VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

SIMULAČNÍ MODEL SVOZU ODPADU PRO NETWORK SIMULATOR 3

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

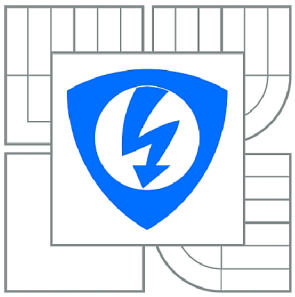
AUTOR PRÁCE
AUTHOR

Bc. LUKÁŠ KOLAJA

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

SIMULAČNÍ MODEL SVOZU ODPADU PRO NETWORK SIMULATOR 3

SIMULATION MODEL OF WASTE COLLECTION FOR NETWORK SIMULATOR 3

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

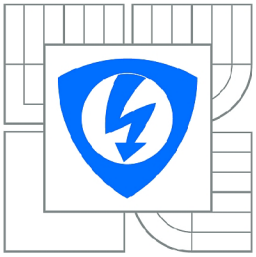
AUTOR PRÁCE — Bc. LUKÁŠ KOLAJA

AUTHOR

VEDOUCÍ PRÁCE — Ing. RADEK FUJDIAK

SUPERVISOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Lukáš Kolaja	***ID:***	134336
Ročník:	2	***Akademický rok:***	2014/2015

NÁZEV TÉMATU:

Simulační model svozu odpadu pro Network Simulator 3

POKYNY PRO VYPRACOVÁNÍ:

Student bude mít za úkol vytvořit v Network Simulatoru 3 (NS-3) model pro svoz odpadu ve městě (metoda řízní bude vycházet ze semestrálního projektu). Model by měl být jednoduše upravitelný a nastavitelný (mělo by být možné v něm simulovat danou metodu ze semestrálního projektu, klasický svoz odpadu a dále by mělo být možné jednoduchými úpravami, které budou popsány, zkoušet další různé metody řízení). Jednotlivé implementované metody pro řízení budou v modelu vyzkoušeny, proměřeny a následně porovnány. Výstupem tedy bude simulační nástroj pro NS-3 a proměřené implementované metody řízení ve vytvořeném simulačním nástroji.

DOPORUČENÁ LITERATURA:

[1] NS3. "NS3 Network Simulator: ns-3 Manual". Release ns-3-dev (ns-3 project). October 2014.
[2] NS3. "NS3 Network Simulator: ns-3 Tutorial". Release ns-3.18.1 (ns-3 project). November 2013.

Termín zadání:	9.2.2015	***Termín odevzdání:***	26.5.2015

Vedoucí práce: Ing. Radek Fujdiak
Konzultanti diplomové práce:

doc. Ing. Jiří Mišurec, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Cílem této diplomové práce je vytvořit aplikaci pro optimalizaci trasy inteligentního svozu odpadu, který spadá do technologií chytrých měst. Nejdříve byla popsána problematika chytrých měst s následným zaměřením na svoz odpadu. V práci je popsáno reálné nasazení systému pro inteligentní svoz odpadu za použití bezdrátové senzorické sítě a také byl vytvořen vlastní komunikační model pro monitorování a svoz odpadu. Dále je také popsána teorie grafů a s tím spojené genetické algoritmy vhodné právě pro optimalizaci trasy svozu odpadu.

Na tomto základě byla vytvořena aplikace napsaná v jazyce C/C++, která pomocí genetického algoritmu počítá optimální trasu v daném grafu, který reprezentuje mapu oblasti svozu. Vstupními daty do aplikace je vektorový obrázek ohodnoceného grafu ve formátu SVG.

KLÍČOVÁ SLOVA

chytrá města, svoz odpadu, C, C++, genetické algoritmy, teorie grafů

ABSTRACT

The goal of this diploma thesis is create an application for route optimalization for waste collection which is one of the technologies of smart cities. At first was described issue of smart cities focused to waste collection. The thesis describes the real deployment of smart waste collection using sensor network and was also designed its own model of smart waste collection. It is also described graph theory and related genetic algorithms which is suitable for waste collection optimalization.

On that basics an application was made in C/C++ language which using a genetic algorithm to compute best possible path in graph which represents a map where waste is collected. By input data to application is vector image of evaluated graph in SVG data format.

KEYWORDS

smart cities, waste collection, C, C++, genetic algorithm, graph theory

KOLAJA, Lukáš *Možnosti pro svoz odpadu v chytrých městech*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2015. 59 s. Vedoucí práce byl Ing. Radek Fujdiak,

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Možnosti pro svoz odpadu v chytrých městech“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Radku Fujdiakovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

(podpis autora)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic

http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

Výzkum popsaný v této diplomové práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Brno . . . . . .

. . . . . .

(podpis autora)

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OP Výzkum a vývoj
pro inovace

OBSAH

Úvod **10**

1 Chytrá města **11**
1.1 Systémy chytrých měst 12

2 Svoz odpadu v chytrých městech **14**
2.1 Měřící senzory 15
2.2 Konvenční svoz odpadu 16
2.3 Monitorování stavu kontejnerů 16
2.4 Vlastní návrh systému svozu odpadu 18

3 Teorie grafů **20**
3.1 Floyd-Warshallův algoritmus 21
3.2 Problém obchodního cestujícího 22
3.3 Genetické algoritmy 24
3.3.1 Myšlenka a algoritmus 24
3.3.2 Výhody a nevýhody genetických algoritmů 28

4 Aplikace pro výpočet optimální trasy **29**
4.1 NS-3 Síťový simulátor 29
4.1.1 Výhody NS-3 30
4.2 O aplikaci 32
4.2.1 Spuštění aplikace 32
4.3 Vstupní data programu 33
4.3.1 Tvorba vstupního SVG souboru 33
4.4 Parsování vstupního SVG souboru 36
4.5 Genetický algoritmus pro výpočet optimální trasy v grafu 38
4.5.1 Křížení 41
4.5.2 Mutace 43

5 Parametrizace genetického algoritmu **44**
5.1 Výchozí nastavení parametrů algoritmu 44
5.2 Vliv velikosti populace 45
5.3 Vliv poměru křížení a mutací 47
5.4 Vliv velikosti zahazování populace 48
5.5 Měření úspory nákladů svozu odpadu 50

6 Závěr **51**

Literatura **52**

Seznam symbolů, veličin a zkratek **55**

Seznam příloh **56**

A Komunikační modely systémů pro svoz odpadu **57**
A.1 Příklad modelu pro svoz odpadu 57
A.2 Navržený model pro monotorování kontejnerů 58

B Obsah přiloženého CD **59**

SEZNAM OBRÁZKŮ

3.1 Příklad jednoduchého grafu. 20
3.2 Vývojový diagram genetického algoritmu 25
4.1 Struktura NS-3. 30
4.2 Programová organizace NS-3. 31
4.3 Příklad vytvořeného grafu. 34
5.1 Graf váhové a časové závislosti na velikosti populace 46
5.2 Graf váhové a časové závislosti na koeficientu křížení 48
5.3 Graf váhové a časové závislosti na poměru zahazování populace . . . 49
5.4 Graf váhové a časové závislosti na počtu uzlů v grafu 50
A.1 Komunikační model systému pro monitorování kontejnerů. 57
A.2 Navržený model pro monitorování stavu kontejnerů. 58

ÚVOD

V dnešním světě jsou odpady velmi důležitou otázkou a problémem především v městských oblastech, jelikož stále dochází k rychlému nárůstu počtu obyvatel a s tím spojené nároky na sběr odpadu. Sběr odpadu je komplexní proces, který vyžaduje použití velkého množství peněz a propracovaného řízení logistiky. V dnešní době však existuje několik pokročilých řešení jak s těmito problémy účinně naložit za co nejmenší náklady, s maximálním možným výsledkem služeb a spokojenosti zákazníka [1].

Pokročilé systémy svozu odpadu pomáhají zlepšit a optimalizovat nejen sběr odpadu všeho druhu, ale i jak je s odpadem dále nakládáno. Takové systémy pak nejen šetří náklady na provoz, ale také zkvalitňují poskytované služby zákazníkům a pomáhá udržovat čisté a zdravé prostředí ve městech [1]. Nejdříve se tedy práce zaměří na systémy používané v chytrých městech.

Cílem této práce je vytvořit aplikaci pro optimalizaci svozu odpadu na základě popisu problematiky současného svozu odpadu a inteligentního řešení svozu odpadu. Pozornost také bude věnována teorií grafů a optimalizačními genetickými algoritmy pro vyhledání optimální trasy, které jsou vhodné pro inteligentní svoz odpadu. Je také navržen vlastní komunikační model pro monitorování a svoz kontejnerů. Pro řešení svozu odpadu byl uvažován síťový simulátor NS-3, který se však při vývoji aplikace neukázal jako dostatečně vhodný. NS-3 tedy pro vytvoření vlastní aplikace nebylo využito a pro tvorbu vlastní aplikace je vybrán programovací jazyk C a C++ a vstupními daty je vektorový obrázek ohodnoceného grafu ve formátu SVG.

