vlny

Autor: Kamil Ondrák

Obor: Inženýrská informatika

Druh práce: Bakalářská práce

Vedoucí práce: Ing. Václav Šmídl, Ph.D.
ÚTIA AV ČR

Konzultant: Dr. Ing. Jan Přikryl, Ph.D.
ÚTIA AV ČR

Abstrakt: Popis práce

Klíčová slova:

Title:

Decentralized control of traffic lights: green wave optimization

Author: Kamil Ondrák

Abstract: Popis práce anglicky

Key words:

Obsah

Úvod

Na mnoha místech silničních komunikací dnes nastává problém s jejich ucpáváním příliš silným automobilovým provozem, přičemž nejčastějším úzkým hrdlem bývají křižovatky. Teoreticky nejjednodušším řešením se jeví přestavba dotčených míst. To je však často také řešení nejnákladnější a v některých místech to ani není možné, například z důvodu nedostatku prostoru. Další možností je omezení počtu vozidel přijíždějících do předmětné oblasti. To sice zlepší průjezdnost v kritickém místě, ale provoz se tím pouze přesouvá jinam, kde tak může vzniknout podobný problém. Jako zajímavá možnost se jeví optimalizace řízení provozu pomocí světelných signalizačních zařízení.

V současné době je většina vytížených křižovatek řízena pomocí poměrně sofistikovaných signálních plánů, které se do jisté míry dokáží přizpůsobovat aktuální dopravní situaci. Tyto plány bývají vypočítány pomocí dlouhodobě vyvíjených metodik (v ČR je to třeba TP81, v USA to popisuje Higway Capacity Manual) na základě změřených dat o hustotě dopravy v daném místě. Pro izolované křižovatky se chovají velmi dobře, nevýhodou toho způsobu řízení však bývá právě izolovanost – každá křižovatka má informace jen o svém nejužším okolí a chybí dynamická koordinace mezi sousedícími křižovatkami.

V oblastech s vysokou intenzitou dopravy, kde změna signálního plánu na jedné světelně řízené křižovatce může mít za následek zhroucení dopravy na křižovatce sousední, se používá jistá forma centralizovaného řízení. Centrálním uzlem řízení je dopravní ústředna, která sbírá údaje z určité oblasti a na základě těchto informací vysílá pokyny do řadičů příslušných křižovatek. Tento postup vede k dobrým výsledkům, ale většímu rozšíření brání vysoká cena instalace těchto systému a také přílišná složitost příslušné optimalizační úlohy pro komplexnější oblasti.

Další možností je centralizované řízení s omezenou autonomií lokálních řadičů. Při tomto způsobu ovládání signalizace zasílá centrála zadaný rámcový řídící plán, zachovává ovšem i lokální adaptivitu – řadiče mohou prodloužit délku určitých fází nebo například při nočním provozu zařazovat jen ty fáze, po kterých je poptávka.

Relativně novým přístupem je decentralizované řízení dopravní signalizace. To funguje na principu multiagentních systémů. Každá křižovatka se pak stává jedním agentem, který jedná s ostatními agenty za účelem vytvoření společné optimální strategie řízení vedoucí ke zlepšení průjezdnosti.

Cílem této práce je seznámit se s tímto decentralizovaným způsobem řízení, navrhnout komunikační strategii, která by pomocí nastavení tak zvaných offsetů mohla vézt ke zlepšení průjezdnosti oblastí, toto řešení implementovat na počítači a prozkoumat jeho důsledky v mikrosimulátoru dopravy Aimsun. Toto vše je prováděno na modelu skutečné oblasti, konkrétně Řevnické ulice v Praze – Zličíně. Jako referenční stav pak slouží řízení dopravní signalizace expertně navrženými signálními plány s pevnou délkou cyklu.

Na začátku práce je představen způsob řízení světelných křižovatek, používané detektory a popis signálních plánů. Následuje popis modelované oblasti a po té sekce o simulátoru Aimsun. Ta představuje některé možnosti, které Aimsun nabízí, ukazuje jaká vstupní data do simulace vstupují a jaké zní vystupují. Součástí je také popis rozhraní Getram Extensions, které se používá pro komunikaci mezi simulátorem a externími programy.

Následující kapitola pak představuje úvod to teorie multiagentních systémů a komunikace v nich.

Ve třetí kapitole je popsána nejprve teoreticky navrhovaná strategie komunikace mezi agenty a pak i představena konkrétní implementace. Ta spočívá v rozšíření stávajícího řešení pro simulace dopravy vyvíjené v Ústavu teorie informace a automatizace (ÚTIA) Akademie věd ČR.

V poslední kapitole se nacházejí výsledky simulací navrženého algoritmu a srovnání stavu v oblasti před a po jeho aplikaci.

Kapitola 1

Popis situace

Aby bylo možné věnovat se simulaci řízení světelné signalizace, je nutné seznámit se s fungováním křižovatek, jejich řadičů a se způsobem, jakým probíhá ovládání signalizačních zařízení. Dále pak bude představena možnost jak pomocí mikrosimulátoru Aimsun modelovat na počítači dopravu ve skutečné oblasti.

1.1 Konstrukce a řízení křižovatky

Křižovatka řízená světelným signalizačním zařízením se skládá z několika prvků. Vedle samotných semaforů je v její blízkosti připojen řadič světelné signalizace, který se stará o ovládání signalizačních prvků. K řadiči pak může být připojeno několik detektorů, poskytujících mu informace o aktuální dopravní situaci.

Detektory se dělí do tří kategorií podle jejich umístění na detektory výzvové, prodlužovací a strategické. Výzvové jsou umístěné na stop čáře, prodlužovací ve určité vzdálenosti před ní (v závislosti na rychlosti vozidel v daném místě) a strategické na výstupu z nájezdních křižovatek, či v případě izolovaných křižovatek na vzdálenějších místech, běžně alespoň 100 metrů před stop čárou. Ne vždy musí být na všech ramenech křižovatky instalovány všechny typy detektorů.

Technicky je nejčastějším řešením detektoru indukční smyčka umístěná ve vozovce. Detektor pracuje na principu změny induktance cívky, která se zjišťuje měřením změny vlastní frekvence oscilačního obvodu, způsobenou přítomností magneticky vodivé látky v jejím okolí. Především na dálnicích se někdy využívá umístění dvou smyček blízko sebe, díky kterým lze získat i podrobnější informace o rychlosti, druhu vozidla a další údaje. Smyčkové detektory jsou léty ověřeným prostředkem pro měření dopravních veličin. Mezi jejich výhody patří nízká pořizovací cena oproti ostatním technologiím, jejich neovlivněnost počasím (především odolnost vůči stavům s nízkou viditelností, jako mlha nebo hustý déšť) a nejlepší přesnost sčítaných dat ze všech běžných používaných technologií [2].

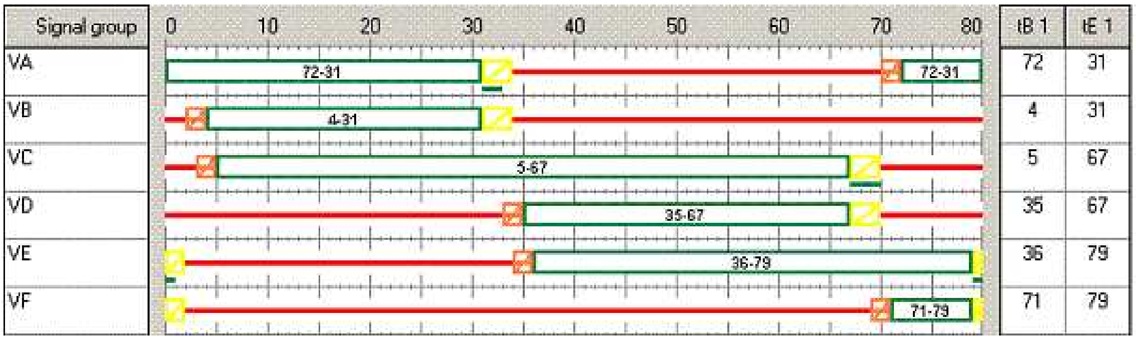
Problémem smyčkových detektorů je náročnost instalace a údržby, které si vyžadují úpravu vozovky a proto i její dočasné vyloučení z provozu. Kvůli umístění detektory trpí mechanickým namáháním, které může způsobit až selhání a nutnost opravy či výměny.

Se smyčkovými detektory se pojí i jistá nepřesnost naměřených údajů. Ta pramení z několika zdrojů. Jedním z nich je doba odezvy detektoru na příjezd a odjezd vozidla ze sledovaného prostoru. Pokud se liší, vzniká nepřesnost v hlášené době obsazenosti. Dále každý detektor po odjezdu vozidla potřebuje jistý čas na to, aby se navrátil do klidového stavu a mohl zaznamenat další vozidlo. Pokud se však během tohoto času dostane nad indukční smyčku další automobil, nemusí být zaznamenán nebo může být považován za součást předchozího. Šum může do výsledků také vnášet například interference od nějakého elektronického zařízení. Tyto potíže komplikují mimo jiné modelování délky front na křižovatkách, které bude ještě zmíněno dále.

Další možností sledování provozu jsou infračervené detektory. Jejich hlavní součástí je kamera, snímající sledovaný prostor v infračervené oblasti. Ve výsledném obrazu pak poměrně snadno rozezná automobily i chodce jakožto zdroje tepelného záření.

Dále existují například video detektory založené na kamerách snímající v oblasti viditelného světla (videodetektory), mikrovlnné nebo ultrazvukové detektory. Tyto však nejsou v současné době v simulované oblasti instalovány.

Smyčková detektory umístěné na Zličíně poskytují dva údaje. Prvním je počet vozidel, která přes detektor během stanovené doby agregace dat (typicky 60, 90 nebo 120 sekund) projela a druhým čas (udávaný jako počet vzorků s odstupem desetina sekundy), po který se během periody detekce ve sledované oblasti vyskytovalo nějaké vozidlo.



Obrázek 1.1: Ukázka tabulky pevného signálního plánu. Vlevo jsou jednotlivé signální skupiny, nahoře časová osa, uprostřed jsou pak zeleně označeny intervaly, ve kterých tyto signální skupiny vysílají znak „volno“.

Řízení křižovatky probíhá pomocí signálních plánů uložených v řadiči nebo dle pokynů z ústředny. Signální plány mohou být pevné nebo proměnné. *Signální plán* se skládá z několika fází, které mají buď pevný sled (mají pevné začátky a jejich délka je v každém cyklu stejná), případně se mohou částečně překrývat (v takovém případě určí přesný okamžik přechodu mezi fázemi řadič podle vyhodnocení údajů z jednotlivých detektorů). *Fáze* jsou složené ze signálních skupin. *Signální skupina* pak značí skupinu světelných signalizačních zařízení, která rozsvěcuje v dané fázi zelenou. Signální plány mají pevnou a jednotnou *délku jednoho cyklu*, kterou označujeme T_c. V předmětné oblasti je to konkrétně 80 sekund. Posledním důležitým pojmem je *offset*. Ten říká o kolik sekund je začátek jednoho cyklu signálního plánu posunut oproti jistému počátečnímu času t_0. Offset je klíčovým parametrem pro tvorbu tak zvané zelené vlny – špatnou volbou offsetů signálních plánu u dvou na sebe navazujících křižovatek můžeme způsobit situaci, kdy vozidla najíždí do následující křižovatky, na níž právě padla červená.

