VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM ŘÍZENÍ DOPRAVY

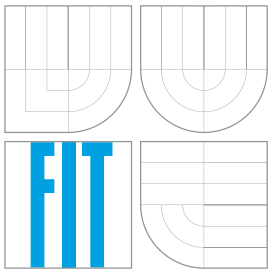
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MATEJ KAČIC

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM ŘÍZENÍ DOPRAVY

TRAFFIC CONTROL SYSTEM

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. MATEJ KAČIC

AUTHOR

VEDOUCÍ PRÁCE — Doc. Dr. Ing. DUŠAN KOLÁŘ

SUPERVISOR

BRNO 2010

Abstrakt

Cílem práce je vytvořit aplikaci, která odsimuluje dopravní situaci na simulačním modelu založeném na realitě, a zvládne řídit světelní signalizaci na základe navrhnutého algoritmu tak, aby cestní provoz byl plynulý a systém měl co největší propustnost. Zaobírá se popisem simulačního modelu, analýzou různých přístupů při návrhu algoritmů při řízení dopravního systému a podrobně popisuje evoluční přístup optimalizace řízení systému křižovatek.

Abstract

Main goal of this thesis is to create an application which can do a simulation of model of road traffic system based on reality and can handle to manage signal control based on the proposed algorithm so that the road traffic should have a great performance and system should have a maximum throughput. It describes a simulation model, different approaches in design of algorithm for management of the road traffic system and decribes in detail evolutionary approach for optimalization system of control crossroads.

Klíčová slova

Řízení dopravy, signální plán, evoluční algoritmy, neurónové síte, simulace, Simlib, GAlib

Keywords

Road traffic control, evolutionary algorithm, neural network, simulation, Simlib, GAlib

Citace

Matej Kačic: Systém řízení dopravy, diplomová práce, Brno, FIT VUT v Brně, 2010

Systém řízení dopravy

Prehlásenie

Čestne prehlasujem, že som vypracoval tento semestrálny projekt samostatne pod vedením pána Doc. Dr. Ing. Dušana Koláře

.....

Matej Kačic

17. mája 2010

Poďakovanie

Veľmi rád by som poďakoval vedúcemu mojej diplomovej práce Doc. Dr. Ing. Dušanovi Kolářovi za jeho pripomienky a odborné rady.

Obsah

Kapitola 1

Úvod

Simulácie rôznych reálnych modelov sveta, algoritmy riešiace problémy pre človeka nepredstaviteľné, toto všetko má na svedomí rýchly vývoj v oblasti hardwaru ako aj softwarového vybavenia. Dopravná situácia vo veľkomestách sa komplikuje s narastajúcim počtom vozidiel. Autá čakajúce v dlhých zástupoch, zvyšovanie nákladov na prepravu a v neposlednom rade negatívny dopad na životné prostredie. Riešenie optimálneho riadenia pre všetky križovatky mesta je pre človeka neriešiteľné. Cieľ je jasný, navrhnúť sofistikované riadenie dopravy v celom meste pomocou výpočtovej techniky.

K moderným dopravným technológiám v dnešnej dobe patrí dynamické riadenie dopravy. Tento projekt sa zaoberá analýzou systému riadenia pomocou svetelnej signalizácie. Analýza je dostatočne podrobná, opisujúca realitu cestnej premávky, pričom definuje základné pojmy dopravného inžinierstva, ako napríklad predraďovací prúd, signálne fázy, bezpečný prejazd a podobne. Ďalej popisuje súčasný spôsob riadenia križovatky, či už pomocou statického signálneho plánu alebo pomocou dynamického či koordinovaného riadenia križovatky resp. križovatiek. Táto diplomová práca nadväzuje na semestrálny projekt riešený v predposlednom semestri magisterského štúdia, ktorého predmetom bola analýza dopravného inžinierstva a návrh grafického užívateľského rozhrania.

Hľadanie optimálnej metódy pre zostavenie signálneho plánu jednej izolovanej križovatky alebo plánu skupiny križovatiek tvoriacich systém je netriviálna úloha. Kapitola 3 sa zaoberá hľadaním metód riešiacich tento problém, pričom sa opiera o matematické aparáty - neurónové siete, multiagentný systém a evolučné algoritmy. Všetky z nich sú podrobne popísané vrátane ich výhod, či nevýhod. Ďalej sa podrobne venuje návrhom agentného systému pre plošné riadenie križovatiek a jeho následnou implementáciou pomocou nástroja *JADE*, pričom dôraz je kladený na priepustnosť celého systému.

Úspešné navrhnutie riadiaceho algoritmu je závislé na kvalitnej simulácii systému križovatiek, ktorá by sa mala čo najviac blížiť k realite. Sú vysvetlené základné pojmy teórie simulácie potrebné pre pochopenie problému. V ďalšom kroku sú podrobne popísané etapy návrhu abstraktného a simulačného modelu. Koniec kapitoly 4 je venovaný spôsobu

implementácie simulácie pomocou knižnice *Simlib/C++*.

Grafické užívateľské rozhranie tvorí základ pre komunikáciu medzi človekom a aplikáciou v počítači, preto musí poskytovať prehľadný, jednoduchý a priateľský prístup pri zadávaní informácií. Návrhom grafického užívateľského rozhrania ako aj jeho implementáciou pomocou knižnice *wxWidgets* sa zaoberá kapitola 5.

Kapitola 6 je venovaná evolučnému prístupu pomocou genetických algoritmov, ktoré sú aplikované na problém optimálneho resp. suboptimálneho riešenia signálneho plánu križovatky i systému križovatiek. Úvod je venovaný stručnému popisu genetických algoritmov a knižnici *GAlib*. V ďalších častiach tejto kapitole sú podrobne rozobrané optimalizácie signálneho plánu jednej a viacerých križovatiek, pre ktoré sú definované kódovania problému a k nim prislúchajúce genetické operátory. Základným kameňom evolučného prístupu je *fitness* funkcia, ktorá ohodnocuje jedincov populácie. Jej návrhu a implementácii sa venuje záver kapitoly.

Záver tejto práce tvoria výsledky dosiahnuté pomocou kombinácie simulácie a genetických algoritmov. Výsledky, vhodne zobrazené pomocou grafov, porovnávajú situáciu pred a po aplikácii genetickej optimalizácie signálneho plánu. V poslednej kapitole sú navrhnuté možné rozšírenia jednotlivých častí aplikácie.

Kapitola 2

Analýza problému

2.1 Križovatky riadené svetelnou signalizáciou

Základom tohto riadenia je svetelná fáza, čo je stav, v ktorom majú určité prúdy vozidiel voľný prejazd a iné prúdy majú prejazd zakázaný. Pre každú križovatku je navrhnutý signálny plán, ktorý môže byť statický, dynamický alebo adaptívny (centrálne riadený). Pri návrhu statického riadenia križovatky postupujeme na základe analýzy dopravnej situácie a výpočtu riadiaceho cyklu tak, aby priepustnosť križovatky bola čo najväčšia. Veľkou nevýhodou tohto systému riadenia je zbytočne dlhé čakanie mimo špičku alebo neprispôsobenie sa aktuálnej dopravnej situácii. Táto časť bola spracovaná z [1].

Pod pojmom križovatky riadené svetelnou signalizáciou rozumieme vytváranie časových medzier s dostatočnou dĺžkou, aby sa zabezpečil plynulý a bezpečný prejazd vozidiel z každej komunikácie v čo najkratšom čase. Svetelné riadenie sa zavádza, aby sa znížili časové straty vedľajšej komunikácie oproti neriadenej križovatke tak, aby tieto straty neboli väčšie ako straty v hlavnom smere. Križovatky riadené pomocou svetelnej signalizácie umožňujú dať prednosť jednotlivým navzájom kolíznym dopravným prúdom. Správne časové rozdelenie prejazdov kolíznych prúdov je základom bezpečnosti i plynulosti cestnej premávky, ako aj zníženie emisií a pohonných látok, a v neposlednom rade zvýšenie kapacity križovatky a zavedenie koordinácie. Väčšina križovatiek sa drží pevne stanoveného časového plánu, čo neumožňuje dynamicky reagovať na prítomnosť vozidla alebo sledovať vytváranie frónt v určitom smere.

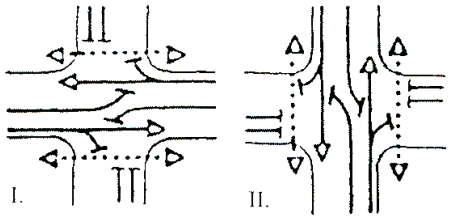
Vjazd do križovatky je upravený tak, aby sa vozidlá vstupujúce do križovatky mohli radiť podľa smeru jazdy, preto dôležitou súčasťou križovatky bývajú predraďovacie pruhy, ktorých počet závisí od povolených smerov a od šírky komunikácie. V zásade platí, že jeden vstupný pruh má potenciál spracovať 150 až 600 vozidiel za hodinu. Nové križovatky počítajú vždy s odbočovacím ľavým predraďovacím pruhom.

Prevádzková schéma križovatky určuje sled dopravných pohybov na základe tvaru križovatky a jej zaťaženia. Tieto pohyby sú rozdelené na prúdy motorových vozidiel, na pešie

prúdy a koľajové prúdy tak, aby sa vylúčila vzájomná kolízia. Podľa smeru jazdy a kolízie môžeme tieto prúdy zladiť resp. realizovať v rovnakom čase, čím vytvoríme fázu križovatky. Fáza je čas, počas ktorého majú povolený smer jazdy všetky navzájom nekolízne dopravné pohyby. Celkový počet fáz závisí od veľkosti a počtu jednotlivých dopravných i chodeckých prúdov križovatky, pričom rozhodujúcim smerom je odbočenie vľavo. Medzi jednotlivými fázami sa nachádza časová strata, ktorá zvyšuje bezpečnosť a zároveň znižuje výkonnosť križovatky. Je to čas potrebný na rozjazd vozidiel, nevyužitý čas pri oranžovom svetle a čas červeného svetla, ak sa počas cyklu viackrát zopakuje. V neposlednom rade je to bezpečnostná časová strata medzi každou fázou znižujúca celkovú dĺžku aktívnej zelenej.

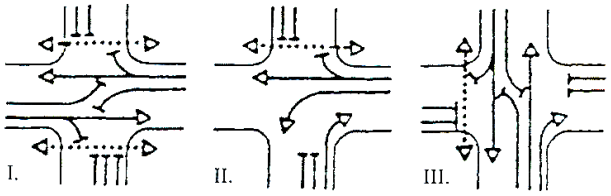
Riadenie križovatky, čiže cyklus križovatky, môže pozostávať z rôzneho počtu fáz. Zvyšovanie počtu fáz je nepriamo závislé dĺžke aktívnej zelenej (so zvyšujúcim sa počtom fáz sa znižuje dĺžka aktívnej zelenej). Prípustné sú tri možné fázy riadenia:

- Dvojfázové riadenie - dĺžka aktívnej zelenej je maximálna, povolenie jazdy dostanú vždy dve strany križovatky, ľavé odbočovacie prúdy musia dať prednosť protiidúcemu vozidlu



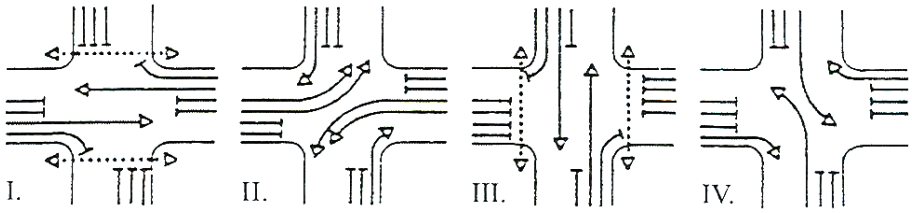
Obrázok 2.1: Dvojfázové riadenie, prevzaté z [1]

- Trojfázové riadenie - dve fázy rovnaké ako pri dvojfázovom riadení, pridáva sa ďalšia fáza z najviac preplneného smeru, ľavý odbočovací prúd tohto smeru ako aj ďalšie nekolízne prúdy dostávajú zelenú, dĺžka aktívnej zelenej sa znižuje



Obrázok 2.2: Trojfázové riadenie, prevzaté z [1]

- Štvorfázové riadenie - môže byť vo viacerých variantoch, väčšinou sa povolí celý daný smer alebo kombinácia prúdov rôznych smerov, dĺžka aktívnej zelenej je minimálna



Obrázok 2.3: Štvorfázové riadenie, prevzaté z [1]

Dĺžka cyklu je daná nemennými medzičasmi, dopravným značením a počtom fáz. Pri voľbe vhodnej dĺžky cyklu je nutné si uvedomiť, že príliš krátky cyklus znižuje kapacitu križovatky a príliš dlhý cyklus vedie k nevyužitiu zelených fáz, kedy križovatkou neprechádza žiadne vozidlo. Štandardne sa dĺžka cyklu pohybuje od 40 do 120 sekúnd.

Neproduktívnym časom v riadení križovatky je medzičas, časový úsek medzi koncom zeleného signálu a začiatkom zeleného signálu nasledujúcej fáze. Funkciou medzičasu je možnosť opustiť kolízny priestor pre tie vozidlá, ktoré vstúpili do križovatky na končiacu zelenú prípadne žltú. Dĺžka medzičasu je daná:

$$t_m = t_v - t_n + t_b \tag{2.1}$$

- t_m - medzičas
- t_v - vyprázdňovací čas, ktorý potrebuje posledné vozidlo na prejazd od stop čiary až do opustenia kolíznej plochy.
- t_n - čas nájazdu je čas potrebný pre prvé vozidlo na prejazd od stop čiary do opustenia kolíznej plochy.
- t_b - bezpečnostný čas je určený pre vozidlá, ktoré nezastavia v čase žltej.

Produktívnou časťou cyklu je dĺžka zelených signálov, pričom platí, že suma všetkých dĺžok zeleného signálu z_i je rovná rozdielu dĺžky cyklu C a sume všetkých dĺžok medzičasov t_{mi}:

$$\sum_{i=1}^{n} z_i = C - \sum_{i=1}^{n} t_{mi}$$

Ďalšou, v poradí druhou podmienkou, je minimálna hodnota zeleného signálu z_i vychádzajúca z platnej normy „Svetelné signalizačné zariadenia pre riadenie premávky na pozemných komunikáciách“. V závislosti na type účastníka cestnej premávky je dĺžka zeleného signálu rovná:

- Automobily - 5 sekúnd
- Električky - 6 sekúnd

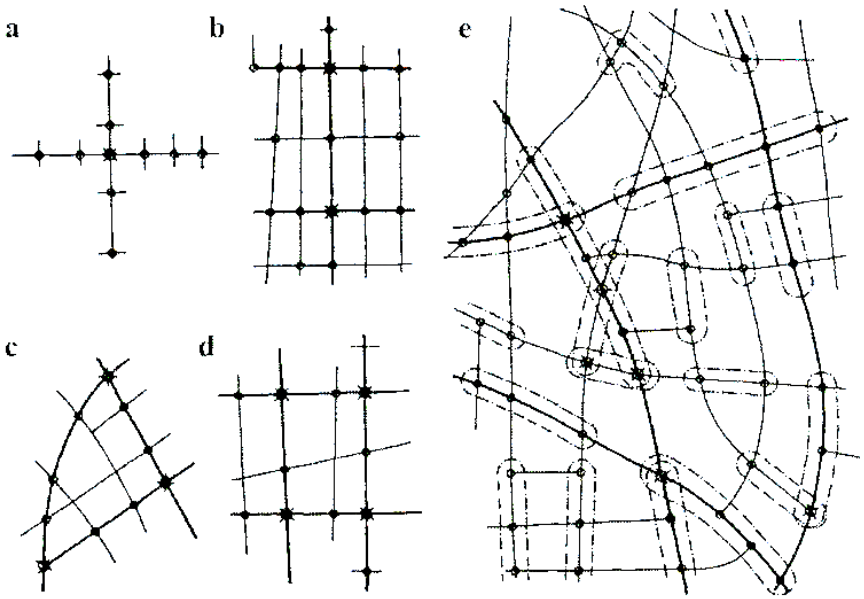
- Chodci - 5 sekúnd

Jednotlivé fázy cyklu riadenia križovatky je možné optimalizovať tak, aby bola priepustnosť križovatky maximálna. V súčasnej dobe sa poradie fáz určuje na základe súčtu medzičasov. Počet kombinácií pri trojfázovom riadení je rovný dvom, naopak štvorfázové riadenie ich má šesť.

2.1.1 Koordinované riadenie dopravy

Koordinované riadenie, zelená vlna, zaručuje plynulý a bezpečný prejazd vozidiel v hlavnom smere optimalizáciou poradia svetelných signálov susedných križovatiek. Oproti izolovanému riadeniu križovatky radikálne zvyšuje priepustnosť systému v hlavnom smere na úkor väčších časových strát priečnej dopravy. Koordinované riadenie môže byť:

- Líniové - koordinácia križovatiek nachádzajúcich sa v jednom smere.
- Plošné - koordinácia všetkých smerov križovatiek systému.



Obrázok 2.4: Možnosti plošnej koordinácie, prevzaté z [1]

Podstata líniovej koordinácie je v posune zelených fáz hlavného smeru na všetkých riadených križovatkách tak, aby kolóna vozidiel prešla systémom bez zastavenia. Časový posun medzi križovatkami je závislý na vzdialenosti križovatiek a rýchlosti dopravného prúdu. Nutnou podmienkou pre líniovú koordináciu je rovnaká dĺžka cyklu. Za základ sa považuje najzložitejšia križovatka koordinačného ťahu.

Pri väčších systémoch, kde bez problémov fungujú líniové koordinácie hlavných ťahov ako samostatné zelené vlny, môže priečna doprava vykazovať časové straty. Pre zvýšenie efektívnosti, priepustnosti systému a zníženiu priemernej doby čakania zahrnieme do koordinácie aj vedľajšie, priečne smery. Plošná koordinácia celého systému je pomerne zložitá, dokonca v niektorých prípadoch nemožná, preto sa v praxi často koordinujú menšie skupiny križovatiek.

2.1.2 Dynamické riadenie dopravy

Vyššou formou riadenia dopravy je centrálne riadenie s cieľom pre maximálnu optimalizáciu systému. Z hľadiska spôsobu návrhu môže byť toto riadenie:

- Centralizované - Všetky riadiace, rozhodovacie i kontrolné funkcie sú situované na jednom centrálnom mieste. Križovatky obsahujú len spínacie jednotky, ktoré reagujú na povel z centrály. Výhoda tohto riadenia spočíva v možnosti okamžitého riadenia dopravy celej oblasti. Naopak značná nevýhoda spočíva v ochromení celého systému pri výpadku centrály.

- Decentralizované - Systém riadenia, kde sú riadiacie, rozhodovacie a kontrolné funkcie realizované v radičoch umiestnených na jednotlivých navzájom komunikujúcich križovatkách. Nevýhodou tohto riadenia je nemožnosť vidieť ucelený stav dopravnej situácie a na rozdiel od centralizovaného riadenia nemožnosť operatívnych zásahov.

- Zmiešané - Tento systém využíva vlastnosti obidvoch predchádzajúcich riadení a tak sa snaží odstrániť ich nedostatky.

Dynamické riadenie teda spočíva v detekcii prítomnosti vozidiel v jednotlivých dopravných prúdoch, pričom detekcia sa uskutočňuje na základe senzorov zabudovaných v križovatke. Zvolenie ideálneho algoritmu je náročné. Vo viacerých systémoch sa používa celočervený režim, čiže ide o spúštanie zelenej na žiadosť. Tento systém je výhodný najmä v noci, keď križovatkou neprechádza veľké množstvo áut.