1 CHYTRÁ MĚSTA

Více než polovina světové populace dnes žije v městských oblastech, které jsou zodpovědné za 60-80% spotřebované energie, vody a vyprodukovaných emisí. Lidé se také stěhují z venkova do měst a předpokládá se, že tento trend bude pokračovat i nadále pro příštích pár let [2].
Městské aglomerace jsou katalyzátorem ekonomického rozvoje (a také místem socio-ekonomického a kulturního pokroku), avšak stále více pociťují problémy, které jsou bezprostředně závislé na stavu a kvalitě infrastruktury a infrastrukturních technologií [3].
Zavedením kvalitní infrastruktury a dalších řešení lze předejít mnoha problémům jako jsou potíže s řízením a nakládáním s odpady, nedostatek zdrojů, znečištění ovzduší, řízení dopravy. Tyto problémy se však stávají něčím víc, než jsou základní problémy technické, materiální nebo fyzické povahy, ale stávají se čím dál více otázkou sociální a organizační. Závisí také na mnoha dalších faktorech, jako je sociální a politická složitost, míra závislosti a konkurenční cíle a hodnoty společnosti [2].

Městské odbory často investují nezávisle na sobě, což se projevuje minimálním sdílením infrastruktur a zdrojů informačních technologií, minimálním sdílením inteligence a informací typu dat ze senzorických sítí - záznamů z kamer apod., duplikacemi v investicích a činnostech a problémy se škálováním správy všech infrastruktur [2].

Zajištění zachování nebo zlepšení životní úrovně v rámci tak rychlého růstu městské populace po celém světě vyžaduje hlubší porozumění konceptu inteligentních měst. Naléhavost kolem těchto výzev přinutilo mnoho měst po celém světě hledat, vyvíjet a realizovat chytřejší způsoby, jak je zvládat. Tato města jsou stále více označována jako „chytrá města“. Tak se dá označit město, které se jeví jako dlouhodobě udržitelné a obyvatelné s velkou mírou komfortu a kvalitními službami [2].

V první kapitole jsou popsány nejčastější systémy používané v chytrých městech a následně v další kapitole se zaměří na problematiku svozu odpadu, jeho požadavky a možná technická řešení.

1.1 Systémy chytrých měst

Mezi technologie a systémy chytrých měst patří například:
Parkování – Příkladem jsou automatické displeje na místních komunikacích indikující počet volných parkovacích míst v parkovacích domech. V případě pouličního parkování jsou využívány magnetické nebo infračervené detektory zabudované v povrchu vozovky. Dalším způsobem může být použití CCTV kamer, které sledují registrační značky vozidel nebo RFID senzorů, kde jsou vozidla vybavena vinětami (tzv. tagy). Mnoho měst má takové systémy a informace jsou šířeny i jinými kanály jako web a mobilní aplikace [4] [5] [3].

Řízení dopravy – Mezi nejdůležitější činnosti systému patří optimalizace a řízení dopravní sítě, informování a navigování, preference městské hromadné dopravy a zajištění návaznosti na další systémy a subsystémy jako jsou tunelové technologie, poskytování informací před jízdou, řízení proměnných informačních tabulí apod. Systém využívá rozsáhlou síť kamer, senzorů nebo dat z GPS, ze kterých se pomocí SW algoritmů vyhodnocuje hustota provozu a sledují se abnormální situace. Řízení dopravy často začíná již dlouho před městem samotným tak, aby bylo možné včas uplatnit systém dynamických dopravních značek, které pomáhají uregulovat dopravu do patřičné hustoty vhodným směrováním přijíždějících vozidel [6] [1].

Inteligentní sítě – Systémy pro vývoj rozvodných sítí nízkého a středního napětí. Hlavním smyslem těchto systémů je zvyšování spolehlivosti dodávek energie za pomoci kontrolních a řídících systémů a integrovaných senzorů, které monitorují chování sítě. Energetická společnost má také dostupné informace v reálném čase o vytížení sítě, kvalitě dodávky nebo jejího přerušení [7].

Městské osvětlení – Systém se skládá z LED světla, čidla jasu, snímače pohybu a komunikační sítě na krátkou vzdálenost. Hlavní funkcí inteligentních pouličních světelných systémů je, že jsou rozsvíceny pouze v případě potřeby. V praxi to funguje tak, že není-li v dosahu žádný člověk nebo projíždějící auto, je produkované světlo staženo na minimum, nebo dokonce zcela vypnuto. V případě že systém zaznamená pohyb osob nebo vozidel, je vlastní světlo rozsvíceno do jisté míry potřebné k osvícení daného místa. Lidé a vozidla vůbec nerozeznají rozdíl oproti běžnému pouličnímu osvětlení. Systém tak snižuje světelný smog měst a spotřebu energie k provozu [8].

Odpad - Systém staví na jednoduchém základu – každý kontejner je vybaven detektorem, který o míře jeho zaplnění informuje centrálu. Systém dále může vyhodnocovat frekvenci používání daného kontejneru, hlásit poruchy či nebezpečí požáru, optimalizovat jízdy svozných vozidel podle statistiky či podle reálné naplněnosti kontejnerů a v neposlední řadě i nastavit cykly potřebné údržby. Systém poskytuje tyto přínosy: snižuje náklady na svoz odpadů a údržbu kontejnerů, čímž snižuje emise oxidu uhličitého a občanům přináší možnost kvalitnějších služeb a udržet snížit stávající výši paušálních plateb za svoz odpadu [3].
Jelikož předmětem práce je právě optimalizace svozu odpadu, jsou následující kapitoly věnovány právě této problematice.

2 SVOZ ODPADU V CHYTRÝCH MĚSTECH

Koncept chytrých měst staví na čitelných strategiích, které vyčíslí prospěšnost městské regulace a nasazení chytrých technologií. Tato prospěšnost může být ekonomická, ekologická či společenská. Největším motivem ale zůstává „úspora v rodinném rozpočtu“, tedy osobní prospěch každého občana z dané strategie města. A oblast nakládání s odpady je jednou z velkých příležitostí [3].

Všechna města, bez ohledu na jejich velikost, zeměpisné umístění nebo jejich ekonomickou úroveň, utratí obrovské množství peněz každý rok za sběr odpadu. Počet kontejnerů umístěných v ulicích a počet sběrných vozů používaných k jejich vyprázdnění jsou obecně odhadnuty na základě počtu obyvatel v dané oblasti. Výsledný odhad je někdy však příliš vysoká nebo naopak nízká. Přirozeným následkem takového odhadu může pak být poskytování nekvalitních služeb svozu odpadu nebo jsou ve výsledku velmi vysoké náklady za provoz např. za pohonné hmoty, protože bylo využíváno zbytečně příliš mnoho sběrných vozidel. Kromě toho, sběr odpadů, bez ohledu na druh materiálu (recyklovaný nebo netříděný), je obvykle stanovený na základě periodického sběru bez zohlednění skutečného stavu a úrovně zaplnění jednotlivých kontejnerů. Výsledkem je pak skutečnost, že se jezdí svážet minimálně zaplněné kontejnery nebo naopak došlo v některých případech k přeplnění kontejneru a tím k snížení hygienických podmínek daného místa [9].

Předpovědět nejvhodnější čas pro svoz daného kontejneru a optimalizovat počet sběrných vozů a kontejnerů je možné jen tehdy, máme-li k dispozici informace o denním vyprodukovaném množství odpadu v daném místě. Díky nízkonákladovým mobilním zařízením, které mají také velmi nízké energetické nároky na provoz, můžeme sbírat informace o vyprodukovaném odpadu a stavu kontejnerů v reálném čase. Tyto informace pak vidí dispečeři ve středisku a systém na základě sběru těchto dat upravuje strategie svozu. Použitím různých typů čidel je dále možné dosáhnout, aby systém pracoval ještě efektivněji nebo dále rozšířil možnosti systému [9].

Hlavní cíle systému pro inteligentní svoz odpadu tedy jsou:

- Optimalizace tras pro sběrné vozy,
- zprostředkovat kvalitní služby zákazníkům,
- monitorování stavu kontejnerů, přehled o údržbě,
- redukce nákladů spojená se svozem odpadu jako je například palivo pro sběrné vozy,
- v případě přesunu, odcizení nebo poškození kontejneru je pro firmy odpovědné za služby velmi důležité, aby bylo možné sledovat umístění jednotlivých kontejnerů v městské oblasti a také redukovat krádeže,
- identifikace a monitorování zaměstnanců pracujících ve službách svozu odpadu,
- zvýšení recyklace a tím tak snižovat dopad na životní prostředí.

Splnění všech těchto cílů je velmi náročné protože některé se mohou navzájem vylučovat a tak je potřeba najít určitý kompromis [9].

2.1 Měřící senzory

Nejčastěji používané typy kontejnerových senzorů a jejich využití:
Senzor zaplnění – ultrazvukový senzor pracující na principu odrazu vln, které sám generuje. V závislosti na čase v jaký se vlna vrátí zpět do senzoru je vyhodnocena vzdálenost překážky respektive odpadu od senzoru. Slouží k určení do jaké míry je kontejner zaplněn [9].

Senzor váhy – určuje množství odpadu v kontejneru. Slouží jako informativní parametr pro vyhodnocování množství odpadu, ale také může poskytnout centrále varování v případě, že je z kontejneru něco odcizeno [9].

Teplotní a kouřové čidlo – slouží k rozpoznání požáru kontejneru a zkracuje příjezdovou dobu záchranných složek [10].

GPS modul – slouží k lokalizaci daného kontejneru, zaznamenává souřadnice a tím lze monitorovat jeho pohyb [9].

