České vysoké učení technické v Praze

Fakulta jaderná a fyzikálně inženýrská

BAKALÁŘSKÁ PRÁCE

Kamil Ondrák

Praha – 2010

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, 3. července 2010 Kamil Ondrák

Poděkování

(. . .) Děkuji Mgr. Markovi Hryciowovi za pozorné pročtení a podnětné připomínky. (. . .)

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 nejjednodušší 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 přesouvá jinam, kde tak může vzniknout podobný problém. Jako zajímavá možnost se jeví co nejlepší řízení provozu pomocí světelných signalizačních zařízení.

V současné době je většina vytížených křižovatek řízena pomocí poměrně sofistikovaných signálních plánů, které se i dokáží přizpůsobovat aktuální dopravní situaci. Tyto plány bývají vypočítány různými dlouhodobě vyvíjenými programy 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 jakákoli koordinace mezi sousedícími křižovatkami.

Na některých místech se používá jistá forma centralizovaného řízení. Ústředna sbírá údaje z určité oblasti a na základě těchto informací vysílá pokyny do řadičů příslušných křižovatek. Tento postup vede k velmi dobrým výsledkům, ale většímu rozšíření brání malá univerzálnost algoritmů používaných v ústřednách.

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

Cílem této práce je seznámit se s tímto decentralizovaným způsobem řízení, navrh-

nout 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 simulátoru dopravy Aimsun. Toto vše je prováděno na modelu skutečné oblasti, konkrétně Řevnické ulice v Praze – Zličíně. Jako referenční stav pak slouží řízení dopravní signalizace signálními plány expertně navrženými metodou Monte Carlo.

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 systému ... (atd, podle toho, co v kapitole bude)

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

1.1 „Křižovatka“

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

Detektory se dělí dělí do tří kategorií podle jejich umístění. (i) výzvové, (ii) prodlužovací a (iii) strategické. (i) jsou umístěné na stop čáře, (ii) cca 30 metrů před ní a (iii) na vzdálenějších místech, typicky 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. Ta pomocí změn magnetického pole ve své okolí zaznamenává projíždějící vozidla. V případě umístění dvou smyček blízko sebe lze získat i podrobnější informace o rychlosti, druhu vozidla a další informace. Další možností 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.

Detektory umístěné na Zličíně poskytují dva údaje. Prvním je počet vozidel, které přes detektor projely a druhým čas, po který se v detekované oblasti vyskytovalo nějaké vozidlo. Bohužel, tato data nejsou naprosto přesná.

(!chyby detektorů!)

Řízení křižovatky probíhá pomocí signálních plánů uložených v řadiči nebo dle pokynů z ústředny. Signální plány mohou být pevné nebo rámcové. Signální plán se skládá z několika fází, které mají buď pevně nebo rámcově dány časy začátků a délku. Fáze jsou složené ze signálních skupin. Signální skupina pak značí skupinu světelných signalizačních zařízení, která rozsvěcuje současně zelenou. Signální plány mají pevnou a jednotnou délku jednoho cyklu. 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.

(možná obrázek sem)

V případě pevných signálních plánů se přepínání fází řídí předem danou tabulkou, která určuje kdy která fáze začíná a jak dlouho má trvat. Navíc může být u fází uložena možnost je prodloužit v případě, že údaje z detektorů oznamují další přijíždějící vozidla v daném směru.

Rámcové signální plány dovolují řadiči větší volnost při zařazování fází. Na základě aktuálních naměřených údajů může řadič volit nejvhodnější fázi z několika momentálně přípustných, může nastat i situace, že některá fáze nebude v rámci cyklu zařazena vůbec.

1.2 Oblast Zličína

Ověření možnosti decentralizovaného řízení dopravní signalizace je v této práci ověřeno na modelu skutečné oblasti: ulice Řevnická v Praze – Zličíně. V ulici se nachází pět světelných křižovatek situovaných v jedné linii. Pro zjednodušení jsou simulovány jen dvě severní křižovatky: 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.