V případě pevných signálních plánů se přepínání signalizovaných znaků řídí předem danou tabulkou, která určuje, kdy která signální skupina rozsvěcí zelenou a jak dlouho má zelená trvat. V případě proměnných signálních plánů může být pro každou skupinu uložena možnost prodloužit interval zelené v případě, že údaje z detektorů oznamují další přijíždějící vozidla v daném směru. Tabulka pevného signálního plánu je na obrázku 1.1.

Obrázek 1.2: Celkový pohled na Řevnickou ulici.

1.2 Oblast Zličína

Ověření možnosti decentralizovaného řízení dopravní signalizace je v této práci provedeno na modelu skutečné oblasti: ulici Řevnické v Praze – Zličíně. V ulici se nachází pět světelných křižovatek situovaných v jedné linii. Pro zjednodušení jsou v této práci simulovány jen dvě severní křižovatky s číselným označením 5.495 a 5.601. První jmenovaná je trojramenná křižovatka ulic Řevnická a Na Radosti, druhá je potom čtyřramenná a je tvořena napojením autobusového terminálu na jedné a obchodního centra Metropole Zličín na druhé straně na Řevnickou. Celá oblast je na obrázku 1.2, dvě simulované křižovatky potom na obrázku 1.3.

Během simulací jsou na obou křižovatkách nastaveny pevné signální plány bez prodlužování fází.

1.3 Simulátor Aimsun

K simulaci dopravní situace se používá softwarový balík GETRAM/AIMSUN od společnosti TSS (Transport Simulation Systems) ve verzi 4.2.16. Jádrem celého ná-

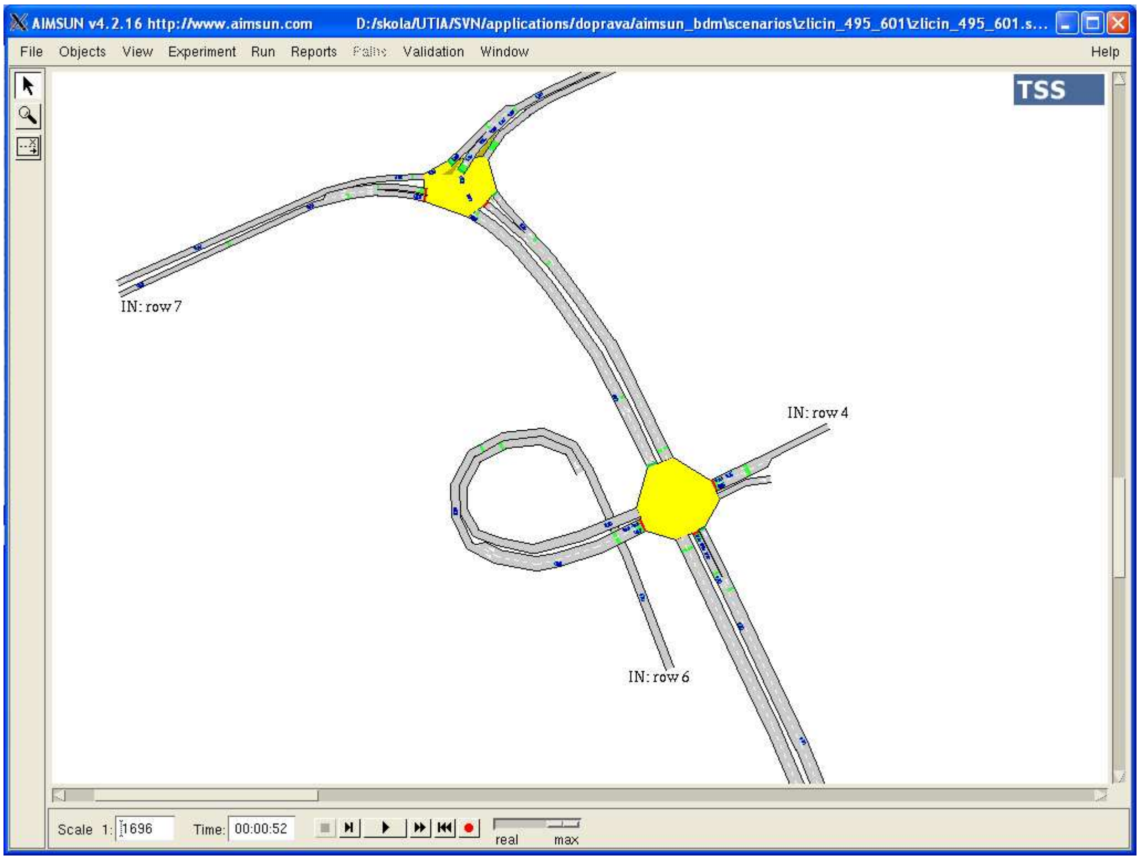
Obrázek 1.3: Pohled na dvě simulované křižovatky, vlevo 5.495, vpravo 5.601.

stroje je program Aimsun, který slouží k samotné simulaci.

Aimsun (Advanced Interactive Microscopic Simulator for Urban and Non-Urban Networks, [8]) je mikrosimulátorem dopravy[1], to znamená, že modeluje polohu jednotlivých vozidel diskrétně během celé doby simulace. Každé vozidlo se řídí určitým modelem chování. Lze nastavit různé styly předjíždění, prudkost brzdění a rozjezdů, ale třeba i ochotu čekat. Je možné simulovat různé druhy vozidel, od osobních přes nákladní auta až po autobusy a hromadnou dopravu vůbec. Vedle vozidel Aimsun samozřejmě nabízí simulaci většiny objektů, které se vyskytují v dopravních sítích: světelných signalizační zařízení, detektorů, atd.

Vstup pro simulátor se skládá ze scénáře a parametrů simulace. Scénář obsahuje popis dopravní sítě, údaje o dopravní zátěži, programy řízení dopravy a plány veřejné dopravy. Popis dopravní sítě představuje geometrickou reprezentaci zkoumané oblasti, tedy rozměry a tvar jednotlivých silničních pruhů a křižovatek, umístění dopravní signalizace, detektorů, zastávek hromadné dopravy a podobně. Dopravní zátěž může být zadaná dvěma způsoby. Jedna možnost je pomocí intenzity dopravy na vstupech, poměrů odbočení na křižovatkách a počátečního stavu sítě, druhá pomocí takzvané O-D matice, která zachycuje počet uskutečněných cest mezi dvo-

[1] Současná verze Aimsun 6 dovoluje již mikro, meso i makrosimulaci



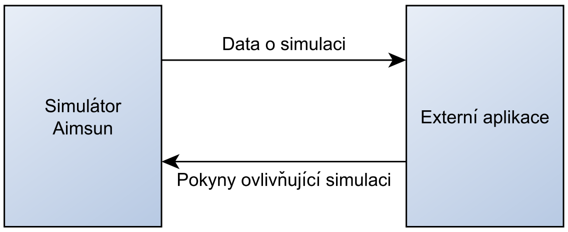
Obrázek 1.4: Simulovaná oblast nahraná v Aimsunu

jicemi vstupních a výstupních bodů. Programy řízení dopravy zahrnují popisy fází a jejich trvání pro řízené křižovatky, přednosti v jízdě pro křižovatky neřízené. Konečně plány veřejné dopravy se skládají z popisů tras, zastávek a jízdních řádů linek hromadné dopravy v simulované oblasti. Parametry simulace pak představují pevná čísla popisující experiment (jako doba simulace) a proměnné hodnoty určené pro kalibraci modelu.

Výstupem za simulace je spojitá grafická reprezentace zkoumané oblasti, statistická data o provozu (tok vozidel, počty zastavení, průměrná rychlost, atd.) a údaje sesbírané z detektorů. Výsledky mohou být vykresleny do grafů, nebo uloženy v textové podobě do souborů či do databáze pro další zpracování.

1.3.1 Getram Extensions

Aimsun má aplikační rozhraní (API), které umožňuje tvorbu rozšiřujících modulů nazvaných Getram Extensions. Aimsun pomocí tohoto rozhraní poskytuje v reálném



Obrázek 1.5: Aimsun a externí aplikace.

čase data ze simulace a naopak přijímá data pro její ovlivňování, jak je naznačeno na obrázku 1.5. Rozšíření se píší buď jako DLL knihovny napsané v C/C++ nebo ve formě skriptů v Pythonu.

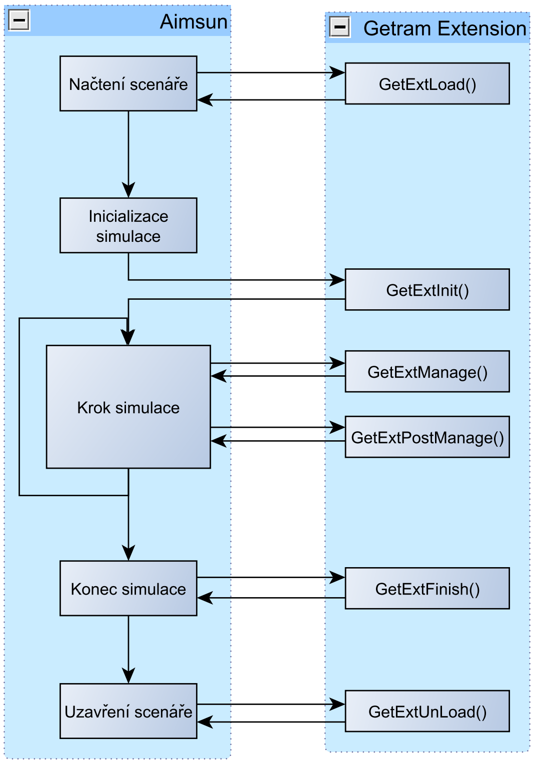
Komunikace mezi rozšířením (Getram Extension) a simulátorem Aimsun probíhá pomocí šesti funkci:

1. *GetExtLoad()* je volána v okamžiku nahrání rozšíření Aimsunem.
2. *GetExtInit()* je volána na začátku simulace.
3. *GetExtManage()* se volá na začátku každého kroku simulace. Slouží k získání aktuálních údajů z detektorů, dat o vozidlech a dalších údajů. Také umožňuje naopak ovládání aktivních prvků v simulované oblasti.
4. *GetExtPostManage()* je funkce podobná předchozí, jen tato je volána na konci simulačního kroku.
5. *GetExtFinish()* se volá na konci simulace a v ní rozšíření dokončuje všechny operace, které si to ještě žádají.
6. *GetExtUnLoad()* je zavolána v okamžiku, kdy Aimsun ukončuje komunikaci s rozšířením.

Průběh komunikace mezi Aimsunem a rozšířením je vidět na obrázku (1.6)

1.4 Emulátor řadiče ELS3

Pro simulace funkcí řadiče křižovatky se používá emulátor ELS3 vyvinutý firmou Eltodo dopravní systémy s.r.o. Ten z reálného řadiče obsahuje algoritmus řízení.



Obrázek 1.6: Podrobný nákres komunikace mezi Aimsunem a Getram Extension.

Ovládání HW periferií (modul detektorů, spínačů signálních skupin, a podobně) jsou nahrazeny výše popsaným simulátorem Aimsun.

V INI souboru emulátoru řadiče jsou uloženy parametry dopravního řešení použitého v simulované oblasti. Dopravní řešení je určeno parametry křižovatky (její konstrukce, umístění detektorů, fáze, signální skupiny, ...) a dopravními vztahy mezi těmito parametry (signální plány, dynamické řízení, ...). Návrh dopravního řešení je dílem dopravního inženýra a jeho struktura přesahuje rámec této práce.