Fáza pri dynamickom riadení má premenlivú veľkosť, ktorá je obmedzená zdola i zhora. Minimálna fáza bola stanovená na päť sekúnd a maximálna fáza sa vypočítava na základne intenzity premávky. Vo veľkých mestách je dôležité priorizovať mestskú hromadnú dopravu, aby sa dodržiaval stanovený cestovný poriadok. Deje sa tak na základe signálov z GPS, teda pozícia a rýchlosť daného dopravného prostriedku. Pre plynulosť stačí podržať zelenú o niekoľko málo sekúnd.

Nasledujúce kapitoly analyzujú a riešia problém optimálneho riadenia systému križovatiek a popisujú spôsob, akým je možné otestovať navrhnuté riešenie.

Kapitola 3

Metódy dynamického riadenia križovatiek

Nájsť optimálnu metódu pre zostavenie signálneho plánu jednej križovatky alebo navrhnúť algoritmus pracujúci nad celou sieťou križovatiek nie je ľahkou úlohou. Existujú rôzne prístupy, ako tento problém začať riešiť. Matematické aparáty, ako napríklad konečné automaty[2] či Petriho siete[3], nie sú dostatočne obsiahle, aby popísali a optimálne riadili tak zložitý systém. Ich hlavnou nevýhodou je nemožnosť prispôsobiť sa nárazovej dopravnej situácii.

3.1 Neurónové siete

Ďalším, v poradí tretím prístupom je výpočtový model neurónových sietí[4], ktorý je zostavený na základe abstrakcie biologických nervových systémov. Základ siete tvorí model neurónu, ktorý spracováva informáciu podľa určitého pravidla. Tieto modely neurónov sú navzájom pospájané väzbami s rôznou váhou. Učenie takejto siete spočíva v aktualizácii týchto váh na základe učiaceho algoritmu. Po skončení učenia sa, siete tieto váhy už nemenia. Vstupom takejto neurónovej siete by mohli byť senzory detekujúce hustotu premávky v každom smere všetkých križovatiek. Naopak výstupom by boli signálne plány pre jednotlivé križovatky. Takýto prístup by vyžadoval použitie viacvrstvových neurónových sietí s hierarchiou križovatiek, skupiny križovatiek a napokon celého systému. Sieť by mohla byť schopná sa naučiť, ako riadiť premávku za normálnych podmienok, či spracovať nárazovú dopravnú vlnu. Vhodným nástrojom pre začlenenie neurónových sietí do systému je open-source knižnica *FANN (Fast Artificial Neural Network Library)* [5], napísaná v jazyku C. Implementuje viacvrstvové neurónové siete a podľa tvrdení uvedených v [5] je 150-krát rýchlejšia ako ostatné knižnice.

3.2 Multiagentný systém

Využitie agentného resp. multiagentého systému[6] pre riadenie skupiny križovatiek je ďalším možným prístupom. Je to systém zložený z viacerých vzájomne komunikujúcich inteligentných agentov. Pomocou tohto prístupu je možné riešiť problémy, ktoré sú pre jedného agenta neriešiteľné. Agent je autonómna, samostatná jednotka situovaná v nejakom prostredí resp. architektúre. Tento spôsob vyzerá byť vhodný pre dopravnú situáciu, preto bol experimentálne vyskúšaný na zjednodušenom modeli. Systém je realizovaný pomocou nástroja *JADE* [7].

Autá v dopravnom multiagentnom systéme sú simulované pomocou ďalších agentov. Tí si medzi sebou posielajú správy, a tým signalizujú rôzne udalosti, ktoré v systéme nastali. Napríklad auto pri príchode na križovatku zasiela správu, čím danú križovatku informuje o svojom príchode. Rovnako križovatka zasiela správy autám čakajúcim vo frontách a povoľuje im prechod. Ďalej sa v systéme okrem križovatiek a áut nachádzajú iní agenti. Generovanie áut zabezpečujú agenti nachádzajúci sa na okrajových častiach systému - na začiatku ulíc. Spustenie celého systému zabezpečuje agent, ktorý nastaví parametre pre jednotlivé križovatky a generátory áut, a následne týchto agentov pridá do hlavného kontajnera. Podobný prístup popisuje aj článok[8] publikovaný na SICE Annual Conference v roku 2008.

Riadenie križovatky zabezpečuje jeden agent spravujúci dvanásť front, pre každý smer tri fronty - pre autá smerujúce doľava, doprava, rovno. Vo fronte sa radia postupne prichádzajúce autá (agenti). Križovatka má štyri stavy, každý povoľuje jazdu z daného smeru rovno, doľava a doprava. Križovatka určí danému agentovi na základe pravdepodobnosti smer jazdy. Táto pravdepodobnosť sa používa na zaistenie hlavného toku áut pri simulácii systému. Agent križovatky má napevno nastavený cyklus, pričom zvýhodnený je hlavný dopravný prúd. Toto riadenie zodpovedá klasickému riadeniu pomocou statického signálneho plánu.

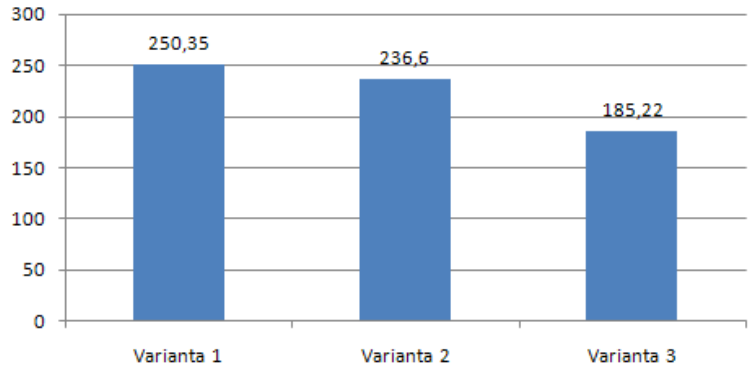
Agent križovatky, navyše oproti predchádzajúcej metóde riadenia, dynamicky prepočítava jednotlivé dĺžky cyklov v závislosti od počtu čakajúcich áut v danom smere. Cyklus je teda rozdelený v pomere podľa počtu čakajúcich áut, pričom sa dodržuje pravidlo minimálnej zelenej (5s). V prípade, že v danom smere nečaká žiadne auto, ostáva smer uzavretý a pokračuje sa cyklicky ďalej.

Pri spustení systému je pre každú križovatku vytvorený agent, riadiaci a reprezentujúci jej činnosť. Agent má k dispozícii údaje o okolitých križovatkách a ich vzdialenostiach. Jednotlivé križovatky medzi sebou priamo nekomunikujú, ale komunikujú nepriamo prostredníctvom agentov reprezentovaných autami. Auto po príchode na križovatku zašle križovatke správu informujúcu o tejto skutočnosti a čaká na odpoveď. Križovatka (ako reakciu na túto správu) uloží meno auta (agenta) do príslušnej fronty. Po prejazde križovatkou je autu zaslaná správa obsahujúca názov križovatky, smer vstupu do tejto križovatky a vzdialenosť

od tejto križovatky (miesto nasledujúcej križovatky môže byť aj výstup zo systému, po ktorom agent ukončí svoju činnosť). Auto následne preruší svoju činnosť po dobu potrebnú na dojazd na ďalšiu križovatku, s ktorou potom komunikuje obdobným spôsobom.

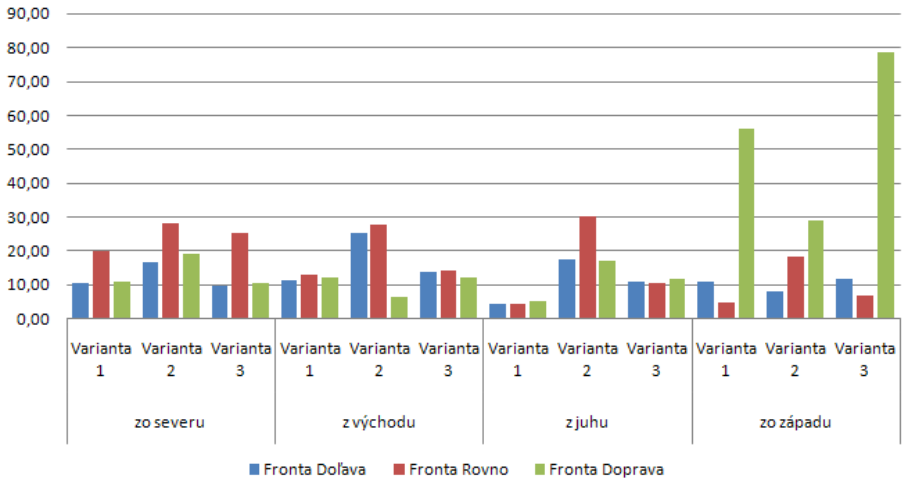
Na model siete križovatiek bolo aplikovaných z dôvodu vyhnutia sa príliš zložitej implementácii niekoľko zjednodušení. Za dobu prejazdu križovatkou autom bola zvolená konštantná hodnota jedna sekunda - vôbec sa neuvažujú prípady ako napríklad rôzna veľkosť jednotlivých križovatiek, rôzna doba potrebná pre opustenie križovatky v smeroch doľava, doprava a rovno, ktoré by nastali v skutočnosti. Taktiež sa neuvažuje obmedzená dĺžka čakacích frónt pre jednotlivé križovatky (čiže ich veľkosť nie je nijako obmedzená), čo je opäť situácia odlišná od reality, ale je nepodstatná pre porovnanie účinnosti jednotlivých metód riadenia križovatiek.

Tento systém riadenia dopravy bol odsimulovaný na reálnej sieti križovatiek v Brne, oblasť Moravské náměstí - Hlavní nádraží. V simulácii sa počítalo s priemernou rýchlosťou áut 40 km/h a so vzdialenosťami nameranými pomocou webmapy. Vstupné údaje pre križovatku boli odhadnuté na základe pozorovania na jednotlivých križovatkách, z čoho boli následne vypočítané pravdepodobnosti odbočenia z daného smeru. Schéma modelovaného systému križovatiek za nachádza v prílohe A.1. Simulácia pre každú variantu trvala 10 min. Výsledky zobrazujú grafy 3.1 a 3.2. Jednotlivé varianty sú označené číslami 1, 2 a 3.



Obrázok 3.1: Priemerný čas strávený v systéme

Z grafov vyplýva, že optimálna metóda je dynamické riadenie premávky pomocou izolovanej križovatky. Avšak pri použití tretej varianty, čiže vytvorenie zelenej vlny, sa zvýšia priemerné čakacie doby vo frontách, ale na druhej strane sa zníži celková doba strávená v systéme. Zjednodušená implementácia multiagentného systému vďaka príliš dlhej dobe simulácie ukazuje, že výber tohto spôsobu riadenia dopravy nebude správnou voľbou.



Obrázok 3.2: Priemerná doba strávená vo frontách križovatky 1

3.3 Evolučné algoritmy

Evolučné algoritmy [9], moderný matematický prístup založený na modeloch evolúcie v prírode, pracujú na princípoch vylepšovania možných riešení tak, že sa preferujú lepšie riešenia, ktoré vznikli z pôvodných horších riešení. Reprodukčný proces je založený na dvoch hnacích silách, na variačných operátoroch kríženia a mutácií a na selekcii kvalitných jedincov. Tento prístup sa používa v mnohých oblastiach umelej inteligencie. Jednou z oblasti je plánovanie a riadenie. Riešenie riadenia systému križovatiek je možné, treba však vhodne zakódovať stav systému do genotypu, na ktorý sa aplikuje genetický algoritmus. Ďalšími dôležitými krokmi je výpočet fitness funkcie, ktorá definuje vhodnosť chromozómu a spôsob určenia výsledku a ukončenia evolúcie. Výsledkom algoritmu bude genotyp, ktorý po rozkódovaní určí signálne plány pre každú križovatku v systéme. Knižnica vhodná pre použitie genetických algoritmov je *GAlib C++ Library* [10].

Je obtiažne vybrať vhodný prístup pre riadenie systému križovatiek, avšak najlepšie vyzerajúcim prístupom sú práve evolučné algoritmy. Za zmienku tiež stoja neurónové siete, ktoré podľa môjho názoru zaostávajú za evolučným prístupom. V kapitole 6 sú podrobne popísané evolučné resp. genetické algoritmy a ich aplikácia na problém riadenia systému križovatiek.

Kapitola 4

Simulácia systému križovatiek

Pre otestovanie funkčnosti systému križovatiek a aplikovanie vhodného typu riadenia je nutné celý systém odsimulovať tak, aby sa simulácia čo najviac priblížila k realite. K tomu potrebujeme veľké množstvo údajov. V prípade multiagentného systému, kde všetky objekty boli navzájom komunikujúci agenti, nebolo treba sa zaoberať otázkou simulácie. Veľkou nevýhodou pri tomto prístupe bola dĺžka simulácie, ktorá je úplne neprípustná pre aplikáciu evolučných algoritmov. Nasledujúca časť vysvetľuje základné pojmy teórie simulácie potrebné pre pochopenie problému. Neskôr sa venuje problému modelovania a simulácie systému križovatiek a ich praktickej implementácii.

4.1 Teória simulácie

Simulácia je metóda získavania nových znalostí o systéme experimentovaním s jeho modelom. Systém môžeme obecne definovať ako množinu elementárnych častí, ktoré majú medzi sebou určité väzby a modelom nazveme systém, ktorý sa snaží napodobniť iný systém [12]. Na systém sa pozeráme zväčša ako na funkčný celok. Pri modelovaní resp. pri vytváraní modelov systémov vychádzame z vopred dostupných znalostí reálneho systému, pričom sa zameriavame resp. formalizujeme tie znalosti modelovaného systému, ktoré sú pre daný účel simulácie podstatné. Nikdy sa nám nepodarí namodelovať úplný model reálneho systému, preto sa musíme uspokojiť s jeho podmnožinou, pričom nie všetky vlastnosti resp. znalosti tejto podmnožiny sú prevediteľné na číslicovom počítači. Ďalším faktorom, ktorý ovplyvňuje výsledok simulácie, sú chyby merania spôsobujúce problémy pri interpretácii výsledku.

Dôvodov, prečo sa zoberať modelovaním a následným experimentovaním s modelom resp. simuláciou, je niekoľko. Dôležitým kritériom je cena simulácie, kde pri porovnaní simulačného experimentu s reálnym systémom, je táto cena tisícnásobne nižšia. Ďalšou výhodou je rýchlosť simulácie, kedy v zlomku sekundy poznáme výsledok veľkého množstva pokusov.

Simulácia prináša so sebou okrem výhod aj nevýhody. Problém validity modelu je ten najdôležitejší, pretože chybný model dáva pri simulácii chybné výsledky. Preto overovanie správnosti modelu by malo byť uskutočnené ešte pred začatím experimentov. Na druhej strane validácia modelu je veľmi náročná. Rozvoj v oblasti IT neustále minimalizuje ďalší problém, ktorým je náročnosť na výpočtový výkon. V minulosti na starých strojoch bolo nemožné vykonávať simulácie väčších modelov, ale v dnešnej dobe tento problém pri priemerne veľkých modeloch nepostrehneme. Nepresnosť a nestabilita simulácie môžu negatívne ovplyvniť resp. znehodnotiť výsledky i v prípade overeného a bezchybného modelu.

Proces modelovania a simulácií je možné rozdeliť do viacerých etáp:

- Vytvorenie abstraktného modelu - formujeme zjednodušený popis reálneho systému, ktorý nezahrňuje všetky naše znalosti o modelovanom systéme, čím dosiahneme prijateľné zjednodušenie. Jedná sa o homomorfné zobrazenie medzi reálnym systémom a abstraktným modelom, čiže pri zjednodušovaní hovoríme o vzťahu $N : 1$. Ďalej zvolíme správny prístup pre popis chovania systému - diskrétny, spojitý alebo kombinovaný. Musíme si ale uvedomiť, že implementácia simulačného modelu na číslicovom počítači je vždy diskrétna.

- Vytvorenie simulačného modelu - jedná sa o zápis abstraktného modelu formou programu, pričom implementácia musí vhodne aproximovať chovanie reálneho systému. Simulačný model musí obsahovať všetky vlastnosti abstraktného systému. Vytvorený model je potreba verifikovať tj. overiť správnosť vzhľadom k abstraktnému modelu. Pri verifikácii overujeme izomorfizmus simulačného a abstraktného modelu. Izomorfizmus v tomto prípade znamená, že simulačný model odpovedá $1 : 1$ k abstraktnému modelu z hľadiska štruktúry a chovania. Validáciu modelu na druhej strane chápeme ako mieru správnosti získaných výsledkov, pričom reprezentuje proces, pri ktorom dokazujeme či pracujeme s adekvátnym modelom simulovaného systému.

- Simulácia - experimentovanie s reprezentáciou simulačného modelu. Popis experimentu môžeme rozdeliť na niekoľko fáz:

  - Príprava experimentu - vytvorenie simulátoru a modelu a ich inicializácia na vhodné hodnoty.
  - Prevedenie experimentu - spustenie simulátoru, riadenie behu simulácie a záznam výsledkov.
  - Ukončenie experimentu - záznam výsledkov a cieľového stavu, zrušenie modelu a simulátoru.

  Každá fáza má svoj vlastný popis experimentu, pričom experimenty môžu byť ľubovoľne opakované. Nasledujúce experimenty môžu brať do úvahy výsledky predchádzajúcich a podobne.

- Analýza a spracovanie výsledkov - overovanie správnosti výsledkov. Porovnávame výsledky s reálne nameranými dátami, prípadne vykonávame štatistické spracovanie výsledkov.

4.1.1 Diskrétna simulácia

Tento typ simulácie je vhodný všade tam, kde nepotrebujeme spojitosť. Vhodným príkladom sú systémy hromadnej obsluhy. Existujú rôzne formalizmy ako popísať štruktúry diskrétnych systémov a ich chovanie v čase. Medzi základné formy popisu patria:

- Popis v simulačnom alebo obecnom programovacom jazyku formou procesov a udalostí.
- Konečné automaty[2]
- Celulárne automaty[12] - popisujú priestorové systémy tvorené mriežkou buniek. Je popisované chovanie buniek v závislosti na stavu ich okolia.
- Petriho siete - graf popisujúci stavy a prechody medzi stavmi. Pomocou nich je možné vyjadriť paralelizmus a nedeterminizmus[12].
- DEVS (Dicrete EVent Specified system) - popis na základe teórie systémov, dovoľuje popis hierarchických modelov a distribuovanú simuláciu[13]
- iné - π-calculus[15], PRAM[14] a pod.

4.1.2 Kvaziparalelizmus

Naimplementovaný systém má obvykle veľké množstvo súbežne bežiacich (paralelných) procesov, kde každý z nich je popísaný sekvenciou príkazov programovacieho jazyka vyjadrujúcich chovanie celej triedy procesov. Pri simulácii musíme zaistiť ich vzájomnú komunikáciu, čo môžeme dosiahnuť zasielaním správ a používaním synchronizácie zdieľaných prostriedkov. Obvykle je počet procesov oveľa vačší ako počet procesorov, dokonca aj ako počet systémom dovolených procesov. Preto musíme paralelné procesy implementovať kvaziparalelne, čo znamená, že vždy beží len jeden proces a ostatné sú pozastavené. Týmto odpadá nutnosť zamykania zdielaných dátových štruktúr. Kvaziparalelizmus je teda forma implementácie paralelných procesov na jednoprocesorovom počítači. To znamená, že paralelné procesy sa neprevádzajú súčasne, ale jeden po druhom. Toto zjednodušenie však prináša so sebou problém, pretože výsledok simulácie závisí na poradí udalostí systému. Preto väčšina systémov zavádza pojem „priorita procesu“, ktorý jednoznačne identifikuje poradie vykonania procesov.