1.3 Simulátor Aimsun

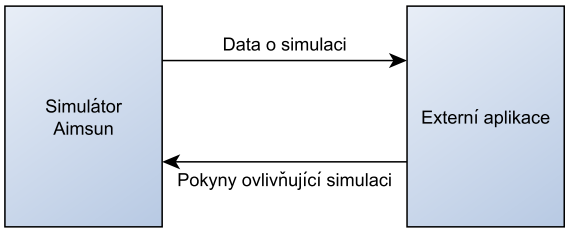
K simulaci dopravní situace se používá softwarový balík GETRAM/AIMSUN od společnosti TSS (Transport Simulation Systems) ve verzi 4.2.16. Jádrem celého nástroje je program Aimsun, který slouží k samotné simulaci.

Aimsun (Advanced Interactive Microscopic Simulator for Urban and Non-Urban Networks, [6]) je mikrosimulátorem dopravy[1]. To znamená, že modeluje polohu jednotlivých vozidel spojitě 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 poptávce, „traffic control plans“ 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 poptávka může být zadaná ve dvou formách. Buď ve formě intenzity dopravy na vstupech, poměrů odbočení na křižovatkách a počátečního stavu sítě, nebo pomocí takzvané O-D matice, která zachycuje počet uskutečněných cest mezi dvojicemi vstupních a výstupních bodů. „Traffic control plans“ 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 představují pevné hodnoty popisující experiment (jako doba simulace) a proměnné hodnoty určené pro kalibraci modelu.

Výstup za simulace jsou textové soubory s číselnými údaji, které se zpracovávají například v MATLABu pomocí `toolboxVGS`.

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



Obrázek 1.1: Aimsun a externí aplikace

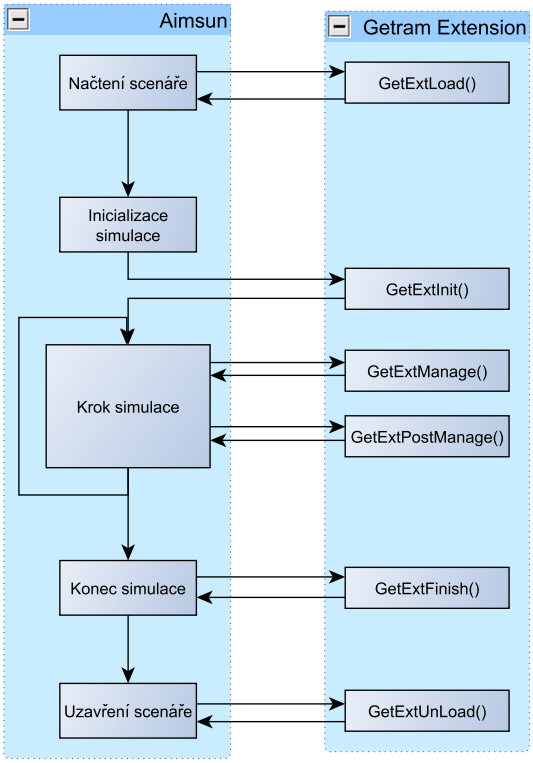
1.3.1 Getram Extensions

Aimsun má aplikační rozhraní (API) jménem Getram Extensions, které umožňuje tvorbu externích aplikací komunikujících se simulátorem. Aimsun pomocí tohoto rozhraní poskytuje v reálném čase data ze simulace a naopak přijímá data pro její ovlivňování, jak je naznačeno na obrázku (1.1). 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 (1.2)



Obrázek 1.2: Podrobný nákres komunikace mezi Aimsunem a Getram Extension

1.4 Emulátor řadiče ELS3

Pro simulace funkcí řadiče křižovatky se používá emulátor ELS vyvinutý firmou Eltodo. Ten z reálného řadiče obsahuje algoritmus řízení, řízení 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 tvoří dopravní řešení 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 hraje nahrazuje modul dopravní ústředny) řadič pak přijímá data pro ovlivnění řízení. Výstupem řadiče je obraz barevné kombinace signálních skupin. Ten se zasílá do Aimsunu na konci simulačního kroku.

