České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

BAKALÁŘSKÁ PRÁCE

Decentralizované řízení dopravní signalizace: nastavení délky cyklu

Vypracoval: Jakub Novotný
Vedoucí práce: Ing. Václav Šmídl, Ph.D.
Konzultant: Dr. Ing. Jan Přikryl, Ph.D.

Praha, 2010

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

Praha, 8. července 2010 Jakub Novotný

Rád bych poděkoval vedoucímu práce, Ing. Václavu Šmídlovi, Ph.D. a odbornému konzultantovi, Dr. Ing. Janu Přikrylovi, Ph.D. za odborné vedení a nemalé úsilí vynaložené při tvorbě této práce.

Název práce:

Decentralizované řízení dopravní signalizace: nastavení délky cyklu

Autor: Jakub Novotný

Obor: Inženýrská informatika

Zaměření: Tvorba softwaru

Druh práce: Bakalářská práce

Vedoucí práce: Ing. Václav Šmídl, Ph.D.

Konzultant: Dr. Ing. Jan Přikryl, Ph.D.

Abstrakt:

V této práci je popsán návrh a implementace algoritmu na řízení dopravy pomocí nastavení délky cyklu řadiče. Tento algoritmus je navržen podle teorie multiagentních systémů a otestován na mikrosimulátoru dopravy AIMSUN. V závěru jsou vyhodnoceny výsledky srovnáním s pevnou referenční hodnotou.

Klíčová slova:

multiagentní systémy, decentralizované řízení, řízení dopravy, AIMSUN

Title:

Decentralized control of traffic lights: setting of cycle time

Author: Jakub Novotný

Abstract:

This work represents design and implemetantion of algorithm, which purpose is to control the traffic lights by setting length of cycle time. This algorithm is designed according to the theory of multiagent systems and tested on microsimulator of traffic AIMSUN. In the end of this work, there are results of tests, compared with fixed reference value of time cycle.

Key words:

multiagent systems, decentralized control, control of traffic lights, AIMSUN

Zadání práce s podpisem děkana

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Stav dopravní situace ve městě bezesporu ovlivňuje život každého z nás a do jeho zlepšení, ať už máme na mysli výstavbu a zlepšování dopravních komunikací, nebo teoretický výzkum, jsou v dnešní době investovány nemalé finanční prostředky. Optimalizované řízení provozu je jednou z možností, jak odlehčit dopravě bez nutnosti zasahování do fyzického stavu silnic.

V současné době se spoléhá hlavně na ruční nastavení parametrů křižovatky z centrály na základě dat z detektorů. Účelem této práce je prozkoumat možnosti řízení novým způsobem, a to decentralizovaným řízením multiagentním systémem. Pomocí programu v C++, kde jsou agenti ovládající křižovatky reprezentováni objektem, a komunikací přes knihovny BDM s mikrosimulátorem dopravy AIMSUN, budeme testovat možnosti výměny údajů, nalezení optimální strategie a změny dopravní situace na křižovatkách.

Nejprve popíšeme mikrosimulátor AIMSUN, použité knihovny a používané parametry. Poté si přiblížíme některé důležité prvky z teorie multiagentních systémů.

V druhé části práce popíšeme běh celého programu, způsob implementace třídy agenta, zpracování údajů a volbu nejvhodnější strategie. Na závěr vyhodnotíme získané výsledky.

Kapitola 2

AIMSUN

AIMSUN (Advanced Interactive Microscopic Simulator for Urban and Non-Urban Networks), je mikrosimulátor dopravy. Podstatou mikroskopické simulace (mikrosimulace) je modelování jízdy jednotlivých vozidel po dané komunikační síti, přičemž se zohledňují všechny parametry infrastruktury i dopravních prostředků, a to včetně chování řidiče. V této kapitole je popsána základní charakteristika mikrosimulátoru AIMSUN. Podrobnější informace lze nalézt například v [3] nebo [4].

2.1 Vstupní data pro AIMSUN

AIMSUN potřebuje pro svůj běh simulační scénář a množinu simulačních parametrů, které definují experiment. Scénář se skládá ze čtyř druhů dat: popis dopravní sítě, plány řízení dopravy, požadovaná dopravní data, a plány hromadné dopravy.

2.1.1 Scénář

Popis dopravní sítě zahrnuje geometrii sítě, popis křižovatek a rozmístění detektorů, které jsou rozmístěny podél dopravní sítě.

Plány řízení odpravy obsahují fáze a jejich délky pro křižovatky, které jsou řízeny dopravními světly. V každé fázi je definováno která signální skupina je průjezdná.

Dopravní data se dají zadat dvěma způsoby:

- Pomocí objemů dopravy v určitých místech dopravní sítě, poměrů odbočení a počátečního stavu

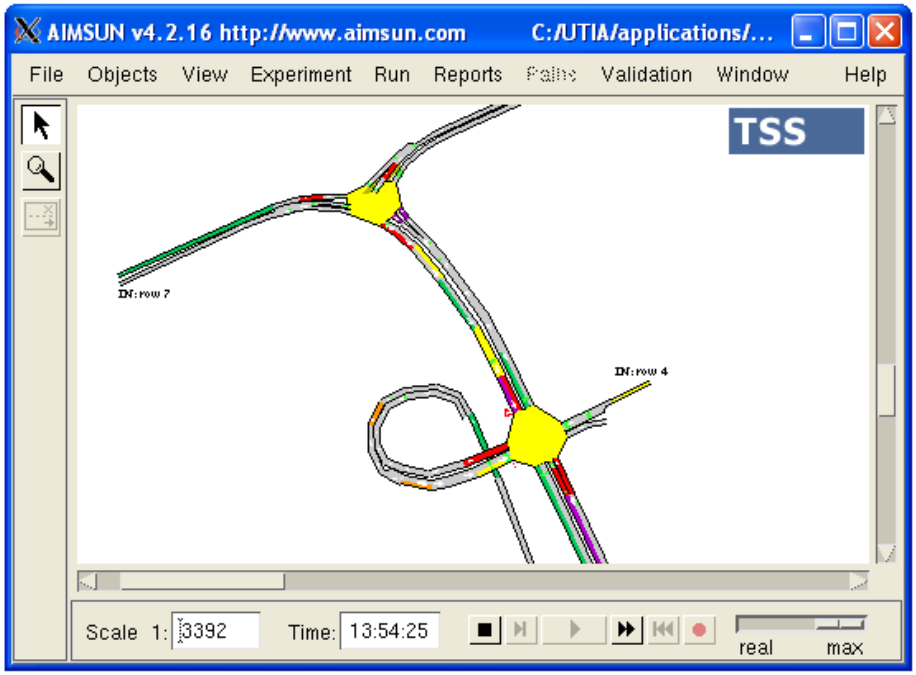
- Pomocí matice, kde prvek na i-tém řádku a v j-sloupci udává kolik jízd se uskuteční z místa i do místa j

V prvním případě se vozidla rozmístí stochasticky podle požadovaných počtů a poměrů odbočení do dopravní sítě, v případě druhém je každému vozidlu přiřazena trasa z místa i do místa j.

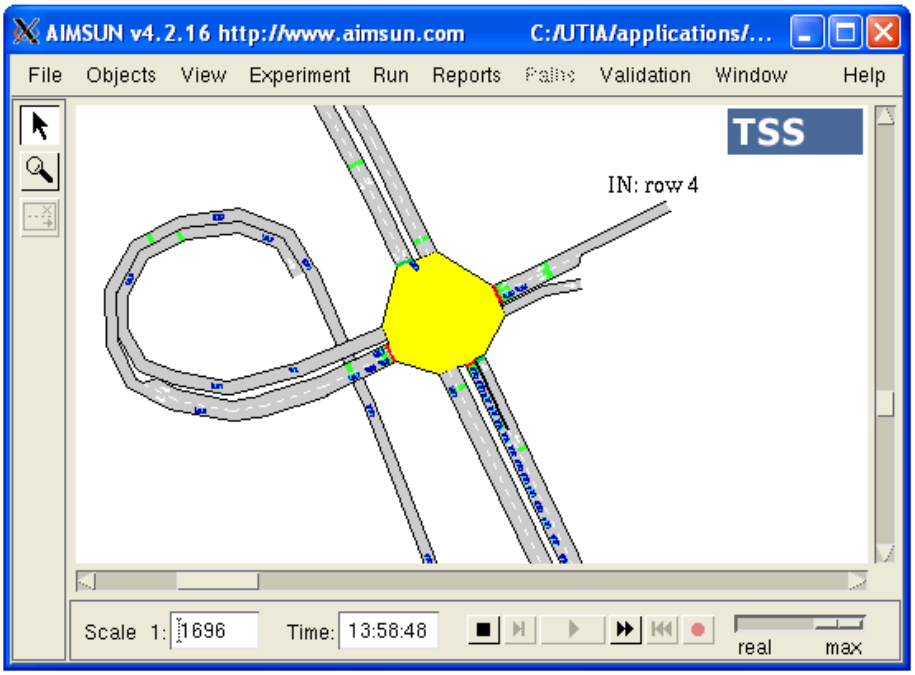
Plány hromadné dopravy obsahují linky autobusů, jejich zastávky a jízdní řády.

2.1.2 Výstupní data AIMSUNu

AIMSUN poskytuje jak statistické výstupy v podobě hodnot doby průjezdu vozidla sítí, počtu zastavení, zpoždění a průměrné rychlosti, tak výstupy z detektorů - počet vozidel, rychlost a doba stání na detektoru, tak i plynule modelovaný grafický výstup.