Zmena stavu diskrétneho systému je popísaná pomocou udalosti, ktorej popis sa líši na základe použitého formalizmu. Napríklad Petriho siete popisujú udalosť formou prechodu

a v konečnom automate je udalosť hranou. Udalosť je atomická a v simulačných jazykoch triviálna operácia. V zásade platí, že akýkoľvek diskrétny systém je možné popísať pomocou udalostí, avšak v praxi je oveľa prehľadnejší popis pomocou procesov.

Proces je postupnosť za sebou vykonávajúcich udalostí, súčasťou ktorých sú prostriedky vyjadrujúce čakanie po zadanú dobu v modelovom čase, prípadne prostriedky vyjadrujúce popis čakania vo fronte a podobne.

4.1.3 Systém hromadnej obsluhy

Diskrétny systém typicky obsahujúci procesy a popis ich príchodov do systému, obslužné linky a fronty nazývame systémami hromadnej obsluhy. Pri simulácii sledujeme informácie o časových priebehoch procesov a doby čakania vo frontách. Tieto informácie získavame väčšinou vo forme štatistík. Vstupom systému sú požiadavky prichádzajúce z jeho okolia. Obvykle definujeme stochastický popis pre tieto požiadavky. Napríklad určíme strednú dobu medzi príchodmi (exponenciálne rozloženie pravdepodobnosti) alebo počet príchodov za jednotku času (Poissonovo rozloženie pravdepodobnosti) [18].

- Exponenciálne rozloženie - doba medzi dvoma po sebe idúcimi výskytmi udalostí. Označujeme ju $X \sim Exp(\lambda)$, kde X je spojitá veličina.
- Poissonovo rozloženie - počet výskytov udalostí za časovú jednotku. Označujeme $Y \sim Po(\lambda)$, kde $Y \in 1, 2, 3, ...$

Parameter λ udáva priemerný počet výskytov udalostí za jednotku času. Obe rozloženia môžeme medzi sebou jednoducho prevádzať.

Systém hromadnej obsluhy modelujeme ako diskrétny systém, kde procesy modelujú paralelné chovanie objektov systému. Obslužné linky modelujeme špeciálnymi objektmi a fronty sú špeciálne zoznamy, do ktorých ukladáme objekty resp. procesy.

4.2 Abstraktný model systému križovatiek

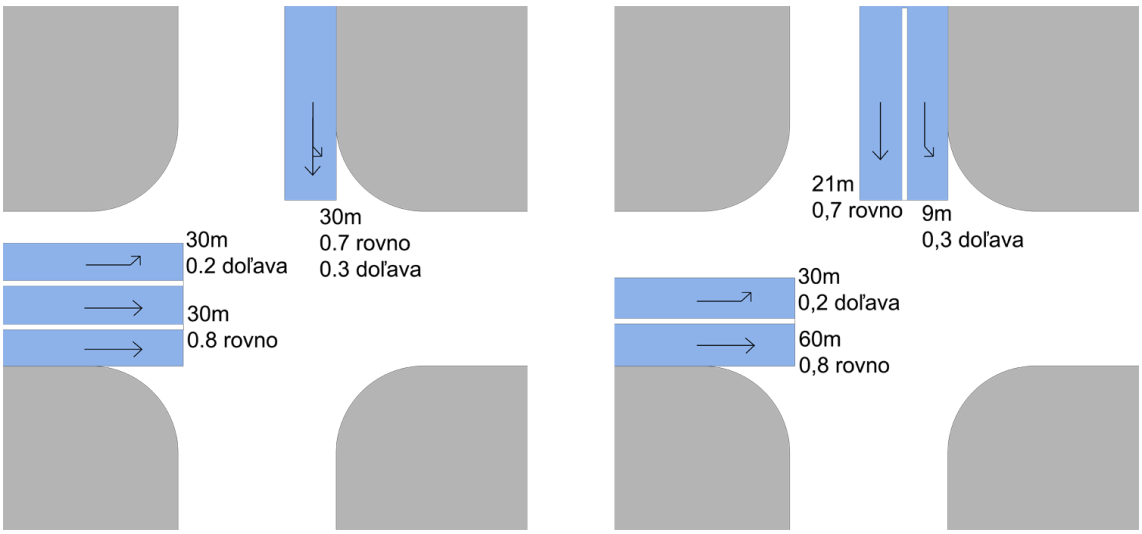
Vytvorenie abstraktného modelu je základným kameňom simulácie. V kapitole 2 sme podrobne definovali reálny model križovatky. Jednotlivé križovatky pospájané cestami tvoria komplexný dopravný systém. Tento model reálneho systému je nutné zjednodušiť resp. nájsť vhodnú mieru zjednodušenia tak ,aby výsledky simulácie boli uspokojivé.

4.2.1 Model križovatky

Definujme najprv model križovatky. Križovatka má viacero vstupných a výstupných smerov. Každý vstupný smer má niekoľko odbočovacích prúdov v závislosti na hustote premávky z daného smeru. Vstupné prúdy tvoria základný kameň križovatky. Ich počet je

z pravidla rovný trom, tj. pravý a ľavý odbočovací prúd a prúd smerujúci rovno. V prípade väčšej križovatky sú niektoré prúdy zdvojené, dokonca až ztrojené.

Za simulačné parametre vstupných prúdov križovatky považujeme dĺžku resp. kapacitu jednotlivých prúdov, ktorá je zásadným faktorom priepustnosti danej križovatky. Inak povedané, so zväčšujúcou sa kapacitou jednotlivých prúdov, ako aj s ich počtom, sa zväčšuje kapacita danej križovatky, čo má za dôsledok zvýšenie jej priepustnosti. Počet a dĺžka prúdov sú limitované fyzickým priestorom určeným pre križovatku, preto nie vždy nám okolnosti dovoľujú postaviť dostatočne dlhé prúdy. Nastavenie týchto parametrov je kľúčovým pre správnosť simulácie a priblíženie sa k reálnemu modelu križovatky. Zdvojenie resp. ztrojenie niektorého zo vstupných prúdov nemusíme uvažovať. V simulačnom modeli nastavíme zdvojenú resp. ztrojenú dĺžku, a tým to úspešne odsimulujeme. Na druhej strane prúd, ktorý obsahuje dva alebo dokonca tri smery, odsimulujeme vhodným nastavením ich dĺžky. V prípade prúdu aút smerujúcich rovno a doľava, s dĺžkou prúdu 30 metrov, s pravdepodobnosťou odbočenia 0.7 rovno a 0.3 doľava, môžeme v simulačnom modeli nastaviť dva prúdy, jeden rovno s dĺžkou 21 metrov a druhý doľava s dĺžkou 9 metrov. Pravdepodobnosti odbočenia ostávajú nezmenené. Týmto sme vhodne aproximovali danú situáciu, preto môžeme tvrdiť, že abstraktným modelom vstupných prúdov sú tri vstupné prúdy (doľava, rovno a doprava) s parametrom ich dĺžky. Problém lepšie znázorňuje obrázok 4.1, kde sú obe varianty prevodu na abstraktný model ukázané.



Obrázok 4.1: Reálny vs. abstraktný model prúdov križovatky

Doba prejazdu medzi hranicou križovatky a vstupom na výstupnú vozovku je závislá na type vozidla a vzdialenosti medzi týmito bodmi. Vzdiadelnosť pre zjednodušenie modelu zanedbávame, v dôsledku čoho nám stačí určiť dobu prejazdu pre každý typ vozidla.

Križovatka sa od inej križovatky líši dovolenými odbočovacími smermi a signálnym plá-

nom. Dopravní inžinieri definujú pre každý vstupný bod križovatky jeho možné výstupy, tj. definujú dovolené smery odbočenia. Potom nasleduje zostavenie signálneho plánu križovatky. V tejto fáze návrhu sa určí počet riadiacich fáz, pričom každá fáza má svoju dĺžku trvania a zoznam prúdov, ktoré majú zelenú. V poslednom rade definujú optimálne poradie fáz. Abstraktný model bude obsahovať všetky tieto vlastnosti. Nutným rozšírením je možnosť dynamicky riadiť určité smery tj. mať možnosť zostaviť plán, ktorý obsahuje jednotlivé fázy a ich dĺžky trvania. Zostavovaním optimálneho plánu pre systém križovatiek sa zaoberá kapitola 6. Vhodným príkladom môže byť signálny plán križovatky tvaru „T“ pri použití troch fáz. Tabuľka 4.1 zobrazuje signálny plán práve takejto križovatky. Predpokladáme, že križovatka má pre každý smer oddelený odbočovací prúd. Vstupy resp. výstupy križovatky sú označené ako INx resp. OUTx.

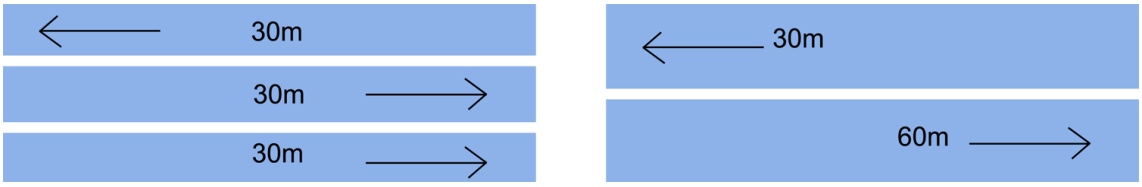
Fáza	Povolené smery	Dĺžka fázy
1	IN1 - OUT2 IN1 - OUT3	30s
2	IN2 - OUT1 IN2 - OUT3	15s
3	IN3 - OUT1 IN3 - OUT2	15s

Tabuľka 4.1: Príklad signálneho plánu

4.2.2 Model cesty

Výstupné prúdy nemusíme uvažovať, pretože z hľadiska simulácie nemajú žiadny význam. Na druhej strane si musíme strážiť, či autá opúšťajúce daný prúd majú kam odchádzať. Túto informáciu získame z ďalšieho modelu, ktorým je cesta. Parametre cesty sú vzdialenosť medzi dvoma križovatkami a počet prúdov, ktorý môžeme, podobne ako pri vstupných prúdoch, zanedbať. Pri x prúdoch cesty x-krát zväčšíme jej dĺžku. Cestu musíme definovať v každom smere, aby sme zachovali jej rozmanitosť. Prevod modelu cesty na jej abstraktný model lepšie znázorňuje obrázok 4.2. Vzdialenosti medzi križovatkami sú dôležité preto, aby sme vedeli vypočítať, koľko áut sa medzi ne zmestí, a aby v prípade preplnenia cesty medzi nimi simulácia zablokovala prejazd áut z križovatky.

Do križovatky vstupuje prúd áut deliaci sa do odbočovacích pruhov. Vstupný prúd aút, ako aj každý odbočovací pruh musíme uvažovať ako samostatnú frontu, pričom auto môže prejsť do odbočovacej fronty iba vtedy, pokiaľ nie je plná. Vstupná fronta môže narastať iba do takej veľkosti, pokiaľ dosiahne hranicu ďalšej križovatky. Výstup áut z križovatky je možný iba v prípade, že fronta nasledujúcej križovatky nie je plná. Týmto spôsobom môžeme simulovať situáciu, pri ktorej autá dostanú zelenú a nemôžu pokračovať v jazde,



Obrázok 4.2: Reálny vs. abstraktný model cesty

pretože by nemali kam odísť.

4.2.3 Model vozidiel

Do reálneho modelu prichádzajú vozidlá s určitou intenzitou. Do abstraktného modelu zaznamenáme počty áut za jednotku času udávané s pomocou Poissonovo časového rozloženia pravdepodobnosti (480 áut za hodinu), ktoré podľa potreby môžeme jednoducho previesť na exponenciálne rozloženie. Tieto údaje musíme poznať, zadať, pri každom vstupe križovatky alebo na vstupoch systému križovatiek. Vo väčšine miest sa stretávame s problémom ako zvládnuť nákladnú dopravu v meste. Tieto nadrozmerné vozidlá majú oveľa pomalší rozbeh, zaberajú štyri krát viac miesta ako bežné vozidlá, zahlcujú križovatky, čo spôsobuje zvýšenie čakacích dôb na vstupoch križovatky, a radikálne prispievajú k tvorbe smogu v mestách. Preto by sa mala nákladná doprava začleniť do simulácie, tj. musíme poznať počty vozidiel nákladnej dopravy, ako aj autobusov za jednotku času.

Vozidlá na križovatke odbočujú v závislosti na ich destinácii. V simulácii by bolo veľmi obtiažne zostavovať pre každé auto v systéme plán, na základe ktorého bude na jednotlivých križovatkách odbočovať. Preto musíme poznať pravdepodobnosť, s akou autá prichádzajúce do križovatky z daného smeru odbočia doľava, doprava alebo rovno, čo môžeme odhadnúť na základe štatistického pozorovania križovatky, alebo nám tieto údaje môže poskytnúť príslušný dopravný inžinier. Celkovo na priemernú križovatku, ktorá má štyri smery a dovoľuje každému smeru odbočiť do všetkých ostatných smerov, pripadá dvanásť rôznych pravdepodobností odbočenia.

Podobne ako vozidlá v skutočnosti, aj vozidlá v simulácii musia niekam odchádzať. Túto skutočnosť je treba brať na zreteľ pri implementácii simulačného modelu.

4.3 Simulačný model systému križovatiek

Po zostavení abstraktného modelu, ktorý je vhodne zjednodušeným modelom reality križovatiek, sa môžeme pustiť do implementácie simulačného modelu. Existuje niekoľko možností ako implementovať simulačný model. Jednou z nich je priama implementácia vo vybranom vyššom programovacom jazyku, čo by znamenalo vývoj na zelenej lúke. Vhodným výberom je však knižnica obsahujúca hlavné štruktúry vrátane možnosti popisu ich správania.

Značnou výhodou v oblasti simulácií je použitie objektového prístupu[1][16]. Nasledujúca časť popisuje simulačnú knižnicu, ktorá využíva objektovo orientovaný prístup.

4.3.1 Knižnica Simlib/C++

Simlib/C++ je simulačná knižnica pre objektovo orientovaný programovací jazyk C++[17]. Umožňuje spojitý, diskrétny i kombinovaný prístup. Popis modelov je objektovo orientovaný na základe abstrakcií. Pracuje s GNU C++ prekladačom, čo zaručuje prenositeľnosť na rôzne platformy (Linux, FreeBSD, Windows - MinGW) [11]. Obsahuje niekoľko bázových tried:

- *Process* - trieda pre modelovanie procesov
- *Event* - trieda pre modelovanie udalostí
- *Facility* - obslužná linka s výlučným prístupom
- *Store* - obslužná linka so zadanou kapacitou
- *Queue* - prioritná fronta
- *Stat, Histogram* - triedy pre zber štatistík

Popis udalostí implementujeme triedou odvodenou od bázovej triedy *Event*, pričom každá trieda musí mať definovanú metódu *Behavior* s príkazmi, ktoré sa vykonajú pri spustení udalosti. Pomocou metódy *Activate* môžeme implementovať periodické vyvolávanie udalostí. Tento spôsob nám, okrem iného, slúži pre generovanie procesov. Globálna premenná *Time* reprezentuje aktuálny modelový čas.

Knižnica definuje celú radu rôznych generátorov pseudonáhodných čísiel:

- *Random()* - rovnomerné rozloženie $\langle 0, 1 \rangle$.
- *Uniform(L, H)* - rovnomerné rozloženie $\langle L, H \rangle$.
- *Exponential(E)* - exponenciálne rozloženie so stredom E.
- *Normal(M, S)* - normálne rozloženie so stredom M a rozptylom S.

Procesy sú odvodené od abstraktnej triedy *Process* a po jej aktivácii je volaná metóda *Behavior*. Beh tejto funkcie je prerušiteľný pri nasledujúcich operáciách:

- Explicitné čakanie pomocou príkazu *Wait(t)*.
- Čakanie vo fronte pri obsadzovaní zariadení alebo skladov v rámci príkazov *Seize* a *Enter*.

[1] Objektovo orientovaný prístup bol od jeho počiatku vyvíjaný pre oblasť simulácií.

- Zastavenie procesu na neurčito pomocou príkazu *Passivate*. Proces je znovu aktivovaný príkazom *Activate*.

Abstraktná trieda *Queue* definuje prioritnú FIFO[2] frontu radenú podľa priority procesov. Táto trieda slúži na definovanie vlastných špeciálnych prvkov. Dôležité sú metódy *Insert*, ktorá vloží proces do fronty, a metóda *getFirst* vracajúca prvý proces v poradí.

Vlastný beh simulácie popisujeme vo funkcii *main*, kde za príkazom *Init(od , do)* nasleduje inicializácia modelov. Simulácia sa spustí príkazom *Run*. Po dokončení simulácie môžeme pristúpiť k analýze nazbieraných údajov. Štatistiky nám poskytnú rôzne informácie o modeloch simulácie, ako napríklad dĺžky frónt, doby čakania vo frontách, celková doba strávená v systéme a podobne. Pre všetky tieto údaje sú zaznamenané ich maximá, minimá, priemerné hodnoty a smerodajné odchýlky.

Sekvenciu *Init*(t_0,t_1)*;...;Run();...;* môžeme ľubovoľne opakovať. Túto možnosť využijeme napríklad pri optimalizácii parametrov modelu.

4.4 Implementácia simulačného modelu

Implementácia simulačného modelu používa knižnicu *Simlib/C++* podrobne popísanú v predchádzajúcej časti.

4.4.1 Model vozidla

Každé vozidlo je v systéme reprezentované abstraktnou triedou *Vehicle* odvodenou od triedy *Process*. Metóda *Behavior* definuje správanie objektu takto:

- proces je pozastavený na určitú dobu v závislosti na type vozidla - simuluje sa tým prejazd križovatky.
- na základe nasledujúceho objektu sa proces vozidla vloží na vstup križovatky *Crossroads* alebo cesty *Road*. V prípade, že nasledujúci objekt je čierna diera *Blackhole*, tak objekt procesu opúšťa systém a zaniká.
- proces zavolá metódu *Passivate*, čím je pozastavená jeho činnosť do doby, pokiaľ iný objekt v simulácii nezavolá metódu *Activate*.

Triedy odvodené od abstraktnej triedy *Vehicle* sú *Car*, *Bus*, *Truck*. V týchto triedach sa reimplementujú len metódy *GetWaitTime* a *GetLength*, ktoré vracajú čas potrebný pre prejazd križovatkou resp. dĺžku vozidla vzávislosti na type odvodenej triedy. Týmto spôsobom sa veľmi ľahko začlenia do systému akékoľvek ďalšie typy vozidiel.

Generator je trieda odvodená od triedy *Event*, simulujúca exponenciálne časové rozloženie príjazdov áut do systému. Hodnoty tohto rozloženia v rôznych generátoroch v systéme

[2] First In First Out

je nutné meniť podľa určitých pravidiel, a tak simulovať nárazovú dopravnú vlnu v systéme. Generátor generuje tri typy vozidiel v závislosti na ich parametroch. Parametrom exponenciálneho rozloženia generátora je súčet parametrov všetkých vozidiel. Následne na základe rovnomerného rozloženia pravdepodobnosti a ich percentuálneho zastúpenia sa generuje výsledný typ vozidla. Vezmime v úvahu tri typy vozidiel. Autá prichádzajú do systému s intenzitou 500 krát za hodinu, podobne autobusy s intenzitou 20 krát za hodinu a nakoniec kamióny 5 krát za hodinu. Z uvedených intenzít vidíme, že sa jedná o Poissonovo rozloženie pravdepodobnosti podrobne popísaného v časti 4.1.3, preto je nutné previesť tieto údaje do exponenciálneho rozloženia. Najprv si určíme percentuálne zastúpenie jednotlivých áut. V poradí to je $0,95$, $0,04$ a $0,01$. Potom súčet intenzít prevedieme na exponenciálne rozloženie, pričom uvažujeme modelový čas v minútach. Výsledok získame ako prevrátenú hodnotu súčtu Poissonových rozložení všetkých vozidiel vynásobenú 60.

$$Gen_{exp} = \frac{60}{Car_{poisson} + Bus_{poisson} + Truck_{poisson}} \tag{4.1}$$

4.4.2 Model križovatky

Predtým, ako si podrobne popíšeme implementáciu modelu križovatky, musíme si definovať triedu *MyQueue*, ktorá rozširuje funkcionalitu triedy *Queue*. Je pridaná podpora pre výpočet dĺžky fronty v metroch, čo je potrebné pre zistenie, či sa vozidlo vstupujúce do nej ešte zmestí. Nová metóda *SeeFirst* má špeciálnu funkčnosť. Vráti element nachádzajúci sa na začiatku fronty, ale na rozdiel od metódy *GetFirst* nechá prvok stále vo fronte. Použitie tejto metódy bude podrobne popísané neskôr.