Kapitola 2

Agentní systémy

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 definici, kterou používá Michael Wooldrige v [7, 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 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 jde o deterministické nebo nedeterministické prostředí, 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 jsou myšleny tři body:

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 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 zaměření agenta na různé cíle jdoucí proti sobě a tím pádem ani jeden z cílů nemůže být nikdy uskutečněn. Na druhou stranu příliš proaktivní agent se zaměří na jeden cíl a kvůli 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 protinávrh vedoucí ke kompromisu.

(...pokračování...)

Kapitola 3

Algoritmus řízení

3.1 Návrh algoritmu

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.

Důležitým prvkem návrhu je funkce která, ohodnotí nastavení offsetu a umožní tak tedy porovnat různé jeho hodnoty a vybrat tu možná nejlepší. Jako míra kvality byl odhad počtu aut, která projedou křižovatkou bez zastavení.

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

$$t_z = t_{zz} + \frac{d}{v_P} + \mathit{offset} \tag{3.1}$$

a

$$t_k = t_z + t_{dz} \ , \tag{3.2}$$

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 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, který o odhady žádal.

Po shromáždění všech předpokladů od sousedů může agent přistoupit k samotnému výpočtu. Ten spočívá v tom, že agent spočítá příspěvky všech jízdních pruhů, u kterých má nějaké informace o časech příjezdů. Pro každý pruh se pak nejprve spočítá délka fronty v čase, kdy se na jeho odjezdu rozsvítí zelená. To znamená:

$$Q_V = Q + \sum_{i=1}^{k} t_i n_i \tag{3.3}$$

kde Q_V je délka tak zvané virtuální fronty, zde jde o délku fronty při rozsvícení zelené, Q je odhadovaná délka fronty, t_i je délka i-tého intervalu, ve kterém přijíždějí vozidla, n_i je počet těchto vozidel a k je počet intervalů s příjezdy před rozsvícením zelené.

Dále se pak spočítá délka virtuální fronty v okamžiku, kdy zelená přejde zpět na červenou. K tomu je třeba rozdělit interval od prvního očekávaného příjezdu nebo rozsvícení zelené (podle toho, co nastane dříve) po zhasnutí zelené (tento interval označíme T) na posloupnost intervalů $(T_i)_{i=1}^{k}$. T_i jsou intervaly typu $\langle a_i; b_i\rangle$, kde $b_i = a_{i+1}\ \forall i \in \{2, \ldots, n-1\}$, a_1 a b_k jsou krajní body intervalu T a a_i jsou chronologicky řazené všechny významné body, tedy body, ve kterých se mění signalizovaný znak nebo ve kterých začíná či končí příjezd vozidel od souseda.

S pomocí tohoto dělení pak můžeme počítat

$$Q_K = Q_V + \sum_{i=1}^{k} u(T_i)\,, \tag{3.4}$$

kde $u(T_i)$ je funkce $u:\ \langle a; b\rangle \to \mathbb{R}$

$$u(T_i) = \begin{cases} n_i & \text{pokud v intervalu } T_i \text{ pouze přijíždějí vozidla} \\ -|T_i| \cdot c_o & \text{pokud v intervalu } T_i \text{ pouze odjíždějí vozidla} \\ n_i - |T_i| \cdot c_o & \text{pokud v intervalu } T_i \text{ přijíždějí i odjíždějí vozidla} \end{cases} . \tag{3.5}$$

Q_K značí konečnou délku fronty, Q_V je virtuální fronta z rovnice (3.3), k je počet intervalů „s různými vjezdy a výjezdy“. $|T_i|$ je délka intervalu T_i, n_i počet vozidel,

která přijedou v intervalu T_i a c_o je empiricky zjištěná konstanta – počet vozidel, která za sekundu opustí křižovatku.