RFID čip – může obsahovat různá data jako je například GPS souřadnice, barva a určení kontejneru, záznamy o údržbě kontejneru apod. RFID může ale také být aktivní a může aktivovat například mechanismus pro otevření víka kontejneru těm, kteří nemohou dosáhnout patřičné výšky, tj. hendikepovaným [10].

2.2 Konvenční svoz odpadu

Standardní svoz odpadu je ve své podstatě založen na pravidelných intervalech. Jak již bylo psáno dříve, neřeší se, do jaké míry jsou kontejnery zaplněny. Do jisté míry se však ale bere v úvahu sociální a ekonomická vyspělost oblasti svozu, roční období (v letních měsících bývá svoz častější), ale stále nemá odpovědné firma za svoz odpadu žádný přehled o aktuálním stavu kontejnerů, takže jsou svážený všechny kontejnery v dané oblasti [11].

2.3 Monitorování stavu kontejnerů

V minulosti se společnosti zabývaly potenciálem různých informačních a komunikačních technologií jako je RFID (Radio Frequency Identification), GIS (Geographic Information Systems), GPS (Geographic Positioning System) a různé dopravní modely pro monitorování a svoz kontejnerů. Některé výzkumy dokonce počítaly se sledováním v reálném čase, avšak narazily na některá omezení jako bylo například malé pokrytí oblasti nebo při použití čistě GSM/GPRS modulů znamenal každý další kontejner velký nárůst nákladů. Řešením bylo implementování bezdrátové senzorické sítě. Taková síť založená na kombinaci ZigBee a GSM/GPRS technologie poskytuje energeticky nenáročný systém a umožňuje přenášet velké množství dat a poskytuje solidní územní pokrytí [12].
Použití ZigBee modulů přináší také výhodu v nižších pořizovacích nákladech a to až o polovinu na jeden kontejner, protože není nutné platit registraci u telefonního operátora a jsou také energeticky méně náročné. Důležitá je také ochrana proti vandalům a obecnému poškození modulů při manipulaci s kontejnerem, proto jsou moduly chráněny robustním plastovým krytím [13].

Takový systém se pak skládá ze tří vrstev (v příloze obr. A.1). První a nejnižší vrstva se skládá z kontejneru, prostřední vrstvou je brána a nejvyšší vrstva je řídící stanice. Na základě této architektury se systém skládá z těchto fyzických složek:[13]

1. Senzorový modul uvnitř kontejneru, obsahuje různé měřící senzory jako je zaplnění kontejneru, váha odpadu, vlhkost, teplota apod. Tyto data pak odesílá pomocí rádiového vysílání ZigBee modulu. ZigBee modul bývá nejčastěji vybaven procesorem ATMega.

2. Brána, která reprezentuje prostřední vrstvu a vytváří most mezi kontejnerem a řídící stanicí. Je složena ze ZigBee modulu kterým shromažďuje data z kontejnerů a přeposílá je pomocí GSM/GPRS modulu do řídící stanice.
   GSM/GPRS modul zajišťuje komunikaci na dlouhou vzdálenost do řídícího střediska a brána jako taková je postavena na bázi operačního systému Linux a zajišťuje funkčnost při použití různých komunikačních technologií.

3. Servery v řídící stanici. Jsou zde uloženy naměřená data a také zde běží aplikace pro sběr dat a sledování stavu kontejnerů. Skládá se z databázového a webového serveru. Přijatá data jsou uložena a běžící aplikace pak tyto data analyzují a vyhodnocují stav konkrétních kontejnerů. Další webové aplikace pak umožňují uživateli správu a sledování kontejnerů přes webové rozhraní. Nashromážděná data pak také mohou sloužit pro výpočet optimální trasy svozu v dané lokalitě.

Provozní princip systému je takový, že senzorový modul je v klidovém stavu a není pak tak energeticky náročný. Modul reaguje jen na základě nějaké události, například je-li do kontejneru vhozen odpad, nebo jestli není zaznamenán požár apod. Po vhození odpadu se modul aktivuje a změří parametry ze senzorů. Po odečtení všech veličin následně odešle data do brány prostřednictvím ZigBee modulu. Řídící servery v tu chvíli naslouchají, zda nějaká brána neodesílá požadavek pro zaslání dat. Brána zažádá o zaslání dát a je vytvořen kanál na bázi TCP/IP prostřednictvím GPRS spojení. Řídící server tak data obdrží a následně je ukládá a dále zpracovává [13].

2.4 Vlastní návrh systému svozu odpadu

Systém bude uvažovat pro výpočet optimální trasy s určitými městskými částmi nebo bloky. Bloky jsou vhodně rozděleny dle velikosti a složitosti a každý takový blok pak bude obsluhovat jeden svážecí vůz. Kontejnery budou vybaveny senzorem zaplnění, senzorem váhy a teplotním čidlem. O odeslání naměřených parametrů a o aktuálním stavu kontejneru bude obstarávat jednoduchý modul SRD (z anglického Short Range Device neboli zařízení na krátkou vzdálenost).
Jedná se o zařízení, které pracuje ve volném frekvenčním pásmu 863–870 MHz, dle podmínek stanovených CEPT (Konference evropských správ pošt a telekomunikací). Podle těchto podmínek lze v Evropě provozovat SRD zařízení ve stanovených pásmech a za daných parametrů. Výhodou těchto zařízení je nízká cena (menší jak u technologie ZigBee), nízká energetická náročnost a malé rozměry. Nevýhodou je horší dostupnost těchto zařízení a taky fakt, že komunikující moduly musí být ve svém vysílacím dosahu, takže není možné pokrýt tak rozsáhlé oblasti v porovnání se ZigBee. Je tedy nutné vhodně umístit příjmač (bránu) pro danou oblast. Nevýhodou je také nutnost naprogramování vlastního komunikačního protokolu, kterým budou moduly v kontejnerech odesílat data do brány.

Brána která bude shromažďovat data bude umístěna například na sloupu veřejného osvětlení nebo budově, tak bude relativně dobře zabezpečen proti krádeži nebo poškození. Může být založena například na mikrokontroléru MSP430, ke kterému je připojen SRD a Ethernet modul. Výhodou je opět nízká cena a energetická nenáročnost. Výpočetní výkon takového modulu by měl bez potíží zvládnout přeposílat objem dat (řádově desítky kB/s v závislosti na výbavě kontejnerů) produkovaný kontejnery. Do řídící stanice se pak data odešlou prostřednictvím internetu, do kterého bude brána připojena kabelem. Internetové připojení bylo vybráno z důvodů dostupnosti a není nutné zavádět další infrastrukturu pro přenos dat do řídící centrály. Zde je však malá nevýhoda takového systému a to ta, že je nutné platit poplatek za internetové připojení. Datový tok však pro tento systém nebude velký, by mělo by tak stačit i nízkorychlostní připojení, takže cena za zprostředkování internetu by neměla být překážkou pro nasazení. Brána bude tedy tvořit takový most mezi kontejnery a řídící stanicí.

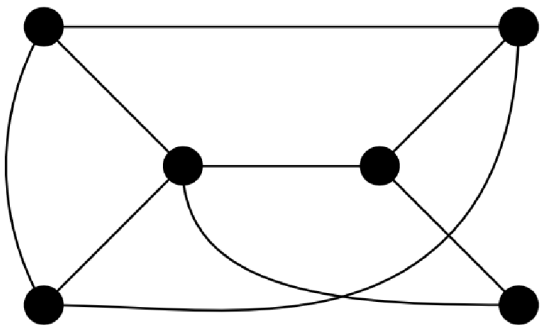
Do řídící stanice se data odešlou pomocí TCP protokolu. Důvodem je nutná spolehlivost doručení dat, protože dojde-li například v místě kontejneru k požáru a teplotní čidlo spolu s modulem vyšlou zprávu o takovém stavu, tak je žádoucí, aby řídící stanice takovou zprávu skutečně obdržela a mohla adekvátně reagovat.

V řídící stanici je pak umístěn databázový, webový a případně výpočetní server, kde se ukládají přijatá data, vyhodnocuje se stav kontejnerů a počítá se optimální trasa pro jednotlivé bloky za pomoci mravenčích algoritmů. Sběrný vůz pak může vložit informace o trase do GPS navigace. Díky tomu, že je systém připojen do internetu může zákazník prostřednictvím webové aplikace odeslat případné požadavky, informovat o stavu kontejneru (krádež, poškození) apod. Takový systém je pak graficky znázorněn v příloze na obr.A.2.

Je-li k dispozici přehled o stavu kontejnerů a jejich zaplnění, je pak nutné definovat kde se kontejnery nachází a sestavit mapu kontejnerů, jejich spojení a ohodnocení. Takovou definici splňují grafy, o nichž se bude pojednávat v následující kapitole.