Obrázek 2.1: Grafický výstup ze simulátoru AIMSUN



Obrázek 2.2: Grafický výstup ze simulátoru AIMSUN - detail

2.1.3 VGS API

VGS API je rozhraní napsané v jazyce C++, které rozšiřuje funkčnost mikrosimulátoru AIMSUN. Jeho dva nejdůležitější úkoly jsou zjednodušení reálné simulace a statistické zpracování výstupních dat.

2.1.3.1 Reálná simulace

AIMSUN umožňuje simulaci reálné dopravní sítě, kdy hodnoty hustoty dopravy odpovídají skutečně naměřeným hodnotám v reálném prostření simulované sítě. Běžně se používají ručně sečtené hodnoty v intervalech jedné hodiny. Údaje se potom vkládají opět ručně do simulátoru.

Pro přesnější simulaci se používají data z detektorů v simulované oblasti. Objem dat je ale pro ruční vkládání neúnosně velký. Proto bylo vyvinuto VGS API, které umožňuje generování vozidel v průběhu simulace. VGS API se předá soubor s uloženými údaji z detektorů, to automaticky rozběhne AIMSUN, do kterého generuje v průběhu simulace vozidla podle reálných dat.

2.1.3.2 Zpracování dat

Druhým důležitým úkolem VGS API je shromažďovat data potřebná pro vyhodnocení experimentu. AIMSUN sice disponuje jednoduchým rozhraním pro vizualizaci dat a jejich export do textových souborů, není ale možné například porovnávat jednotlivé scénáře simulací. VGS API proto periodicky ukládá všechny klíčové ukazatele jak pro jednotlivé segmenty dopravní sítě, tak i pro celý simulovaný systém. Uživatel má po ukončení simulace k dispozici údaje o počtu zastavení vozidla, o jeho zpoždění, průměrné rychlosti, době jízdy a době stání, o dopravím toku a hustotě dopravy na jednotlivých segmentech.

Jedním z nejpodstatnějších údajů, které nám VGS API poskytuje, je délka fronty pro daný jízdní pruh. Tento údaj je prakticky nemožné určovat podle dat z detektorů, podstatně zkreslených chybami dvojího typu. Jednak vzniká chyba při přejezdu vozidla z pruhu do pruhu v oblasti detektorů, kdy jedno vozidlo zaznamenají oba detektory. Vozidlo by se potom přičetlo do obou front. K druhému případu chyby dochází, pokud je mezi dvěma vozidly tak malý rozestup, že je detektor považuje za jedno. V tomto případě naopak dochází k samovolnému vytrácení vozidel. Pokud bychom chtěli počítat délku fronty jako rozdíl počtů vozidel, která do křižovatky přijela a která ji opustila, docházelo by ke kumulaci této chyby v průběhu simulace. Hodnota délky fronty se ve VGS API získá sečtením vozidel s menší rychlostí než 3,6 km/h po segmentech jízdního pruhu.

2.1.4 Řadiče

K řízení signálních skupin křižovatky se používá tzv. řadič. V reálném případě se jedná o počítač napojený na dopravní ústřednu, signální skupiny křižovatky a její detektory. v případě simulace je použit emulátor řadiče ELS3 firmy ELTODO. Ten má implementován pouze algoritmus řízení. O simulaci detektorů a přepínání signálních skupin se stará AIMSUN.

Parametry křižovatky (signální skupiny, fáze, detektory, ...) a dopravní vztahy nad těmito parametry (signální plány, dynamické řízení, ...) se načítají na začátku simulace z konfiguračního souboru. Jsou součástí dopravního návrhu a v průběhu simulace se nemění. Přenastavují se pouze vnější parametry.

Vstupní a výstupní data se do řadiče načítají přes komunikační API. Na začátku každého

simulačního kroku se z AIMSUNu předají údaje z detektorů. Po zpracování dat program určí hodnotu vnějších parametrů pro řízení křižovatky, a na konci kroku simulace je předá řadiči pomocí knihovny BDM, která zastává roli dopravní ústředny v reálném případě.

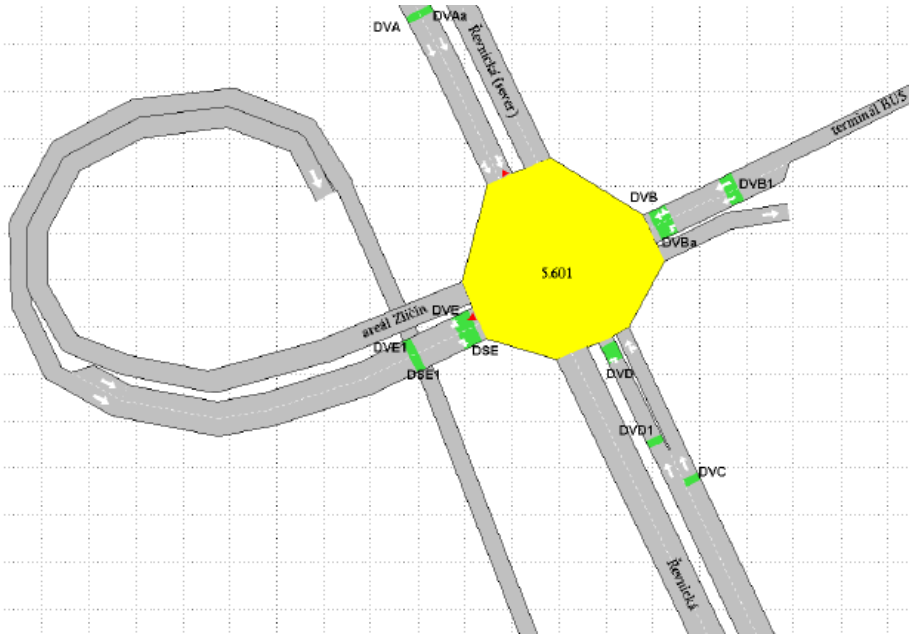
Vnějšími parametry jsou

- délka cyklu - čas, za který se vystřídají všechny fáze
- offset - posunutí začátku cyklu oproti globálnímu času

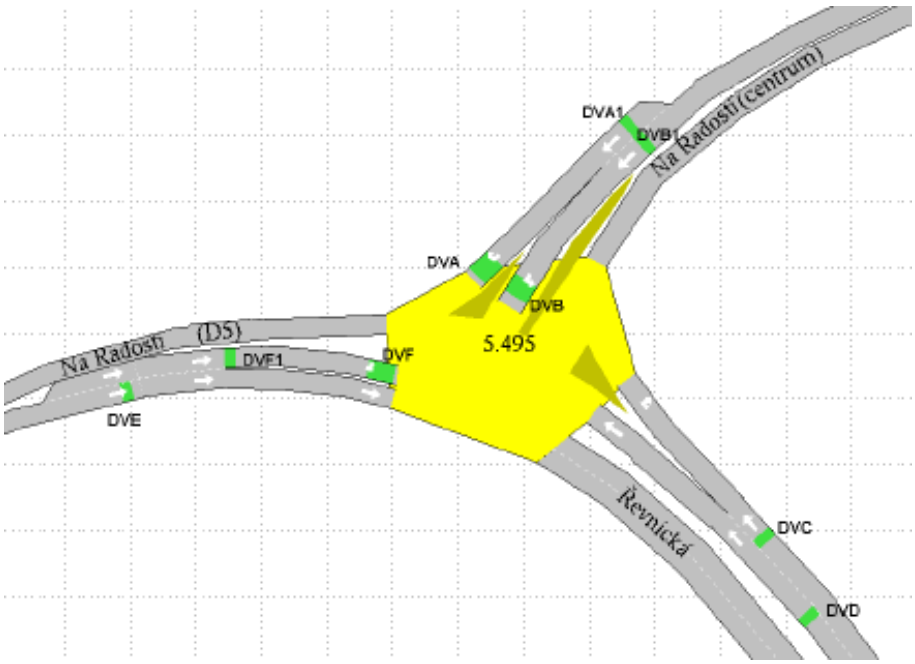
Úkolem této práce je řídit provoz pouze pomocí nastavení délky cyklu. To změní i délky jednotlivých fází, jejichž součet se musí délce cyklu rovnat, neovlivní však jejich poměr, který je konstantní. To znamená, že se nezmění ani poměr doby průjezdnosti a neprůjezdnosti dopravních pruhů.

2.1.5 Oblast simulace

Pro simulaci bylo použito schéma dvou křižovatek na ulici Řevnické, sestavené podle skutečné situace. Následující schémata znázorňují křižovatky s označením 495 - Na Radosti a 601 - terminál BUS, jejich pruhy (VA, VB, VC, VD, VE, VF a VA, VAa, VB, VBa, VC, VD, VE, Se) a detektory, znázorněné zelenými obdélníky.



Obrázek 2.3: Křižovatka 601



Obrázek 2.4: Křižovatka 495

Kapitola 3

Multiagentní systémy

3.1 Úvod

Multiagentní systém je druh distribuované umělé inteligence. Tento systém se skládá z jednotlivých výpočetních prvků, tzv. agentů, které musí mít dvě základní schopnosti. Zaprvé musí být schopni autonomní akce rozhodnutí - zjistit jak nejlépe dosáhnout požadovaných cílů - a zadruhé je to schopnost interakce s ostatními agenty. V druhém případě nejde jen o pouhou výměnu dat, ale o typ kolektivní aktivity - návrh, potvrzení, odmítnutí.