Definice funkce $u(T_i)$ jak je zapsána vztahem (3.5) není úplná. Ještě je nutné doplnit omezení

$$|u(T_i)| \begin{cases} \leq Q_i & \text{pokud v intervalu } T_i \text{ pouze odjíždějí vozidla} \\ \geq n_i & \text{pokud v intervalu } T_i \text{ přijíždějí i odjíždějí vozidla} \end{cases}, \tag{3.6}$$

kde Q_i je délka fronty na začátku intervalu T_i.

Tyto podmínky zarčují, že pokud z jízdního pruhu jen odjíždějí vozidla, nemůže jich odjet více než jich čeká na začátku ve frontě a že pokud vozidla zároveň přijíždějí a odjíždějí, projede jich bez zastavení maximálně tolik, kolik dorazí od souseda.

Pokud vyjde Q_K záporné, představuje (odhadovaný) počet aut, která mohou křižovatkou projet bez zastavení a tato hodnota se tedy odečte od hodnocení dané křižovatky (a tím se tedy hodnocení zvýší).

Vystává otázka jak zjistit aktuální délky fronta na jednotlivých ramenech 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 v reálné situaci, ale používají se hodnoty, které lze získat ze simulátoru Aimsun. Podrobnější informace o modelování délky fronty představuje například [2].

Při samotném vyjednávání pak mají agenti dvě role, jeden z nich je označen jako *pasivní*, zatímco druhý je v pozici aktivního. Pasivní agent má pevně nastavený offset a jen reaguje na pokyny aktivního. Pokyny jsou 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 jejich offsetu nevedla k zlepšení

hodnocení. Hledání nejlepšího offsetu probíhá ve třech krocích. Nejprve se porovná aktuální offset, offset zvýšený o 8 sekund a offset snížený o 8 sekund. Vybere se nejlepší ze tří možností a pokračuje se s ním stejným způsobem, jen 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 agentům 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. Stejným způsobem jako při hledání vlastního nejlepšího offsetu zkusí odhadnout změnu `ratingu` při posunu sousedova offsetu o ± 4, 2 a 1 sekundu. Pokud má nejlepší hodnocení nenulová změna, zašle se sousedovi žádost o tuto změnu 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.

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 toho 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 se 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 `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.

V knihovně jsou mj. 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`, tedy 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);
```

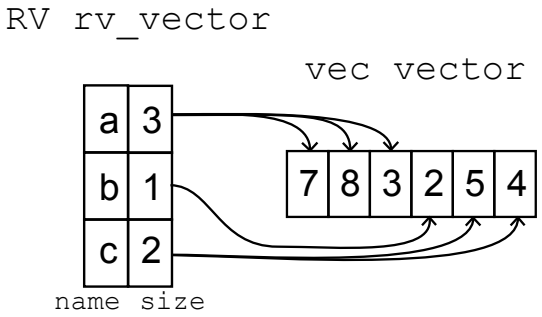
přičemž první ze způsobů pro vektor nealokuje paměť. To je pak nutné udělat funkcí setsize(). Následně je pak možné uložit do vektoru jednotlivé prvky a to například jedním z následujících příkazů.