Crossroads križovatka je odvodená od triedy *Event*, jej funkčnosť je riadiť svoje vstupné fronty. Pri riadení vykonáva plán stanovený objektom *Controller*. Správanie tejto triedy by sme mohli popísať nasledovne:

- Zavolá metódu *GetNextPhase()* objektu *Controller*, ktorá vráti signálnu fázu. Inak povedané, vráti všetky povolené smery a dĺžku „zelenej“.
- Nastáva celočervený režim.
- Čaká 3 sekundy, čo zabezpečí bezpečný prejazd aj takých vozidiel, ktoré prešli križovatkou pri zopnutí červenej.
- Dá zelenú všetkým povoleným smerom.
- Zavolá metódu *Activate(Time+l)*, kde l je dĺžka aktuálnej fázy. Týmto sa riadenie križovatky znovu aktivuje, až keď uplynie doba „zelenej“.

Táto trieda ďalej obsahuje dvanásť objektov typu *Lane*, ktorým sa modelujú jednotlivé vstupné prúdy križovatky. Objekt križovatky pri zavolaní metódy *In*, ktorej parametrom je vozidlo vstupujúce do križovatky, najprv zistí, odkiaľ ide a kam má vozidlo namierené.

Následne ho vloží do príslušnej fronty. Metóda *IsFull* s príslušnými parametrami identifikujúcimi frontu vracia informáciu o tom, či je daná fronta plná alebo nie. Súčasťou objektu križovatky je štatistický objekt *Histogram*, ktorý zabezpečuje štatistiku o dobách strávených vo frontách križovatky. Táto štatistika je dôležitá pri optimalizácii riadenia, pretože ukazuje, či optimalizovanie nezvýhodnilo málo používané smery.

Objekt typu *Lane* je odvodený od abstraktnej triedy *Process* a jeho úlohou je modelovať vstupné prúdy križovatky. Zapúzdruje v sebe objekt typu *MyQueue* a má definované nasledovné správanie, ktoré beží v nekonečnom cykle až do ukončenia simulácie. Objekt pomocou metódy *IsFull* poskytuje informáciu o tom, či je jeho fronta plná.

- Čaká pokiaľ je fronta prázdna.
- Zistí, aké vozidlo sa nachádza na začiatku fronty.
- Pokračuje, pokiaľ má kam vozidlo odísť (fronta nasledujúceho objektu nie je plná), inak čaká.
- Vloží vozidlo do fronty nasledujúceho objektu.
- Zapíše do štatistiky križovatky dobu čakania.
- Aktivuje proces vozidla.

4.4.3 Model cesty

Objekt modelujúci cestu, teda objekt *Road*, obsahuje podobne ako objekt *Lane* vlastnú frontu. Správanie tohto objetku je však úplne inak definované, ale podobne beží v nekonečnom cykle:

- Čaká pokiaľ je fronta prázdna.
- Zistí, aké vozidlo sa nachádza na začiatku fronty.
- Vygeneruje číslo pomocou generátoru pseudonáhodných čísiel rovnomerného rozloženia.
- Na základe vygenerovaného čísla určí smer odbočenia pre vozidlo.
- V prípade, že nasledujúci objekt je križovatka, tak čaká, pokiaľ príslušný vstupný prúd križovatky nie je plný, a až po ukončení čakania, vloží vozidlo do príslušného vstupného prúdu.
- Vyberie z fronty vozidlo na prvej pozícii.
- Vybrané vozidlo je aktivované.

Parameter vzdialenosti medzi modelmi križovatky, generátora a čiernej diery udáva dĺžku fronty v metroch. Objekt pri každom pokuse o vstúpenie na cestu kontroluje, či vozidlá majú kam ísť. Tento údaj má však význam len medzi dvoma križovatkami. Pri ostatných objektoch nastavujeme nulovú veľkosť, čím simulujeme nekonečnú frontu. Vymoženosť nekonečnej fronty používame špeciálne medzi generátorom a križovatkou, vďaka čomu sa vozidlá, ktoré križovatka nachádzajúca sa za generátorom neobslúžila, radia v nekonečnej fronte.

Blackhole čierna diera je objekt ukončujúci systém križovatiek. Jeho funkčnosť je ukončenie procesov doňho vstupujúcich a zbieranie štatistík o celkovej dobe strávenej v systéme. V prípade potreby by sme mohli rozšíriť funkcionalitu tohto objektu tak, aby mohol simulovať nejaký preplnený systém s určitým potenciálom priepustnosti.

4.5 Proces simulácie

Simulácia pracuje v určitom dopredu definovanom časovom rozmedzí, ktorého časovou jednotkou je jednotka modelového času, v našom prípade minúta. Knižnica *Simlib/C++* definuje metódu *Init(od, do)*, ktorej parametrami sú dva časy určujúce dĺžku simulácie. Je zrejmé, že dĺžka simulácie, aj keď sa jedná o modelový čas, ovplyvňuje dĺžku trvania simulácie na procesore. Preto je vhodné nastaviť časové rozmedzie tak, aby sme počas neho získali dostatočné množstvo relevantných štatistických údajov, a aby dĺžka trvania simulácie na procesore nebola zbytočne dlhá. Vhodným rozmedzím preto bude interval $0 - 120$ minút resp. $0 - 2$ hodiny.

Po inicializácii časového rozmedzia simulácie inicializujeme model celého systému. Tento proces inicializácie obnáša prejdenie všetkých objektov a ich vlastností, ktoré si užívateľ zadal pomocou grafického uživateľského rozhrania[3] a vytvorenie k nim príslušných simulačných objektov presne popisujúcich model systému. Po dokončení inicializácie spustíme simuláciu metódou *Run*, ktorá väčšinou do pár sekúnd skončí.

Po skončení simulácie máme k dispozícii veľké množstvo štatistických údajov. Zaujímavé pre nás sú hlavne čakacie doby v jednotlivých frontách a ich dĺžky. Mali by sme brať do úvahy priemerné aj maximálne hodnoty týchto údajov. Maximálna hodnota údajov nás informuje o extrémoch, ktoré nastali počas simulácie. Tieto extrémy sú pre nás dôležité, pretože odzrkadľujú situáciu na cestách, kedy je z ničoho nič daný smer preťažený. Tento jav na cestách môžeme radiť medzi javy náhodné. Na druhej strane priemerná hodnota nám dáva stabilné údaje o dopravnej situácii po skončení simulácie.

V tejto časti je nutné vykonať validáciu simulačného modelu vzhľadom k reálnemu modelu. Budeme teda skúmať, či výsledky simulácie odpovedajú skutočne nameraným hodnotám. Reálne hodnoty je veľmi obtiažne získať, preto často vytvoríme model križovatky spolu so vstupnými údajmi a výstupnými resp. očakávanými údajmi na „papieri“.

[3] Grafické užívateľské rozhranie je podrobne popísané v kapitole 5

Následne overujeme takto navrhnutý fiktívny model.

4.5.1 Radič križovatiek

Riadenie jednotlivých križovatiek systému vykonáva objekt *Controller*, ktorého správanie je definované nasledovne:

- Križovatka sa pomocou metódy *Register* zaregistruje u kontroléru. Na základe registrácie je križovatke pridelený unikátny identifikátor.
- Objekt križovatky počas simulácie volá metódu *GetNextPhase()* s parametrom identifikátora a kontrolér mu vráti povolené smery a dĺžku trvania fázy.

Systém riadenia by nebol možný bez objektu *SignalPlan*, ktorý zapúzdruje počet fáz a pre každú fázu jej dovolené smery a dĺžku trvania fázy. Objekt *PhasePlan* na druhej strane udržuje informáciu o optimálnom poradí fáz a je zväčša používaný plánovacím algoritmom, ktorému sa podrobne venuje kapitola 6.

Kapitola 5

Grafické užívateľské rozhranie

Po dôkladnej analýze problému svetelného riadenia križovatiek a po zistení, že simulácia systému križovatiek potrebuje veľké množstvo vstupných štatistických údajov, je na rade návrh grafického užívateľského rozhrania. Pomocou tohto rozhrania sa musí dať navrhnúť akýkoľvek model dopravného systému, ktorý musí spĺňať podmienky podrobne definované v časti 4.2 popisujúcej abstraktný model systému križovatiek.

Simulačná knižnica, ako už bolo spomenuté, je prenositeľná a kompatibilná s prekladačom *GNU C++*. Na základe tohto predpokladu bola vybraná knižnica pre tvorbu grafického užívateľského rozhrania *wxWidgets*, ktorá spĺňa všetky uvedené požiadavky [19]. Knižnica taktiež obsahuje základné prvky grafického užívateľského rozhrania, ako napríklad menu, panel nástrojov, scrollbar a podobne. Doimplementovať však musíme spôsob vytvárania modelu.

Samotná aplikácia pozostáva z jedného hlavného okna (obrázok 5.1), ktoré obsahuje menu, panel nástrojov, grafickú návrhovú plochu a lišty nastavenia pre dynamické zadávanie vlastností práve označeného komponentu plochy.

5.1 Pracovná plocha modelu

Pracovná plocha modelu tvorí základný kameň grafického užívateľského rozhrania. Umožňuje efektívne a vizuálne spravovanie modelu systému križovatiek a taktiež vhodne graficky zobrazuje výsledok simulácie. Nasledujúca časť sa zaoberá popisom grafických elementov a objektového návrhu plochy.

5.1.1 Popis grafických elementov plochy

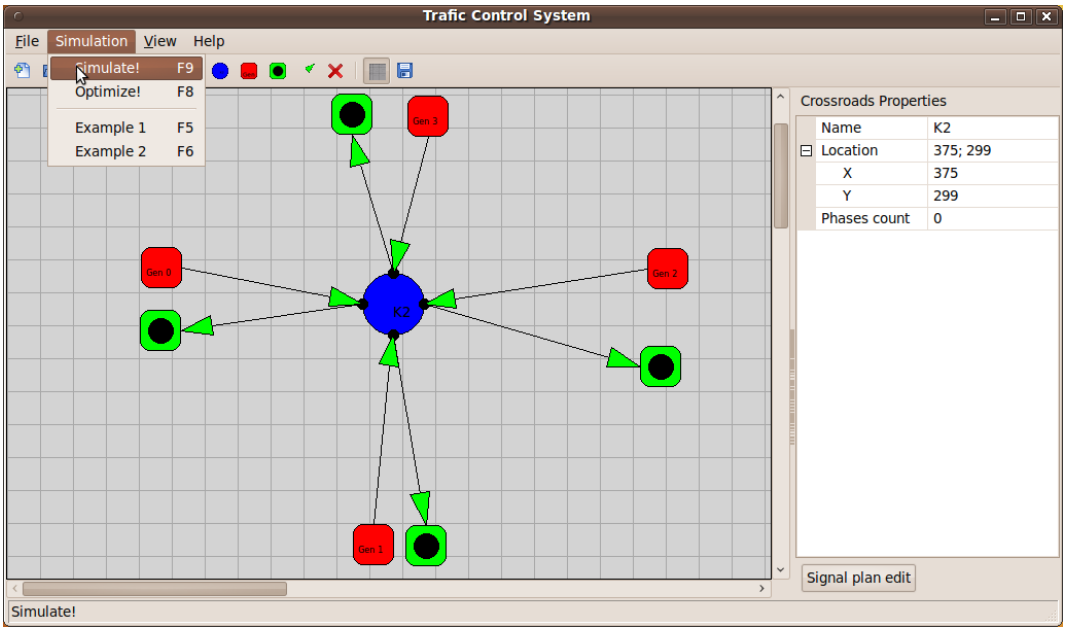
Návrhová plocha pozostáva z jednotlivých komponent reprezentujúcich simulačné prvky tvoriace model systému. Komponenty sú umiestnené na paneli nástrojov, pričom vždy je aktívny len jeden komponent. Základné komponenty návrhovej plochy teda sú:

- 

: *Generator*, prvok reprezentujúci okraj systému, generovanie áut na základe exponenciálneho časového rozloženia pravdepodobnosti.
-  : Čierna diera *Black hole*, ukončenie systému na jeho okraji, zber štatistík.
- 

: Objekt križovatky
-  : Orientovaná šípka, vzťah relácie, znázorňuje tok vozidiel medzi križovatkami, generátorom a križovatkou, medzi križovatkou a čiernou dierou.

Jednotlivé komponenty sa pospájajú pomocou orientovanej šípky, čím vytvoria systém - simulačný model. Vlastnosti každého komponentu sa dajú nastavovať pomocou lišty nastavenia. Napríklad objekt generátora poskytuje nastavenie polohy a zadanie počtu vozidiel za jednotku času. Vzdialenosť medzi križovatkami ako aj počet odbočovacích pruhov sú vlastnosťami relácie medzi križovatkami.



Obrázok 5.1: GUI - celkový pohľad

5.1.2 Popis objektového návrhu plochy

Objekty plochy uchovávajú všetky informácie pre grafické užívateľské rozhranie ako aj informácie potrebné pre vytvorenie simulačného modelu, vrátane jeho parametrov. Vykresľovanie resp. prekresľovanie návrhovej plochy nastáva pri každom pohybe či zmene rozmerov

okna. Táto činnosť pozostáva z cyklického prechádzania zoznamu všetkých objektov, z ktorých sa vyberú parametre potrebné pre vykreslenie. Podobné prechádzanie zoznamu je v prípade kliknutia myšou, pri ktorom sa zisťuje na aký objekt užívateľ klikol.

Objekty plochy boli navrhnuté takto:

- *GenericObject* - abstraktný objekt obsahujúci vlastnosti ako napríklad meno objektu a pozícia. Dôležitá je tiež jeho funkcionalita, pretože zapúzdruje metódy pre výpočet *BoundingBoxu*[1], stredu telesa a zásahu telesa *HitTest* použitého pri vyhľadávaní, či užívateľ klikol resp. zasiahol objekt.

- *GeneratorObject* - objekt odvodený od objektu *GenericObject*. Obsahuje ukazateľ na nasledujúci objekt, ukazateľ na príslušný simulačný objekt a hodnoty intenzity všetkých typov vozidiel, s akou vozidlá prichádzajú do systému. Objekt ďalej vykonáva prepočet z Poissonovho časového rozloženia na exponenciálne časové rozloženie a vypočítava percentuálne zastúpenie jednotlivých vozidiel. Výpočet bol podrobne popísaný v časti 4.4 venovanej popisu simulačného modelu.

- *BlackholeObject* - jednoduchý objekt odvodený od objektu *GenericObject*.

- *CrossroadsObject* - objekt odvodený od objektu *GenericObject*. Obsahuje ukazatele na všetky prvky, ktoré sú v relácii s križovatkou. Ďalej zapúzdruje objekty *Signalplan* a *PhasePlan* podrobne definované v časti 4.5.

- *Siminfo* - objekt obsahujúci maximálny čas a veľkosť preťažených smerov. Zobrazuje sa formou štvorca vyplneného farbou podľa výšky preťaženia.

- *ArrowObject* - objekt odvodený od objektu *GenericObject* a zaisťuje konzistenciu celého systému tým, že vytvára relácie medzi ostatnými objektmi. Obsahuje údaje o pravdepodobnosti odbočenia z daného smeru. Podľa tejto pravdebodobnosti sa autá radia do odbočovacích prúdov. Ďalším údajom je vzdialenosť medzi dvoma objektmi nachádzajúcimi sa v relácii pomocou tohoto objektu.

Všetky objekty návrhovej plochy majú definované vstupno-výstupné metódy [2] pre jednotlivé atribúty a metódy určené pre ukladanie atribútov do *XML*, ktorým sa zaoberá časť 5.4 tejto kapitoly.

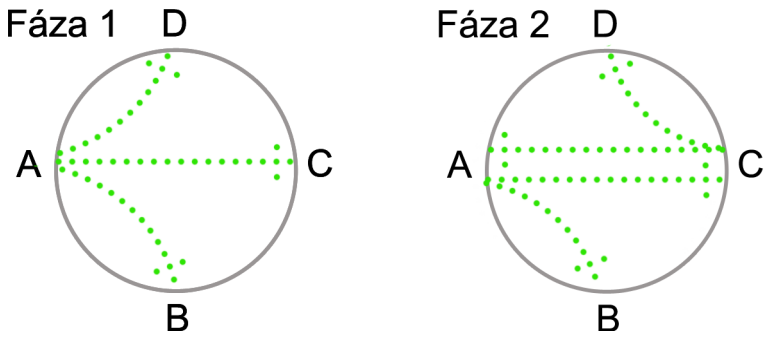
Posledným najdôležitejším objektom je objekt *AreaObject*, ktorý nesie v sebe dva zoznamy objektov. Jeden zoznam obsahuje všetky objekty typu *ArrowObject*, čiže objekty tvoriace prepojenia, a druhý zoznam pozostáva z ostatných objektov plochy. Zoznamy sa prechádzajú pomocou iterátorov, čo je najoptimálnejší spôsob v jazyku *C++*.

[1] obalové 2D teleso

[2] v ang. terminológii Setters/Getters

5.2 Tvorba signálneho plánu

Možnosť zadávania signálneho plánu so sebou prináša problém veľkého množstva vstupných údajov, pričom zadaný signálny plán musí byť konzistentný vzhľadom ku križovatke. Inak povedané nesmie dovoliť pridať do plánu neexistujúci smer. Pri tvorbe signálneho plánu je ponechaná užívateľovi určitá voľnosť, ktorá dovoľuje zadávať veľké množstvo variácií povolených smerov. Uvažujme príklad, pri ktorom dopravný inžinier navrhol signálny plán križovatky nasledovne. Jedná sa o križovatku, ktorá ma štyri vstupy a štyri výstupy, pričom vstupy sú označené písmenami A, B, C, D v poradí proti smeru hodinových ručičiek. Písmenami L, R, P sú označené odbočovacie smery (vľavo, rovno a vpravo). V prvej fáze majú zelenú smery AL, AR, AP, pričom v druhej fáze sa rozhodol ponechať zelenú hlavnému smeru AR a povoliť smery AP, CR a CL. Pre lepšie pochopenie je príklad popísaný na obrázku 5.2 a ukazuje rôzne typy variácií pri zadávaní signálneho plánu. Na koniec obrázok 5.2 ukazuje okno zadávania signálneho plánu.