3.1.1 Historie

Multiagentní systémy jsou na poli počítačové vědy relativní novinkou. Studium tohoto tématu probíhá od začátku osmdesátých let dvacátého století. Větší pozornosti se jim dostalo v polovině let devadesátých s rozvojem internetu.

3.1.2 Agent

Neexistuje obecně uznávaná definice agenta. Přikloníme se k definici použité v publikaci [6].

Definice 3.1 (Agent): Agent je počítačový systém umístěný do nějakého prostředí, který je schopen autonomní akce k přiblížení se navrženým cílům. ▶

Agent v naprosté většině případů nemá celkovou kontrolu nad prostředím. Prostředí může ovlivňovat jen částečně. Obecně, stejně jako v našem případě, je prostředí nedeterministické. Stejná akce provedená dvakrát za sebou nemusí vést ke stejnému výsledku.

3.2 Druhy prostředí

Způsob práce agentů se liší podle druhu prostředí, ve kterém pracují. Podle [6] se prostředí dají klasifikovat následovně:

- Deterministické vs. nedeterministické
- Dostupné vs. nedostupné
- Statické vs. dynamické

Deterministické prostředí je takové, ve kterém má každá jednotlivá akce předem daný efekt. Prostředí je dostupné, pokud agent může zjistit jeho úplný stav v kteroukoliv dobu. Statické prostředí se na rozdíl od dynamického mění pouze vlivem akcí vyvolanými agenty. V diskrétním prostředí existuje pevné konečné číslo možných vjemů a akcí.

V našem případě je prostředí nedeterministické (agent pouze odhaduje vliv přenastavení parametru), nedostupné (hodnoty se měří pouze jednou za 90 sekund, a ještě jsou zkreslené) a dynamické (intenzita dopravy se mění nezávisle na akcích agenta). Je zřejmé, že prostředí s těmito vlastnostmi znesnadňuje rozhodování a kontrolu vyvolaného výsledku.

3.3 Interakce agentů

3.3.1 Stavy prostředí a preference agentů

Mějme pro jednoduchost 2 agenty. Označme si je i a j. Předpokládejme, že máme množinu

$$\Omega = \{\omega_1, \omega_2, ...\}$$

obsahující všechny možné stavy prostředí, v kterém agenti operují. Aby byl agent schopen efektivně ovlivňovat prostření, musí být schopen ohodnotit, jak je pro něj daný stav příznivý. Hodnocení daného stavu agenta i a j formálně definujeme jako funkce

$$u_i : \Omega \to \mathbb{R},$$

$$u_j : \Omega \to \mathbb{R}.$$

Čím je stav ω příznivější pro agenta i, tím je větší hodnota funkce u_i.

Definice 3.2 (Uspořádání na množině všech stavů): Mějme 2 stavy prostředí ω_1, ω_2. Řekněme, že stav ω_1 je preferován agentem i nad stavem ω_2, pokud platí $u_i(\omega_1) \geqq u_i(\omega_2)$. Značíme

$$\omega_1 \succeq_i \omega_2.$$

Stav ω_1 je silně preferován agentem i nad stavem ω_2, pokud platí $u_i(\omega_1) > u_i(\omega_2)$. Značíme

$$\omega_1 \succ_i \omega_2$$

Relace $\succeq_i$ je zřejmě uspořádání, protože má všechny potřebné vlastnosti.
Reflexivitu:

$$\forall \omega \in \Omega : \omega \succeq_i \omega$$

Tranzitivitu:

$$\forall \omega_1, \omega_2, \omega_3 \in \Omega : \omega_1 \succeq_i \omega_2 \wedge \omega_2 \succeq_i \omega_3 \Rightarrow \omega_1 \succeq_i \omega_3$$

Porovnatelnost:

$$\forall \omega_1, \omega_2 \in \Omega : \omega_1 \succeq_i \omega_2 \vee \omega_2 \succeq_i \omega_1$$

Relace $\succ_i$ zjevně nesplňuje podmínky reflexivity.

3.3.2 Akce agentů

Nyní popíšeme, jak mají agenti možnost ovlivňovat prostředí. Opět předpokládejme existenci dvou agentů i a j. Obecně mají různí agenti různou oblast působnosti. Množina

$$A = \{a_1, a_2, ...\}$$

znázorňuje množinu všech akcí, které jsou agenti schopni provézt. Na tyto akce reaguje prostředí přechodem do nějakého stavu $\omega \in \Omega$. Formálně můžeme tento přechod zapsat jako funkci

$$\tau : A \times A \to \Omega.$$

3.4 Strategie

Popišme zde způsob, jak se agent rozhodne pro realizaci určité akce. Agent je nyní schopen ohodnotit, který stav prostředí je pro něj příznivější než jiný, neví však jak budou reagovat ostatní agenti, není schopen určit tudíž, i za předpokladu, že by systém byl

deterministický, do jakého stavu systém přejde. K výběru optimální akce se používají prvky z teorie her. Zadefinujme nejprve v souladu s touto teorií základní pojmy.

Definice 3.3 (Dominance množiny): Mějme 2 podmnožiny $\Omega_1, \Omega_2 \subset \Omega$. Řekneme, že Ω_1 je pro agenta i dominantní nad množinou Ω_2, pokud platí

$$\forall \omega \in \Omega_1, \forall \omega' \in \Omega_2 : \omega \succeq_i \omega'.$$

Řekneme že Ω_1 je pro agenta i silně dominantní nad množinou Ω_2, pokud platí

$$\forall \omega \in \Omega_1, \forall \omega' \in \Omega_2 : \omega \succ_i \omega'.$$

Abychom používali terminologii teorie her, budeme nyní akce $a_i \in A$ jednotlivých agentů nazývat strategiemi.

Definice 3.4 (Množina výsledků): Nazvěme množinu všech stavů, do kterých může prostředí přejít při hraní strategie $a_i \in A$, množinou možných výsledků. Označme ji

$$a_i^* \subset \Omega.$$

Definice 3.5 (Dominance strategie): Řekneme, že strategie a_i je dominantní nad strategií a_j, pokud je množina a_i^* dominantní nad množinou a_j^*. Strategie a_i je silně dominantní nad strategií a_j, pokud je množina a_i^* silně dominantní nad množinou a_j^*. ▶

Racionálně uvažující agent tedy vyloučí všechny strategie a_i, jestliže existuje strategie a_j, která nad strategií a_i silně dominuje. K zúžení výběru zbývajících strategií slouží Nashova rovnost. Pro zjednodušení uvažujme 2 agenty, i a j. Dvě strategie, a_1 a a_2 jsou v Nashově rovnosti, pokud za předpokladu že agent i zvolí strategii a_1, je nejvýhodnější strategií pro agenta j je strategie a_2 a zároveň pokud agent j zvolí strategii a_2, je pro agenta i nejvýhodnější strategií a_1.

3.4.1 Použití pro výběr délky cyklu

Délka cyklu řadiče křižovatky je parametr, který je pro všechny agenty ve skupině zahrnující křižovatky do zelené vlny společný. Nesmí tedy dojít k situaci, kdy by každý agent nastavil jinou délku cyklu. Množina strategií $A = \{a_1, a_2, ...\}$ je tedy v našem případě množinou všech nastavitelných délek cyklu Tc_k v dané situaci. Mějme funkci $u_{i_{Tc}} : A \to \mathbb{R}$, jejíž funkční hodnota v bodě Tc_j udává předpokládaný zisk pro agenta i

při délce cyklu Tc_j. Funkci $u_i(\omega) = u_i(\tau(Tc_k, Tc_l))$ můžeme zadefinovat následovně (pro jednoduchost předpokládejme existenci dvou agentů i a j)

$$u_i(\tau(Tc_k, Tc_l) = \begin{cases} u_{i_{Tc}}(Tc_k), & k = l \\ -\infty, & k \neq l \end{cases},$$

kde hodnota $-\infty$ vyjadřuje jakýsi kolaps systému při nastavení různých délek cyklu. To však znamená, že žádná strategie není silně dominantní nad jinou. Zároveň za předpokladu, že agent i zvolí strategii Tc_l, agent j nemůže udělat lépe, než že zvolí stejnou strategii. To znamená že pro všechny strategie $Tc_l \in A$ platí, že jsou v Nashově rovnosti samy se sebou. Takto definovaná funkce nám v souladu s teorií zaručí, že agenti nenastaví různé délky cyklu, potřebujeme ale dodatečné kritérium kterou délku cyklu vybrat.

3.4.2 Globálně nejlepší řešení

Komunikace agentům dovoluje vzájemně si předat předpokládané zisky pro určitou délku cyklu. Nejprve musíme určit množinu možných délek cyklu tak, aby v každém kroku simulace byla pro všechny agenty společná. Určíme tedy přirozená čísla n a dTc. Množina možných strategií A bude

$$A = \{Tc_{-n}, Tc_{-n+1}, ..., Tc_{n-1}, Tc_n\},$$

$$Tc_i = Tc + i \cdot dTc,$$

kde Tc je délka cyklu v minulém kroku simulace. Každý agent je pak schopen sestavit množinu součtů zisků

$$U = \{u_{Tc_{-n}}, u_{Tc_{-n+1}}, ..., u_{Tc_{n-1}, u_{Tc_n}}\},$$

kde

$$u_{Tc_i} = \sum_k u_{k_{Tc}}(Tc_i).$$

Agent poté zvolí takovou délku cyklu Tc_i, pro kterou platí

$$u_{Tc_i} = \max(U).$$

Toto je výběr globálně nejlepšího řešení, kde agent upřednostní takový čas délky cyklu, u kterého se předpokládá největší součet zisků od všech agentů nad časem, u kterého předpokládá největší zisk pro sebe.