ELS3 má pro komunikaci s okolím vlastní API. V každém simulačním kroku přes něj obdrží z Aimsunu stavy svých detektorů. Od knihovny BDM (která zde nahrazuje modul dopravní ústředny a o níž se bude řeč později) řadič pak přijímá data pro ovlivnění řízení. Výstupem řadiče je obraz barevné kombinace signálních skupin. Ten se zasílá do Aimsunu na konci každého simulačního kroku.

Kapitola 2

Agentní systémy

Decentralizované řízení staví na principu multiagentních systémů. Ten spočívá v nasazení několika autonomních jednotek (agentů) do dané oblasti, kteří se snaží společně řídit jistý systém, o kterým mají úplné, nebo alespoň částečné informace.

2.1 Inteligentní agent

Pokud mluvíme o agentních systémech, je nutné předem si vyjasnit základní pojem agent. Bohužel neexistuje přesná obecně přijímaná definice. Pro účely této práce můžeme použít například tu, kterou používá Michael Wooldrige v [9, kap. 1]: „*Agent* je počítačový program, který je *umístěn* v nějakém *prostředí* a který je schopen *autonomního jednání* v tomto prostředí za účelem dosažení určených cílů.“ Autonomií se zde myslí schopnost samostatného jednání bez zásahu člověka nebo jiných programů. Tato definice se zatím vyhýbá pojmu *inteligentní*, tomu se budeme věnovat později.

Agent z definice má tedy určitý druh čidel, kterými může získat některé údaje o svém okolí, omezený seznam akcí, které může provádět a jistý algoritmus, který určuje jakou z akcí (pokud vůbec nějakou) v aktuálním čase za zjištěných podmínek provést.

Komplexnost rozhodovacího procesu závisí mimo jiné i na prostředí. To můžeme klasifikovat z mnoha pohledů. Zda lze či nelze získat úplné informace o jeho současném stavu, jestli je deterministické nebo nedeterministické, tedy jestli stejná akce povede vždy ke stejnému konečnému stavu. Dále je možné rozlišit mezi statickým a dynamickým prostředí. Ve statickém prostředí jsou veškeré změny stavu dílem pouze agenta, zatímco dynamické okolí se mění ať agent jedná nebo ne. Podobných rozdělení existuje mnoho, ale toto pro představu postačí. Nejlépe řiditelné je samozřejmě prostředí, které je plně popsatelné, deterministické a statické, bohužel řízení dopravy je případem přesně opačným.

Naprosto typickým příkladem jednoduchého agenta umístěného v prostředí je termostat ovládající topení za účelem udržení stále teploty v místnosti. Tento agent má jedno čidlo, kterým určuje jestli je teplota nižší než nastavená požadovaná, a dvě možné akce: zapnutí a vypnutí vytápění. Rozhodovací proces se pak skládá ze dvou pravidel:

$$\text{nízká teplota} \longrightarrow \text{zapnout vytápění}$$
$$\text{správná teplota} \longrightarrow \text{vypnout vytápění}$$

Takto postavený program se dá považovat za agenta, ale pravděpodobně jen málokdo by ho označil za agenta inteligentního. Problém je už samotná definice inteligence. Proto se omezíme na to, že inteligentní agent je takový, který dokáže flexibilně reagovat. Flexibilita znamená tři vlastnosti:

reaktivita – inteligentní agent vnímá své okolí a dle takto získaných údajů reaguje na jeho změny tak, aby splnil své nastavené cíle;

proaktivita – inteligentní agent je schopný předvídat chování systému a na základě těchto předpovědí aktivně jednat tak, aby splnil své nastavené cíle;

sociální schopnosti – inteligentní agenti na sebe mohou vzájemně působit tak, aby splnili své nastavené cíle.

Požadavky na vysokou reaktivitu a proaktivitu jdou proti sobě. Je třeba najít vyvážený poměr mezi oběma. Extrémně reaktivní agent totiž na základě neustále přijímaných podmětů stále koná nějakou akci. Ovšem tímto způsobem se může dostat do stavu, kdy podměty způsobují střídavé zaměření agenta na různé cíle, kterou nejsou slučitelné a tím pádem ani jeden z cílů nemůže být nikdy uskutečněn. Na druhou stranu příliš proaktivní agent se zaměří na jeden cíl a kvůli tomu ignoruje, že počáteční podmínky, které ho k plnění cíle dovedly, už nejsou platné. Najít proto tu správnou míru je obtížná úloha.

Mezi sociální schopnosti nepatří jen rutinní výměna informací, kterou dnešní počítače provádějí prakticky neustále, ale především schopnost vyjednávat a spolupracovat při plnění společných nebo individuálních cílů. Mezi běžné akce tak patří navrhování změn spolu s přínosem, které by tato změna přinesla, jejich zvažování a následné přijetí, odmítnutí či sestavení protinávrhu vedoucího ke kompromisu.

2.2 Komunikace mezi agenty

Agenti mezi kterými probíhá komunikace musí být schopní zastávat v dialogu aktivní, pasivní nebo případně obě role. Aby toto bylo možné, budeme předpokládat, že agenti mohou přijímat a zasílat zprávy jistou komunikační sítí.

Michale N. Huhns a Larry M. Stephens v [9, kap. 2] uvádějí, že zprávy zasílané agenty lze dělit na dva základní typy: *prohlášení* a *dotazy*. Každý agent by měl být schopen přijímat informace. Agent zastávající pasivní roli v dialogu je navíc schopen odpovídat na otázky, to znamená umět přijmout a zpracovat dotaz a odpovědět na něj odesílateli nějakým prohlášením. Aktivní agent je pak takový, který sestavuje a odesílá dotazy a prohlášení přijímá. Díky tomu aktivní agenti mohou mít jistou formu kontroly nad pasivními.

V případě rovnocenných agentů je nutné, aby všichni zvládali jak aktivní, tak pasivní roli v dialogu.

Pro umožnění komunikace mezi agenty se definuje společný komunikační protokol. Ten může mít binární či n-ární. Binární umožňuje komunikaci vždy jen mezi dvěma agenty, n-ární dovoluje jednomu agentovi rozeslat zprávu několika příjemcům zároveň. Protokol je určen datovou strukturou skládající se z pěti položek

1. odesílatel

2. příjemce (příjemci)

3. jazyk v protokolu

4. kódovací a dekódovací funkce

5. akce, která by měla být vykonána příjemcem

Kapitola 3

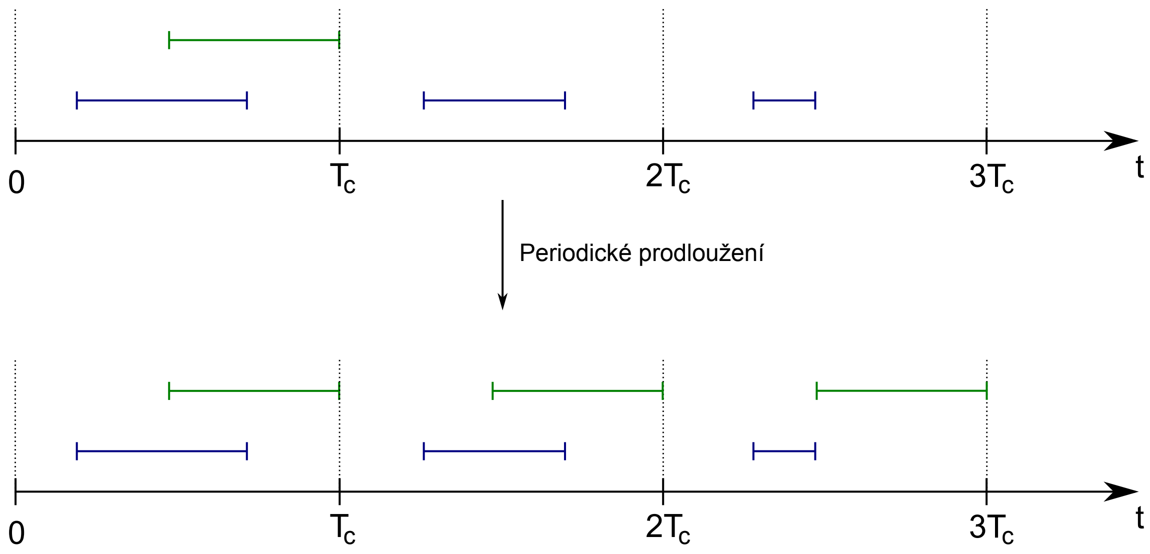
Algoritmus řízení

Úkolem v této práci bylo navrhnout algoritmus vyjednávání mezi jednotlivými křižovatkami tak, aby koordinovanou změnou svých offsetů vytvořili zelenou vlnu a tedy aby projelo co nejvíce vozidel bez zbytečného zastavování. V této kapitole bude takovýto algoritmus představen a následně bude popsána jeho implementace do již fungujícího simulačního prostředí.

3.1 Návrh algoritmu

Navržená strategie řízení se skládá z agentů, kteří jsou přiřazeni k jednotlivým křižovatkám. Každý agent může ovládat offset signálního plánu u svého řadiče křižovatky, od něj naopak získává údaje z detektorů popisující aktuální stav provozu. Navíc sousedící agenti mezi sebou mohou komunikovat. Toho využívají především k tomu, aby zjistili nastavení offsetu na vedlejších křižovatkách, respektive přesněji aby věděli kdy mohou od sousedů očekávat příjezd vozidel. Ze získaných informací pak agent usoudí, jestli by změna sousedova offsetu nemohla přinést zlepšení situace a pokud ano, pokusí se tuto změnu vyjednat.

Jedním z nosných prvků návrhu je funkce, která ohodnotí konkrétní nastavení offsetu a umožní tak tedy porovnat různé jeho hodnoty a vybrat tu možná nejlepší. Za tuto míru kvality byl zvolen odhad počtu aut, která projedou křižovatkou bez zastavení. Do výpočtu jsou zahrnuta jen vozidla ze směrů od jiných řízených křižovatek.



Obrázek 3.1: Periodické prodloužení intervalu svícení zelené: Modře jsou označeny intervaly, kdy přijíždějí vozidla, zeleně interval, kdy dle signálního plánu svítí zelená (se započítáním offsetu). Horní obrázek potom představuje stav před prodloužením, dolní po něm.

Pro výpočet hodnocení (v programu nazývaném `rating`) je třeba pro každý jízdní pruh na vjezdech do křižovatky znát délku fronty a očekávané časy příjezdů vozidel od sousední křižovatky. Délka fronty se řeší později, příjezdy agent pak získává přímo od souseda. Ten je počítá podle vztahu

$$t_\mathrm{z} = t_\mathrm{zz} + \frac{d}{v_\mathrm{P}} + \mathit{offset} \tag{3.1}$$

a

$$t_\mathrm{k} = t_\mathrm{z} + t_\mathrm{dz} \;, \tag{3.2}$$

kde t_z je čas začátku příjezdu aut a t_k čas konce. Dále pak t_zz je čas začátku svícení zelené, d vzdálenost ke křižovatce, která žádá o časy příjezdů, v_P průměrná rychlost vozidel, *offset* nastavený offset a t_dz délka svícení zelné. K těmto dvěma vypočítaným hodnotám se ještě přidává předpokládaný počet aut. Jeho odhad je získán z hustoty provozu v minulém cyklu. Křižovatka zasílající odhady takovýchto trojic údajů předává několik – podle toho, kolik fází rozsvěcí zelenou ve směru připouštějícím jízdu k sousedovi žádajícímu o odhady.

Hodnocení je pak tedy počítáno po jednotlivých jízdních pruzích. Pro každý je nutné provést periodické prodloužení času svícení zelené odpovídající signální skupiny tak, aby v každém okamžiku intervalu od času 0 po čas posledního předpovězeného příjezdu vozidel bylo jasné, jaký znak bude v tomto jízdním pruhu signalizován. Tento interval je dále značen T. Princip periodcikého prodloužení je představen na obrázku 3.1.