3 TEORIE GRAFŮ

Grafem se v teorii grafů uvažuje něco jiného než je například graf funkce nebo například výsečový graf. *Graf G* je dvojice (V, E). Prvky množiny V nazýváme vrcholy a prvky $E \subseteq \{\{u, v\} | u, v \in V\}$ zase hrany. Hrana $e \in E$ je neuspořádaná dvojice vrcholů, tedy $e = \{u, v\}$ pro $u, v \in V$. Vrcholy *u, v* jsou konce hrany *e*. Jelikož platí $v \in e$, tak lze říci, že *v* náleží do *e*. Existence hrany se dá také vyjádřit: „vrcholy *u, v* jsou spojeny hranou *e*, „z *u* vede hrana do *v*, „vrchol *u* sousedí s vrcholem *v*. Vrcholy, se kterými je vrchol *u* spojen hranou, se nazývají sousedi vrcholu *u*. I když je graf spíše abstraktním pojmem, lze jej zakreslit na papír nebo v počítačovém programu. Vrcholy se zakreslí jako „body“ a hrany jako „spojnice bodů“, které je spojují (obr. 3.1) [14].



Obr. 3.1: Příklad jednoduchého grafu.

Grafem je možné zachytit velkou škálu souvislostí mezi objekty, například silniční síť. Křižovatky se znázorní jako vrcholy, a silnice, jako hrany. Každá silnice má svojí délku nebo odpovídající ohodnocení. Tuto délku můžeme připsat ke každé hraně a tímto způsobem dostaneme ohodnocený graf. Graf *G* je pak graf *G = (V, E)*, ke kterému přidáme funkci $h : E \to R$, která každé hraně *e* přiřadí hodnotu *h(e)*. Může však nastat případ že dvě křížovatky budou spojeny několika různými silnicemi. Jinými slovy, že dva vrcholy budou spojeny dvěma či více různými hranami. Běžné grafy nám neumožňují takové věci zachytit. Museli bychom pojem grafu zobecnit tak, že každé hraně přiřadíme její násobnost $N : E \to N$. Násobnost vyjadřuje, kolikrát je hrana v grafu zastoupena. Tím dostaneme multigrafy. Jiné zavedení multigrafů pracuje se zobrazením, které každé hraně přiřadí dva vrcholy, které jsou její konce. To už je blíže tomu, jak multigrafy používá programátor. V některých aplikacích se můžeme použití multigrafů (a násobných hran) vyhnout tím, že na jednu hranu umístíme fiktivní vrchol/křižovatku, který hranu rozdělí na dvě [14].

Existuje několik algoritmů a metod jak s takovými grafy pracovat a využít je pro řešení problémů jako je hledání kořenu grafu, nejkratší cesty v grafu od jednoho uzlu k druhému apod [14].

3.1 Floyd-Warshallův algoritmus

Jelikož se při řešení daného problému bude používat matice vzdáleností mezi uzly, tak Floyd-Warshallův algoritmus je vhodný pro orientované grafy, které neobsahují záporně ohodnocené hrany. Algoritmus pak najde nejkratší orientované cesty mezi každou dvojicí vrcholů. Navíc ze všech cest stejné délky vybere tu s nejmenším počtem hran. Pro výpočet vzdálenosti každé dvojice vrcholů lze také n-krát použít Dijkstrův algoritmus (na každý vrchol). Lepší možností je však použít Floyd-Warshallův algoritmus, který počítá všechny vzdálenosti přímo, proběhne rychleji než n-krát použitý Dijkstrův algoritmus a ještě se snadněji implementuje. Vrcholy grafu se očíslují od nuly do n. Vzdálenosti mezi každou dvojicí vrcholů se budou ukládat do matice $n * n$. Celý systém Floyd-Warshallova algoritmu spočívá v tom, že vzdálenosti se nepočítají přímo, ale v n iteracích. V i-té iteraci se vypočítá matice Di . Hodnota $Di[u, v]$ je délka nejkratší cesty z u do v , která smí procházet pouze přes vrcholy $\{0, 1, ..., i\}$. Jinými slovy $Di[u, v]$ je délka nejkratší cesty v podgrafu indukovaném vrcholy $\{0, 1, ..., i\}$. V nulté iteraci se začne s maticí $D0$. Hodnota $D0[u, v]$ je délka hrany uv , pokud z u vede hrana do v , nula na diagonále a v případě nesousedících uzlů se do pozice dosadí nekonečno (nebo velmi vysoké číslo vzhledem k ostatním ohodnocením hran). Matice $D0$ je tedy upravená matice sousedností, která místo jedniček obsahuje délky hran a místo nul mimo diagonálu nekonečna). V poslední iteraci se skončí s maticí Dn , která už bude obsahovat hledané vzdálenosti, protože cesty mezi u a v smí procházet přes všechny vrcholy [15].

Kód 3.1 pro výpočet matice vzdáleností vypadá takto (numOfNodes - počet uzlů, eval - dvojrozměrné pole vzdáleností mezi uzly) [15]:

```
for(int k = 0; k < numOfNodes; k++)
    for(int i = 0; i < numOfNodes; i++)
        for(int j = 0; j < numOfNodes; j++)
            if(eval[i][j]>eval[i][k]+eval[k][j])
                eval[i][j]=eval[i][k]+eval[k][j];
```

Code 3.1: Kód Floyd Warshallova algoritmu

Zde je uveden příklad počáteční a konečné matice vyhodnocené Floyd Warshallovým algoritmem pro 5 uzlů. Je vidět že po vyhodnocení jsou někdy nahrazeny i váhy hran vedoucí přímo z jednoho uzlu do druhého, jelikož byla nalezena cesta s menší celkovou váhou vedoucí přes jiné uzly.
Příklad počáteční a koncové matice po výpočtu:

$$\begin{pmatrix} 0 & 5 & \infty & 400 & \infty \\ 100 & 0 & 30 & \infty & \infty \\ \infty & 20 & 0 & 500 & 5 \\ 20 & \infty & 500 & 0 & 60 \\ \infty & \infty & 10 & 20 & 0 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 5 & 35 & 60 & 40 \\ 75 & 0 & 30 & 55 & 35 \\ 45 & 20 & 0 & 25 & 5 \\ 20 & 25 & 55 & 0 & 60 \\ 40 & 30 & 10 & 20 & 0 \end{pmatrix}$$

3.2 Problém obchodního cestujícího

V teorii grafů problém obchodního cestujícího se chápe jako nalezení nejkratší cesty, která je určena k návštěvě daného počtu měst a návratu do počátečního bodu. Neorientovaná úloha může být uváděna v teorii grafů nějak takhle: Nechť $G = (V, A)$ je graf, kde $V = \{v_1, ..., v_n\}$ je vrchol (nebo uzel) a $A = \{(v_i, v_j)|v_i, v_j \in V, i \neq j\}$ je hrana s nezápornými náklady (nebo vzdálenostmi) matice $C = (c_{ij})$ spojená s A. TSP spočívá v určení minima nákladů hamiltonovského cyklu v úloze grafu. Souměrné implicitní používání neorientované hrany, spíše než orientovaného oblouku, může být také výslovně stanovené vztahem $c_{ij} = c_{ij}$. Úloha obchodního cestujícího patří mezi NP-úplné úlohy [16].
Obchodní cestující chce tedy navštívit n různých měst a poté se navrátit domů. Chce stanovit pořadí cesty, tak aby cestovní vzdálenost byla minimalizovaná, když navštíví každé město ne více než jednou. Ačkoli slovní definice TSP je poměrně jednoduchá, v reálném světě je poměrně obtížné takové řešení dostat [17].
Řešený problém se svozem odpadu má jednu změnu co se definice takového problému týče. Po ohodnocení hran může docházet k místům, kde trasa do následujícího navštěvovaného uzlu bude vhodnější zvolit přes uzly, které již byly jednou navštíveny, jelikož výsledná váha trasy může být nižší než trasa vedoucí přímo do onoho uzlu po jediné hraně, která má však váhu hrany vysokou. Tudíž trasa může tedy ve výsledku vypadat např: A > B > C > D > E, ale z bodu C do bodu D může být výhodnější se navrátit do bodu B a z něj teprve přejít do bodu D, jsou-li tyto dva uzly spojeny.
V situaci n-počtu uzlů, kterákoliv permutace n uzlů přináší možné řešení. V důsledku toho musí být n! možných cest ohodnoceno v celém vyhledávacím prostoru. Při počtu 5 uzlů je počet cest 12. Při 20 je to však již $6.08 * 10^{16}$. Při 50 uzlech je to nakonec obrovské číslo $3 * 10^{62}$.

Z těchto počtů pak vyplývá že již řešení pro téměř stovku uzlů nepůjde vypočítat jen za pomoci hrubé síly či podobných systematických hledání možných kombinací aniž by nalezení netrvalo v takovém případě několik dní, ale je nutné nasadit komplexnější řešení.

Jelikož v reálném světě na trasách svozu odpadu mohou nastat případy jako jsou jednosměrky, dopravní zácpy, uzavírky apod., je tedy potřeba hrany ohodnotit v obou směrech. Váha cesty z bodu A do bodu B tak nemusí mít stejné ohodnocení jako cesta z bodu B do bodu A. Jedná se tak o asymetrickou úlohu obchodního cestujícího, která má nesymetrickou matici nákladů C (v grafickém vyjádření jde o orientovaný graf). Existuje pak tedy různé ohodnocení cest mezi dvěma vrcholy ($c_{ij} \neq c_{ij}$) [17].

V následující kapitole se práce bude zabývat jakým způsobem v takovém grafu vyhledat optimální trasu průchodu všemi uzly za pomoci matice vzdáleností.