3.4.3 Rozšíření

V našem případě se zabýváme pouze dvěma křižovatkami. V případě většího počtu křižovatek patřících do různých skupin zelené vlny se bude optimalizace provádět pro každou skupinu zvlášť. Jak dále popíšeme v 4.3.2, pro danou skupinu je důležitý i údaj o délce cyklu okolních agentů, neboť ovlivňuje předpoklad o hustotě provozu. Komunikace se rozšíří o tento údaj a každá skupina ho zahrne do výběru strategie.

Kapitola 4

Popis implementace

4.1 Použité knihovny

4.1.1 IT++

IT++ je knihovna jazyka C++ zahrnující třídy pro matematické výpočty a komunikaci. Umožňuje práci s vektory a maticemi podobným způsobem jako v nástroji MATLAB. Použity jsou třídy `vec` a `Array`.

4.1.1.1 Třída vec

Třída `vec` reprezentuje vektor hodnot číselného typu `double`. Přetížením operátorů kulatých a hranatých závorek a operátoru `=` umožňuje jednoduchou inicializaci vektoru pomocí textového řetězce a zjednodušuje přístup k jednotlivým hodnotám. Kód

```
1  vec a = "0.1 0.2 0.3";
2  cout << a(0) << endl;
3  cout << a[1] << endl;
```

inicializuje vektor o dimenzi 3 s hodnotami 0,1, 0,2 a 0,3 a poté vypíše první a druhý prvek vektoru.

4.1.1.2 Třída Array

Tato šablonová třída umožňuje pracovat s polem o proměnných předem daného libovolného typu. Rozměr pole se může v průběhu programu měnit.

4.1.2 BDM

Knihovna BDM (Bayesian Decision Making) je navržena pro programy používající Bayesovu statistiku. Zde jsou použity hlavně třídy `Datalink`, `RV`, `UI` a `Setting` sloužící pro komunikaci s mikrosimulátorem AIMSUN, načítání hodnot z konfiguračních souborů a komunikaci mezi agenty.

V knihovně BDM je také definována třída `Participant`, implementující metody pro komunikaci , od které je odvozena třída našeho agenta.

4.1.2.1 Třída RV

Třída `RV` implementuje pole popisků k vektorům třídy IT++. K přístupu k jednotlivým prvkům slouží přetížený operátor závorky `(int index)`. K určení veličiny uložené ve vektoru se používají metody

name jméno veličiny uložené v příslušném vektoru

length rozměr RV vektoru

size(i) rozměr veličiny, kterou značí popisek na pozici `i`

4.1.2.2 Třídy UI a Setting

UI a Setting slouží mimo jiné k načtení vybraných dat z konfiguračních souborů. Kód

```
1    UI::get(inputs,set,"inputs",UI::compulsory);
```

zapíše na adresu proměnné `inputs` hodnotu v objektu `set` třídy `Setting`, označenou jako ¨`inputs`¨. Objekt `set` reprezentuje načtený konfigurační soubor. `UI::compulsory` značí, že daná hodnota musí být zadána.

4.1.2.3 Třída Datalink

Tato třída definuje spojení mezi dvěma vektory, mezi kterými zprostředkovává spojení pomocí metody `filldown`.

Následující kód popisuje vytvoření dvou vektorů `from` a `to`, popsaných `RV` vektory `rv_from` a `rv_to`, a zkopírování odpovídajících údajů z `from` do `to` pomocí třídy `Datalink`. Výsledkem je, že vektor `to` bude obsahovat hodnoty 1 a 2.

```
1  RV a = RV ( "{ a }");
2  RV b = RV ( "{ b }" );
3  RV c = RV ( "{ c }" );
4
5  RV rv_to = a;
6  rv_to.add ( b );
7
8  RV rv_from = rv_to;
9  rv_from.to ( c );
10
11 datalink dl ( rv_to, rv_from );
12 vec from ( "1 2 3" );
13 vec to = dl.pushdown ( total );
```

4.2 Třída Lane

Každá instance třídy Lane reprezentuje jeden dopravní pruh. Po inicializaci se do příslušných proměnných uloží textové řetězce, podle kterých se přiřazují k pruhu data z detektorů a AIMSUNu, jako je například maximální délka fronty na dopravním pruhu za jednu simulační periodu.

4.3 Třída LaneHandler

Tato třída zprostředkovává agentovi přístup k údajům z jednotlivých pruhů. Stará se o zaznamenávání a statistické zpracování dat z AIMSUNu. Také je zde implementován odhad čekací doby automobilu do průjezdu křižovatkou, což je údaj používaný v hodnotící funkci délky cyklu agenta.

4.3.1 Fronta

Agent předává třídě LaneHandler údaj o maximální délce fronty za vzorkovací periodu. Jedná se o největší počet automobilů v příslušném jízdním pruhu, jejichž rychlost nepřesahuje určitou hodnotu. V našem konkrétním případě je tato mezní rychlost nastavena na 3,6 km/h. Tento údaj je však pro naše potřeby zkreslený, a to hlavně tím, že ve většině případů se délka cyklu neshoduje se vzorkovací periodou. Uvažujme jízdní pruh s poměrem délky zelené 0,5. Pokud bude například délka cyklu 180 sekund, a budeme li předpokládat, že začátek jedné periody se bude shodovat s časem, kdy na příslušném semaforu naskočí zelená, bude v první vzorkovací periodě pruh průjezdný po dobu 45 sekund, ale v další nebude průjezdný vůbec a fronta se za stejné hustoty provozu zvětší.

4.3.2 Odhad fronty

K odhadu fronty je použito metody tzv. exponenciálního zapomínání. Metoda vychází z plovoucího průměru, který je vhodný pro odhad nekonstantní hodnoty. Označíme si veličinu $\overline{q_i}$ jako plovoucí průměr fronty v kroku i, kam budeme ukládat posledních n měření. Označme si velikost fronty naměřenou v i-tém kroku jakoq_i. V kroku $i+1$ je hodnota plovoucího průměru před přičtením poslední naměřené hodnoty rovna

$$\overline{q_i} = \frac{1}{n} \sum_{j=0}^{n-1} q_{i-j}.$$

Obsahuje tedy všech posledních n naměřených hodnot. $\overline{q_{i+1}}$ se tedy rovná

$$\overline{q_{i+1}} = \frac{1}{n} \left(n\overline{q_i} - q_{i-n+1} + q_{i+1}\right) = \overline{q_i} - \frac{1}{n} q_{i-n+1} + \frac{1}{n} q_{i+1}.$$

Abychom nemuseli ukládat posledních n měření, zjednodušíme algoritmus tím, že místo kroku $i-n+1$ odečteme od průměru jeho hodnotu podělenou délkou průměrovacího okna n. Výpočet přejde na tvar

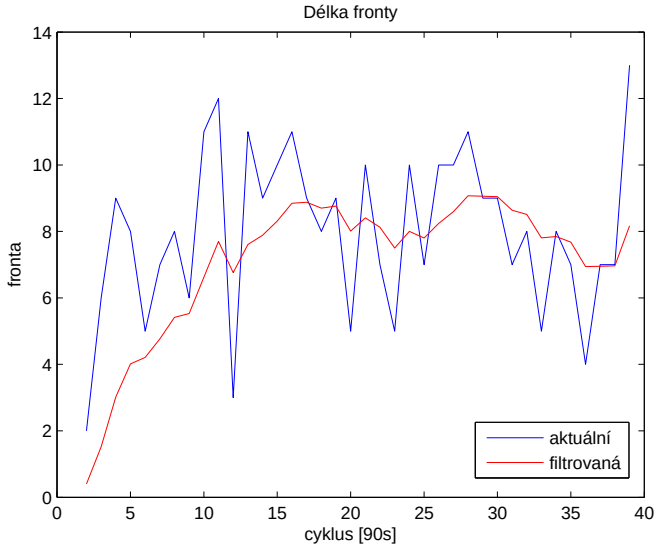
$$\overline{q_{i+1}} = \frac{1}{n} \left(n\overline{q_i} - \overline{q_i} + q_{i+1}\right) = \overline{q_i} + \frac{1}{n} \left(q_{i+1} - \overline{q_i}\right).$$

$\frac{1}{n} \left(q_{i+1} - \overline{q_i}\right.$ je vlastně přírůstek v kroku $i+1$, označme si ho jako Δq_{i+1}, a faktor $1/n$ můžeme chápat jako jeho váhu w. Po dosazení dostáváme jednoduchý tvar

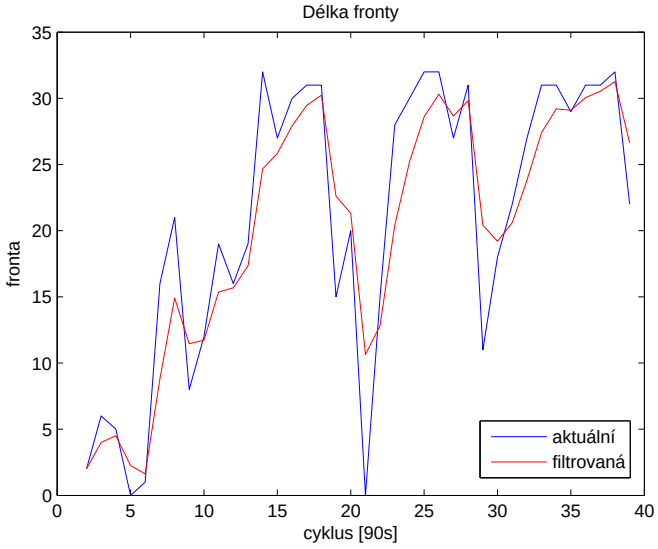
$$\overline{q_{i+1}} = \overline{q_i} + w\Delta q_{i+1}.$$

Průměr s exponenciálním zapomínáním se tak nazývá proto, že je v něm v i-tém kroku uložena i první hodnota, má však zanedbatelnou váhu $\left(\frac{n-1}{n}\right)^i$. Jedná se o jaksi zjednodušenou verzi Kalmanova filtru. Grafy 4.1 a 4.2 znázorňují rozdíl mezi okamžitou a

filtrovanou délkou fronty při váze $w = 0.2$. Graf 4.1 pro konstantní délku cyklu 80 sekund, graf 4.2 pro délku cyklu periodicky se měnící od 40 do 120 sekund.