Interval T poté rozdělíme na posloupnost intervalů $(T_i)_{i=1}^k$. T_i jsou intervaly typu $\langle a_i; b_i \rangle$, kde $b_i = a_{i+1}\ \forall i \in \{1, \dots, k-1\}$, a_1 a b_k jsou krajní body intervalu T a a_i jsou chronologicky řazené všechny *významné body*, Těmi se rozumí body, ve kterých se mění signalizovaný znak nebo ve kterých začíná či končí příjezd vozidel od souseda. Každý interval T_i lze tím pádem rozdělit na čtyři druhy, podle toho jestli v něm přijíždějí nebo nepřijíždějí vozidla a jestli svítí nebo nesvítí zelená. Řekneme, že T_i je

(i) typu 1 $\iff$ v jeho průběhu přijíždějí auta do fronty a svítí zelená

(ii) typu 2 $\iff$ v jeho průběhu přijíždějí auta do fronty a nesvítí zelená

(iii) typu 3 $\iff$ v jeho průběhu nepřijíždějí auta do fronty a svítí zelená

(iv) typu 4 $\iff$ v jeho průběhu nepřijíždějí auta do fronty a nesvítí zelená

Pomocí tohoto dělení pak počítáme délku virtuální fronty na konci intervalů T_i následujícím způsobem:

$$Q_V^{(i)} = Q_V^{(i-1)} + u(T_i)\,, \tag{3.3}$$

kde $Q_V^{(i)}$ je délka virtuální fronty na konci i-tého intervalu. Funkci $u(T_i)$ je definována takto: $u : \langle a; b \rangle \to \mathbb{R}$

$$u(T_i) = \begin{cases} |T_i| \cdot (c_o - \rho_i) & \text{pokud } T_i \text{ je typu 1} \\ |T_i| \cdot \rho_i & \text{pokud } T_i \text{ je typu 2} \\ -|T_i| \cdot c_o & \text{pokud } T_i \text{ je typu 3} \\ 0 & \text{pokud } T_i \text{ je typu 4} \end{cases}. \tag{3.4}$$

kde $|T_i|$ je délka intervalu T_i, ρ_i hustota přijíždějících vozidel během intervalu T_i a c_o je empiricky zjištěná konstanta – počet vozidel, která za sekundu opustí křižovatku (tedy vlastně hustota odjíždějících vozidel).

Definice funkce $u(T_i)$ jak je zapsána vztahem (3.4) není úplná. Ještě je nutné doplnit omezení

$$(\forall i \in \{1, \dots, k\})\,(T_i \text{ je interval typu 3}) : \quad -u(T_i) < Q_V^{(i-1)} \tag{3.5}$$

Tato podmínka říká, že pokud z fronty pouze odjíždějí vozidla, nesmí se stát, aby volná kapacita po vyprázdnění fronta byla započítána do hodnocení. V tuto chvíli totiž nepřijíždějí žádná vozidla, která by křižovatkou projela bez zastavování.

Pokud vyjde v nějakém okamžiku $Q_V^{(i)}$ záporné, představuje (odhadovaný) počet aut, která mohou v tomto intervalu křižovatkou projet bez zastavení a tato hodnota se tedy odečte od hodnocení dané křižovatky (a tím se hodnocení zvýší).

Vyvstává otázka jak zjistit aktuální délky front na jednotlivých jízdních pruzích jen pomocí údajů z detektorů, které jsou k dispozici. Ukazuje se, že toto není triviální problém a jeho řešení přesahuje rámec této práce. Z toho důvodu nejsou v programu délky front odhadovány tak, jak by to bylo nutné v reálné situaci, ale používají se hodnoty, které lze získat ze simulátoru Aimsun pomocí VGS API. Podrobnější informace o modelování délky fronty představuje například [4]. VGS API fronty počítá tím způsobem, že projde celý příslušný jízdní pruh a v něm do fronty započítá ta vozidla, která jedou nižší než stanovenou hraniční rychlostí.

Při samotném vyjednávání pak mají agenti dvě role, jedna z nich je zvaná *pasivní*, druhá je pak tedy *aktivní*. Pasivní agent má pevně nastavený offset a jen reaguje na pokyny aktivního. Pokyny tvoří buď žádost o očekávané časy příjezdů nebo návrh na změnu offsetu. Na první z nich agent vždy odpovídá zasláním požadovaných údajů, u druhého pak zvažuje, jestli by změna v celkovém součtu přinesla zlepšení `ratingu`. Z tohoto popisu už pak vyplývá role aktivního agenta. Ten stejně pracuje se žádostmi o časy příjezdů, navíc ale aktivně mění svůj offset a pokouší se vyjednat změnu u sousedů.

Vyjednávací cyklus pak probíhá v několika krocích. Nejprve si všichni agenti vyžádají očekávané příjezdy vozidel na základě offsetů nastavených v minulém cyklu. S přihlédnutím k těmto očekáváním pak aktivní agenti spočítají `rating` svého nastavení offsetu a pokusí se zjistit, jestli by nějaká změna vlastního offsetu nevedla ke zlepšení hodnocení.

Hledání nejlepšího vlastního offsetu probíhá ve třech krocích. Nejprve se porovná aktuální offset, offset zvýšený o 8 sekund a offset snížený o 8 sekund. Vybere se nejlépe hodnocený z těchto možností a pokračuje se s ním stejným způsobem, jen další uvažovaná změna je ± 4 sekundy. V posledním kroku je pak otestován posun o ± 2 sekundy.

Když aktivní agenti naleznou své nejlepší offsety, rozešle se všem účastníkům zpráva o nalezení stabilního stavu, která obsahuje nové hodnoty očekávaných příjezdů vozidel. Nyní aktivní agenti vyzkouší, jestli by k dalšímu zlepšení nevedla změna offsetu u některého z jejich sousedů. Podobným způsobem jako při hledání vlastního nejlepšího offsetu zkusí odhadnout změnu `ratingu` při posunu sousedova offsetu o ± 4 a nejlepší ze tří možností se zašle jako žádost o posun offsetu sousedovi spolu se změnou `ratingu`, kterou by přinesla.

Každý pasivní agent poté sesbírá všechny návrhy a otestuje, který z nich má největší součet změny ratingu u něj samotného a změny u navrhovatele a ten potom přijme za vlastní. Pokud by všechny návrhy přinesly zápornou změnu, jsou zamítnuty a žádná změna nenastává. V každém případě jsou pak rozeslány informace o novém stabilním stavu a s nimi opět očekávané příjezdy.

Poté se ještě zasílání žádostí opakuje, jen s posunem o ± 2 a ± 1 sekundu.

Tímto každá křižovatka nalezne svůj konečný offset. Problém nastává při zaslání tohoto offsetu do řadiče křižovatky. Od toho není garantována okamžitá akce, způsob jakým žádaného offsetu dosáhne je jen v jeho režii a než se tak stane může trvat i několik cyklů. Z tohoto důvodu se vypočtený offset neposílá hned po nalezení, ale agent vždy v pěti cyklech napočítá optimální offset, z těchto pěti hodnot spočítá průměr a až ten se následně předá řadiči pro zpracování. Tím je také snížena reaktivnost agenta a zamezí se případným přehnaným reakcím na chvilkové změny poptávky, kterým stejně není možné se přizpůsobit.

3.2 Použité knihovny

Pro usnadnění práce a také z důvodu lepšího začlenění do současného řešení se v programu používají tři volně dostupné knihovny: `IT++`, zjednodušující práci s vektory, maticemi a poli, `BDM` (Bayesian Decision Making), která obsahuje užitečné nástroje pro práci s popisy k vektorům a obsahuje také nástroj pro ukládání průběžných hodnoot z experimentu a `libconfig`, sloužící především pro práci s konfiguračními soubory. První dvě knihovny jsou distribuovány pod GPL licencí, třetí pak pod licencí LGPL.

3.2.1 IT++

IT++ je knihovna pro C++, která obsahuje třídy a funkce pro provádění některých matematických operací, zpracování signálů a další. Pro účely této práce jsou zajímavé právě funkce matematické.

Pro zacházení s vektory jsou v knihovně mimo jiné zavedeny typy `vec` a `ivec`. První jmenovaný je vektor obsahující prvky typu `double`, tedy čísla s desetinou čárkou, druhý je pak složen z prvků `int`, čili čísel celých. Práce s vektory je velmi intuitivní, navíc lze často používat syntaxi podobnou jako v programu MATLAB.

Vektor můžeme nadefinovat jedním z těchto způsobů

```
vec my_vector;
vec my_vector(10);
```

přičemž první z nich pro vektor nealokuje paměť. To je pak nutné udělat funkcí setsize(). Následně je pak možné uložit do vektoru jednotlivé prvky a to například jedním z následujících příkazů.

```
vec  a = "0 0.7 5 9.3";          // tedy a = [0 0.7 5 9.3]
ivec b = "0:5";                  // tedy b = [0 1 2 3 4 5]
vec  c = "3:2.5:13";             // tedy c = [3 5.5 8 10.5 13]
ivec d = "1:3:5,0:2:4";          // tedy d = [1 3 5 0 2 4]
vec e("1.2,3.4,5.6");            // tedy e = [1.2 3.4 5.6]
```

Navíc lze i-tému prvku vektoru `a` přistupovat pomocí a(i) nebo a[i].

Díky přetížení operátorů lze s vektory přímo provádět běžné matematické operace, jako jsou:

```
a+b     // součet vektorů
a+5     // přičtení čísla 5 ke všem prvkům vektoru
a*b     // skalární součin vektorů
a*9     // vynásobení všech prvků vektoru číslem 9
```

a tak podobně.

Práce s maticemi pak funguje podle obdobných pravidel.

Dále se v knihovně nachází třída `Array`. Díky ní je možné vytvářet pole prvků libovolného typu (včetně výše popsaných vektorů) a provádět s nimi mnohé operace. Příkladem užití právě pro pole vektorů je například následující kód:

```
Array<vec> pole_vektoru(2);
my_vec_array(0) = "2 4 5 0 3";;
my_vec_array(1) = "0.1 0.2 0.3 0.4 0.3 0.2 0.1";
```

U takto definovaného pole pak lze pak intuitivně přistupovat k jeho prvkům pomocí pole_vektoru(i) a navíc provádět operace jako jsou posuny prvků, výběr podmnožiny prvků nebo třeba prostředního prvku a mnohé další.

3.2.2 BDM

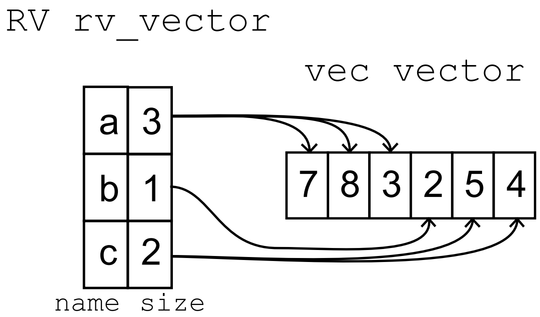
Knihovna `BDM` (Bayesian Decision Making) se zabývá, jak název napovídá, bayesovským rozhodováním. Ovšem v této práci jsou z ní přímo použity jen některé třídy, jmenovitě `UI`, `RV`, `datalink` a `logger`. BDM navíc definujeme třídu `Participant`, která je základem rozhodovacích procesů a od které se pak odvozuje agent pro řízení dopravy.