3.3 Genetické algoritmy

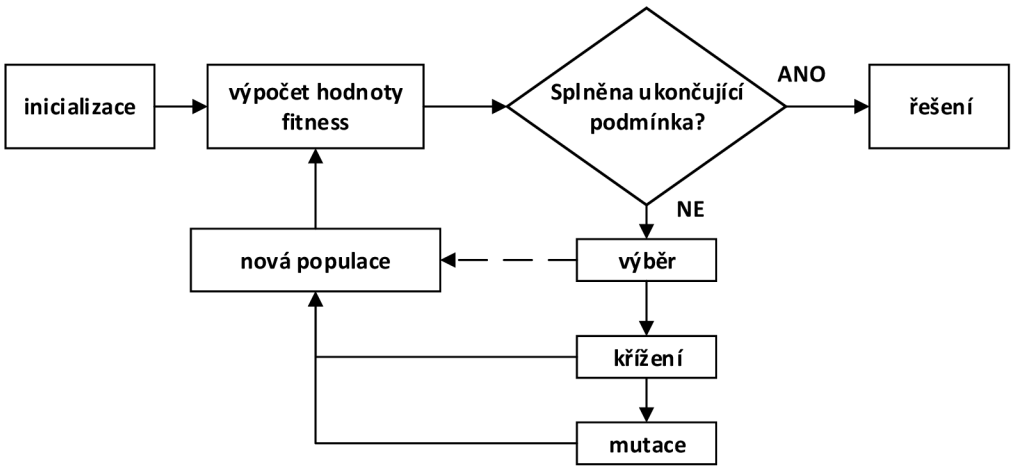
Vznik genetických algoritmů se datuje do 60. let dvacátého století. Základní myšlenky těchto algoritmů vycházejí z Darwinovy teorie o vývoji druhů. Od doby, kdy zakladatel této vědní disciplíny John Holland popsal základní principy a vlastnosti těchto algoritmů, prošel obor dlouholetým vývojem a množství praktických nasazení ukázalo jeho smysluplnost.
Genetické algoritmy jsou metodou pro matematickou optimalizaci nejen NP-složitých úloh, jakou je např. úloha obchodního cestujícího, ale nachází uplatnění i v mnoha technických oborech (např. optimální nastavení leteckých motorů, atd.) [18].

3.3.1 Myšlenka a algoritmus

Základní myšlenka je inspirována evoluční teorií, kdy mají přežívat maximálně odolní a schopní jedinci, kteří předávají své genetické vybavení (geny) další generaci. Analogicky v genetických algoritmech jedinci představují jednotlivá řešení daného problému, které je kódované v tzv. genomu (každý jedinec tedy nese jeden genom). Většinou dané řešení bývá kódované binárně, pro kódování se může ale použít jakákoliv abeceda. Jedinci tvoří tzv. populaci. Populace v daném čase, na kterou jsou aplikovány vývojové procesy, je generace. V rámci populace se měří „síla“ jedince (v tomto případě kvalita řešení) pomocí tzv. fitness funkce. Čím vyšší je hodnota fitness daného jedince, tím vyšší má šanci na přežití či možnost podílet se na další generaci (v rámci úlohy obchodního cestujícího bude fitness odpovídat hodnotě nalezené trasy) [19] [18].

Vlastní algoritmus by se dal shrnout do následujících bodů a vývojového diagramu 3.2:

1. Inicializace
   (a) t = 0.
   (b) Náhodně vytvoř populaci P(t) velikosti N.
   (c) Pro každého jedince z P(t) urči jeho hodnotu fitness.
2. Hlavní cyklus
   (a) Dokud není splněna podmínka pro ukončení, dělej
      i. Proveď výběr (selekci) – vyber jedince, kteří se přímo přenesou z P(t) do P(t +1) .
      ii. Proveď křížení.
         A. Vyber z P(t) dvojice k reprodukci.
         B. Aplikuj na dvojici křížení.
         C. Potomky zařaď do P(t +1) .
      iii. Proveď mutaci.
         A. Vyber z P(t +1) jedince k mutaci.
         B. Aplikuj mutaci na každého jedince.
         C. Zmutovaného jedince zařaď do P(t +1) .
      iv. t = t +1.
      v. Pro každého jedince v P(t) vypočti jeho hodnotu fitness.
   (b) Vrať genom jedince s nejvyšší hodnotou fitness.



Obr. 3.2: Vývojový diagram genetického algoritmu

Základní činností genetických algoritmů jsou pak 3 operace:

- Selekce
- Křížení
- Mutace

Tyto operace se vždy v dané generaci aplikují nad celou populací a výsledkem je nová generace. Tento proces se opakuje do té doby, než se v nově vytvořené generaci vyskytne jeden, nebo více jedinců s požadovanými vlastnostmi [19] .

Selekce

Tento první krok slouží k výběru jedinců z populace, kteří se mohou stát rodiči. Metod, jak tento krok provést, je hned několik. Většinou pracujeme s kvalitou jedince, tedy fitness hodnotou jeho chromozómu.

Vážená ruleta je jedna z prvních používaných metod. Každý jedinec dostane na pomyslné ruletě takový podíl, jaký odpovídá jeho fitness hodnotě. Podle procentuálního rozdělení vytvoříme kruh rulety rozdělený na N částí úměrných hodnotám pravděpodobnosti. S takto vytvořenou ruletou pak provádíme losování pro výběr každého dalšího rodiče.

Turnajová metoda vybírá vždy náhodně skupinu jedinců z populace. Tato skupina musí mít vždy minimálně dva jedince, ale můžeme vybírat i větší skupiny. Vítězem turnaje ve skupině se stává jedinec s nejvyšší fitness hodnotou.

Při **ořezávání**, což je další metoda selekce, se všichni jedinci seřadí podle své fitness hodnoty. Tuto řadu pak podle libovolně zvoleného parametru rozdělíme na dvě části. Z té části s nízkými fitness hodnotami nebudeme možné rodiče vybírat vůbec, a z druhé části můžeme vybírat podle jakéhokoliv deterministického či náhodného pravidla.

Náhodný výběr je nejjednodušší metoda selekce, která nijak nezohledňuje kvality jedinců. Jako rodiče jsou náhodně vybíráni jedinci z celé populace [19].

Křížení

Tato operace navazuje na selekci. Stejně jako v přírodě, rodiče si vymění část svého genetického kódu, tedy část chromozómu. Nejjednodušší metoda je jednobodové křížení, kdy se náhodně zvolí bod v chromozómu. Tato hranice rozdělí chromozom na dvě části a ty se mezi potomky vymění.
Uveďme to na příkladu dvou jedinců s těmito chromozómy:

```
X = 1100101|1100
Y = 1001110|0101
```

Křížením vzniknou dva nové chromozómy:

```
P = 11001010101
Q = 10011101100
```

Teď je potřeba jen vybrat, zda necháme do další generace postoupit oba nové jedince, nebo si náhodně vybereme jen jednoho z nich. Mužeme také volit i vícebodové křížení a kód potomka může vznikat různými kombinacemi z více než dvou rodičů. Není ani vyloučeno, že by se mohli do další generace dostat i někteří rodiče a to dokonce aniž by se zúčastnili křížení. Tato možnost se ovšem používá jen velmi malou pravděpodobností [19].

Mutace

Jde o poslední operaci genetického algoritmu. U každého jedince z nové generace procházíme celý chromozóm a s velmi malou pravděpodobností měníme hodnoty některých genů z 0 na 1 a obráceně. Význam mutace je v tom, že se může v dané generaci objevit vlastnost, kterou dosud žádný jedinec neměl a nemohl ji tedy předat potomkům. Smyslem mutace je také vytváření zcela nových jedinců, tedy zvyšování různorodosti populace a prevence uváznutí v lokálním extrému funkce. V některých implementacích se dokonce dynamicky mění poměr podle míry konvergence populace. Pokud se pravděpodobnost mutace nastaví velká, ztrácí se výhoda genetického algoritmu a prohledávání stavového prostoru přechází v prohledávání náhodné. Aplikace mutací dokazuje, že u malých populací by měla být nastavená velká pravděpodobnost mutace a opačně. Dá se to vysvětlit tím, že u malých populací tak mutace napomáhá k prohledání větší části stavového prostoru, zatímco u velkých populací je velikost prohledaného stavového prostoru zaručena velikostí populace.[19].

3.3.2 Výhody a nevýhody genetických algoritmů

Hlavní výhodou genetických algoritmů je velmi široké uplatnění v různých oblastech vědy a výpočtů. Vykazují velmi dobré výsledky u úloh s rozsáhlými množinami přípustných řešení, které jsou jinak výpočetně velmi náročné. Dále nevyžadují speciální znalosti o cílové funkci.
Jejich nevýhodou je, že nenaleznou vždy optimální výsledek, protože mohou uváznout v některém z lokálních optim. Vždy najdou pouze jeden výsledek a vyžadují velké množství vyhodnocování účelové funkce, proto mohou být i pomalé. Někdy vyžadují dlouhé ladění parametrů (jako pravděpodobnosti křížení a mutace, počet jedinců v generaci, atd.), než se dojde k dobrému výsledku. Některé z těchto problémů lze překonat vhodným výběrem algoritmu pro výběr jedinců ke křížení a vhodnou mutací [18].