Obrázek 4.1: Porovnání okamžité a filtrované délky fronty
- délka cyklu 80 s



Obrázek 4.2: Porovnání okamžité a filtrované délky fronty
- délka cyklu 40-120 s

V reálném případě je potřeba vypočítat délku fronty podle záznamů z detektorů.

Prostý výpočet délky fronty například jako rozdíl počtu vozidel, která odjela, a která přijela však není možný, v důsledku problémů popsaných v 2.1.3.2. Výpočet délky fronty není předmětem zkoumání této práce. Podrobnější pojednání o tomto problému lze najít například v [1].

Protože časté přenastavování délky cyklu působí řadičům problémy, přenastavuje se tento parametr jednou za časový úsek, který je několikanásobně delší než 90 sekund. Je tedy nutné modelovat časový průběh fronty. Při rozšiřování testovacího prostředí tento model ovlivní i údaj o počtu vozidel, které přijedou ze sousední křižovatky. Tento údaj závisí na délce cyklu a není agentovi dostupný přímo. Bude tedy nutné rozšířit komunikaci agentů, popsanou v 4.5.3.3, o tento údaj.

4.3.3 Odhad čekací doby

Jednou z nejdůležitějších částí logiky, implementované v třídě `LaneHandler` je odhad čekací doby vozidel stojících v, nebo přijíždějících do příslušného jízdního pruhu. Tento odhad se používá při výběru a ohodnocení délky cyklu.

Odahad se provádí zjednodušenou mikrosimulací po čas `T`. Do proměnné `q` se uloží aktuální hodnota fronty. Simulace začíná v čase `t = 0`. V každém kroku simulace se zvětší fronta o hustotu provozu (počet aut za sekundu), odhadovanou z údajů z detektorů, a pokud se čas nachází ve průjezdné fázi, zmenší se o konstantu, nazývanou saturovaný tok, která udává maximální počet aut, které projedou křižovatkou. K čekací době vozidel, uložené v proměnné `wt`, se potom přičte délka zbývající fronty a čas `t` se zvětší o jednu sekundu.

Odhad probíhá ve dvou fázích, realizovaných cykly typu `for`. V první běží čas do největší části `T`, soudělné s `tc`, v druhé po zbytek `T`. Níže uvádím popis všech proměnných.

tc délka cyklu, pro kterou je odhad prováděn (parametr funkce)

T doba odhadu

ro hustota provozu odhadovaná z dat z detektorů

cars_in_avg filtrovaná data z detektorů (atribut třídy)

queue_avg filtrovaná naměřená délka fronty (atribut třídy)

q délka fronty v průběhu odhadu

wt součet čekacích časů

green_time_ratio poměr délky zelené k délce cyklu (atribut třídy)

gt doba, kdy je křižovatka pro pruh průjezdná

tc_rest čas v daném cyklu

tc_last zbytek odhadovací periody

gt_last délka zelené ve zbytku odhadovací periody

delta korekční parametr vyjadřuje zkrácení doby průjezdnosti vlivem pomalých rozjezdů a podobně

Odhad je implementován v metodě `getWT`:

```
1   double getWT ( int tc ) {
2       const int T = 450; // 5 scykl
3       double ro = cars_in_avg/90;
4       double q = queue_avg;
5       double wt = 0;
6       double gt = tc*green_time_ratio - delta;
7       int t_rest;
8       int tc_last = T % tc;
9       double gt_last = tc_last*green_time_ratio;
10      for ( int t = 0; t < (T-tc_last); t++ ) {
11          t_rest = t%tc;
12          // zelena
13          if ((tc-gt)/2<=t_rest&&t_rest<(tc+gt)/2)
14              q -= saturated_stream;
15          if ( q < 0)
16              q = 0;
17          wt += q;
18          q += ro;
19      }
```

```
20          // zbytek
21          for ( int t = 0; t < tc_last; t++ ) {
22              // zelena
23              if ((tc_last-gt_last)/2<=t&&t<(tc_last+gt_last)/2 )
24                  q -= saturated_stream;
25              if ( q < 0)
26                  q = 0;
27              wt += q;
28              q += ro;
29          }
30          return wt;
31      }
```

4.4 Hlavní smyčka

Hlavní smyčka programu se stará o synchronizaci všech potřebných parametrů a provádění iterací. Před započetím opakování kroku simulace nastaví podle konfiguračního souboru počáteční parametry mikrosimulátoru dopravy AIMSUN, jako jsou délka cyklu a fáze řadičů, inicializuje jednotlivé agenty a zavolá jejich metody `fromSettings` a `Validate`.

4.4.1 Krok simulace

Délka kroku simulace je pevně nastavena na 90 sekund shodně s periodou sběru dat z AIMSUNu. V jednom kroku se nejprve načtou data z AIMSUNu do vektoru `glob_dt`, který se následně předá každému z agentů jako parametr jejich metody `adapt`. Následně se zprostředkuje komunikace mezi agenty tak, že se naplní fronta zpráv voláním metody `broadcast` jednotlivých agentů. Z této fronty se postupně odebírají zprávy, které se předávají agentům parametrem jejich metody `receive` Po ukončení komunikace naplní vektor výstupů `glob_ut` voláním metod `act` hodotami parametrů pro AIMSUN, předá se mu a vyčká se do dalšího kroku.

4.5 Třída agenta

Třída `TrafficAgentCycleTime` je odvozena od obecného návrhu agenta `BaseTrafficAgent`. Jsou v ní implementovány metody pro načtení dat, komunikaci a nastavení délky cyklu za účelem řízení jedné konkrétní křižovatky.

4.5.1 Stručný popis algoritmu

Cíl algoritmu je zlepšit dopravní situaci nastavením délky cyklu řadiče na takovou hodnotu, při níž bude celková čekací doba všech vozidel minimální. Nejprve agent odhadne délku cyklu, která je ideální pro něj, poté vyšle zprávu, zda je tato hodnota vyšší nebo nižší než aktuální. Pokud je nadpoloviční většina agentů pro změnu stejným směrem, pokouší se najít hodnotu ideální pro všechny.

4.5.2 Výpočetní metody

4.5.2.1 Metoda getWT

Tato metoda vrací součet čekacích dob, sečtený přes všechny dopravní pruhy pro danou délku cyklu předanou parametrem `tc`. V cyklu projde instance tříd `LAneHandler`, na které má uloženy ukazatele v proměnné `lanehs` typu `Array<LaneHandler*>`. Pole se inicializuje v metodě `validate` předka `BaseTrafficAgent`.

4.5.2.2 Metoda findIdealTc

Pro určení, zda vznést návrh na zvýšení či snížení délky cyklu, slouží metoda `findIdealTc`. Ta zjišťuje čekací doby pro všechny možnosti rozsahu od minimální do maximální hodnoty, určené atributy `minTc` a `maxTc`. Vrátí takovou délku cyklu, pro kterou je čekací doba nejmenší.

Princip použití součtu čekacích dob automaticky upřednostňuje dopravní pruhy s větší vytížeností, což je žádoucí. Ostatní pruhy však zcela nepotlačuje. Tímto způsobem by se mělo dosáhnout optimální hodnoty pro celou křižovatku.

4.5.3 Přepsané metody předka

Hlavní smyčka programu automaticky volá v každém cyklu metody určené k načtení dat, komunikaci a nastavení dalšího řízení.

4.5.3.1 Metoda validate

K předání požadovaných hodnot parametrů v průběhu simulace slouží vektor `actions`. Významy těchto hodnot vysvětluje `rv_actions` třídy `RV`. V metodě `validate`, která se volá na začátku simulace, přidáme do `rv_actions` hodnotu `Tc`. Tím vlastně sdělíme AIMSUNu, že budeme přenastavovat délku cyklu. Protože délka jednotlivých fází se nepřenastavuje automaticky, přidáme do `rv_actions` také označení těchto fází. V praxi je tento postup realizován kódem

```
1  rv_action = RV("Tc",1);
2  rv_action.add( RV(stage_names, ones_i(stage_names.length())));
```

kde je v proměnné `stage_names` uložen seznam fází.

Následující metody se volají v každém cyklu simulace.

4.5.3.2 Metoda adapt

Na začátku každého simulačního cyklu se jako první volá u všech agentů metoda `adapt`. V parametru se jí předá adresa vektoru výstupních údajů simulace `glob_dt`, k jejichž zpracování je metoda určena. Zde se předávají údaje z detektorů a délky front instancím třídy `LaneHandler` a poté se volá jejich metoda `countAvgs`, která počítá filtrované hodnoty způsobem popsaným v sekci 4.3.2.