Třída `UI` (User Info) slouží pro ukládání a čtení libovolných uživatelských dat. Zde se používá pro načtení konfigurace pro simulátor a pro jednotlivé agenty a dále jako formát pro ukládání a posílání zpráv mezi agenty.

Účelem třídy `RV` je poskytovat popisné informace k datovému vektoru. Proměnná typu `RV` se skládá z jedné nebo několika položek. Každá taková položka má svoje jméno (`name`) a délku – počet prvků, které tomuto jménu odpovídají. Součet délek všech takovýchto položek se potom musí rovnat délce vektoru, který je danou proměnnou typu `RV` popisován. Fungování je ilustrováno na obrázku 3.2.

Pro kopírování dat mezi vektory existuje třída `datalink`. Vytváří spojení mezi dvěma datovými vektory na základě shodně pojmenovaných prvků v k nim příslušných popisných vektorech. Po propojení vektoru s podvektorem pak funkce datalinku umožňují snadné kopírování dat oběma směry.

Na závěr `Logger` je třída určená pro ukládání dat z programu. Tvoří abstraktní vrstvu mezi programem a samotným zápisem dat. Při použití se jen na začátku na-



Obrázek 3.2: Vektor typu `RV` (vlevo) fungující jako popis k vektoru typu `vec` (vpravo). Každá položka z `rv_vector` má v levém sloupci své jméno (`name`) a v pravém velikost (`size`). V tomto případě se tedy podvektor `c` rovná vektoru [5 4]

staví v jaké formě chceme získat výsledné údaje (např. uložit do paměti, do souboru, do databáze, atd.) a dále třídu používáme bez ohledu na tuto volbu. Je tím zajištěna flexibilita pro případ, že se změní požadavky na výstupy z programu.

3.2.3 Libconfig

Konfigurační parametry pro simulaci se ukládají do souboru ve formátu, který zavádí knihovna `libconfig`. Jde o textový formát, který je stručnější a pro člověka lépe čitelný, než XML.

Podporovaný formát souborů představuje hierarchickou strukturu. *Konfigurace* se skládá ze skupiny *nastavení*, která přiřazuje *jménům hodnoty*. Hodnotou může být *skalár*, *pole*, *skupina* nebo *seznam*. Skupiny nastavení je možné do sebe dále vnořovat, čímž knihovna nabízí velmi flexibilní a přitom stále přehledný způsob pro ukládání konfiguračních souborů.

3.3 VGS API

Velmi důležitým úkolem při simulaci reálné oblasti v mikrosimulátoru Aimsun je vložení reálných vstupních intenzit dopravy do zkoumaného modelu. Běžným postupem je ruční sčítání vozidel v dané oblasti, zpravidla v hodinovém rastru. Takto získaná data je možné vložit do simulátoru Aimsun jako hodinové zátěže, ovšem toto je třeba provést také ručně.

Přesnější možností je získat intenzity provozu z dat získaných přímo z dopravních detektorů instalovaných v předmětné oblasti. To však znamená ruční zadání stovek údajů do simulátoru. Právě z tohoto důvodu vzniklo VGS API. To se stará o nastartování Aimsunu a vpouštění vozidel do oblasti. Vjezdy vozidel se při tom řídí údaji v externím souboru, obsahujícím intenzity na jednotlivých ramenech. Uživatel pak jen zadá jméno a umístění tohoto souboru a vše ostatní se děje automaticky. Navíc lze volit mezi různými rozdělení pravděpodobnosti vjezdů, přičemž nejčastěji používané je rovnoměrné či Poissonovo rozdělení.

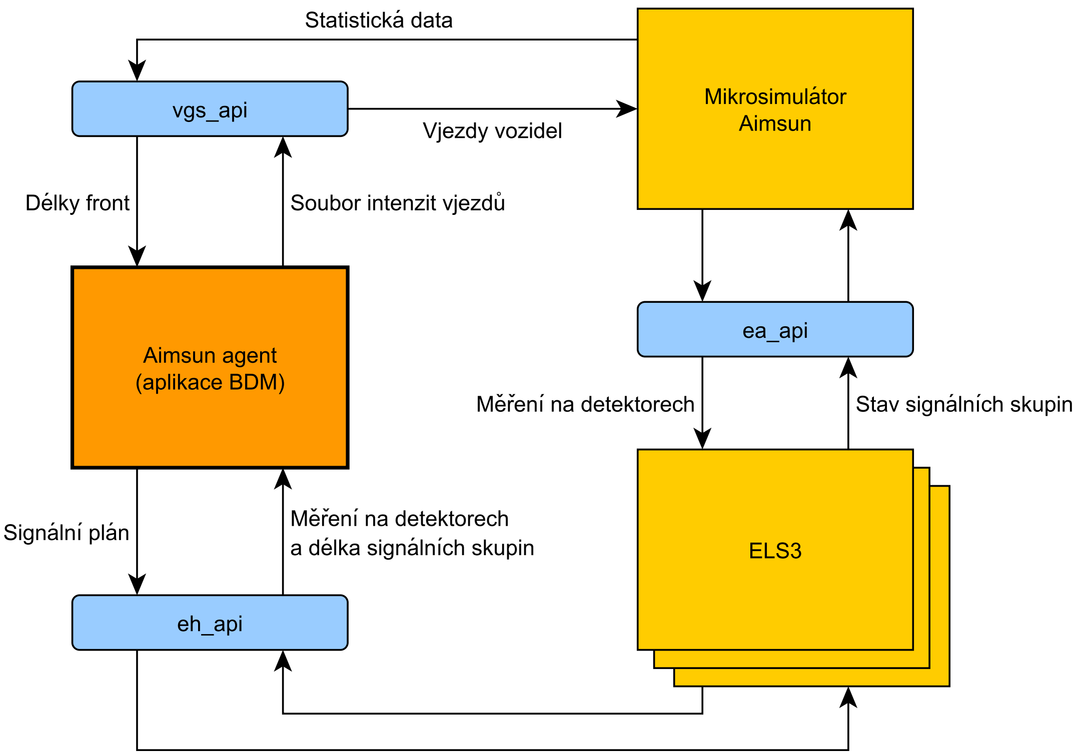
Vedle práci se vstupní daty pro simulaci se VGS API stará i o zpracování dat výstupních. Aimsun sice zvládá export výsledků simulací do textových souborů a obsahuje i nástroje pro jejich vizualizaci, neumožňuje přímo ale některé důležité věci jako je například porovnání výsledků ze dvou různých scénářů. VGS proto v průběhu celého experimentu ukládá údaje o sledované oblasti a to jak pro jednotlivé sekce, tak pro celý systém. Ve výsledku uživatel získává informace o počtu zastavení vozidel, o jeho zpoždění, průměrné rychlosti, době jízdy a době stání, o dopravím toku a hustotě dopravy na jednotlivých segmentech či o délkách front vozidel. Údaje o délkách front jsou specialitou VGS, Aimsun tento údaj pro jednotlivé jízdní pruhy přímo neposkytuje a VGS má proto implementován vlastní algoritmus pro jejich sčítání.

Implementací VGS API je DLL knihovna napsaná v jazyce C pro 32 a 64 bitové systémy Windows XP a výše. Navíc je součástí VGStoolbox, sada skriptů pro zpracování výstupních dat v programu Matlab.

3.4 Struktura programu

Fungování simulačního prostředí tak jak bylo navrženo v ÚTIA je na obrázku (3.3). Skládá se ze tří hlavních bloků, mikrosimulátoru Aimsun, emulátorů řadičů ELS3 a z tzv. Aimsun agenta.

Aimsun agent představuje řídící prvek celého systému. Přes `vgs_api` spouští Aimsun a stejným způsobem od něj průběžně přijímá hodnoty sledovaných dopravních veličin. Dále prostřednictvím `eh_api` získává údaje z řadičů křižovatek. Na základě



Obrázek 3.3: Komunikace jednotlivých komponent simulačního prostředí

všech sesbíraných údajů pak sestavuje pokyny pro změny v signálních plánech a opět pomocí `e_api` tyto pokyny zasílá řadičům.

Na závěr komunikace mezi mikrosimulátorem a emulátory řadičů, které obsahuje údaje z detektorů v jednom směru a ze stavů signálních skupin ve druhém, probíhá přes `ea_api`.

3.5 Program main_loop.exe

Simulace je obsluhována z programu *main_loop.exe*. Ten se stará o načtení konfigurace, spuštění simulátoru Aimsun a zajišťuje komunikačního prostředníka mezi jednotlivými agenty.

Konfigurační soubor obsahuje několik prvků. Seznam `list`) `agents` se skládá z popisu jednotlivých agentů. Především je definována třída, jejíž instance agenta představuje a unikátní jméno agenta (`name`). Další proměnné jsou pak záleží na použité třídě pro konstrukci agenta, každá může mít své specifické požadavky. Skupina (`group`) `logger` určuje, jak název napovídá, třídu použitou pro logování údajů během simulace. Skupina `system` obsahuje parametry pro simulátor a konfigurační soubor uzavírají ostatní užitečné proměnné nespadající do výše uvedených kategorií.

Komunikaci program zajišťuje obsluhou fronty zpráv a ta vlastně definuje základ použitého komunikačního protokolu. Zprávy jsou ve formátu knihovny `libconfig` a obsahují několik povinných a libovolný počet volitelných složek. Nezbytnými údaji jsou řetězec `to`, obsahující příjemce zprávy a řetězec `what`, který značí předmět zpráva. Další rozšíření zpráv je pak na potřebách použitých agentů.

Program se dá rozdělit na samostatné bloky tak, jak je zobrazeno na obrázku 3.4. Z tohoto nákresu bude vycházet následující popis.

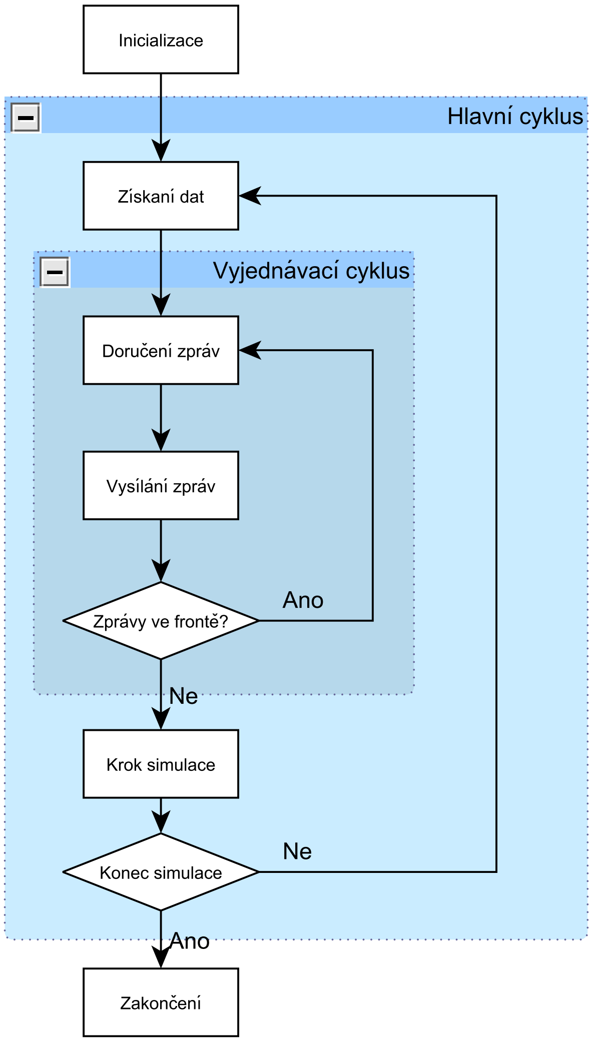
Během *inicializace* je načten konfigurační soubor, jehož jméno se předává jako jediný parametru při spouštění `main_loop.exe`. Po načtení jsou parametry ze skupiny `system` předány VGS API, které se postará o spuštění Aimsunu s těmito parametry. Jde především o intenzitu provozu na vstupech do dopravní sítě a délku simulace.