4 APLIKACE PRO VÝPOČET OPTIMÁLNÍ TRASY

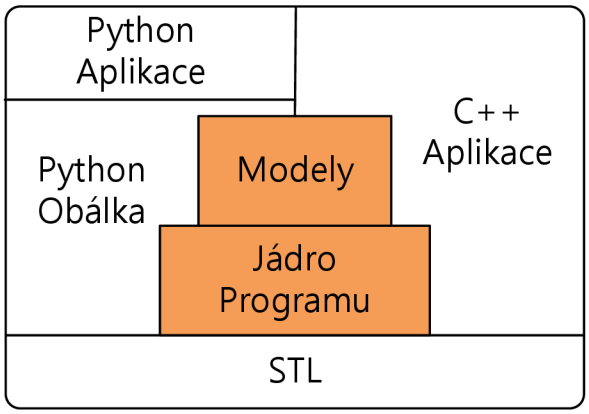
Z počátku byla vyvíjená aplikace založena na implementaci genetického algoritmu do prostředí NS-3, kde měl být definován graf za pomocí modulů které již NS-3 obsahuje, jednotlivá spojení ohodnotit váhou a následně by se vypočítala optimální trasa v takovém grafu a byl by zobrazen výsledek.

4.1 NS-3 Síťový simulátor

NS-3 simulátor sítě je zaměřen především na výzkum a vzdělávací použití a je univerzální simulační nástroj, který může být použit k simulaci různých typů sítí. Jádro NS-3 je sada knihoven, které poskytují široký rámec nástrojů pro vývoj síťových simulací [20].
Kromě toho také balíček NS-3 obsahuje několik modelů běžných síťových protokolů, které můžeme pomocí funkce jednoduše použít v simulacích. Tyto modely nejsou omezené pouze pro síťové protokoly, ale obsahuje například modely pohybu, šíření ztrát v sítí nebo modely spotřeby energie [21].

K získání výsledků simulace existují v modelu simulátoru záchytné systémy a díky nim můžeme poté přistupovat k výsledkům simulace, tato funkce se nazývá trasování. Jádro simulátoru dále také obsahuje logovací záznamy či testovací jednotku, která se používá pro kontrolu stávajícího kódu a detekci chyb. Jednotka také testuje, jestli se model bude chovat očekávaným způsobem [21].

NS-3 je psán výhradně v jazyce C++ (na rozdíl od jeho předchůdce NS-2, který používal kombinaci C++ a OTcl), ale nabízí také možnosti psaní skriptů a simulací v Pythonu. NS-3 je ve své podstatě C++ knihovna, která poskytuje sadu modelů pro síťové simulace, realizované jako C++ objekty zabalené prostřednictvím skriptovacího jazyka Python (obr. 4.1). Umožňuje také běh v reálném čase při připojení k virtuálním jednotkám běžících na skutečných zařízeních. NS-3 má velkou podporu využití pod operačním systémem Linux, ale existují i metody použití pod systémem MS Windows [21].

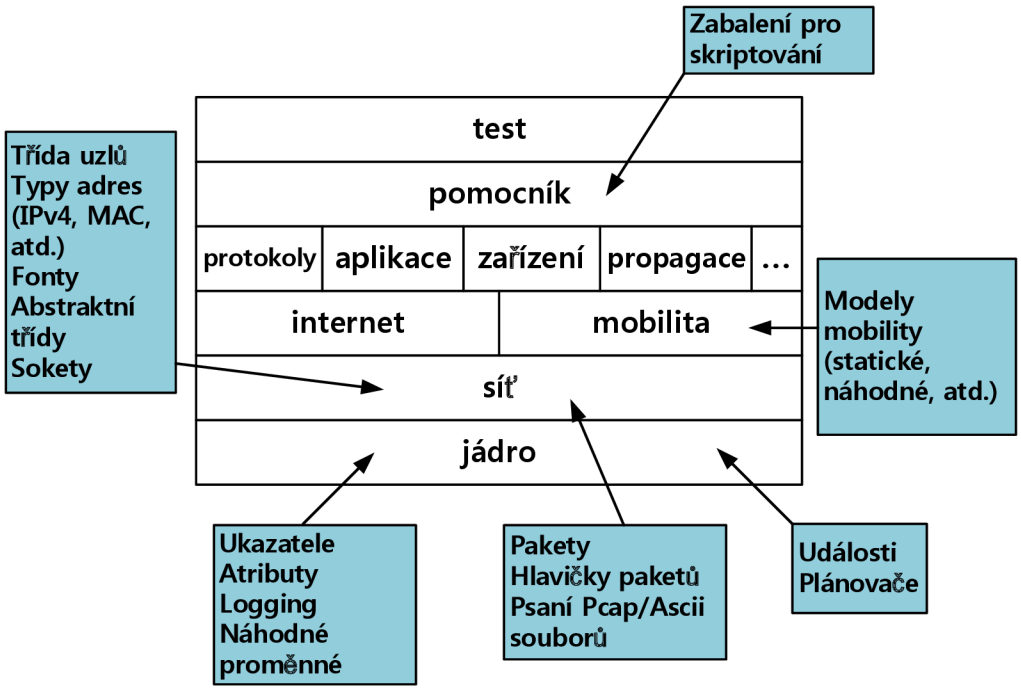


Obr. 4.1: Struktura NS-3.

4.1.1 Výhody NS-3

Ve srovnání s ostatními obdobnými simulátory jako je například OPNET, QualNet, které jsou komerční nebo volně šiřitelné OMNeT++, SSFNet, J-Sim jsou výhody NS-3 následující:

- Ns-3 využívá programovací jazyky C++ nebo Python a umožňuje jejich plnou podporu. Některé simulátory totiž používají pouze své specifické jazyky bez možnosti rozšíření.
- V NS-3 může uživatel využít tzv. pomocníky (helpers), kteří umožňují jednodušší používání některých funkcí. Uživatel má pak na výběr jestli dá přednost jednodušší práci s pomocníky nebo bude tvořit simulaci na nízké úrovni rozhraní, kde lze nastavit více parametrů apod.
- Simulační události jsou jednoduše vytvořené události a ty je možné přesně naplánovat, kdy se mají uskutečnit.
- Dalším rozdílem oproti jiným simulátorům je fakt, že NS-3 nemá vlastní vývojové prostředí. To umožňuje uživatelům využívat například příkazový řádek nebo implementovat nástroje např. pro vizualizace nebo generování výstupu ze simulace. Je tedy pouze na uživateli, jaké vývojové prostředí si zvolí.



Obr. 4.2: Programová organizace NS-3.

Při vlastním vývoji praktického řešení výpočtu optimální trasy v grafu s použitím NS-3 se postupem času ukázalo jako značně nevýhodné, jelikož nebylo možné ve vnitřních modulech NS-3, které jsou navíc spolu provázané, definovat například vzdálenosti mezi jednotlivými uzly. Například při použití modulu point-to-point není žádný atribut nebo proměnná, která by dokázala takové ohodnocení reprezentovat, stejný problém se vyskytuje i u dalších modulů, které slouží k propojení jednotlivých uzlů, ať už jsou to koncová zařízení, nebo routery. Kvůli provázanosti modulů s nástroji pro zobrazení NS-3 jako je například NetAnim nebo Python Visualizer také nebylo možné alespoň zobrazit vytvořený model grafu a sledovat vývoj počítané trasy nebo výslednou trasu pouze zobrazit, jelikož tyto nástroje vyžadují předem definovanou strukturu dat a v případě řešeného problému takové struktury nebylo možné dosáhnout.
Samotné definování vstupního grafu a následný výpočet trasy se tak staly problémem, které natolik ztížily vývoj řešené problematiky, že bylo rozhodnuto vydat se vlastní cestou bez využití potencionálních výhod, které NS-3 nabízí. Aplikace je tedy tvořena od začátku jako konzolová aplikace psána v jazyce C/C++. Tak lze bez omezení definovat vstupní data, následně je vhodně upravit pro výpočet optimální trasy svozu.

4.2 O aplikaci

Program je napsán v jazyce C a C++ jako konzolová aplikace ve vývojovém prostředí Code::Blocks verze 13.12. Pro správnou kompilaci kódu je nutné mít v nastavení kompilování: Project»Build options... a v nově otevřeném okně, záložce Compiler settings a v její záložce Compiler Flags zaškrtnutou položku „Have g++ follow the C++11 ISO C++ language standart [-std=c++11]“. V kódu se totiž využívá funkcí z nového standardu jazyka C++.
V následujících kapitolách jsou popsány jednotlivé funkce aplikace a jejich implementace.

Program byl testován a měřen na stolním počítači s konfigurací procesoru Core 2 Duo E8200 pracujícím na taktu 2,66GHz a 4GB pamětí RAM pracujících na taktu 800MHz. Operační systém Windows 7 64-bit.

4.2.1 Spuštění aplikace

Na přiloženém CD, ve složce „svozGA“ je stejnojmenný projektový soubor po jehož otevření se spustí vývojové prostředí Code::Blocks (je-li nainstalováno) a je vidět zdrojový kód programu. Program je již nastaven vhodně pro kompilaci a spuštění klávesou F9. V podsložce „source“ jsou umístěny vstupní SVG soubory.
Program lze také spustit přímo přes spustitelný soubor umístěný na CD ve složce „Aplikace“, kde je opět složka se zdrojovými daty a samotná spustitelná aplikace.
Pro spuštění aplikace je nutné si složky „svozGA“ a „Aplikace“ stáhnout na lokální umístění, jelikož po ukončení běhu jsou vytvořeny dva textové soubory - „result“, kde je vypsána nalezená trasa a její váha a „matrix“, kde je vypsána matice s ohodnocením jednotlivých cest mezi uzly. Soubory se vytvoří ve stejném umístění, kde je spustitelný soubor aplikace.