4.5.3.3 Metoda receive a broadcast

Po načtení a zpracování dat začíná komunikace. Ta je realizována metodami `broadcast` a `receive`, které se volají střídavě. Přesněji řečeno, hlavní smyčka nejprve volá metody `receive` každého agenta tak dlouho, dokud nevyprázdní frontu zpráv. Ta je reprezentována polem proměnných typu `Setting`, které mají nastaveny čtyři parametry:

from určuje od koho zpráva přichází

to má hodnotu jména agenta, pro kterého je zpráva určena

what obsahuje co posíláme

value obsahuje hodnotu spojenou s tím co posíláme

Každý agent má v atributu `name` uloženo jméno křižovatky, kterou ovládá. Toto jméno je uloženo v konfiguračním souboru a odpovídá označením ze sekce 2.1.5. Pokud je jméno agenta stejné jako parametr zprávy `to`, vyjme se zpráva z fronty a předá se metodě `receive`.V momentě, kdy je fronta prázdná, zavolají se metody `broadcast` všech agentů. Metoda `broadcast` opět plní frontu zprávami, které chce agent vyslat. Po jejím ukončení se opět volá `receive`. Střídavé volání těchto dvou metod probíhá, dokud `broadcast` produkuje nějaké zprávy.

V našem případě probíhá odesílání ve dvou krocích. V obou se posílají zprávy všem ostatním agentům. Nejprve agent sdělí, jestli chce délku cyklu zvýšit, snížit, nebo ponechat stejnou. Pokud je pro zvýšení více než polovina agentů, začíná druhá fáze. V té každý agent pošle všechny délky cyklu v předem daném rozsahu a k nim očekávané zisky. Rozsah je od minimální do stávající při odhlasování snížení a od stávající do maximální při zvýšení. Zisk se počítá jako rozdíl čekací doby při navrhované a stávající délce cyklu pomocí metody popsané v 4.3.3. Odeslání těchto zpráv je realizováno kódem

```
1  int broadcast_width = (max_tc - min_tc) + 1;
2
3  for ( int i = 0; i < broadcast_width; i++ ) {
4    int time_cycle = min_tc + i*stepTc;
5    double profit = getProfit(time_cycle);
6
7    // send tc
8    stringstream time_cycle_stream;
9    time_cycle_stream << "tc_" << (i);
10   send2allNeighbours( set, time_cycle_stream.str(), time_cycle
     );
11
12   // send profit
13   stringstream profit_stream;
14   profit_stream << "profit_" << (i);
15   send2allNeighbours( set, profit_stream.str(), profit );
16 }
```

Agent tedy vygeneruje pro každou hodnotu délky cyklu index, podle kterého se k ní dá přiřadit hodnota zisku. V metodě `receive` se podle tohoto indexu sčítají zisky do pole `received_profit_sum`.

4.5.3.4 Metoda act

Pokud se agenti dohodli na změně délky cyklu, probíhá její přenastavení v metodě `act`. Nejprve se projde pole `received_profit_sum` a vybere se z něj maximum. Poté se podle indexu maximálního zisku určí ideální délka fronty.

Měření ukázala, že časté změny cyklu působí emulátorům řadičů problémy. Proto se ideální hodnota průměruje plovoucím průměrem a nastavuje se jednou za 5 cyklů.

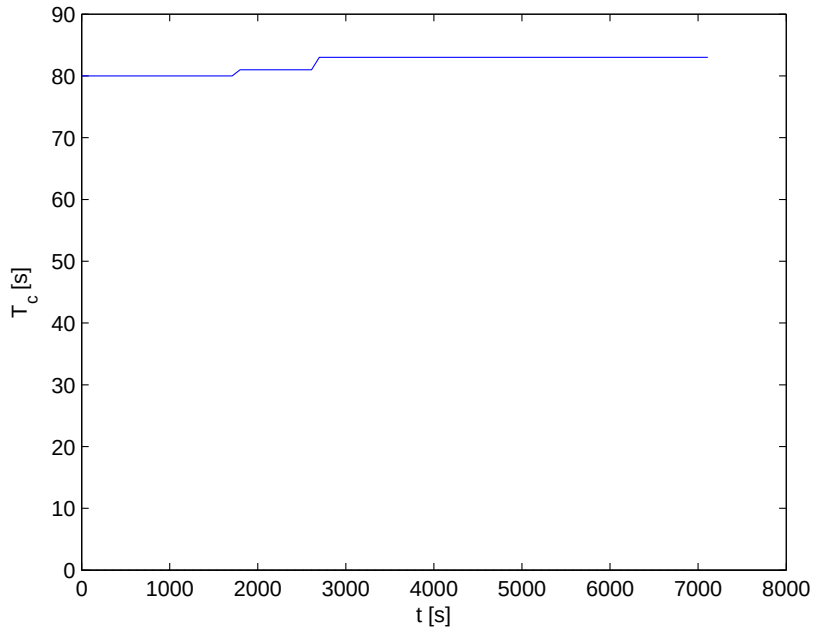
Kapitola 5

Výsledky

Měření byla prováděna v oblasti popsané v kapitole 2.1.5 při použití dvou scénářů s konstantními hustotami provozu a jedním scénářem se záznamem reálné situace o délce 24 hodin. Pro porovnání sloužily simulace s pevně nastavenou délkou cyklu, a to hlavně na 80 sekund, neboť se tato hodnota používá na obou křižovatkách při absenci zásahů z dopravní centrály. Porovnávají se hodnoty počtů zastavení, doby průjezdu a doby zastavení vozidel zprůměrované přes krátký časový úsek, který je násobkem kroku simulace. Simulace vždy začíná s délku cyklu nastavenou na 80 sekund. Tato hodnota se začíná měnit až po osmi krocích simulace kvůli postupnému zpracovávání vstupních dat. Pro zpracování výsledků byly použity výstupy z VGS API popsané v kapitole 2.1.3.

5.1 Konstantí scénář 1

Pro tento scénář se při porovnání se simulacemi při konstantních délkách cyklu, nastavených na 70 a 90 sekund, ukázala délka cyklu 80 sekund jako optimální. Následující graf ukazuje vývoj nastavované délky cyklu. Je vidět, že se drží v rozsahu optimální hodnoty 80 sekund +- 4s a neosciluje.



Obrázek 5.1: Průběh délky cyklu

	T_c 70s	T_c 80s	$/T_c$ 80s $/T_c$ 70s
Tok vozidel [voz/h]	2314,00	2313,00	-0,0 %
Průměrná doba jízdy [s]	159,15	142,61	-10,4 %
Průměrné zpoždění [s]	84,91	68,37	-19,5 %
Průměrná rychlost [km/h]	27,78	29,09	4,7 %
Hustota vozidel [voz/km]	16,80	15,25	-9,2 %
Průměrná doba zastavení [s]	69,27	53,66	-22,5 %
Počet zastavení [-]	2,45	2,08	-15,2 %

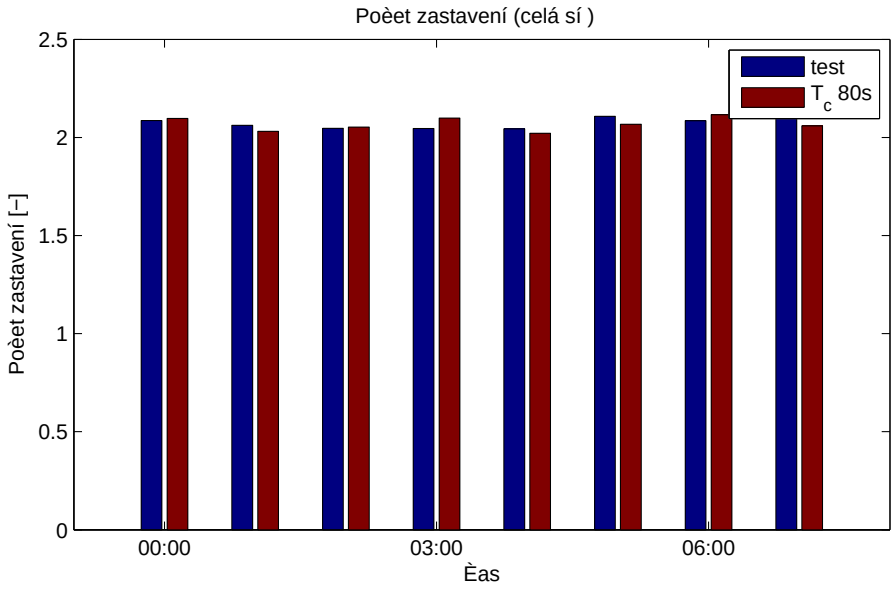
Tabulka 5.1: Tabulka naměřených hodnot a jejich rozdílů při konstantních délkách cyklu 70 a 80 sekund

	T_c 90s	T_c 80s	$/T_c$ 80s $/T_c$ 90s
Tok vozidel [voz/h]	2311,00	2313,00	0,1 %
Průměrná doba jízdy [s]	144,57	142,61	-1,4 %
Průměrné zpoždění [s]	70,34	68,37	-2,8 %
Průměrná rychlost [km/h]	28,87	29,09	0,8 %
Hustota vozidel [voz/km]	15,47	15,25	-1,4 %
Průměrná doba zastavení [s]	55,68	53,66	-3,6 %
Počet zastavení [-]	2,06	2,08	0,9 %