Následně je dle konfigurace vytvořeno pole ukazatelů na agenty `Ags`. Při konstrukci jednotlivých agentů je volána jejich funkce `from_setting()` načítající konkrétní parametry každého agenta.

Poté se vytváří instance třídy `logger` a její provázání s agenty. V tu chvíli se pouští i funkce `ds_register()` umožňující aktualizaci datalinků pro spojení mezi agenty a simulátorem. Na závěr tohoto bloku programu je provedena inicializace fronty zpráv a zaregistrování vektorů které se budou v průběhu simulace logovat.

Dále se pokračuje ústředním for cyklem. Část `získání dat` v sobě zahrnuje uložení logovaných hodnot ze simulátoru a pak především uložení vektor d_t do proměnné `glob_dt` a jeho odeslání agentům ke zpracování funkcí `adapt()`. Poté přichází na řadu vyjednávací cyklus.

Vyjednávací cyklus se zabývá obsluhou fronty zpráv. Nejprve proběhne *doručení zpráv*, které jsou aktuálně ve frontě. V prvním kroku vyjednávacího cyklu je fronta zpráv samozřejmě prázdná, v následujícím se od konce prochází a adresovaným agentům jsou předány příslušné zprávy. To se děje zavoláním funkce `recieve()` se zprávou jako parametrem. V případě, že adresát není nalezen a zprávu tedy nelze



Obrázek 3.4: Strukuta fungování programu `main_loop.exe`, podrobněji popsaná v podkapitole 3.5

doručit je oznámeno varování, ale pokračuje se dál v činnosti.

Po vyprázdnění fronty nastává čas pro *vysílání zpráv*. To spočívá ve spuštění funkce `broadcast()` u všech agentů. V ní mají příležitost vložit do fronty zpráv libovolný počet sdělení pro ostatní agenty, nemusí ovšem vysílat žádné.

Další postup závisí na stavu fronty zpráv. Pokud je prázdná, tedy žádný agent nevyslal žádnou zprávu, končí vyjednávací cyklus a pokračuje se v běhu programu. V případě, že fronta prázdná není, cyklus se opakuje a proběhne další doručování zpráv z fronty. Vyjednávání může být ukončeno ještě v případě, kdy proběhne stanovený maximální počet vyjednávacích cyklů, nastavený v proměnné `max_cycles`.

Limit by však měl být nastaven tak, aby k jeho dosažení nedocházelo, pokud se tak stane, mělo by to být signálem problému v programu.

Poslední část hlavního cyklu je nazvaná *krok simulace*. Spočívá nejprve v zavolání funkce `act()` u každého agenta. To je okamžik, kdy mohou agenti ovlivnit řízení signalizace. Právě zde probíhá zkopírování vypočítaných hodnot do vektoru `glob_ut`, tedy do proměnné představující vektor vstupních hodnot u_t. Na závěr hlavního cyklu je celá simulace posunuta o jeden krok dále voláním funkcí `step()` u loggeru `L`, `Ds` a u všech agentů. Délka kroku je pevně nastavena na 90 s, je to z důvodu, že v Praze jsou údaje z detektorů sbírány právě v tomto intervalu.

Tato smyčka se opakuje dokud není dosažen v konfiguračním souboru nastavený čas simulace. *Zakončení* spočívá již jen v zapsání log souboru.

3.6 Implementace navrženého algoritmu

Algoritmus popsaný v kapitole 3.1 je implementován pomocí agentů, kteří jsou instancemi třídy `GreenWaveTrafficAgent`.

Třída je odvozena od předka jménem `BaseTrafficAgent`. Ten slouží jen jako prakticky prázdný agent, po přiřazení k nějaké křižovatce nekoná žádnou akci a umožňuje tak simulovat systém bez decentralizovaného řízení. Předkem této třídy je třída `Participant` z knihovny BDM. `Participant`, tedy účastník, představuje základní jednotku v rozhodovacím procesu. Každý účastník má své jméno (`name`) a je schopen ukládat výsledky. Už právě třída `Participant` deklaruje existenci funkcí `adapt()`, `broadcast()`, `recieve()`, `act()` a `step()`. Tyto jsou v něm definovány jako virtuální (`virtual`), tedy pokud je v potomkovi redefinujeme, je zaručeno volání těchto nových funkcí.

3.6.1 Veřejné funkce

Veřejné (`public`) funkce ve třídě `GreenWaveTrafficAgent` přepisují funkce definované v předcích a představují rozhraní pro komunikaci s hlavním programem a tím i ostatními agenty.

První po vytvoření instance agenta proběhne zavolání funkce `from_setting()`. V té se nejprve volá stejnojmenná funkce předka, která načítá z konfigurace základní parametry, jmenovitě:

lanes Informace o jízdních pruzích,

neighbours Jména agentů u sousedních křižovatek,

green_names Jména jednotlivých signálních skupin,

green_starts Čas, ve který příslušné signální skupiny rozsvěcí zelenou,

green_times Délka trvání svícení zelené dané signální skupiny, uváděna jako poměr doba svícení/délka cyklu,

a některé další, které pro tuto práci nejsou důležité.

Popis jednotlivých jízdních pruhů si zaslouží podrobnější rozebrání. Ten totiž pro agenta představuje stěžejní část informací a topologii jím řízené křižovatky. Do konfiguračního souboru se uvádějí všechny takové jízdní pruhy, které končí nějakou stop čárou dané křižovatky. Každý jízdní pruh pak má následující parametry: `sg`, což je signální skupina do které patří, `inputs` je seznam detektorů na vjezdu do tohoto pruhu, `outputs` je seznam detektorů na sousedních křižovatkách, ke kterým může vozidlo z tohoto pruhu dojet. Pokud se vozidla mohou dostat do míst, kde žádný detektor není, je zde za každý takovýto směr uveden „falešný“ detektor `DUMMY_DET`. Dále konfigurace obsahuje `input_distances`, pole vzdáleností vstupních detektorů od stop čáry, podobně `output_distances` je pole vzdáleností výstupních detektorů (u „falešných“ nehraje roli), `alpha` je pole poměrů odbočení k jednotlivým výstupním detektorům, `queue` jméno fronty, která se na daném pruhu tvoří a konečně `beta` je koeficient, kterým se násobí délka fronty během výpočtů, které agent provádí.

Agent po získání informací o jízdních pruzích pro každý vytvoří instance tříd `Lane` a `LaneHandler`. ty představují mezivrstvu mezi agentem a fyzickým jízdním pruhem se skutečnými detektory.

Dále se ve funkci `from_setting()` nastavují hodnoty konstant a výchozí hodnoty některých proměnných. Konkrétně jde o

car_leaving_time Čas v sekundách, ze který jedno auto opustí frontu (c_0)

VP Průmerná rychlost jízdy automobilu mezi křižovatkami na volné silnici (v_P)

actual_time Uchovává aktuální čas simulace (v sekundách)

negot_cycle Počet již proběhlých vyjednávacích cyklů od posledního průměrování

cycle_count Hranice počtu cyklů, po jejímž dosažení nastává průměrování získaných offsetů

total_offset Hodnota součtu dosud vyjednaných offsetů od posledního průměrování

negot_start Krok, kterým se začíná při hledání ideálního offsetu u souseda

negot_limit Krok, kterým se končí při hledání ideálního offsetu u souseda

find_best_start Krok, kterým začíná hledání nejlepšího vlastního offsetu

find_best_limit Krok, kterým končí hledání nejlepšího vlastního offsetu

passive Indikuje zda agent zastává pasivní nebo aktivní roli

Hodnoty konstant lze z výchozích hodnot přepsat uvedením proměnné v konfiguračním souboru.

V závěru jsou je pak z konfigurace načten výchozí offset, role agenta a připraven vektor výstupních hodnost `outputs` spolu s jeho popisem `rv_outputs`. Na úplném konci je ještě zapnuto logování hodnot offsetů pro pozdější vyhodnocení.

Další volanou funkcí v agentech je `adapt()`. Podobně jako `from_setting()` nejprve spouští stejnomenou funkci u svého předka. V té probíhá vyplnění vektorů `inputs` a `queues` daty z Aimsunu, uloženými ve vektoru `glob_dt`. Údaje z `queues` se předají příslušným `LaneHandler`ům. Dále je nastaven `planned_offset` na hodnotu získanou v minulém cyklu a hodnoty stavových proměnných jsou vráceny do svých výchozích hodnot.

Funkce `recieve()` zpracovává přijaté zprávy od ostatních agentů. Na začátku je přijatá zpráva rozdělena do jednotlivých proměnných.

Dále se činnost liší podle „předmětu“ zprávy, tedy podle řetězce uloženého v části `what`. V případě přijetí žádosti o očekávané časy příjezdů je jen uloženo jméno odesílatele do seznamu žadatelů `requesters`.

Pokud jsou naopak obsahem přijaté zprávy očekávané časy příjezdů automobilů od souseda, aktivní agent najde nový optimální offset pro svůj signální plán a spočítá `rating`, pasivní provede jen tento výpočet. Na konci je stavová proměnná `new_stable_state` nastavena na `true`, což určuje další činnost agent ve fázi vysílání zpráv.

Obdržení zprávy s hlavičkou „`stable_state`“ znamená, že některý ze sousedů změnil své nastavení offsetu a zasílá tedy nové hodnoty očekávaných příjezdů. Pokud je aktuální hodnota `negot_offset` větší nebo rovna `negot_limit`, zkouší pomocí funkce `find_best_exps()`, která z hodnot 0, +`negot_offset` nebo −`negot_offset` by po přičtení k sousedově offsetu znamenala nejlepší rating. Hodnota `negot_offset` je snížena na polovinu a je připraveno odeslání žádosti pro souseda. Pokud byla proměnná `negot_offset` již nižší než než `negot_limit`, přepíná se agent do konečného stavu, neboť již nemá další prostor k vyjednávání se sousedy.

Při přijetí se zprávy se žádostí o změnu offsetu agent zkoumá, zda je součet změny hodnocení po aplikaci navrženého offsetu a zlepšení hodnocení u souseda, které od změny offsetu očekává, větší než nula. Takováto změna se přijímá a podle ní se upravuje `planned_offset`. V každém případě se agent připraví na zaslání aktuálních hodnot očekávaných příjezdů.

Když přijde zpráva s předmětem, který `GreenWaveTrafficAgent` neumí zpracovat, předává ji předkovi. Pokud by se stalo, že ani ten si se zprávou neporadí, pošle zprávu svému předkovi (tedy třídě `bdm::Participant`), který v takovém případě vyvolá varování o nezpracovatelné zprávě.

Funkce `broadcast()` vysílá zprávy do fronty zpráv a to v závislosti na hodnotách agentových stavových proměnných. Pokud není prázdný seznam `requests`, sestaví odpověď pro každého z agentů uvedeného v seznamu a ten následně vyprázdní. Dále pak za každou ze stavových proměnných, která se rovnají `true` sestaví příslušnou zprávu a hodnotu změní na `false`.