```
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
```



Obrázek 3.1: 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`). Například podvektor `c` má tedy tvar [5 4]

a tak podobně.

Práce s maticemi pak funguje podle obdobných pravidel.

(Array)

3.2.2 BDM

Knihovna `BDM` (Bayesian Decision Making) se zabývá, jak název napovídá, bayesovským rozhodováním. Ovšem v této práci jsou z ní použity jen třídy `UI`, `RV`, `datalink` a `logger`.

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 3.1.

Pro kopírování dat mezi vektory existuje třída `datalink`. Vytváří spojení mezi dvěma datovými vektory na základně 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 nastaví 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.

(...rozvést nebo zrušit...)

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 údaje o počtu zastavení vozidla, 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 pro 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 (3.2). Skladá se ze tří hlavních bloků, mikrosimulátoru Aimsun, emulátorů řadičů ELS3 a z tzv. Aimsun agenta.

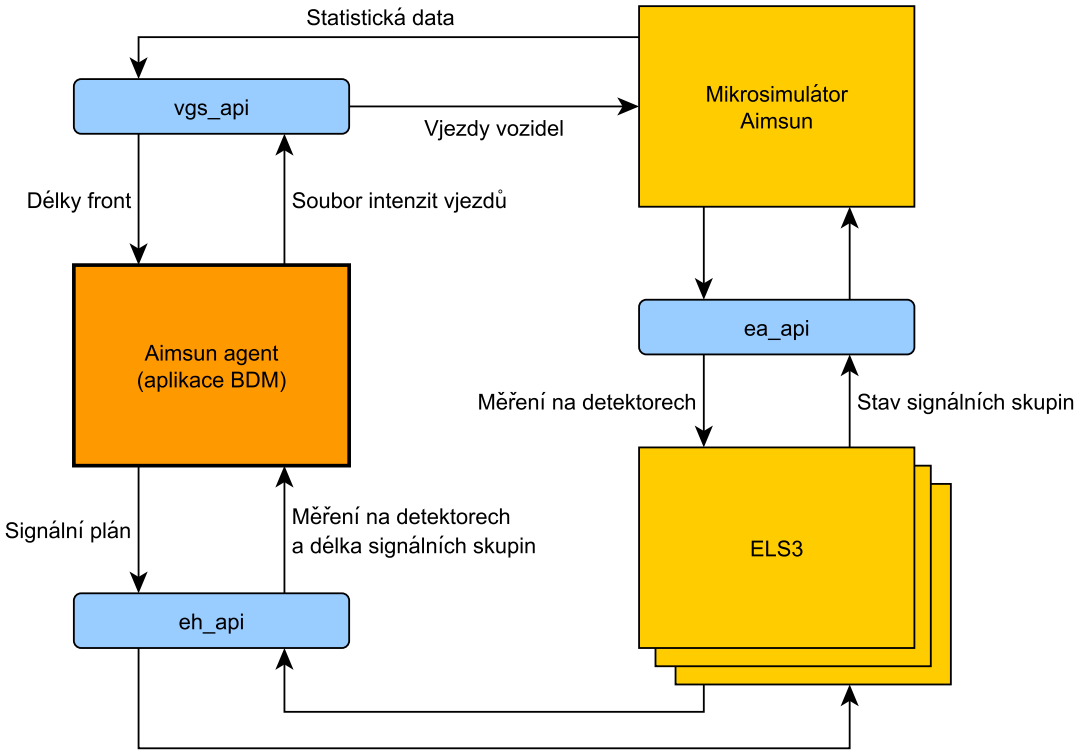
Aimsun agent představuje řídící prvek celého systému. Přes VGS API spouští Aimsun a stejným způsobem od něj průběžně přijímá hodnoty sledovaných dopravních veličin. Dále přes `eh_api` získává údaje z řadičů křižovatek. Na základě všech sesbíraných údajů pak sestavuje pokyny pro změny v signálních plánech a opět pomocí `eh_api` tyto pokyny zasílá řadičům.

Na závěr komunikace mezi mikrosimulátorem a emulátory řadičů probíhá přes `ea_api`.

3.5 Program `main_loop.exe`

(!upravit podle obrazku!)

Simulace je obsluhována z programu *main_loop.exe*. Ten se stará o načtení konfigurace, spuštění simulátoru Aimsun a zajišťuje komunikačního prostředníka mezi jednotlivými agenty. Diagram jeho fungování je na obrázku 3.3. Program se spouští



Obrázek 3.2: Struktura celé simulace

s jedním parametrem, který představuje jméno konfiguračního souboru. Konfigurační soubor používá formát, který načítá knihovna `libconfig`.