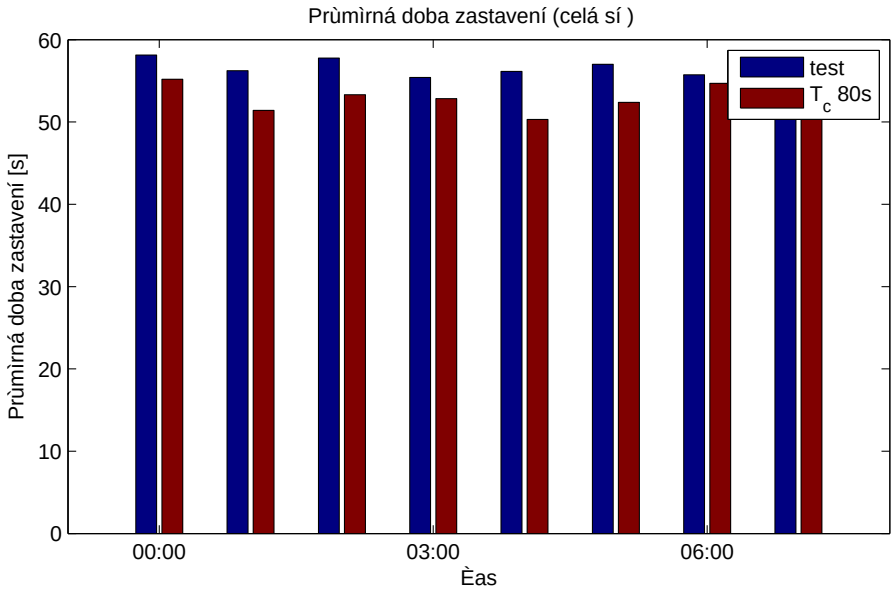
Tabulka 5.2: Tabulka naměřených hodnot a jejich rozdílů při konstantních délkách cyklu 90 a 80 sekund

Tabulky ukazují, že při odchylce +- 10 sekund od 80-ti dochází ke zhoršení ve všech parametrech.

Z důvodu malé změny délky cyklu oproti referenčním hodnotám se také příliš nemění stav dopravy, jak je vidět na grafech znázorňujících průměrné počty a doby zastavení za hodinu v průběhu osmihodinové simulace.



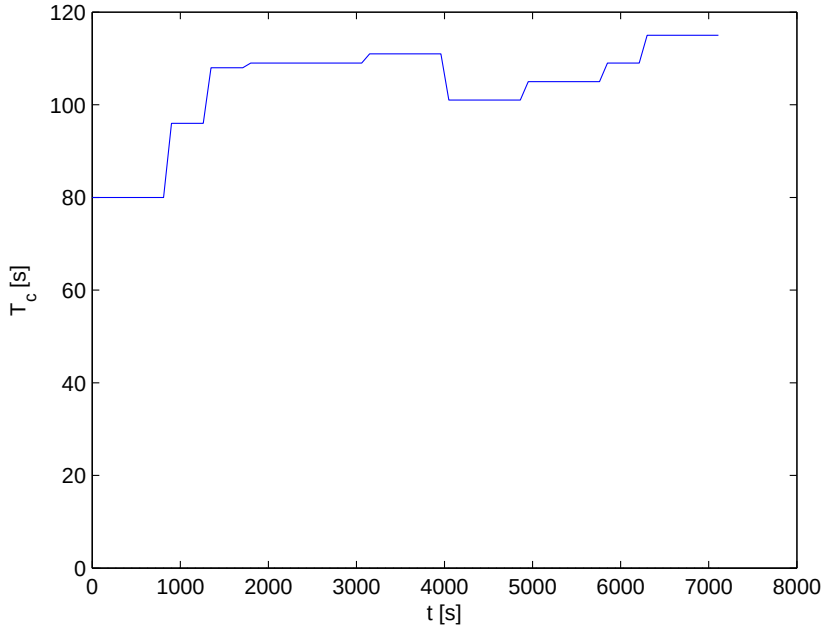
Obrázek 5.2: Průměrné počty zastavení



Obrázek 5.3: Průměrné délky zastavení

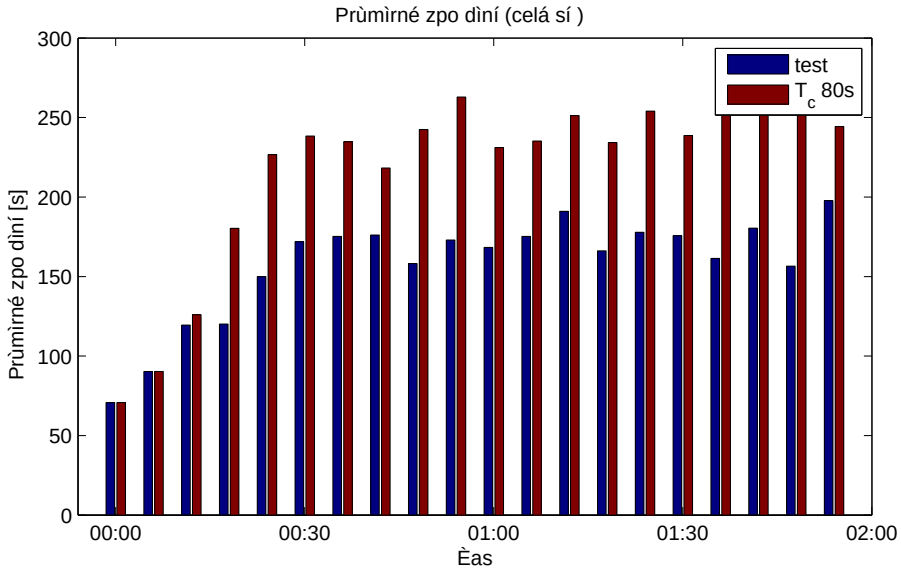
5.2 Konstantí scénář 2

Tento scénář simuluje podstatně vyšší hustotu provozu než scénář předchozí. Délka cyklu 80 sekund se zde již jeví jako nedostačující. Při testech program nastavoval délku cyklu mezi 80-ti a 120-ti sekundami, což znázorňuje tento graf.

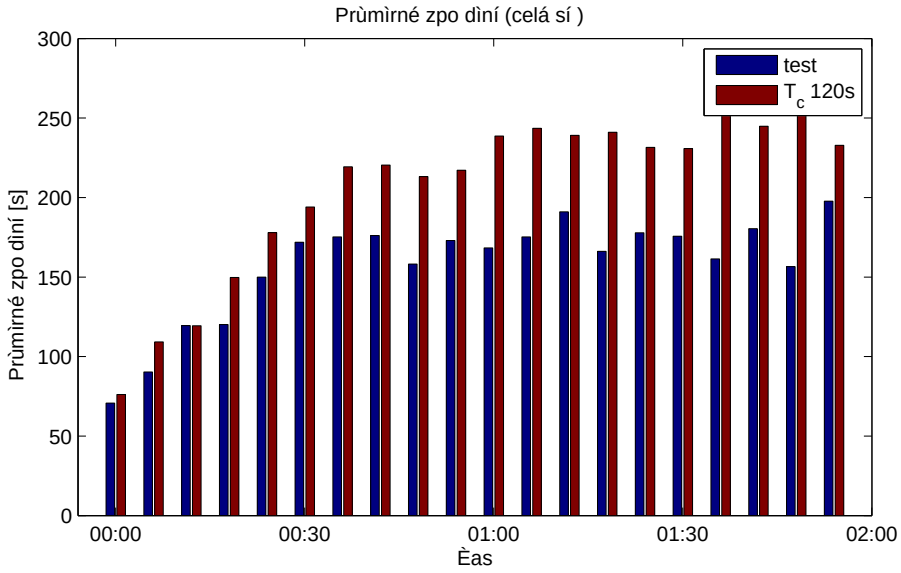


Obrázek 5.4: Průběh délky cyklu

Při porovnání s těmito konstantními hodnotami experiment vykazuje značné zlepšení.



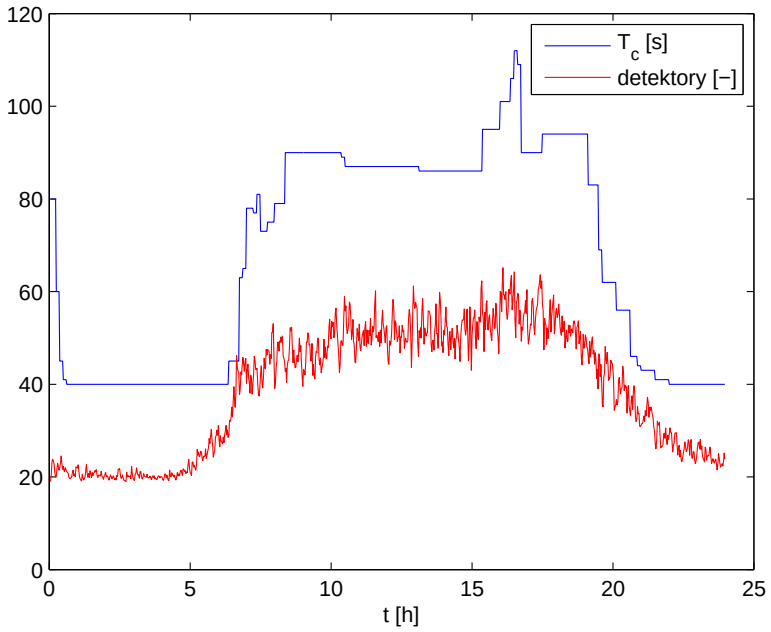
Obrázek 5.5: Průměrné zpoždění v porovnání s konstantní hodnotu délky cyklu 80s