Ve funkci `act()` se spočítaná hodnota `planned_offset` přičítá k hodnotě proměnné `total_offset` a to až do chvíle, než je v něm tolik hodnot, kolik stanovuje limit `cycle_count`. Když se tohoto limitu dosáhne, vydělí se `total_offset` tímto limitem, čímž se získá průměrná hodnotná offsetu z posledních cyklů. Průměr se pak znormalizuje, aby byl z intervalu $\langle 0; T_c \rangle$ a uloží se do vektoru u_t, představovaným proměnou `global_ut`. Tím se tato hodnota dostane k řadiči křižovatky.

O zápis aktuálních hodnot do logovacího souboru se stará `log_write()`. Ukládá se dvousložkový vektor. První prvek obsahuje vypočítaný offset `planned_offset`, druhý jeho hodnocení `planned_rating`.

Poslední volaná veřejná funkce agenta je `step()`, která jen posouvá interní ukazatel času v agentovi o délku kroku simulace.

3.6.2 Chráněné funkce

Chráněné (`protected`) funkce zajišťují většinu výpočtů v agentovi, starají se o sestavení očekávaných časů příjezdů vozidel k sousedům, hledání optimálního offsetu, počítání hodnocení a tak podobně.

Funkce `expected_cars()` sestavuje údaje o příjezdech vozidel šechny sousedy. Odhady jsou založeny na aktuální hodnotě offsetu uložené v proměnné `planned_offset` a jsou na závěr uloženy do vektoru `outputs` určenému k odeslání.

Výpočet probíhá podle rovnic (3.1) a (3.2). Počet vozidel je získán jako součin odhadu počtu vozidel, která projedou jízdním pruhem za dobu rozsvícení zelené a příslušného poměru odbočení α_i, kde i je číslo výstupního směru. Výpočet poštu vozidel, která jízdním pruhem projedou je záležitostí funkce `expected_output()` ze třídy `LaneHandler`. V současnosti tato funkce vrací počet vozidel, která projela v minulém cyklu.

Pro nalezení nejlepšího nastavení vlastního offsetu slouží rekurentní funkce `int find_best_offset(cont int center, int interval)`. Ta hledá ze tří hodnot offsetů: `center` + `interval`, `center` a `center` − `interval` hledá tu s nejlepším hodnocení. Nalezená hodnota se pak použije jako první parametr pro další volání sebe sama, jako nový interval se vezme polovina předchozího. Toto volání probíhá, dokud `interval` nedosáhne hranice `find_best_limit`. Pak je funkce ukončena a je vrácen nalezený offset.

Funkce zkoumající efekty změn offsetu u souseda má prototyp `int find_best_exps (const int offset_change, const string neighbour, double &rating_change)`.

Výstupem je nalezená změna offsetu. Parametr `offset_change` značí o kollik sekund bude agent zkoušet posunout sousedův offset, `neighbour` je jméno zkoumaného souseda a do proměnné `rating_change` bude uložena změn hodnocení, kterou nalezený nový offset pravděpodobně přinese oproti současnému stavu. Funkce si nejprve připraví hodnoty očekávaných příjezdů tak, jak by pravděpodobně vypadaly při změně sousedova offsetu v kladném a v záporném smyslu o hodnotu `offset_change`. Následně pak vypočítá hodnocení s těmito a bez těchto změn. Změna s nejlepším hodnocením je návratovou hodnotou funkce.

Počítání hodnocení probíhá ve funkci `double count_rating(const int offset, const vec recieved_exps, const RV rv_recieved_exps)` několika vnořených cyklech. Vnější probíhá přes všechny (vjezdové) jízdní pruhy křižovatky. Pro každý pruh následuje ve vnořené smyčce hledání očekávaných příjezdů ve vektoru přijatých dat `recieved_exps`. Pokud o nich nejsou pro zkoumaný pruh informace, nebude se započítávat do hodnocení.

V případě nalezení nějakých předpokladů jsou tyto uloženy do vektorů `t_cars_begin` (začátky příjezdů), `t_cars_end` (konce příjezdů) a `cars_density` (hustoty vozidel) pro další výpočet. Ten začíná zjištěním rozsvícení zelené v signální skupině příslušné danému jízdnímu pruhu, se započítáním offsetu předanému funkci v parametru `offset`. Intervalem svícení zelené je pak funkcí `expand_greens()` s periodou T_c zopakován tak, aby pokryl celý časový úsek, pro který jsou k dispozici informace o příjezdech vozidel.

Ze znalosti očekávaných příjezdů a stavu semaforů během nich pak vychází vypočítávání příspěvku k hodnocení. Ten probíhá v `do – while` cyklu. Pro jeho řízení je definována proměnná `t_act`, představující ukazatel na právě zpracovávaný okamžik ve zkoumaném intervalu. Její hodnota začíná na 0 a pokud dosáhne hodnoty proměnné `t_limit`, způsobí ukončení smyčky. Do proměnné `t_limit` je uložen čas posledního zhasnutí zelené, získaný z výše zmíněné funkce `expand_greens()`. `t_act` se tedy během výpočtu posouvá po významných bodech intervalu a v každém takovém bodě se ověřuje, jakého typu (dle definice z kapitoly 3.1) je následující interval. Podle toho se zjistí příslušná délka virtuální fronty $Q_V^{(i)}$ a pokud je tato záporné, odečte se od hodnocení `rating`.

Po té, co funkce projde všechny jízdní pruhy, vrátí výslednou hodnotu uloženou v proměnné `rating` jakožto hodnocení nastavení s daným offsetem a danými předpoklady o příjezdech vozidel.

`GreenWaveTrafficAgent` pak ještě obsahuje několik pomocných funkcí usnadňující a zpřehledňujících výpočty. Jde například o převod libovolné proměnné na typ `string`, nalezení indexu minimálního prvku ve vektoru či hledání indexu (pořadí v konfiguračním souboru) zadané signální skupiny.

Za zmínku stojí funkce `int normalize_offset(int offset, bool zero=true)`. Ta provádí posun zadaného `offsetu` tak, aby zapadl do žádaného intervalu. Při vynechání druhého parametru jde o interval $\left\langle -\frac{T_c}{2}; \frac{T_c}{2} \right\rangle$, pokud má druhý parametr hodnotu `false`, pak $\langle 0; T_c \rangle$. Druhá forma normalizace se provádí před zaslání offsetu řadiči křižovatky, ten totiž hodnotu v daném intervalu očekává. První forma je důležitá pro průměrování offsetů. Pokud by nebyla použita (nebo pokud by se používala druhá), nebyl by ve výsledném průměru zahrnut fakt, že hodnota x a $x + T_c$ je v případě nastavení offsetu totožná.

Kapitola 4

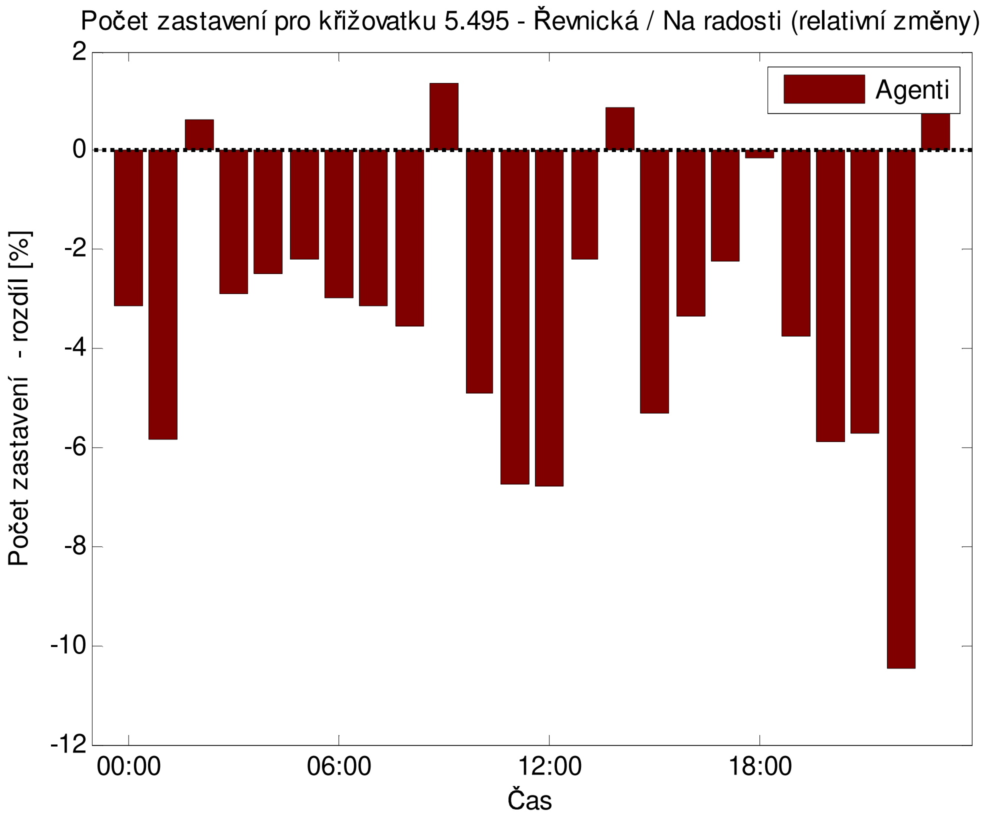
Výsledky simulací

Ověření chování navrženého řídícího algoritmu bylo provedeno na dvou scénářích. Oba simulují dobu 24 hodin (od půlnoci do půlnoci), přičemž první z nich představuje konstantní příjezdy na všech ramenech po celý den, druhý pak zachycuje reálný provoz zaznamenaný detektory dne 12. prosince 2007. Pro každý ze scénářů bylo provedeno porovnání několika dopravních veličin v případě řízení jen pevnými signálními plány (tento stav je označován jako *reference*) a při řízení pomocí agentů, kteří nastavují pevným signálním plánům offsety algoritmem navrženým v kapitole 3.1 (což je dále označováno jako *agenti*).

4.1 Konstantní vjezdy

Scénář s konstantními vjezdy do oblasti nevykazuje z globálního zlepšení v žádné ze sledovaných veličin zlepšení, jak ukazuje tabulka 4.1 a to ani v parametrech, které by měl algoritmus nejvíce optimalizovat, v počtu zastavení a průměrné době zastavení. VGS API bohužel neposkytuje globální číselné údaje pro jednotlivé křižovatky, je tedy nutné spolehnout se pouze na grafy. Obrázek však 4.1, že na křižovatce Řevnická / Na Radosti nastává zlepšení v počtu zastavení vozidel.

Podrobněji to lze vidět na číslech z jednotlivých jízdních pruhů, která již k dispozici jsou. Konkrétně na jízdních pruzích vedoucích od sousední řízené křižovatky, pro které je primárně prováděna optimalizace, je zlepšení zřetelné. Například pro pruh



Obrázek 4.1: Relativní změny počtu zastavení na křižovatce 5.495. Graf zobrazuje relativní velikost rozdílu oproti referenčnímu systému

odbočující vlevo je průměrná doba jízdy při řízení agenty o 13 % nižší. Některé parametrů ukazují tabulky 4.2 a 4.3.

Bohužel u druhé křižovatky již toto zlepšení nenastává a to ani ve směru od první.

Další grafy a tabulky porovnávající referenční systém s distribuovaně řízením se nacházejí v příloze A.

4.2 Reálný provoz

Ani při simulaci se skutečnými vjezdy není při pohledu na globální statistiky v tabulce 4.4 znatelné žádné zlepšení. Při pohledu na jednotlivé křižovatky je ale opět vidět znatelné snížení počtu zastavení v křižovatce 495, jak zobrazuje graf na obrázku 4.2. Bohužel toto s sebou znovu nese i zhoršení tohoto parametru na křižovatce 601.