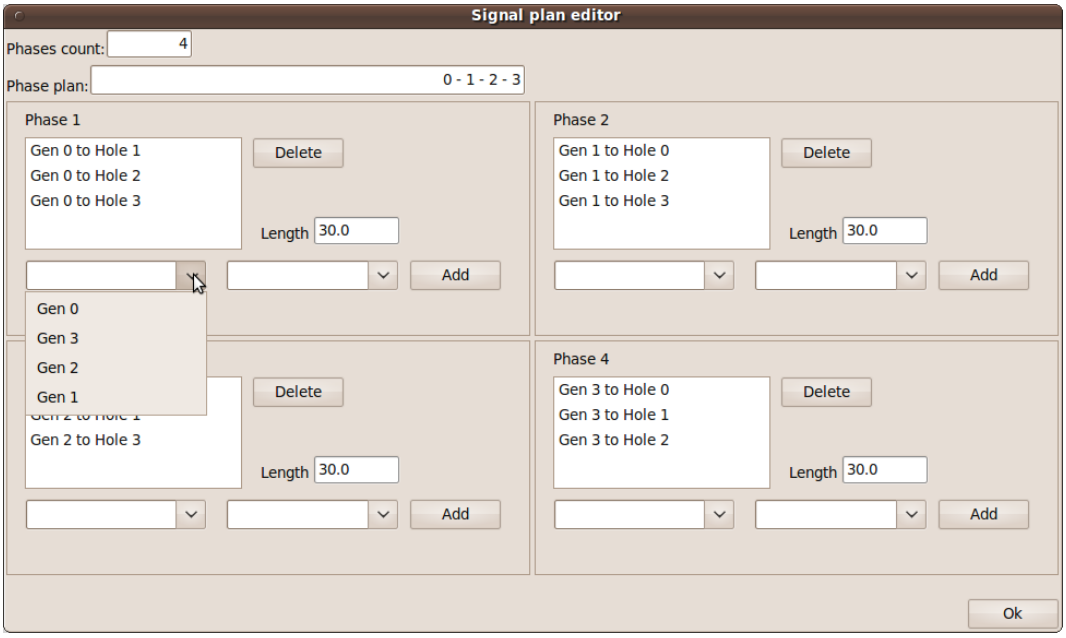


Obrázok 5.2: Príklad signálneho plánu

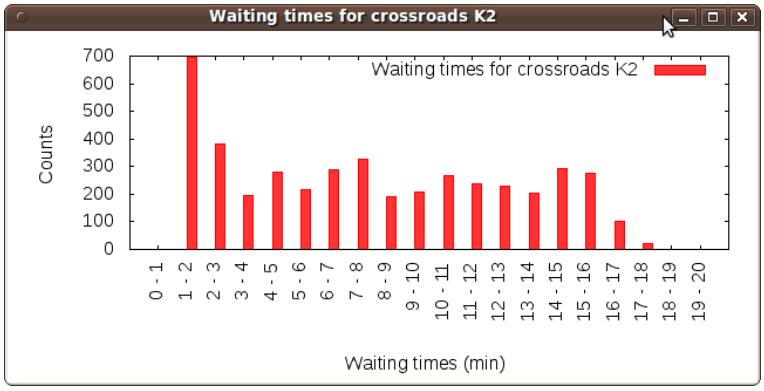
5.3 Grafické znázornenie simulácie

Simuláciu systému križovatiek je možno ovládať pomocou panela nástrojov, kedy po jej dokončení sa objavia simulačné okná, v ktorých sú zobrazené:

- štatistické hodnoty vstupných i odbočovacích frónt križovatiek a ciest.
- celková štatistika systému - histogram doby, ktorú vozidlá strávili v systéme a iné.
- grafické zobrazenie histogramu čakacích dôb pre každú križovatku (obrázok 5.4).
- textová detekcia preťažených smerov.
- grafická detekcia preťažených smerov zobrazená priamo v grafickom modeli návrhovej plochy (obrázok 5.5).



Obrázok 5.3: GUI - pohľad na okno zadávania signálneho plánu

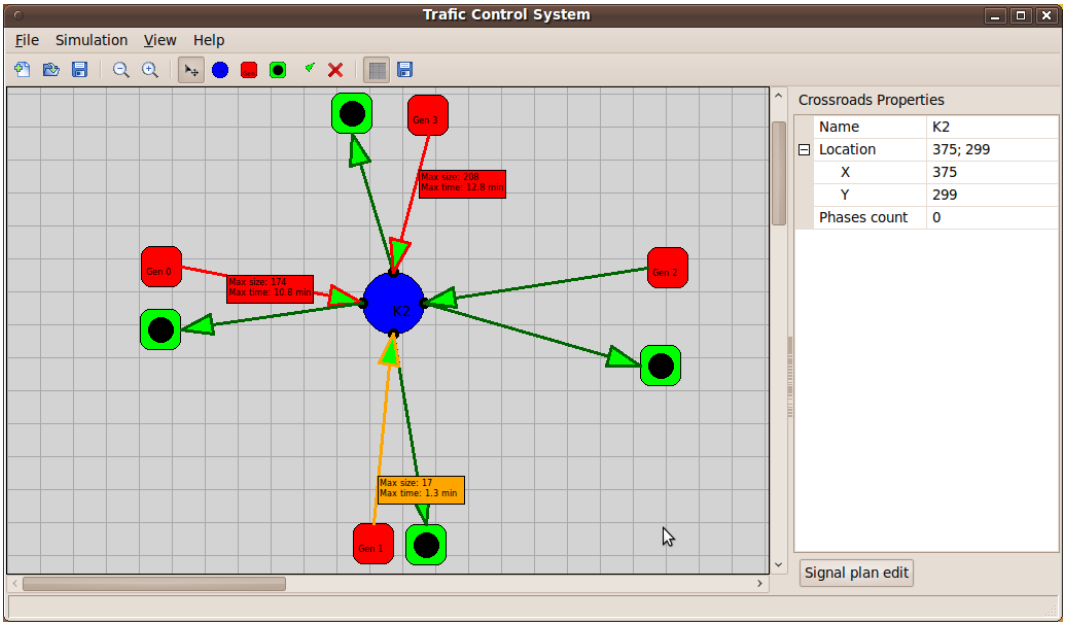


Obrázok 5.4: GUI - grafické zobrazenie histogramu

- simulačný log - podrobný výpis všetkých výsledných štatistík systému.

Vďaka tomuto spôsobu zobrazovania dát je možné vidieť, ako efektívne resp. neefektívne algoritmus pre riadenie križovatky pracuje. Vo väčšine prípadov si vystačíme s grafickými výsledkami simulácie, ktoré nám na prvý pohľad ukážu preťažené smery a ich špičky. Pre podrobnejšie skúmanie sú určené histogramy čakacích dôb križovatiek resp. simulačný log. Grafické zobrazenie histogramu vo forme grafu je vytvárané pomocou programu *gnuplot* [20] tak, že aplikácia vytvorí potrebné súbory pre program gnuplot. Následne sa zavolá tento program s potrebnými parametrami, výsledkom čoho je graf v obrazovom formáte

png[3], ktorý je potom načítaný späť do okna aplikácie.



Obrázok 5.5: GUI - celkový pohľad po simulácii

5.4 Uloženie projektu do XML

Uloženie projektu do súboru tak, aby sa po jeho opätovnom načítaní dal obnoviť stav celého systému so zachovaním integrity, nie je triviálny problém. Je nutné navrhnúť štruktúru súboru a systém, akým sa budú uložené dáta obnovovať do pôvodného stavu. Po analýze prepojení objektov a ich atribútov bol vybraný formát súboru *XML*[21], ktorý je čitateľný ľudským okom, a zároveň je prenositeľný na rôzne platformy.

Skratka *XML* znamená *eXtensible Markup Language*, čo v preklade znamená rozšíriteľný značkovací jazyk, ktorý dovoľuje vytvárať nové značky pre popis nejakého dokumentu. Je to veľmi štruktúrovaný a pohodlný formát. Oproti binárnym súborom, ktoré sú viazané platformou, *XML* formát má značnú výhodu. Stačí jeden syntaktický analyzátor pre všetko.

Pre zjednodušenie práce s formátom *XML* bola vybraná knižnica pre jazyk *C++ TinyXML* [22], ktorá je jednoduchá a ľahko sa integruje do programu. Knižnicu ďalej popisovať nebudeme.

[3]Portable Network Graphics - grafický formát pre ukladanie obrázkov.

5.4.1 Popis nových XML značiek

Objekty plochy majú pridelené vlastné XML značky, pretože ich vlastnosti resp. atribúty objektov sa navzájom líšia. Každý objekt obsahuje niekoľko dcérskych značiek definujúcich štruktúrované atribúty alebo zoznamy. Podrobnú definíciu značiek zobrazuje tabuľka 5.1.

Značka	Atribúty	Popis značky	Dcérske značky
⟨Project⟩	name	hlavná značka projektu	⟨DrawingObjects⟩ ⟨ArrowObjects⟩
⟨DrawingObjects⟩	-	zoznam objektov plochy	⟨Generator⟩ ⟨Cross⟩ ⟨BlackHole⟩
⟨ArrowObjects⟩	-	zoznam objektov relácie	⟨Arrow⟩
⟨Generator⟩	name	objekt generátora	⟨Position/⟩ ⟨Generation/⟩
⟨Cross⟩	name	objekt križovatky	⟨Position/⟩ ⟨Signalplan⟩
⟨BlackHole⟩	name	objekt čiernej diery	⟨Position/⟩
⟨Arrow⟩	From, To, KvadrantFrom, KvadrantTO	objekt relácie resp. cesty	⟨Distance/⟩ ⟨DirProbability/⟩ ⟨LaneX/⟩
⟨Position/⟩	X, Y	ukladá štruktúru pozície	-
⟨Generation/⟩	Cars, Buses, Trucks	generovanie vozidiel	-
⟨Signalplan⟩	count	signálny plán križovatky	⟨Phases⟩
⟨Phases⟩	-	zoznam fáz signálneho plánu	⟨Phase⟩
⟨Phase⟩	length	fáza signálneho plánu	⟨AllowedDirs⟩
⟨AllowedDirs⟩	-	zoznam povolených smerov	⟨FromLane/⟩
⟨FromLane/⟩	from, lane	štruktúra povoleného smeru	-
⟨Distance/⟩	meters	vzdialenosť križovatiek	-
⟨DirProbability/⟩	Left, Direct, Right	pravdepodobnosť odbočenia	-
⟨Lane1/⟩ ⟨Lane2/⟩ ⟨Lane3/⟩	length	vstupný prúd križovatky	-

Tabuľka 5.1: Popis XML značiek.

Kapitola 6

Genetické algoritmy

Táto kapitola je venovaná popisu genetických algoritmov a ich implementácii v jazyku *C++*. Neskôr je podrobne definovaný problém riadenia križovatky resp. systému križovatiek pomocou tohto matematického aparátu.

6.1 Stručný popis genetických algoritmov

Genetické algoritmy[9] sú najrozšírenejším typom evolučných algoritmov, pričom produkujú veľmi dobré výsledky pri riešení zložitých optimalizačných úloh. Jedná sa o moderný matematický prístup založený na modeloch evolúcie v prírode. Pri návrhu genetického algoritmu je potrebné riešiť tieto etapy:

- Reprezentácia problému - dôležitá časť realizácie genetických algoritmov. Príliš jednoduchá informácia vedie na zložitejšiu fitness fukнciu a naopak. Chromozóm kódujeme najčastejšie binárnou, permutačnou alebo stromovou reprezentáciou. Každý typ reprezentácie vyžaduje špecifické genetické operátory.
- Počiatočná populácia - väčšinou náhodné generovanie populácie. Typická veľkosť je 30 až 100 jedincov.
- Evaluácia indivíduí - ohodnocuje jednotlivých jedincov populácie, jedná sa o časovo najnáročnejšiu etapu. Ohodnotenie vykonáva funkcia fitness, ktorá je prevrátenou hodnotou účelovej funkcie. Pod pojmom účelová funkcia si predstavujeme stavový priestor, v ktorom hľadáme minimum. Inak povedané, účelová funkcia $u(i)$ udáva hodnotu funkcie i-teho genómu.
- Operátory selekcie - vytvára novú populáciu výberom s možným opakovaním zo starej populácie. Najbežnejší je ruletový resp. náhodný výber, kde pravdepodobnosť výberu každého jednotlivca je úmerná jeho fitness. Ďalším častým spôsobom je turnajový výber založený na výbere najlepšieho jedinca z vybraných jedincov populácie. Celý

proces sa opakuje N-krát, kde N je počet jedincov v novej populácii. Existujú ešte aj iné algoritmy, ako napríklad pravdepodobnostný výber alebo usporiadaný výber.

- Operátory rekombinácie resp. kríženia - tvoria základ genetických algoritmov. Existuje ich celá rada možností. Základom je náhodný výber jednotlivcov, u ktorých dôjde k výmene génovej informácie tak, že od bodu kríženia dochádza k výmene génov. Táto operácia sa vykonáva s pravdepodobnosťou 70%, čo má za dôsledok, že časť jedincov je reprodukovaná bez výmeny génov. Ďalšími spôsobmi sú jedno a viac bodové kríženie, uniformné kríženie alebo čiastočne mapované kríženie (*Partially mapped crossover*)[23].

- Operátory mutácie - jedná sa o operátor s malým výskytom okolo $0,1\%$ až 1% a je zdrojom nových informácií. Vplyv mutácie na druhej strane môže mať aj katastrofálne dôsledky. Medzi základné operátory mutácie patrí napríklad operátor inverzie, ktorý vymieňa poradie dvoch náhodne vybraných prvkov.

- Obnova populácie - môže byť čiastočná, kedy sa 20% až 50% rodičovskej populácie nahradí, alebo generatívna, ktorá obnoví celú populáciu rodičov.

- Veľkosť populácie - nemusí byť vždy konštantná a jej veľkosť ovplyvňuje rýchlosť konvergencie.

- Ukončenie algoritmu - podľa teorému konvergencie genetických algoritmov[1] je dosiahnutie globálneho optima zaručené v čase $t \to \infty$ za predpokladu, že bola použitá mutácia a elitizmus. Preto je vhodné zvoliť iný spôsob ukončenia evolúcie, ako napríklad nevyhovujúci rast $fitness$ u jedincoch populácie, malá diverzita populácie, prípadne ukončenie na základe počtu generácií.

Keď to celé zhrnieme, tak na začiatku je inicializovaná, obvykle náhodne, prvá populácia, ktorá obsahuje určitý počet kandidátnych riešení. Jeden cyklus algoritmu nazývame generáciou. Riešenia sú v každom cykle algoritmu vyhodnotené $fitness$ funkciou, ktorá určí ich vyspelosť. Potom nastáva fáza reprodukcie, kedy sú najlepšie riešenia vyberané z väčšou pravdepodobnosťou ako riešenia horšie, a postupne sú na ne aplikované genetické operátory kríženia a mutácie. Najlepší jedinci sa vyberajú do ďalšej generácie a celý cyklus sa opakuje, pokým sa nenájde riešenie alebo neprídeme k ukončovaciemu kritériu.

6.2 GAlib

GAlib alebo *Genetic Algorithms libary* [25] je knižnica určená pre prácu s genetickými algoritmami napísaná pre jazyk *C++*. Bola vyvinutá na americkej univerzite MIT[2]. Knižnica

[1] teorém je podrobne popísaný v literatúre [9]
[2] Massachusetts Institute of Technology

sa dá použiť na platformách založených na systémoch UNIX (Linux, MacOSX, FreeBSD a podobne) a tiež na platformách Windows s použitím mingw prekladaču. Tento nástroj ďalej zahŕňa podporu pre paralelnú implementáciu. Prakticky nemá žiadne obmedzenia, čo sa týka reprezentácie dát, či operátorov.

6.2.1 Výber Genetického algoritmu

GAlib používa štyri základné typy algoritmov líšiace sa genetickými operátormi, výberom populácie, dĺžkou trvania resp. zastavovacím kritériom:

- *GASimpleGA* - ako už napovedá názov, ide o jednoduchý algoritmus, ktorý v každej generácii používa úplne novú populáciu, pričom je možné zvoliť elitizmus, kedy sa najlepší jedinec z poslednej generácie prenáša do nasledujúcej.
- *GASteadyStateGA* - jedná sa o algoritmus, ktorý používa prekrývajúce sa populácie. Je možné určiť, aká časť populácie bude nahradená.
- *GAIncrementalGA* - inkrementálny algoritmus, v ktorom je možné definovať spôsob, ako sa budú včleňovať nové generácie do súčasnej populácie. Potomkovia môžu nahradzovať svojich rodičov, dokonca aj iné jedince.
- *GADemeGA* - u tohto typu je vyvíjaných niekoľko paralelných populácií zároveň, pričom sa používa algoritmus *SteadyState*. V každej generácii je niekoľko jedincov presunutých medzi generáciami.

Knižnica tiež umožňuje vytvorenie vlastného typu genetického algoritmu alebo je možné upraviť existujúci.

6.2.2 Reprezentácia

Dátová štruktúra, ktorá aproximuje daný problém, nazývame reprezentáciou. Definovanie takejto funkcie je akýmsi druhom umenia, pretože sa nedá presne definovať, ako má vyzerať štruktúra pre nejaký konkrétny problém. Nároky na štruktúru sú minimálna veľkosť a schopnosť reprezentovať všetky pre nás zaujímavé problémy. V terminológii genetických algoritmov tomu hovoríme genóm. Pod pojmom gén si predstavujeme jedinú časť dátovej štruktúry resp. genómu. Štruktúru navrhujeme tak, aby nemohla reprezentovať nezrealizovateľné riešenie. Počet možných reprezentácií je nekonečný. *GAlib* ponúka celú radu preddefinovaných štruktúr:

- *GA1DBinaryStringGenome* - genóm typu string obsahujúci len gény 0 alebo 1
- *GA1DArrayGenome*$\langle T \rangle$ - jednorozmerný genóm.
- *GAListGenome*$\langle T \rangle$ - reprezentuje genóm s variabilnou sekvenciou (tradičný zoznam).

- *GARealGenome* - jedná sa o genóm pozostávajúci s reálnych čísiel.
- *GAStringGenome* - genóm reprezentuje skupina znakov, používa sa napríklad pri radení.
- *GATreeGenome*$\langle T \rangle$ - reprezentuje genóm stromovej štruktúry, strom môže byť až n-árny.

Pre štruktúry, ktoré v názve majú $1D$, existujú aj variácie pre dvoj- a troj-rozmerný genóm. Písmeno T pri definícii triedy označuje použitie šablón jazyka C++, ktoré umožňujú genericky programovať triedy i funkcie[17].

6.2.3 Definícia genetických operátorov

GAlib má v sebe preddefinované tri základné genetické operátory inicializáciu, kríženie a mutáciu pre každý typ genómu. Preddefinovanie týchto operátorov je zásadný krok pred samotným spustením výpočtu, pretože operátory sú ostro viazané na reprezentáciu a riešený problém. Inicializáciu implementujeme zvyčajne náhodne, ale treba mať na mysli jej zmysluplnosť. Operátor kríženia vyberáme zásadne podľa typu problému a jeho kódovania a nakoniec operátor mutácie môžeme implementovať napríklad pomocou náhodnej funkcie *GAFlipCoin* (predstavuje hod mincou) tak, že pri každom géne spočítame výsledok tejto funkcie, a v prípade kladného výsledku urobíme mutáciu.

Objektívna funkcia nám poskytuje informáciu o tom, ako dobre vyhovuje daný jedinec nášmu účelu. Následne je potom táto informácia transformovaná tak, aby bolo možné určiť vhodnosť jedinca k reprodukcii. Dobre či zle napísaná objektívna funkcia má na svedomí výsledok celého genetického algoritmu[24].

6.2.4 Riadenie evolúcie

Evolúciu môžeme riadiť dvoma spôsobmi. Prvým je zavolanie metódy *evolve*, ktorá vykoná celú evolúciu za nás. Jej správanie je definované typom genetického algoritmu. Druhým spôsobom je použitie nekonečného cyklu a zvolenie si vlastného ukončovacieho kritéria. Telo cyklu potom tvorí volanie metódy *step*, čiže prevedenie jedného kroku evolúcie. Tento spôsob je dobre použiteľný pri koevolúcii (viacero navzájom komunikujúcich genetických algoritmov prebiehajúcich súčasne) [24].