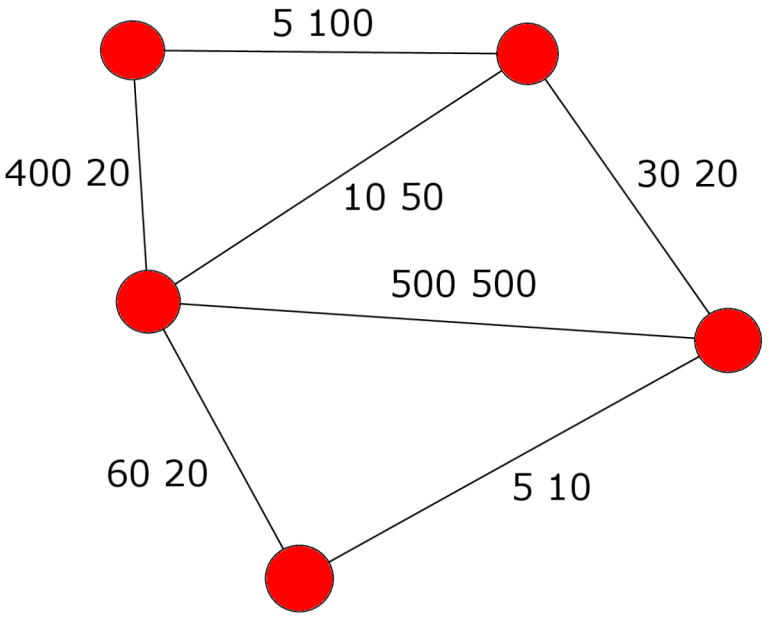
4.3 Vstupní data programu

Vstupními daty do samotné aplikace je vektorový obrázek uložený ve formátu SVG (Scalable Vector Graphics). SVG je zároveň i značkovací jazyk, který definuje dvojrozměrnou vektorovou grafiku pomocí XML. Výhodou takového formátu je možnost pozdějšího zpracování takového souboru pomocí XML parseru, který ze struktury souboru dokáže vybrat důležitá data a informace z jednotlivých tagů a jeho atributů a uložit je do proměnných v aplikaci.

4.3.1 Tvorba vstupního SVG souboru

Vstupní SVG soubor je tvořen ve vektorovém grafickém editoru Inkscape verze 0.91, která používá SVG jako svůj nativní formát, umožňuje přímou editaci XML a je tak vhodná pro tvorbu vstupního souboru pro vytvářenou aplikaci.
Po spuštění editoru je vidět plátno na které přidáme objekty nástrojem „Tvorba kruhů, elips a oblouků“, které budou představovat jednotlivé uzly (kontejnery). Jednotlivým uzlům je nutné upravit jejich čísla ID, aby poté souhlasily a reprezentovaly index v matici vzdáleností. Klávesovou zkratkou Shift+Ctrl+O zobrazíme okno s vlastnostmi objektu. Uzly tedy definujeme číselně od nuly vzestupně až po n-počet uzlů.
Všechny jednotlivé uzly je poté nutné spojit nástrojem „Vytvořit diagramové spojky“, spoj je nutné vytvořit přesně od středu do středu uzlu, kde se nabídne bílý čtverec, dojde tak k provázání uzlů a hran, kde hrana pak ve svých atributech *inkscape:connection-start* a *inkscape:connection-end* má uložena ID čísla počátečního a konečného uzlu. Takové spojení pak reprezentuje hranu grafu. Tyto hrany je pak nutné nakonec ohodnotit, jelikož ohodnocení hrany může být z bodu A do bodu B odlišné jako z bodu B do bodu A, je tedy důležité definovat obě váhy dané hrany. Klávesovou zkratkou Shift+Ctrl+X zobrazíme XML editor a označíme hranu kterou chceme ohodnotit.
V XML editoru pak vyplníme pole „Název atributu“ textem *dist* a do pole „Hodnota atributu“ atributu napíšeme ohodnocení hrany ve formátu „*xx yy*“, kde „*xx*“ je váha hrany směrem od počátečního uzlu do konečného a „*yy*“ z konečného do počátečního. Počet cifer není omezen, v aplikaci bylo používáno 2-4 ciferná čísla pro ohodnocení. Zápis pak může vypadat například „*60 420*“.
Co se týče tvorby grafu, tak je důležité dávat si pozor aby byly všechny uzly napojené alespoň na jednu hranu a přesně od středu do středu uzlu, jinak nedojde ke správnému propojení uzlů v XML struktuře.

Graf může tedy vypadat následovně (doplněno o textové pole s popisky ohodnocení jednotlivých hran):



Obr. 4.3: Příklad vytvořeného grafu.

Číselná velikost ohodnocení hran je závislá na několika faktorech jako je vzdálenost mezi uzly, hustota provozu na dané trase, sjízdnost trasy apod., jedná se tedy o relativní váhu a dá se podle ní vyjádřit jak velké jsou náklady na průjezd takovou hranou. Nižší hodnota váhy tedy znamená nižší náklady a tedy lepší volbu pro zvolení cesty.

Struktura vstupního SVG souboru. Soubor lze otevřít v textovém editoru jako je například PSPad. Vypsány jsou jen hlavní značky a atributy které obsahují informace důležité pro potřebu programu:

```
<ellipse
style="fill:#ff0000;fill-rule:evenodd;stroke:#000000;..."
  id="0"
  cx="109.72222"
  cy="88.010353"
  rx="31.944445"
  ry="30.092592"
  inkscape:label="#path3603" />
<path
  style="fill:none;fill-rule:evenodd;stroke:#000000;..."
  d="M 141.6621,88.490251 355.09717,91.69712"
  id="path3649"
  inkscape:connector-type="polyline"
  inkscape:connector-curvature="3"
  inkscape:connection-start="#0"
  inkscape:connection-end="#1"
  dist="5 100" />
```

Ve značce *ellipse* nejsou pro potřeby programu žádné data potřebné pro výpočet trasy, avšak je nutné zjistit požet uzlů. Ve značce *path* jsou informace o ID uzlu ve kterém daná hrana začíná a ID uzlu ve kterém končí, dále v atributu *dist* jsou vidět definované ohodnocení hrany v obou směrech.

4.4 Parsování vstupního SVG souboru

Pro parsování se v aplikaci využívá C++ knihovna TinyXML-2, která dokáže jednoduše parsovat XML soubory a vytvoří objektový model dokumentu, ze kterého lze následně data číst, modifikovat nebo je zapisovat. Knihovna byla stažena z internetových stránek "https://github.com/leethomason/tinyxml2", kde je také dostupná online dokumentace ke knihovně.

```
XMLDocument xmlDoc;
XMLError eResult = xmlDoc.LoadFile(filepath);
XMLCheckResult(eResult);

XMLElement * pRoot = xmlDoc.FirstChildElement("svg");
if (pRoot == nullptr) return XML_ERROR_FILE_READ_ERROR;
XMLElement * pElementG = pRoot->FirstChildElement("g");
if (pElementG == nullptr) return XML_ERROR_PARSING_ELEMENT;
XMLElement * pElementPath = pElementG->FirstChildElement("path");
if (pElementPath == nullptr) return XML_ERROR_PARSING_ELEMENT;
```

Po spuštění aplikace se v hlavní funkci *main* volá jako první funkce pro parsování potřebných dat z SVG souboru *load_XML*. Výše uvedeným kódem tato funkce načte vstupní soubor a následně pomocí funkcí z knihovny se postupně dostane přes celou strukturu až ke značkám, které obsahují užitečné data, konkrétně do značky *path*. Kód také ošetřuje je-li struktura XML taková jaká by měla být a z jaké chceme data parsovat. V případě nesprávné struktury souboru nebo jeho špatného načtení dojde k vypsání chybové hlášky pomocí makra.

```
const char * szAttributeText = nullptr;
while (pElementPath != nullptr)
{
   szAttributeText = pElementPath->Attribute("...connection-start");
   if (szAttributeText == nullptr) return XML_ERROR...;
   string str = szAttributeText;
   int strStart = atoi(str.substr(1).c_str());
```

Následně se projdou cyklem *while* všechny značky *path* přičemž v každém průchodu se uloží záznam o ID čísle počátečního uzlu. Jelikož jsou data uložená jako řetězce, je nutné je převést na celá čísla funkcí *atoi*.

```
    szAttributeText = pElementPath->Attribute("...connection-end");
    if (szAttributeText == nullptr) return XML_ERROR...;
    str = szAttributeText;
    int strEnd = atoi(str.substr(1).c_str());
```

Zde je vyparsován údaj o konečném uzlu hrany a opět je řetězec převeden na celé číslo.

```
    szAttributeText = pElementPath->Attribute("dist");
    if (szAttributeText == nullptr) return XML_ERROR...;
    string strDist = szAttributeText;
    vector<string> x = split(strDist, ' ');
    a[strStart][strEnd] = atoi(x[0].c_str());
    a[strEnd][strStart] = atoi(x[1].c_str());
    pElementPath = pElementPath->NextSiblingElement("path");
}
```

Z atributu *dist* vyparsujeme řetězec například *„60 420“*, který následně pomocí mezery funkcí *split* oddělíme a převedeme na celá čísla 60 a 420. Ohodnocení hrany je pak zapsáno do dvojrozměrného vektoru do pozic podle ID čísel uzlů.