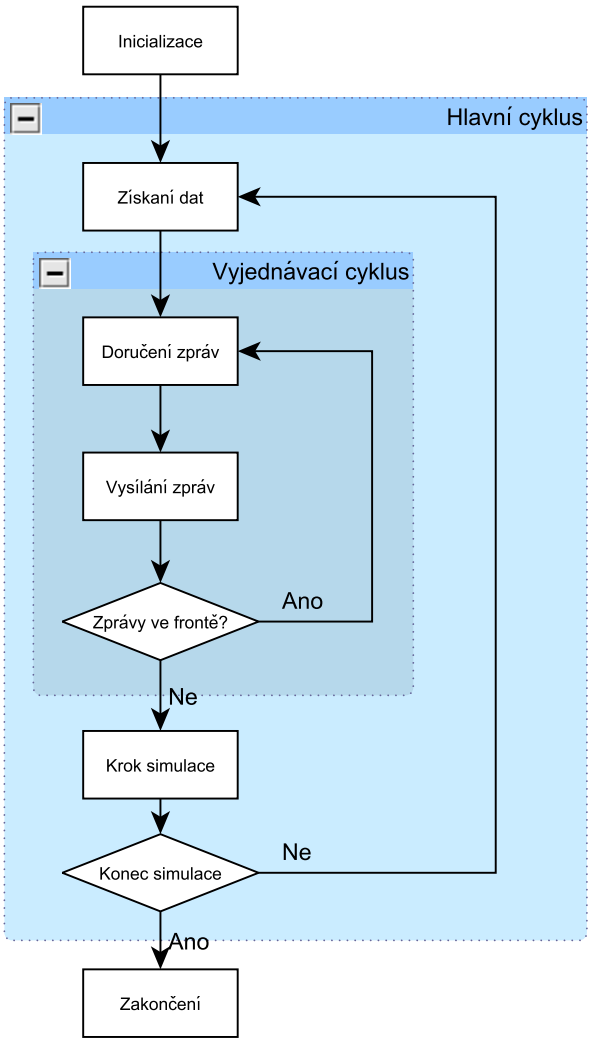
Program funguje tak, že na začátku načte konfigurační soubor. Pokud konfigurační soubor není zadán, program skončí s chybou. Po načtení je spuštěn simulátor Aimsun s parametry zapsanými ve skupině `system`. Jde především o intenzitu provozu na vstupech do dopravní sítě a délku simulace.

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.

(...)

Pak již následuje ústřední for cyklus. Na jeho začátku se zapíší údaje do logu. Poté jsou přečtena aktuální data ze simulátoru a předána agentům ke zpracování funkcí



Obrázek 3.3: Strukuta fungování programu `main_loop.exe`, podrobněji popsaná v podkapitole 3.5

`adapt()`. Pokračuje se vyjednávacím cyklem.

Vyjednávací cyklus se zabývá obsluhou fronty zpráv. Nejprve frontu prohledá frontu zpráv a příslušným agentů předá jim určené zprávy zavoláním funkce `recieve()`. V případě, že nemožnosti doručit oznámí varování, ale pokračuje dál v práci. Nakonec všichni agenti mají možnost nějaké zprávy do fronty přidat – k tomuto účelu existuje funkce `broadcast()`. Tato smyčka je ukončena ve chvíli, kdy žádný z agentů nevyšle žádnou zprávu nebo pokud se zopakuje tolikrát, kolik je nastavený limit v proměnné `max_cycles`.

Po ukončení vyjednávání je u každého agenta zavolána funkce `act()`. To je oka-

mžik, kdy mohou agenti ovlivnit řízení signalizace. Práce 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 funkce `step()` u loggeru `L`, `Ds` a u všech agentů.

Tato smyčka se opakuje dokud není dosažen v konfiguračním souboru nastavený čas simulace. Na závěr je zapsán log soubor a celá simulace je ukončena.

3.6 Implementace navrženého algoritmu

Kapitola 4

Výsledky simulací

Závěr

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] 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.

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

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

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

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

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

Přílohy