Štatistický objekt *GAStatistics* poskytuje informácie o aktuálnom stave objektov genetického algoritmu, ktoré sa dajú použiť pri ladení algoritmu, či ako súčasť ukončovacieho kritéria. Druhým zaujímavým objektom je *GAPopulation*, ktorý je kontajnerom na genómy, a obsahuje štatistiky populácie ako napríklad priemer, maximum a minimum hodnôt objektívnej funkcie. Zapúzdruje tiež úpravu mierky *fitness* fukcie vzhľadom k objektívnej funkcii (*Scaling scheme*).

6.3 Optimalizácia signálneho plánu

Pri návrhu optimalizácie signálneho plánu budeme vychádzať z etáp návrhu genetického algoritmu a popisu knižnice *GAlib*. Skôr ako sa pustíme do samotného návrhu, definujme si požiadavky na genetický algoritmus. Ten by mal po ukončení evolúcie vrátiť jedinca, z ktorého sa po rozkódovaní zostaví nový signálny plán križovatky.

6.3.1 Reprezentácia

Začnime popisom reprezentácie problému. V kapitole 4.2 sme sa podrobne venovali popisu signálneho plánu navrhnutého dopravným inžinierom. Budeme teda vychádzať z tohto návrhu, v ktorom je pre nás rozhodujúca signálna fáza pozostávajúca z jednotlivých povolených smerov a dĺžka tejto fázy.

Gén je definovaný číslom fázy. Číslovanie fáz začína nulou a končí hodnotou o jedna menšou ako celkový počet fáz n. Gén je teda množina:

$$Gene = \{0, 1, \ldots, n\} \tag{6.1}$$

Genóm definujeme ako usporiadanú n-ticu génov o určitej veľkosti n:

$$Genome = \{g; g \text{ je usporiadaná n-tica}(g_1, g_2, \ldots, g_n) \wedge g_i \in Gene\} \tag{6.2}$$

Tabuľka 6.1 ukazuje príklad genómu jednej štvorfázovej križovatky s celkovým počtom 12 génov.

i	1	2	3	4	5	6	7	8	9	10	11	12
Genome (g_i)	0	3	3	2	1	0	3	3	2	2	1	0

Tabuľka 6.1: Príklad genómu o veľkosti 12

V statickom riadení sa jednotlivé signálne fázy striedajú dookola, pričom dopravný inžinier určí len optimálne poradie fáz. Počet možností pri štvorfázovom riadení je šestnásť, čo by znamenalo pre evolučný algoritmus veľmi malú rozmanitosť. Preto zvolíme dĺžku genómu oveľa väčšiu, ako je počet fáz, pričom dĺžku volíme tak, aby súčet dĺžok všetkých fáz genómu bol približne rovný desiatim minútam. Tento čas poskytuje dostatočne veľkú rozmanitosť v plánovaní. Pri štvorfázovom riadení a priemernej dĺžke fázy dvadsať sekúnd je ideálna dĺžka genómu rovná 24.

V knižnici *GAlib* bude tento genóm reprezentovaný jednorozmerným typom genómu $GA1DArrayGenome\langle int\rangle$ o dĺžke n, ktorú je možné parametrizovať.

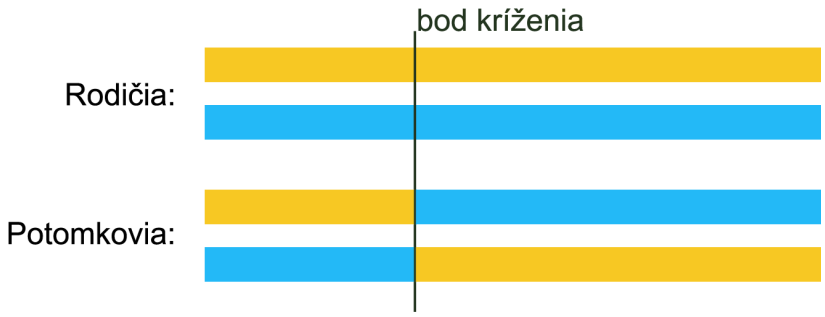
6.3.2 Definovanie genetických operátorov

Na základe definície reprezentácie problému resp. genómu je nutné definovať operátory používané genetickým algoritmom. Operátory musia striktne dodržiavať definíciu genómu.

Konkrétne sa jedná o operátory:

- Inicializácia - jednotlivé gény obsiahnuté v génóme sú inicializované náhodne pomocou generátoru *GARandomInt* s parametrami intervalu $\langle 0, n)$, kde n je počet fáz križovatky.
- Mutácia - Na základe generátora náhodných čísiel *GAFlipCoin*, hod mincou s určitou pravdepodobnosťou, vyberieme prvok, ktorý náhodne zmeníme v rozsahu od nula do n resp. do hodnoty počtu fáz križovatky. Tradičný operátor mutácie používaný pri plánovaní, *SwapMutator*, vymieňa náhodne dva vybrané gény. Tento spôsob mutácie je nedostačujúci, pretože výmena prvkov nám nedáva potrebnú rozmanitosť, čiže sa nedokážu žiadnym spôsobom zmeniť počty jednotlivých typov génov.

Nový operátor kríženia nie je nutné definovať, pretože daný genetický algoritmus knižnice definuje kríženie genómov vhodne použiteľným spôsobom. Štandardne používa jednobodové kríženie, pri ktorom algoritmus najlepšie konverguje. Jednobodové kríženie náhodne vyberá bod kríženia dvoch jedincov, z ktorých opätovne vznikajú dvaja noví jedinci. Operáciu dobre ilustruje obrázok 6.1. Do úvahy by pripadali aj ďalšie druhy kríženia, napríklad dvojbodové kríženie prípadne uniformné kríženie.



Obrázok 6.1: Jednobodové kríženie

6.3.3 Fitness funkcia

Najdôležitejším krokom pri návrhu evolučných algoritmov je práve *fitness* funkcia, ktorej cieľom je stanovenie vyspelosti jednotlivých potomkov. Požiadavky na túto funkciu sú jednoznačné. Musí byť rýchla a schopná správne ohodnotiť všetky možné kombinácie génov jednotlivých potomkov.

Hlavným problémom v našom prípade je rýchlosť, pretože každý genóm je treba odsimulovať a získať výsledky, čo je časovo náročná operácia. Vhodným nastavením dĺžky simulácie by sa dala znížiť časová náročnosť tejto operácie, na druhej strane si musíme uvedomiť, že simulácia systému je založená na pravdepodobnosti a štatistike, ktorá má tým

väčšiu vypovedajúcu hodnotu, čím väčší počet udalostí spracovala. Je potrebné nájsť optimum medzi rýchlosťou simulácie a časovou náročnosťou. Keď vezmeme do úvahy, že dĺžka trvania signálneho plánu po optimalizácii je približne 8 minút, tak pre otestovanie daného genómu je postačujúci trojnásobok tejto doby resp. 30 minút. Dĺžku trvania signálneho plánu sme získali ako súčin priemernej dĺžky fázy v sekundách a dĺžky genómu.

Po odsimulovaní daného genómu nám jej výsledky poskytnú veľké množstvo informácií a my musíme vybrať len jednu, podľa ktorej vieme objektívne ohodnotiť vyspelosť jedinca. Vhodnými sa pritom javia maximálne či priemerné údaje o dĺžkach alebo čakacích dobách frónt všetkých vstupných smerov. Optimalizovanie podľa čakacích dôb je nemožné, pretože ich výpočet je závislý na opúšťaní vozidiel z fronty, čiže čakaciu dobu je možné vypočítať, keď vozidlo opustilo frontu. V prípade optimalizovania podľa tohto časového údaju by došlo síce k minimalizovaniu čakacej doby, ale rapídne by vzrástli dĺžky frónt. Príčinou tohto javu je diskrétnosť simulácie resp. nemožnosť vypočítať čas strávený vo fronte bez toho, aby z nej vozidlo odišlo. Môžeme teda povedať, že údaj o čakacej dobe nemá vypovedajúcu hodnotu pre genetický algoritmus. Budeme teda optimalizovať na základe dĺžky frónt, pričom vyberáme maximálnu alebo priemernú hodnotu. Výber maximálnej hodnoty, by podľa mojej mienky, nebol zrovna šťastnou voľbou, pretože maximálna hodnota odzrkadľuje dopravnú špičku, čiže stav veľmi sa líšiaci od normálnej prevádzky. Toto tvrdenie potvrdzuje aj experiment[3], ktorý porovnáva výsledky simulácie pred aplikovaním optimalizácie a po jej aplikovaní dvoma rôznymi metrikami. Experiment popisuje tabuľka 6.2, v ktorej optimalizácia pomocou priemernej hodnoty dĺžky frónt má značne lepšie výsledky.

		Dĺžka fronty		Čakacia doba (min)	
Experiment	Smer	Priemer	Max	Priemer	Max
Bez GA	0	172	356	10	22
	3	119	305	7	17
GA a maximálna dĺžka	0	6	41	0.3	2.0
	3	7	43	0.4	2.0
GA a priemerná dĺžka	0	5	33	0.3	1.6
	3	1	5	0.1	0.5

Tabuľka 6.2: Porovnanie metrík optimalizácie

Metód, akými stanovýme účelovú funkciu je niekoľko. Treba však vybrať takú metódu, ktorá vhodným spôsobom spravodlivo ohodnotí dané smery. Pri optimalizácii plánu musíme počítať s faktom, že riadenie bude uprednostňovať preplnenejšie smery na úkor tých vedľajších nepreplnených. Domnievam sa, že vhodnou metrikou bude súčet všetkých dĺžok vstupných frónt. Účelová funkcia hľadá minimum, čiže stav plánovania, pri ktorom

[3] popis simulovaného modelu je uvedení v kapitole 7

je daný súčet minimálny. Vylepšením tejto metriky je namiesto súčtu dĺžok robiť súčet druhých mocnín dĺžok, čo v tomto prípade viac znevýhodňuje, penalizuje, preplnenejšie smery. Matematicky by sme mohli túto funkciu zapísať ako:

$$u = \sum_{i=1}^{n} l_i^2 \tag{6.3}$$

kde l_i je priemerná dĺžka fronty i-tého vstupného smeru a n je počet vstupných smerov križovatky. *Fitness* funkcia hľadá najlepšieho jedinca čiže maximum, a preto je rovná prevrátenej hodnote účelovej funkcie:

$$f = \frac{1}{u} \tag{6.4}$$

Implementácia metódy *fitness* pomocou knižnice *GAlib* je teda založená na týchto krokoch:

1. parametrom metódy je ohodnocovaný genóm.
2. vytvorí a inicializuje sa model systému križovatiek pre dobu trvania 30 minút.
3. do riadenia križovatky sa nahrá rozkódovaný genóm resp. plán riadenia.
4. spustí sa simulácia.
5. na základe priemerných dĺžok vstupných frónt získaných zo simulácie sa vypočíta hodnota účelovej funkcie.
6. zrušia sa všetky objekty simulácie.
7. vrátená je hodnota *fitness* resp. prevrátená hodnota účelovej funkcie.

6.3.4 Riadenie evolúcie

Spusteniu a riadeniu evolúcie predchádza inicializácia parametrov a operátorov vybraného evolučného algoritmu. Vyberali sme spomedzi štyroch možných algoritmov, pričom najlepšie konvergujúcim je *GAIncrementalGA* resp. inkrementálny algoritmus, v ktorom sú nové generácie včleňované do súčasnej populácie. Následne algoritmu priradíme reprezentáciu a novo definované operátory. Hodnoty inicializácie parametrov algoritmu boli zvolené na základe odporúčaní literatúry [9] a experimentálne overené. Jedná sa o hodnoty:

- veľkosť populácie - 10
- pravdepodobnosť kríženia - 0,7
- pravdepodobosť mutácie - 0,1

Ukončovacie kritérium je závislé na posledných 150-tich generáciách, z ktorých sa počíta aktuálna konvergencia jedincov ohodnotených *fitness* funkciou vyjadrenou v percentách. Keď algoritmus dosiahne konvergenciu 0,99, evolúcia je ukončená. Môžeme teda povedať, že evolúcia sa ukončí, keď najlepšie ohodnotenie *fitness* funkciou posledných 150-tich generácií je približne rovnaké. Následne je najlepší jedinec rozkódovaný a prenesený do riadenia križovatky.

Optimalizácia vzhľadom k dĺžkam jednotlivých fáz je posledným krokom po optimalizácii genetickým algoritmom. Táto optimalizácia spočíva v zoskupovaní rovnakých po sebe idúcich fáz. Pri združovaní týchto fáz dochádza k zmene dĺžky danej fázy tak, že výsledná dĺžka fázy je vynásobená počtom práve zoskupovaných fáz. Za po sebe idúce fázy sa považujú aj prvá a posledná, pretože plán je vykonávaný cyklicky. Názornú ukážku tejto optimalizácie ukazuje tabuľka 6.3, pričom predpokladáme, že dĺžka každej fázy je 30 sekúnd a je znázorňovaná dolným indexom pri čísle fázy.

Plán po GA	0_{30}	3_{30}	3_{30}	2_{30}	1_{30}	0_{30}	3_{30}	3_{30}	2_{30}	2_{30}	1_{30}	0_{30}
Plán po zoskupení fáz	0_{60}	3_{60}		2_{30}	1_{30}	0_{30}	3_{60}		2_{60}		1_{30}	

Tabuľka 6.3: Príklad optimalizácie plánu pomocou zoskupovania fází

6.4 Optimalizácia signálnych plánov systému križovatiek

Optimalizácia celého systému križovatiek je oveľa zložitejším i časovo náročnejším problémom. Budeme však vychádzať z návrhu optimalizácie riadenia jednej križovatky, ktorý rozšírime tak, aby bolo možné optimalizovať celý systém so zachovaním konvergencie. Jednotlivými časťami návrhu genetických algoritmov sa budeme zaoberať postupne, podobne ako v predchádzajúcej časti.

6.4.1 Reprezentácia

Reprezentácia problému musí obsahovať vhodne zakódované riadenia všetkých križovatiek v systéme. Vychádzame zo zakódovania jednej križovatky, pri ktorom gén obsahoval jednu z jednotlivo očíslovaných signálnych fáz, a genóm bol množinou týchto génov. Túto myšlienku rozšírime na systém križovatiek tak, že dodefinujeme nový gén, ktorý je závislý na križovatke, čiže môže nadobúdať len hodnoty kompatibilné s križovatkou, a dodefinujeme nový typ genómu. Tento genóm bude postupne za sebou pozostávať z genómov jednotlivých križovatiek.

Matematicky môžeme definovať množiny génov v závislostiach na počte fáz n_j danej križovatky j:

$$Gene_j = \{0, 1, \ldots, n_j\} \tag{6.5}$$

Genóm jednej križovatky j o veľkosti n definujeme podobne ako v predchádzajúcej časti:

$$Genome_j = \{g; g \text{ je usporiadaná n-tica } (g_1, g_2, \ldots, g_n) \wedge g_i \in Gene_j\} \tag{6.6}$$

A genóm systému potom definujeme ako množinu genómov jednotlivých križovatiek. Mohutnosť množiny je počet križovatiek n:

$$Genome = \{h; h \in \{h_1, h_2, \ldots, h_n\} \wedge h_j \in Genome_j\} \tag{6.7}$$

Reprezentáciu názorne ilustruje tabuľka 6.4, v ktorej je uvedený príklad genómu dvoch križovatiek, pričom prvá z nich je riadená dvojfázovým plánom a druhá je riadená klasickým štvorfázovým signálnym plánom.

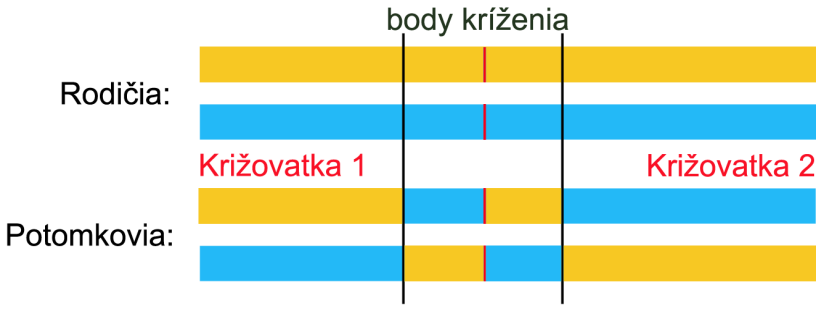
	Križovatka 1 ($j = 0$)								Križovatka 2 ($j = 1$)							
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$Genome(g_{ij})$	0	1	1	0	1	0	1	0	0	3	3	2	1	0	3	3

Tabuľka 6.4: Príklad genómu dvoch križovatiek o veľkosti 16 génov.

6.4.2 Definovanie genetických operátorov

Definícia reprezentácie problému sa oproti reprezentácie jednej križovatky zmenila, preto je nutné definovať resp. upraviť operátory používané genetickým algoritmom tak, aby sa zachovala logika zakódovania. Konkrétne sa jedná o operátory:

- Inicializácia - jednotlivé gény obsiahnuté v genóme sú inicializované náhodne po skupinách o veľkosti genómu križovatky pomocou generátora *GARandomInt* s parametrami intervalu $\langle 0, n_j)$, kde n_j je počet fáz križovatky j.
- Mutácia - operátor mutácie sa taktiež musí pridržiavať logiky, aby nevniesol do genómu nezmyselnú mutáciu. Základom je opäť generátor náhodných čísiel *GAFlipCoin* vyberajúci prvok, ktorý náhodne zmeníme v rozsahu od nula do parametru n_j resp. počtu fáz j-tej križovatky.
- Kríženie - musí zachovávať logiku zakódovania a to tak, aby po procese kríženia nevznikol genóm, ktorý je v rozpore s definíciou reprezentácie. Ako základ použijeme jednobodové kríženie, ktoré aplikujeme postupne na všetky podgenómy jednotlivých križovatiek. Vzniká tak nová forma viacbodového kríženia. Situáciu pred a po krížení ilustruje obrázok 6.2.



Obrázok 6.2: Kríženie systému križovatiek

6.4.3 Fitness funckia

Stanovenie účelovej funkcie pre celý systém bude iba rozšírením účelovej funkcie jednej križovatky. Treba však myslieť na to, že zmena plánu jednej križovatky ovplyvní druhú a naopak. Preto konvergovať musí celý systém zároveň. Využijeme opäť metriku priemernej dĺžky vstupných frónt, pričom budeme robiť súčet druhých mocnín vstupných smerov všetkých križovatiek systému, z ktorých hľadáme minimum. Matematicky účelovú funkciu u vyjadríme ako:

$$u = \sum_{i=1}^{n} \sum_{j=1}^{n_i} l_{ij}^2 \tag{6.8}$$

v ktorej l_{ij} je priemerná dĺžka fronty i-tého vstupného smeru j-tej križovatky, n_i je počet vstupných smerov j-tej križovatky a n je počet križovatiek.

Fitness funkcia je, podobne ako pri výpočte *fitness* funkcie jednej križovatky, prevrátenou hodnotou účelovej funkcie. Kroky implementácie sú totožné s *fitness* funkciou z predchádzajúcej časti, preto ich ďalej popisovať nebudeme.