Obrázek 5.6: Průměrné zpoždění v porovnání s konstantní hodnotu délky cyklu 120s

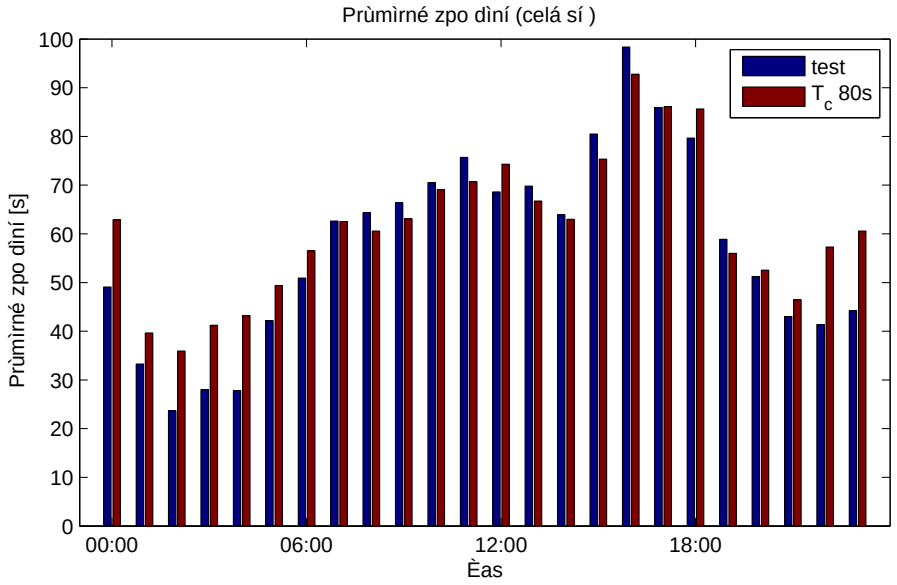
5.3 Reálný scénář

Tento scénář simuluje reálnou situaci naměřenou dne 12. prosince 2007. Protože se zde hustota provozu mění v závislosti na čase, byly prováděny simulace o délce 24 hodin. Následující graf zobrazuje délku cyklu a součet vozidel, které zaznamenaly detektory za simulační periodu (90 sekund).

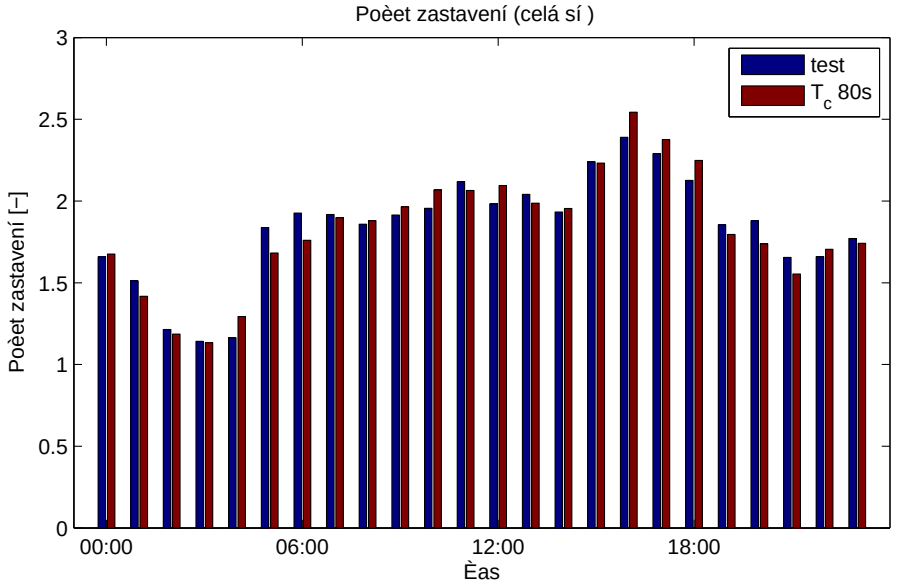


Obrázek 5.7: Graf délky cyklu $T_c[s]$ a záznamů detektorů [-]

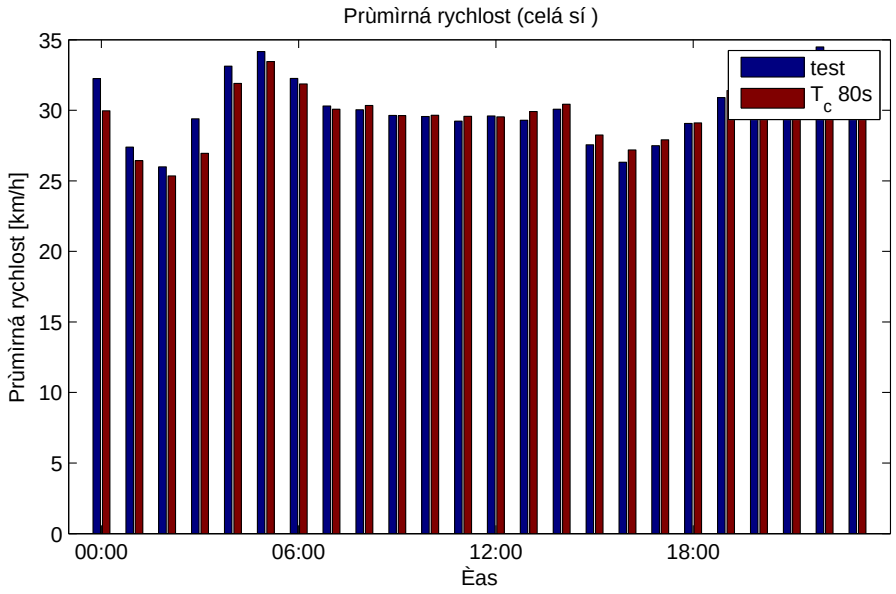
Je vidět, že program reaguje na zvýšení hustoty provozu prodloužením cyklu.



Obrázek 5.8: Graf průměrného zpoždění v porovnání s konstantní délkou cyklu $T_c = 80s$



Obrázek 5.9: Graf průměrných počtů zastavení v porovnání s délkou cyklu $T_c = 80s$



Obrázek 5.10: Graf průměrných rychlostí v porovnání s konstantní délkou cyklu $T_c = 80s$

Nastavením nižší délky cyklu algoritmus zkracuje čekání v době, kdy je nízká hustota provozu. Při vysoké hustotě provozu dosahuje podobné výsledky, jako referenční měření. Při některých hustotách provozu dosahuje mírně horších výsledků oproti referenci s pevnou délkou cyklu 80 sekund. Jedná se zřejmě o provoz, pro který je délka referenční hodnota ideální a algoritmus nedosahuje takové přesnosti, aby nastavil tuto hodnotu.

Kapitola 6

Závěr

Úkolem této práce bylo navrhnout základní algoritmus na decentralizované řízení dopravní signalizace za použití multiagentního systému a provedením základních měření zjistit, jestli může být tento způsob řízení cestou se zdárným koncem. Algoritmus měl za úkol řídit dopravní signalizaci pouze nastavováním délky cyklu při zachování ostatních parametrů. K nalezení optimálního řízení dostal k dispozici údaje z detektorů a délky maximálních front.

Algoritmus byl implementován v jazyce C++ za pomocí knihoven BDM a IT++ a odzkoušen na mikrosimulátoru dopravy AIMSUN. Pro testování byly použity dva scénáře s konstantními hustotami provozu a jeden se skutečnou dopravní situací. K porovnání sloužily výsledky simulací při konstantní hodnotě délky cyklu rovnající se 80-ti sekundám.

Výsledky měření ukázaly, že pokud se intenzita provozu pohybuje na standardní úrovni, na kterou je referenční délka cyklu optimalizovaná, algoritmus dopravní situaci většinou nezlepší. Pokud však dojde ke zhoršení, není nijak výrazné. Při výrazně vyšší nebo nižší intenzitě provozu, než na jakou je délka cyklu optimalizována, dochází vlivem algoritmu k podstatnému zlepšení.

Pro reálné nasazení algoritmu by bylo zapotřebí věnovat ještě mnoho času na jeho zdokonalení, odstranění nedostatků při zpracování dat, či zpřesnění některých parametrů. Pod dojmem dosavadních výsledků si ale myslím, že decentralizované řízení multiagentním systémem je ten pravý směr ve vývoji řízení dopravy.

Zlepšení výsledků se také dá očekávat při možnosti nastavení více parametrů najed-

nou, například délky cyklu a offsetu. Tyto dva parametry jsou vzájemně propojené. Složitost vyjednávání by se samozřejmě dosti zvětšila, neboť hodnota jednoho parametru by ovlivňovala rozhodování o druhém, a to ve smyslu ne jednoho, ale celé skupiny agentů. Touto kombinací by se však mohlo dosáhnout rapidního zlepšení.

Literatura

[1] Pecherková, P.; Duník, J.; Flídr, M.: *Robotics, Automation and Control*, kapitola Modelling and Simultaneous Estimation of State and Parameters of Traffic System. In-Tech, Croatia, 2008, s. 319–336.

[2] Roozemond, D. A.: Using intelligent agents for pro-active, real-time urban intersection control. *European Journal of Operational Research*, ročník 131, č. 2, 2001: s. 293–301, ISSN 0377-2217.

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

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

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

[6] Wooldridge, M.: *Multi Agent Systems*. MIT Press, Březen 2005.