	Reference	Agenti	/Agenti /Reference
Tok vozidel [voz/h]	2333,00	2333,00	0,0 %
Průměrná doba jízdy [s]	143,57	150,48	4,8 %
Směrodatná odchylka doby jízdy [s]	54,74	61,35	12,1 %
Průměrné zpoždění [s]	69,34	76,26	10,0 %
Směrodatná odchylka zpoždění [s]	53,97	60,51	12,1 %
Průměrná rychlost [km/h]	28,85	28,04	−2,8 %
Směrodatná odchylka rychlosti [km/h]	10,74	11,06	3,0 %
Hustota vozidel [voz/km]	15,40	16,16	4,9 %
Průměrná doba zastavení [s]	54,49	61,10	12,1 %
Směrodatná odchylka doby zastavení [s]	50,70	56,61	11,7 %
Počet zastavení [-]	2,10	2,23	5,9 %

Tabulka 4.1: Rozdíl v měřených parametrech v celé simulované oblasti při konstatních vjezdech.

	Reference	Agenti	/Agenti /Reference
Průměrná doba jízdy [s]	16,88	14,68	−13,0 %
Průměrné zpoždění [s]	12,66	10,46	−17,4 %
Průměrná doba zastavení [s]	10,06	8,53	−15,2 %
Počet zastavení [-]	0,64	0,49	−24,4 %

Tabulka 4.2: Rozdíl v některých parametrech na jižním příjezdu do křižovatky 495, pruh pro odbočení vlevo.

	Reference	Agenti	/Agenti /Reference
Průměrná doba jízdy [s]	3,68	3,74	1,6 %
Průměrné zpoždění [s]	0,51	0,57	11,7 %
Průměrná doba zastavení [s]	0,14	0,33	137,6 %
Počet zastavení [-]	0,04	0,04	−6,9 %

Tabulka 4.3: Rozdíl v některých parametrech na jižním příjezdu do křižovatky 495, pruh pro odbočení vpravo.

	Reference	Agenti	/Agenti /Reference
Tok vozidel [voz/h]	1399,00	1400,00	0,1 %
Průměrná doba jízdy [s]	144,13	155,18	7,7 %
Směrodatná odchylka doby jízdy [s]	72,94	111,59	53,0 %
Průměrné zpoždění [s]	69,98	81,02	15,8 %
Směrodatná odchylka zpoždění [s]	72,23	110,88	53,5 %
Průměrná rychlost [km/h]	29,64	29,41	−0,8 %
Směrodatná odchylka rychlosti [km/h]	11,06	11,63	5,1 %
Hustota vozidel [voz/km]	9,35	9,93	6,2 %
Průměrná doba zastavení [s]	56,56	67,64	19,6 %
Směrodatná odchylka doby zastavení [s]	68,62	107,90	57,3 %
Počet zastavení [-]	2,05	2,16	5,3 %

Tabulka 4.4: Rozdíl v měřených parametrech v celé simulované oblasti při reálných vjezdech.

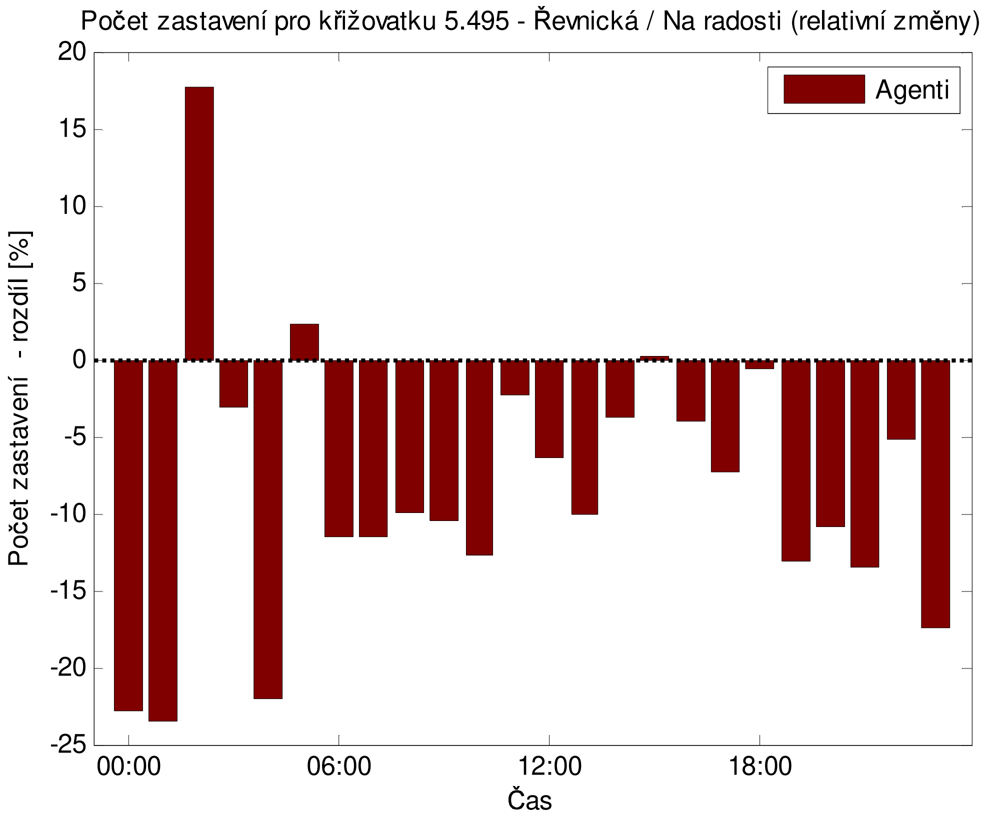
Především kolem 17. hodiny zde nastává výrazný nárůst délek front, který s sebou nese zvýšení počtu zastavení vozidel. Fronta nejspíše dosahuje takové délky, že ji není možné vyprázdnit během jednoho cyklu a proto algoritmus, který při počítání nastavení offsetů vypočítává hodnocení v závislosti na počtu aut, která projedou po vyprázdnění fronty, není schopen vybrat optimální offset.

Další grafy a tabulky k tomuto scénáři lze nalézt v příloze B.

4.3 Zhodnocení

V současné se verzi se implementace navrženého algoritmu neukázala jako řešení přinášející lepší průjezd vozidel celou oblastí. Přesto byl zaznamenám alespoň dílčí úspěch na jedné z křižovatek, který může naznačovat jistý potenciál této metody řízení provozu.

Původem problémů je pravděpodobně nepřipravenost algoritmu na příliš velkou hustotu provozu, při které se tvoří dlouhé fronty. Dokud je provoz na nízkých hladinách, udržuje se počet zastavení pod nebo na hranici hodnot, které jsou v referenčním



Obrázek 4.2: Relativní změny počtu zastavení na křižovatce 5.495. Graf zobrazuje relativní velikost rozdílu oproti referenčnímu systému

systému. Pokud se provoz zvýší a začnou se tvořit dlouhé fronty, systém pravděpodobně není schopen ohodnotit jednotlivá nastavení offsetu (protože nestíhá vyprázdnit fronty) a proto ani nemůže najít optimální offset. Řešením by mohlo být rozšíření hodnotící funkce tak, aby v takovém případě docházelo alespoň k postupnému zkracování fronty, je-li to v danou chvíli možné.

Další cestou ke zlepšení by mohlo být odstranění některých zjednodušení, která jsou v současném modelu použita. Jde především o předpovědi o hustotách provozu. Ty jsou nyní založeny jen jen počtu vozidel v jednom (posledním) cyklu a neobsahují informace o průběhu hustoty v čase. Dá se předpokládat, že třeba v případě existence nějaké fronty pojede po rozsvícení zelené část vozidel v malých vzájemných vzdálenostech a po vyprázdnění fronty pak provoz prořídne. Také by se dala zlepšit práce s údaji z detektorů. Ty nyní nejsou nijak filtrovány pro odstranění šumu a nepřesností.

Další směr vývoje by také mohlo být implementování nějakého algoritmu pro modelování a odhadování délek front jen na základě údajů z detektorů

Otevřenou otázkou prozatím zůstává chování algoritmu při modelování větší oblasti. Především bude nutné prozkoumat vhodné rozdělení agentů na pasivní a aktivní. Je možné, že toto rozdělení se ukáže jako naprosto nevhodné a nutné tyto role dynamicky přepínat, například dle šachovnicového systému, nebo postupným šířením aktivních agentů po celé řízené síti.

Závěr

Decentralizované řízení je jedním z moderních přístupů k řešení problémů nejen s dopravou na silnicích. Tato práce se snažila prozkoumat tento způsob kontroly pro případ dopravní signalizace v malé izolované oblasti.

Byl navržen algoritmus na bázi agentů přiřazených k jednotlivým křižovatkám, kteří se snaží koordinovaným nastavováním offsetů signálních plánů dosáhnout zelené vlny, tedy stavu, kdy automobily mohou projíždět s minimálním počtem zastavení. Algoritmus je následně implementován v jazyce C++ a začleněn do simulačního prostředí používaném v ÚTIA AV ČR, postaveném kolem mikrosimulátoru dopravu Aimsun. Následně byly provedeny simulace na modelu skutečné oblasti a porovnání stavu provozu se situací, kdy je dopravní signalizace řízena pevnými signálními plány.

Vyhodnocení výsledků ukázalo, že algoritmus může přinést zlepšení průjezdnosti do některých částí oblasti, ve které je nasazen, při pohledu na celý simulovaný prostor však tento pokrok zanikl v problémech, které vznikly jinde. Přesto je možné, že další prací na algoritmu by se povedlo některé nedostatky odstranit a rozšířit tak okruh, pro který přináší vylepšení.

Seznam použitých zdrojů

[1] Bazzan, A. L. C.: Opportunities for multiagent systems and multiagent reinforcement learning in traffic control. *Auton Agent Multi-Agent Syst*, 2009: s. 342–375.

[2] Klein, L. A.: Traffic Detector Handbook: Third Edition – Volume I. Technická Zpráva FHWA-HRT-06-108, U.S. Departement of Transportation, Federal Highway Administration, Říjen 2006.

[3] Lindner, M. A.: *libconfig - A Library For Manipulating Structured Configuration Files*. Říjen 2007.

[4] Pecherková, P.; Duník, J.; Flídr, M.: *Robotics, Automation and Control*, kapitola Modelling and Simultaneous Estimation of State and Parameters of Traffic System. InTech, Croatia, 2008, ISBN 978-953-7619-18-3, s. 319–336.

[5] Roozemond, D. A.: Using intelligent agents for pro-active, real-time urban intersection control. *European Journal o Operational Research*, 2001: s. 293–301.

[6] TSS: *Getram v4.2 getting started - User's manual*. Říjen 2003.

[7] TSS: *GETRAM Extensions VERSION 4.2 - User's manual*. Květen 2004.

[8] TSS: *Getram v4.2 - User manual*. Únor 2004.

[9] Weiss, G. (editor): *Multiagent systems: a modern approach to distribuated artificial intelligence*. MIT Press, 1999, ISBN 0-262-23203-0.

Přílohy

Příloha A

Výsledky simulací se scénářem s konstantními vjezdy

Tato příloha obsahuje vybrané grafy a tabulky vzniklé při vyhodnocování navrženého algoritmu na scénáři s konstantními vjezdy do oblasti.

Příloha B

Výsledky simulací se scénářem se skutečnými

Tato příloha obsahuje vybrané grafy a tabulky vzniklé při vyhodnocování navrženého algoritmu na scénáři simulujích reálné vjezdy vozidel do oblasti tak, jak byly zaznamenány 12. prosince 2007.