Kapitola 7

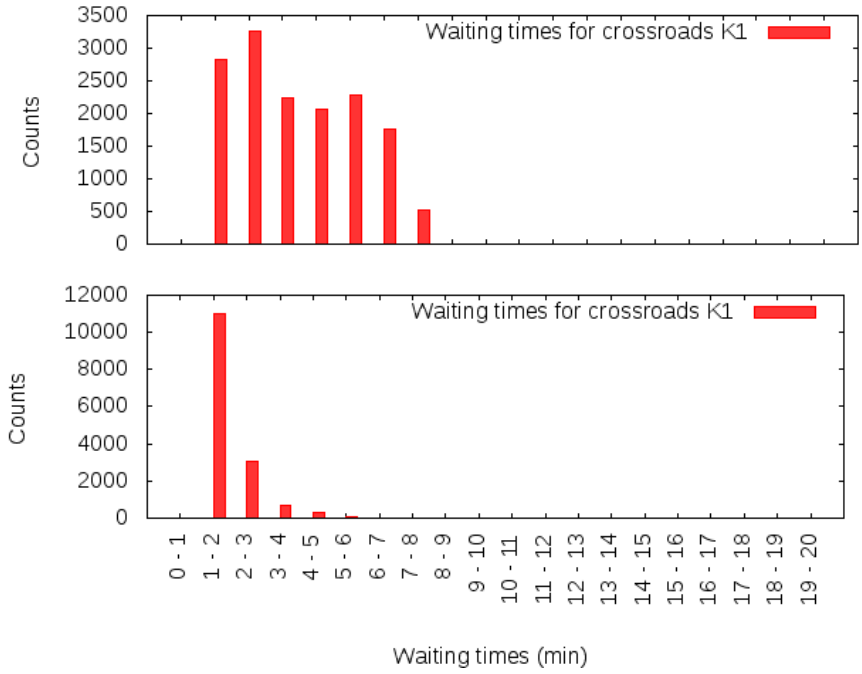
Výsledky a zhodnotenie práce

Táto kapitola sa venuje popisu a vyhodnocovaniu troch testov, ktorými boli odskúšané základné vlastnosti aplikácie. Výsledky sú porovnávané pomocou histogramov zobrazujúcich dobu strávenú v systéme, pred a po ukončení experimentov. Celá aplikácia je plne prenositeľná a bola testovaná na systémoch Linux a FreeDSD.

7.1 Test 1

Prvý testovací model tvorí jedna križovatka, na ktorej vstupoch sú štyri generátory. Generátory Gen0 a Gen2 simulujú hlavný smer, pričom generujú 2000 áut, 20 autobusov a 5 vozidiel nákladnej dopravy za hodinu. Vedľajší smer majú generátory Gen1 a Gen3 generujúce 500 áut, 5 autobusov a 2 vozidlá nákladnej dopravy za hodinu. Hlavný smer má zhodne na obidvoch stranách nastavené pravdepodobnosti odbočenia v poradí 20 %, 60 % a 20 % a dĺžky prúdov 20 m, 30 m a 20 m pre smer doľava, rovno a doprava. Riadenie je štvorfázové, pričom každá fáza povoľuje všetky prúdy z jedného smeru a dĺžka fáz je zhodne 30 sekúnd.

Pred optimalizáciou, ako môžeme vidieť na obrázku A.2, je hlavný smer pomerne silno upchatý. Svedčia o tom priemerné čakacie doby vo vstupných frontách, ktorých hodnoty sú 2 a 4 minúty pri priemernom počte 68 a 134 čakajúcich vozidiel. Po optimalizácii klesli priemerné čakacie doby hlavného smeru na 0,4 a 0,3 minúty, čo je približne 12-násobné zlepšenie. Vedľajšie smery zaznamenávajú mierne zvýšenie čakacích dôb. Optimalizácia urobila systém viac stabilnejším a znížila čakacie doby vo frontách vstupných smerov. Porovnanie histogramov čakacích dôb zobrazuje obrázok 7.1. Optimalizácia tohto príkladu trvala približne 2 minúty.

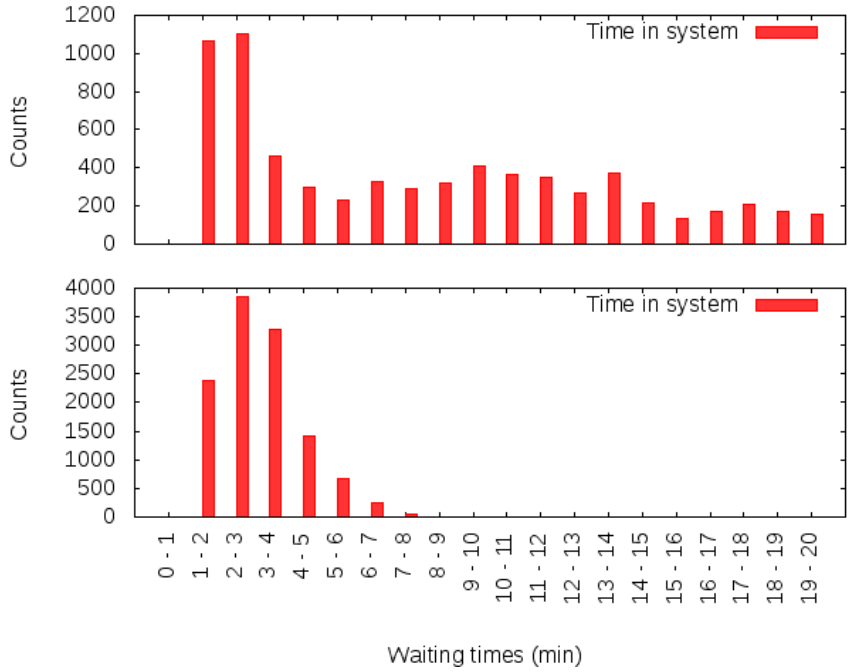


Obrázok 7.1: Test1: Histogram čakacích dôb pred a po optimalizácii

7.2 Test 2

Druhým testovacím modelom je model systému dvoch križovatiek vzdialených od seba 500 metrov. Hlavný smer tvoria generátory Gen10 a Gen21, ktoré generujú 1000 áut, 50 autobusov a 10 nákladných vozidiel za hodinu. Vedľajšie smery generujú 500 áut za hodinu pomocou generátorov Gen11, Gen12, Gen20 a Gen22. Pravdepodobnosti odbočenia z hlavného smeru sú 10 %, 80 % a 10 % a pre odbočenie z vedľajšieho smeru 40 %, 20 % a 40 %. Väčšie hodnoty pravdepodobnosti odbočenia sú v prospech hlavného smeru. Všetky odbočovacie prúdy majú dĺžku nastavenú zhodne na 20 metrov. Riadenie je opäť štvorfázové, pričom každá fáza povoľuje všetky prúdy z jedného smeru a dĺžka všetkých fáz je 30 sekúnd.

Stav pred optimalizáciou jasne ukazuje na preťaženie systému, čo je spôsobené zlou synchronizáciou križovatiek. V hlavnom smere sa priemerné čakanie vo frontách pohybuje medzi 6 až 16 minútami a vo vedľajších smeroch je až 18 minút. Histogramy čakacích dôb križovatiek naznačujú vysoké čakacie doby v oboch križovatkách. Po optimalizácii sú priemerné čakacie doby hlavného smeru približne v rozmedzí 0,5 až 1 minúty, pričom vedľajšie smery majú priemernú čakaciu dobu okolo 1 minúty. Zásadné vylepšenie ukazuje aj histogram doby strávenej v systéme 7.2. Celkovo optimalizácia trvala 5 minút a priniesla so sebou približne 30-násobné zníženie čakacích dôb.

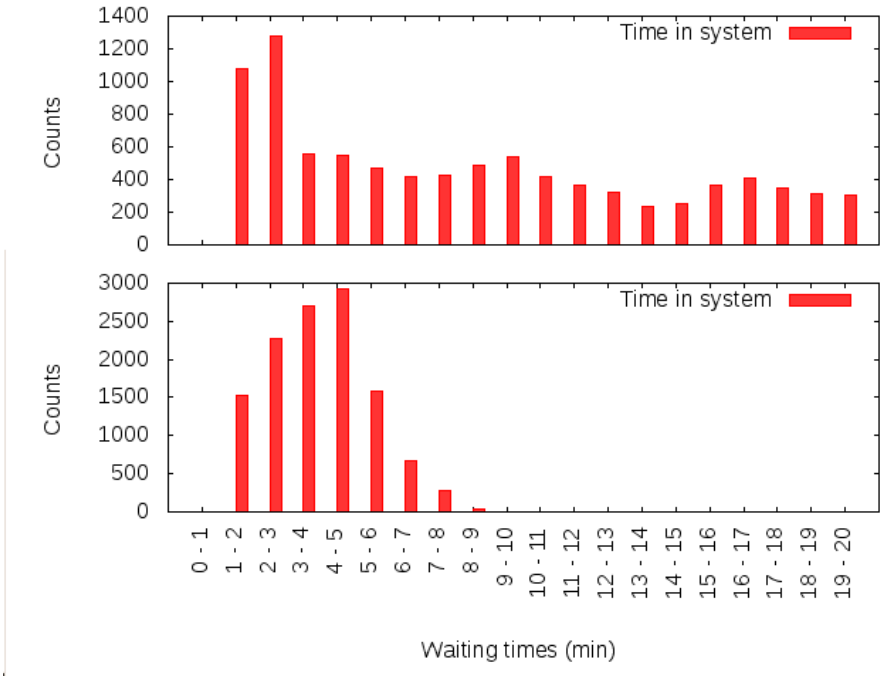


Obrázok 7.2: Test2: Histogram čakacích dôb pred a po optimalizácii

7.3 Test 3

Posledný testovací model modeluje systém štyroch križovatiek vzdialených od seba 500, 800 a 700 metrov. Jedná sa o test líniovej koordinácie hlavného smeru. Generátory Gen1 a Gen6 generujú hlavný dopravný ťah v počte 800 áut, 20 autobusov a 5 vozidiel nákladnej dopravy za hodinu. Všetky ostatné generátory sú priradené vedľajšiemu smeru a generujú 300 áut, 5 autobusov a 2 vozidlá nákladnej dopravy za hodinu. Všetky smery odbočenia sú nastavené tak, aby väčšina vozidiel mala tendenciu vstupovať do hlavného smeru. Dĺžky odbočovacích vstupných prúdov sú nastavené zhodne na 20 metrov. Riadenie je opäť štvorfázové, pričom každá fáza povoľuje všetky prúdy z jedného smeru a dĺžka všetkých fáz je 30 sekúnd. Model simulovaného systému nájdeme v prílohe A.5.

Stav pred optimalizáciou ukazuje, že systém nie je vôbec koordinovaný. Preťažený je hlavný smer a niekoľko vedľajších. Čakacie doby vo frontách hlavného smeru sa pohybujú od 3 do 7 minút, pričom veľké čakacie doby vykazujú vedľajšie smery, a to približne od 4 do 21 minút. Optimalizácia trvala 15 minút a priniesla výrazné zlepšenie. Hlavný tok zaznamenáva čakacie doby od 0,3 do 0,7 minúty a vedľajší tok od 0,2 do 1 minúty. Optimalizácia je teda približne 10-násobná. Obrázok 7.3 porovnáva histogramy času stráveného v systéme pred a po optimalizácii.



Obrázok 7.3: Test3: Histogram čakacích dôb pred a po optimalizácii

7.4 Zhodnotenie práce

Statické riadenie vo forme periodického sa opakovania signálneho plánu dopravy je v dnešnej dobe nepostačujúce. Súčasné metódy dynamického riadenia, ako napríklad celočervený režim alebo centralizované riadenie nemusia byť dostačujúce. Zavedenie synchronizovania križovatiek tzv. „zelenej vlny“ do systému križovatiek sa na prvý pohľad zdá byť výhodné, ale v skutočnosti sa predlžujú doby strávené v systéme, čo má za dôsledok zvyšovanie škodlivých emisií do ovzdušia. Preto táto práca poskytla alternatívne riešenie tohto problému.

Evolučný prístup dokáže zostaviť suboptimálny signálny plán pre aktuálnu dopravnú situáciu. Systém križovatiek tým získa potenciál maximálnej možnej priepustnosti vozidiel. Synchronizáciou križovatiek sa pri tomto prístupe nemusíme zaoberať, pretože optimalizácia vždy nájde suboptimálne riešenie pre celý systém.

Mnohé mestá majú zastarané dopravné riadenie. Zavedenie evolučného prístupu do dopravných systémov nemusí byť zložitou záležitosťou. Prvým krokom by bolo zavedenie senzorov, ktoré by vhodne monitorovali dopravnú situáciu všetkých smerov, a tým poskytli údaje pre genetický algoritmus. Údaje by sa v pravidelných intervaloch posielali do centrály, kde by boli podkladom pre simuláciu a optimalizáciu. Novo zostavené signálne plány križovatiek by sa po optimalizácii prenášali do radičov jednotlivých križovatiek. Tento návrh je však potrebné dopracovať do posledného detailu, aby optimalizácia nebola na úkor bezpečnosti cestnej premávky.

Myslím, že táto práca poskytla dobrú alternatívu k súčasnému riadeniu dopravy. Pozitívne výsledky potvrdzujú aj simulačné experimenty, kde 10 až 20-násobná optimalizácia a zníženie priemerných čakacích dôb pod jednu minútu ukazujú, že evolučný prístup má budúcnosť v riadení dopravy.

7.5 Metriky kódu implementovanej aplikácie

- Počet efektívnych riadkov: 3092
- Počet súborov: 29
- Veľkosť statických dát: 951 KB
- Veľkosť spustiteľného súboru: 3,8 MB (Linux, 64-bit architektúra)
- Knižnice: wxWidgets 2.8.11, TinyXML 2.6.1, GAlib 2.47

Kapitola 8

Rozšírenia do budúcnosti

Táto kapitola popisuje možné vylepšenia grafického užívateľského rozhrania aplikácie, spôsobu simulácie systému križovatiek a návrhu optimalizácie riadenia pomocou genetických algoritmov.

8.1 Grafické užívateľské rozhranie

V prípade grafického vzhľadu aplikácie by bolo možné urobiť nasledujúce vylepšenia:

- Lepšie graficky znázorňovať prvky plochy - objekt generátora, križovatky, relácie a pod.
- Pridať možnosť označiť viacero objektov zároveň. Podobne ako v grafických programoch sa pomocou obdĺžnika označia objekty. Do úvahy by pripadalo aj označovanie pomocou myši a klávesy Shift.
- Pridať Snap-in funkcionalitu, ktorá by mohla byť nápomocná pri vytváraní relácie.
- Graficky zobraziť križovatku, ktorá má nakonfigurované vstupné prúdy.
- Graficky zadávať signálny plán.
- Možnosť graficky zadávať vstupné prúdy.
- Nájsť graficky lepší spôsob ako zobraziť výsledky simulácie.

8.2 Simulácia systému križovatiek

U abstraktného modelu križovatky by boli prípadné rozšírenia zamerané na jeho komplexnosť. To znamená uvažovať:

- rôzne počty vstupných i odbočovacích prúdov

- rôzne vzdialenosti medzi hranicami križovatiek rôznych smerov a podobne
- pridanie iných druhov vozidiel
- pridanie električiek
- pridanie chodcov ako ďalšieho účastníka cestnej premávky
- uprednostňovanie vozidiel mestskej hromadnej dopravy
- komplexná simulácia systému, kde by sa v závislosti na čase menili hodnoty generátorov, a tým by sa odsimulovali dopravné nárazové vlny

8.3 Optimalizácia riadenia križovatiek

V prípade optimalizácií sa ponúka viacero možností jej vylepšenia. Prvou z nich je zvýšenie rýchlosti optimalizácie, pričom sú použiteľné dve alternatívy:

- paralelizácia na viacerých procesoroch. Knižnica GAlib ponúka takúto možnosť, pričom používa PVM[1] pre distribuovanú paralelnú implementáciu.
- Paralelizácia pomocou procesorových jednotiek grafickej karty. Je to relatívne nový postup používaný pri časovo náročných výpočtoch. Používa sa architektúra CUDA[2], ktorá je až 2600-krát rýchlejšia ako výpočet genetických algoritmov na jednoprocesorovom systéme [26].

Ďalšou možnosťou by bolo zamyslenie sa nad rozdelením systému križovatiek na podsystémy a nad ich jednotlivou optimalizáciou. Je možné, že takto oddelené podsystémy by mohli vykazovať lepšie výsledky. Do úvahy by tiež pripadalo určenie vyššej priority dôležitým dopravným tokom na úkor menej dôležitých tokov.

Možným vylepšením by mohlo byť pridanie možnosti zvolenia všetkých parametrov modelu úplne náhodne. Jedná sa o parametre nastavenia generátorov, pravdepodobnosti odbočenia na každej križovatke a vzdialenosti medzi križovatkami. Takto navrhnutý náhodný model systému križovatiek môže poskytnúť cenné informácie pri návrhu optimálneho algoritmu, ako aj analýzy pre dopravných inžinierov.

[1] Parallel Virtual Machine[27]
[2] Compute Unified Device Architecture[28]

Kapitola 9

Záver

V tejto práci bola podrobne analyzovaná cestná doprava riadená pomocou svetelnej signalizácie a jej optimalizácia pomocou evolučného prístupu. Genetické algoritmy ako súčasť evolučného prístupu sú vhodné pre riešenia problémov týkajúcich sa plánovania a táto práca ukázala, že môžu byť nápomocné aj pri optimalizácii plánu jednej križovatky alebo dokonca plánoch skupiny križovatiek tvoriacich dopravný systém. Aplikácia môže byť užitočná pre dopravných inžinierov, ktorý majú takto možnosť odsimulovať nimi navrhnutý systém.

Na základe analýzy súčasných spôsobov riadenia dopravy a teórie simulácie sa získali poznatky, ktoré boli nápomocné pri tvorbe abstraktného modelu systému križovatiek. Neskôr, pomocou knižnice Simlib/C++, bol implementovaný simulačný model, ktorý spĺňa všetky vlastnosti abstraktného modelu križovatiek. Grafické užívateľské rozhranie bolo navrhnuté tak, aby bolo možné jednoducho a efektívne zostavovať simulačný model pomocou menších komponentov grafickej plochy vhodne pospájaných pomocou objektu relácie. Takto zostavený model je možné odsimulovať a získať štatistické výsledky potrebné pre analýzu.

Celý systém bol otestovaný na vhodných testovacích príkladoch, ktoré ukázali v niektorých prípadoch až 30-násobné zníženie čakacích dôb vo frontách križovatiek. Doba optimalizácie sa pohybovala od 1,5 minúty do 15 minút v závislosti na počte vozidiel v systéme. Táto doba by mohla byť rapídne znížená použitím paralelného prístupu pri optimalizácii alebo pri simulácii, ktorá je počas optimalizácie vykonávaná veľmi často.

Práca mi poskytla priestor pre praktické overenie znalostí získaných počas štúdia, hlavne z oblasti jazyka C++, z objektového návrhu, z návrhu užívateľského rozhrania a v neposlednom rade z oblasti riešenia optimalizačných problémov.

Literatúra

[1] KALAŠOVÁ, Alica, PAĎO, Jozef. *Dopravné inžinierstvo*. Žilina: EDIS, 2003. 165 s. ISBN 80-8070-076-0.

[2] ČEŠKA, M., VOJNÁR, T., SMRČKA, A. *Teoretická informatika : Studijní opora*. [s.l.] : [s.n.], 2009. 167 s.

[3] ČEŠKA, Milan, et al. *Petriho sítě : Studijní opora*. [s.l.] : [s.n.], 2006. 238 s.

[4] MEHROTRA, K., MOHAN, C., RANKA, S. Elements of Artificial Neural Networks The MIT Press, 1997. ISBN 0-262-13328-8

[5] NISSEN, Steffen, et al. *Fast Artificial Neural Network Library (FANN)* [online]. 2004 [cit. 2010-01-03]. Dostupné z WWW: <http://leenissen.dk/fann/>.

[6] ZBOŘIL, František. *Agentní a multiagentní systémy : Studijní opora*. [s.l.] : [s.n.], 2006. 65 s.

[7] Telecom Italia. *Java Agent DEvelopment Framework* [online]. 2001 [cit. 2010-01-03]. Dostupné z WWW: <http://jade.tilab.com/>.

[8] SENDA, Y., TANEV, I.,SHIMOHARA, K. *On the Possibility of Priority-Based Road Traffic Control* [online]. 2008 [cit. 2010-01-01]. Dostupné z WWW: <http://ieeexplore.ieee.org>.

[9] SCHWARZ , Josef, SEKANINA, Lukáš. *Aplikované evoluční algoritmy : Studijní opora*. [s.l.] : [s.n.], 2006. 100 s.

[10] WALL, Matthew. *GAlib C++ Library* [online]. 1996 [cit. 2010-01-03]. Dostupné z WWW: <http://lancet.mit.edu/ga/>.

[11] PERINGER, Peter. *SIMulation LIBrary for C++* [online]. 2002, 29.11.2009 [cit. 2010-01-03]. Dostupné z WWW: <http://www.fit.vutbr.cz/ peringer/SIMLIB/>.

[12] PERINGER, Peter. *Modelování a simulace: Studijní opora*. [s.l.] : [s.n.], 2008. 84 s.

[13] ZEIGLER, B., PRAEHOFER, H., KIM, T. *Theory of Modelling and Simulation, 2nd edition*. Academic Press, 2000

[14] REIF, J. *Synthesis of Parallel Algorithms.* Morgan Kaufmann, 1993 ISBN 155860135X

[15] MILNER, Robin. *The Polyadic pi-Calculus: A Tutorial.* 1991. 49 s. Dostupné z WWW: <http://www.lfcs.inf.ed.ac.uk/reports/91/ECS-LFCS-91-180/>.

[16] CAPRETZ, Fernando. *A brief history of the object-oriented approach.* ACM Digital Library, 1993. 6 s.

[17] Open Standards. *ISO/IEC 14882: Programming languages - C++* [online]. 2003 [cit. 2008-05-03]. Dostupné z WWW: <http://www.open-std.org/JTC1/SC22/WG21/docs/papers/2007/n2461.pdf>

[18] FAJMON, Břetislav, RUŽIČKOVÁ, Irena. *Matematika 3.* UMAT FEKT, 2004

[19] SMART, Julian, HOCK, Kevin. *Cross-Platform GUI Programming with wxWidgets.* [s.l.] : [s.n.], 2005. 700 s. ISBN 0-13-147381

[20] WILLIAMS, T., KELLEY, C. *Gnuplot* [online]. 2010 [cit. 2010-04-03]. Dostupné z WWW: <http://gnuplot.sourceforge.net/>.

[21] *Extensible Markup Language (XML)* [online]. 2010 [cit. 2010-05-03]. Dostupné z WWW: <http://www.w3.org/XML/>

[22] THOMASON, Lee. *TinyXML* [online]. 2010 [cit. 2010-03-03]. FIT VUT Brno, Dostupné z WWW: <http://www.grinninglizard.com/tinyxml/>.

[23] SATOLA, Richard. *Genetické algoritmy* [online]. 2004 [cit. 2010-05-03]. Dostupné z WWW: <https://www.fit.vutbr.cz/study/courses/EVO/private/VYUKA/_09/P1/GA_sato.pps>.

[24] ŽALOUDEK, Luděk, et al. *Tutorial ke knihovně GAlib pro C++.* [s.l.] : [s.n.]

[25] WALL, Matthew. *GAlib: A C++ Library of Genetic Algorithm Components* [online]. 2006 [cit. 2010-04-19]. Dostupné z WWW: <http://lancet.mit.edu/ga/dist/galibdoc.pdf>.

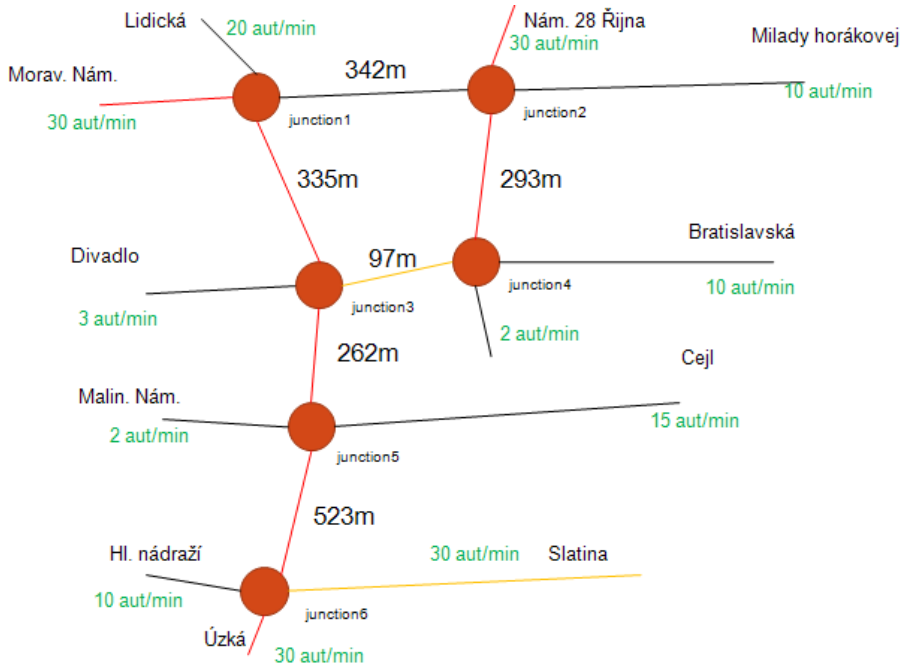
[26] POSPÍCHAL, Petr, JAROŠ, Jiří . *GPU-based Acceleration of the Genetic Algorithm* [online]. 2009 [cit. 2010-04-19]. Dostupné z WWW: <http://www.gpgpgpu.com/gecco2009/7.pdf>.

[27] Computer Science and Mathematics. *PVM: Parallel Virtual Machine* [online]. 2010 [cit. 2010-05-10]. Dostupné z WWW: <http://www.csm.ornl.gov/pvm/>.

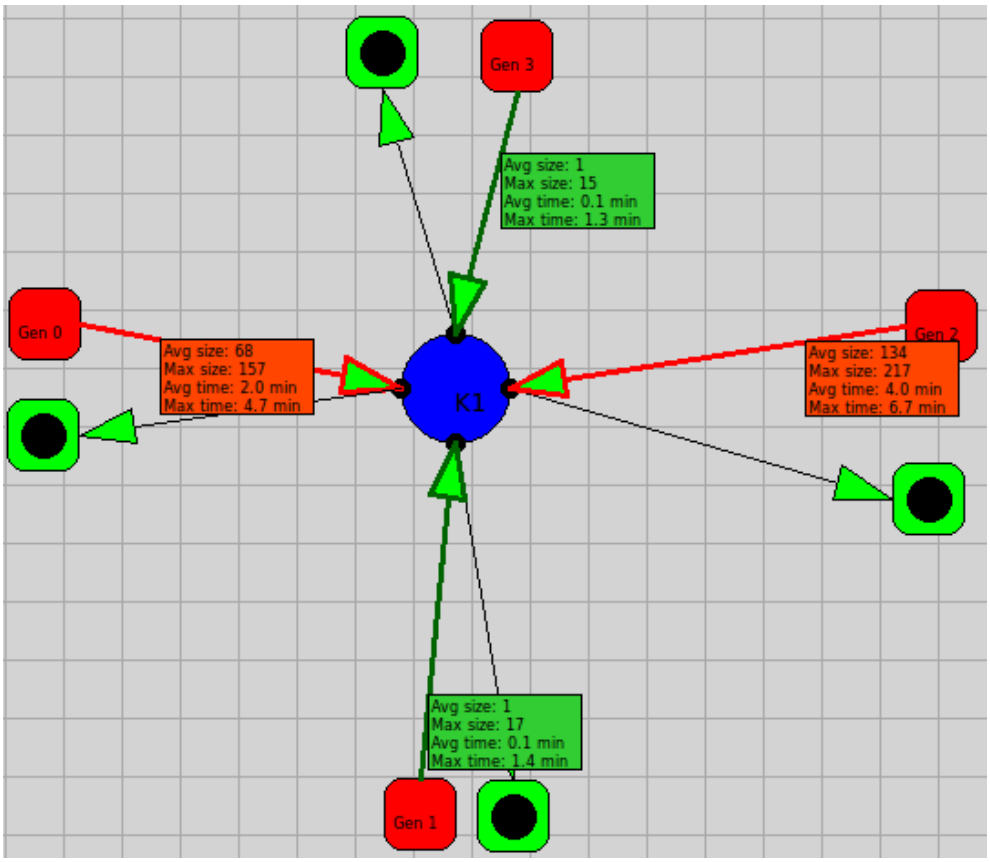
[28] Nvidia. *CUDA Zone* [online]. 2010 [cit. 2010-05-10]. Dostupné z WWW: <http://www.nvidia.com/object/cuda_home_new.html>.

Dodatok A

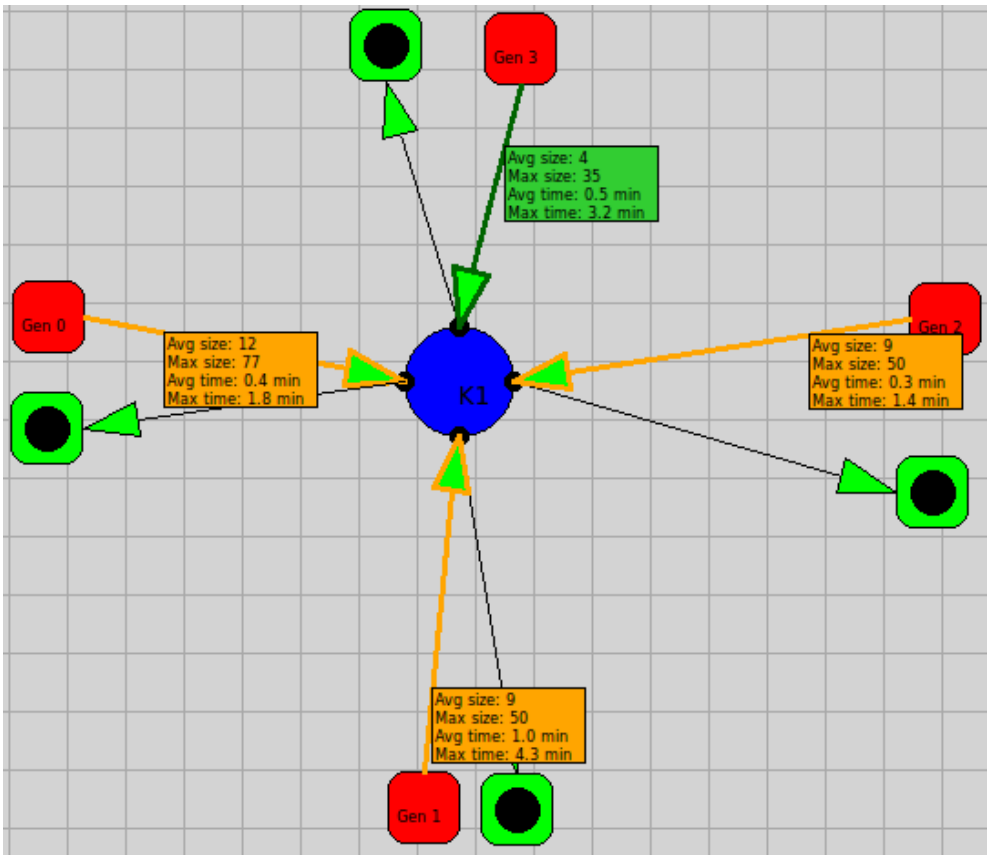
Obrazové prílohy



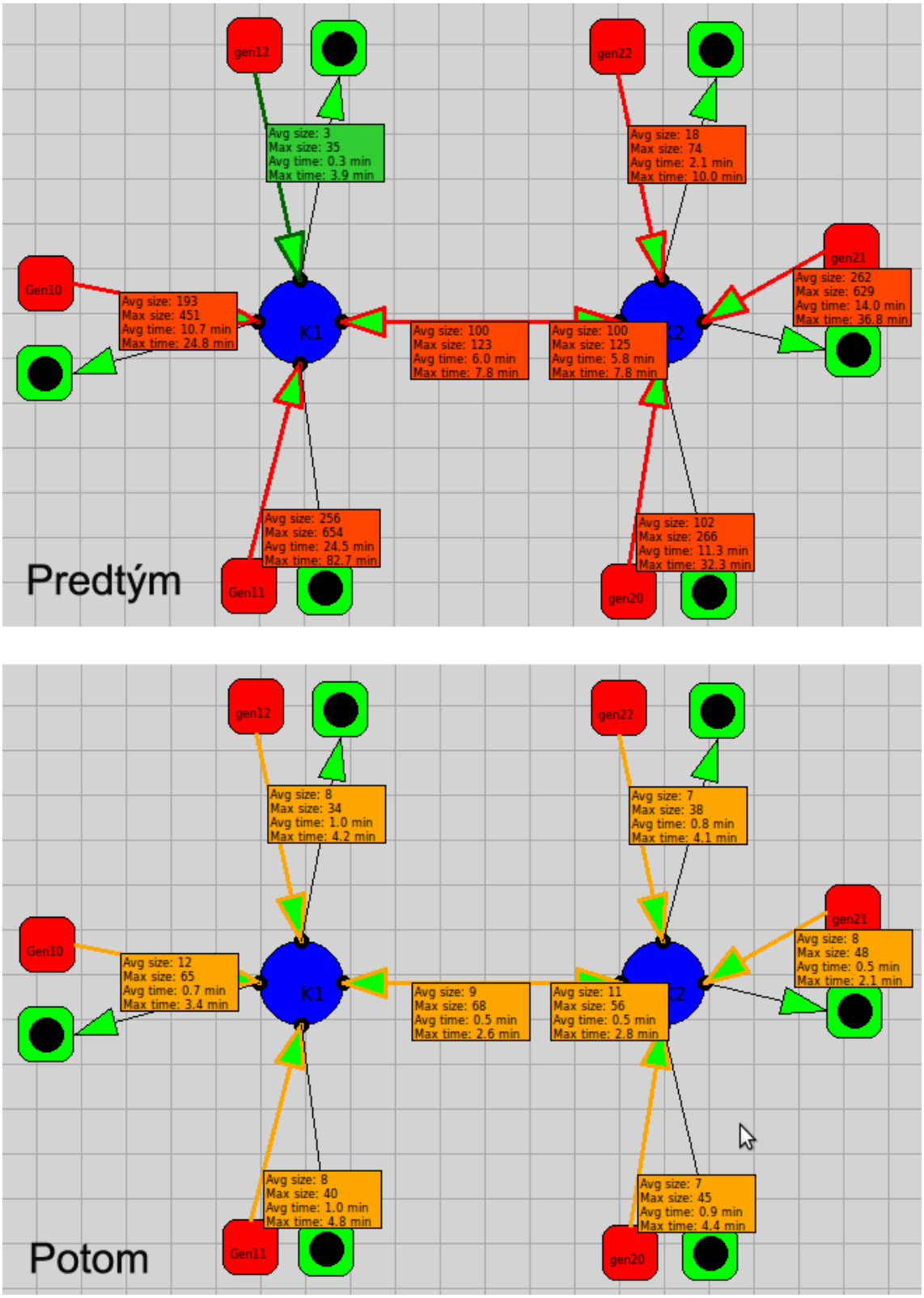
Obrázok A.1: Schéma simulovanej križovatky pri použití agentného prístupu



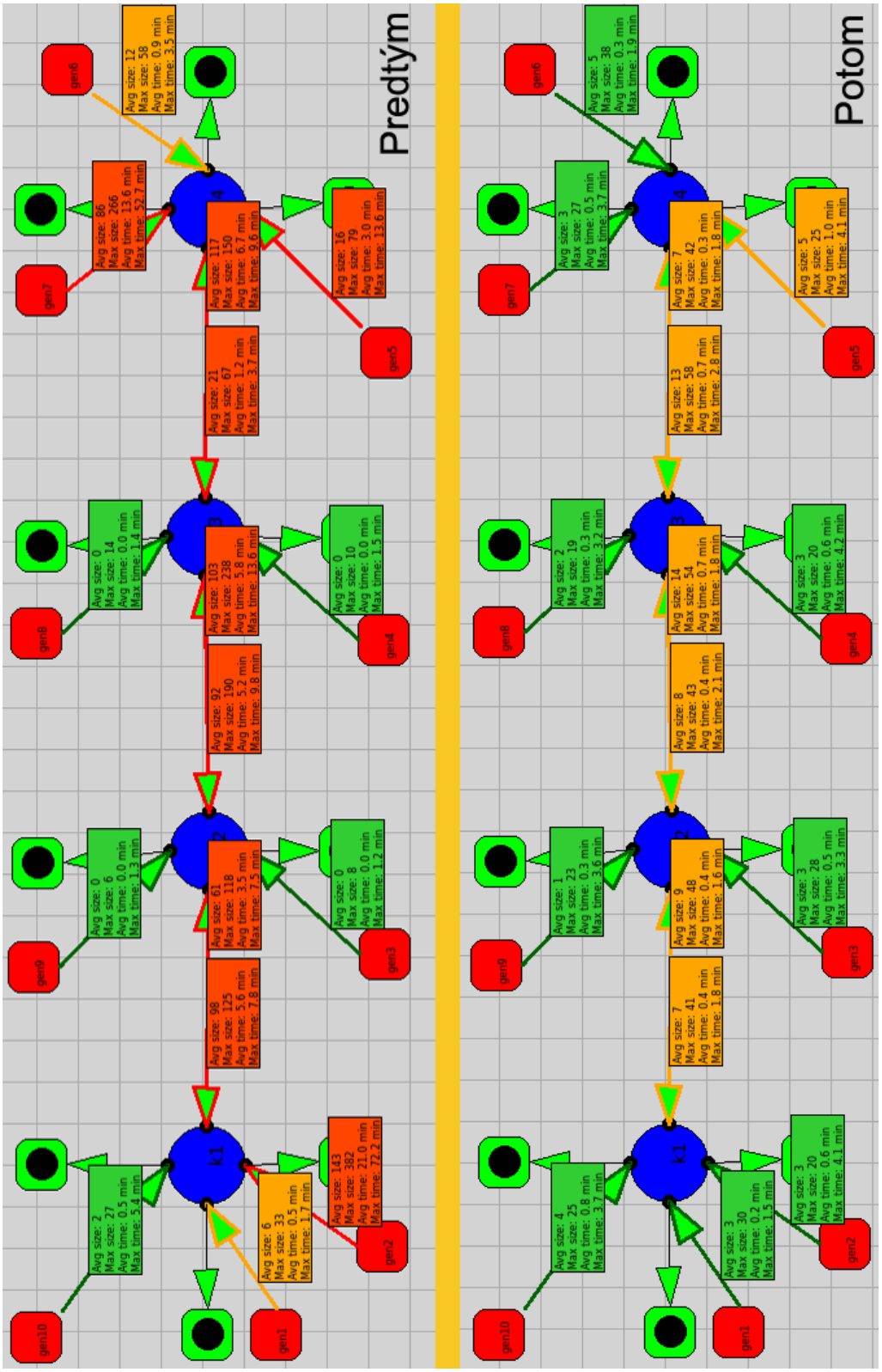
Obrázok A.2: Test1: Systém križovatiek pred optimalizíciou



Obrázok A.3: Test1: Systém križovatiek po optimalizácii



Obrázok A.4: Test2: Systém križovatiek pred a po optimalizácii



Obrázok A.5: Test3: Systém križovatiek pred a po optimalizácii