Podobně je napsána funkce *nodeCount*, která také prochází strukturu souboru, ale hledá značky *ellipse* a z počtu nalezených takových značek vrátí údaj o celkovém počtu uzlů. Tento údaj je pak využíván při samotném vyhledávání optimální trasy.

Jakmile jsou všechny data ze souboru vyparsována, dostaneme tedy počáteční matici vzdáleností mezi uzly které jsou spolu spojené hranou. Pro dopočítání všech vzdáleností mezi všemi dvojicemi uzlů se využije Floyd-Warshallova algortimu. Poté je matice vzdáleností kompletní a připravena pro vstup do hlavního těla výpočtu genetického algoritmu.

4.5 Genetický algoritmus pro výpočet optimální trasy v grafu

Na teoretických základech byl sestaven genetický algoritmus vhodný pro hledání optimální trasy v grafu. Základem je níže uvedená vytvořená struktura, která definuje za pomocí vektoru posloupnosti uzlů a k tomu je svázaná odpovídající celková váha této trasy.

```
typedef struct {
vector<int> nodes; // vektor s uzly, kterymi se prochazi
int length; // celkova vaha cesty
} Path;
```

Inicializace algoritmu začíná vygenerováním jedné trasy, která reprezentuje posloupnost uzlů od 0 vzestupně po n (poslední nalezené ID číslo uzlu). Z této jediné trasy je následně vygenerován takový počet cest, kterou udává proměnná *PATHS*, což je zároveň velikost populace, se kterou bude algoritmus počítat. Posloupnost uzlů ve vektoru je v takto generovaných cestách náhodně zamíchána a každé cestě je spočítána její celková váha což je ekvivalent pro fitness hodnotu. Takto je vytvořena prvotní generace cest se kterou algortimus pracuje. Uzel na nultém uzlu je z náhodného zamíchání a pozdějších mutací a křížení vynechán, jelikož tento uzel reprezentuje svážecí depo ze kterého vůz vyjíždí a zase se do něj po projetí trasy vrací.
Příklad vygenerované cesty:

```
path = 0 1 2 3 4 >> 0 4 1 3 2
```

Nyní již následuje hlavní cyklus genetického algoritmu, ve kterém je nejdříve celý vektor vygenerovaných cest seřazen podle vypočtené celkové váhy trasy od nejmenší po největší. Takové seřazení při velkém počtu populace (více jak 1000) má vliv na časovou náročnost algoritmu, tuto operaci však není možné nijak urychlit či optimalizovat, jelikož je nezbytná pro funkčnost navrženého algoritmu.

Po seřazení dojde k selekci cest, které se budou dále křížit a mutovat, aby vytvořily novou generaci. Selekce byla zvolena metodou ořezávání a to tak, že se z vektoru odstraní předem definované procento nejhorších cest a se zbytkem nejlepších cest se pak pracuje dále.

Jak velké procento cest, které se mají při selekci odstranit, bude předmětem měření. Je však jasné, že bude-li procento malé, počet nově generovaných cest pak nebude tak velký a může trvat větší počet iterací, než dojde k vygenerování lepší cesty. Naopak bude-li procento odstraněných cest velké, bude pak méně cest které se mohou zkřížit, dojde také k nárůstu časové náročnosti algoritmu, protože je nutné vygenerovat více cest.
V dalším kroku následuje generování nových potomků pomocí křížení a mutací. Pro křížení je nastaven maximální poměr proměnnou *CROSSOVER_MAX*. Z té pak vychází kolik cest se nově nalezne pomocí křížení a kolik pomocí mutace. Bude-li populace cest nastavena na 100, poměr zahazovaných cest 0,8 (80% cest se odstraní v selekci) a maximální poměr křížení bude 0,5 (50%), pak po odstranění zbude 20 cest, ze kterých se musí nově vytvořit dalších 80 cest, aby byla znovu vytvořena populace o 100 cestách a z těchto nových 80 cest bude maximálně 40 vytvořených křížením a minimálně 40 mutací.
Maximální počet cest křížením proto, protože během běhu algoritmu může docházet k vytváření duplicitních tras a takové trasy je nevhodné křížit, jelikož z nich nezískáme trasu lepší, ale opět se vytvoří duplikát a navíc se zvýší časové nároky na nalezení optimální trasy. Duplicitních tras může být pak natolik, že například z maxima 40 cest bude možné křížením v dané iteraci vytvořit pouze 30 a zbývajících 10 se dorovná křížením. Poměr mezi cestami vytvořenými křížením a mutací tak může být v určitých případech plovoucí a umožňuje algoritmu jistou pružnost.

Jako první proběhne křížení, jsou vybrány dvě náhodné cesty z populace která prošla selekcí a pokud tyto cesty nejsou duplicitní, tak jsou předány do funkce *crossover* kde se zkříží, je vypočítána celková váha nově nalezené trasy a výsledek je zapsán do vektoru cest. Po navršení maximálního počtu cest křížením se přejde na mutace. Výběr cesty pro mutace je poměrně jednoduchý, začíná se od nejlepší prozatím nalezené trasy. Průběžně tak mutují cesty vzestupně jdoucí podle váhy dokud není aktuální počet cest v populaci stejný jako při inicializaci.

Takovým způsobem je pak každá generace zpracovávána, dokud není splněna podmínka pro ukončení hlavního cyklu algoritmu. Podmínkou pro ukočení je zvoleno porovnání aktuálního nalezeného výsledku oproti nejlepšímu výsledku o daný počet generací zpět. Nezmění-li se nejlepší nalezená trasa za posledních například 200 generací, bude se nalezená trasa považovat za optimální a dojde k zobrazení její váhy a také k výpisu nalezené posloupnosti uzlů jak do konzole, tak do souboru „result.txt“, který lze poté najít v adresáři s projektem.

Algoritmus se dá vyjádřit těmito body:

1. Inicializace
   (a) Načti data z SVG souboru.
   (b) Dopočítej nejkratší vzdálenosti pro všechny uzly pomocí Floyd-Warshallova algoritmu.
   (c) Vytvoř prvotní cestu jdoucí z nuly vzestupně do celkového počtu uzlů.
   (d) Z prvotní cesty vytvoř definovaný počet cest s náhodně promíchanými posloupnostmi uzlů.
   (e) Pro každou trasu urči její celkovou váhu.
2. Hlavní cyklus
   (a) Pokud se trasa za daný počet generací zlepšila, proveď:
      i. Seřaď trasy vzestupně od nejmenší nalezené celkové váhy
      ii. Proveď selekci – smaž určité procento cest.
      iii. Proveď křížení.
         A. Vyber z vektoru náhodné dvojice k reprodukci a otestuj, zda-li nejsou shodné.
         B. Aplikuj na dvojici křížení.
         C. Potomky zařaď do vektoru.
      iv. Proveď mutaci.
         A. Vyber z generace jedince k mutaci.
         B. Aplikuj mutaci na každého jedince.
         C. Zmutovaného jedince zařaď do vektoru.
      v. Srovnej aktuální váhu nalezené trasy s váhou o daný počet iterací zpět.
   (b) Finálně seřaď cesty vzestupně podle celkové váhy.
   (c) Vypiš nejlepší nalezenou trasu a její váhu.

4.5.1 Křížení

Funkce pro křížení - *crossover* byla nejdříve navržena tak, že docházelo k porovnávání každé dostupné cesty se všemi ostatními a hledala se první shoda v čísle ID uzlu na stejném indexu (kromě nuly). Pokus se našly dvě takové trasy, tak došlo k zápisu první trasy *p1* do trasy výsledné od začátku po index včetně něj, kde byla nalezená shoda, poté se přeskočilo na druhou trasu *p2* a zapisovaly se uzly co následovaly po indexu až do konce trasy *p2*. Pokud však byl při zápisu nalezen uzel, který již byl obsažen v cestě *p1*, tak se kombinace těchto cest považovala jako neplatná a začala se hledat nová dvojice cest. Zde znázorněno příkladem jaké možnosti mohly nastat:

```
p1 = 0 2 4 3 5 1
p2 = 0 1 3 4 2 5
```

V tomto případě nemohlo dojít ke křížení, jelikož neexistuje na žádném indexu shoda v číslech uzlů.

```
p1 = 0 4 2 3 1 5
p2 = 0 3 2 1 4 5
pR = 0 4 2 1 4 5 - ERR
```

V tomto případě z počátku docházelo ke křížení, jelikož byla nalezena shoda na druhém indexu (indexy jsou číslovány od nuly), avšak po přeskočení na *p2* bylo zjištěno, že již obsahuje uzel číslo 4, který už výsledná cesta *pR* obsahuje, takže cesta byla označena jako neplatná a nebyla zapsána do vektoru cest.

```
p1 = 0 1 4 2 3 5
p2 = 0 4 1 2 5 3
pR = 0 1 4 2 5 3
```

Zde jsou splněny všechny podmínky pro tento systém křížení, je vypočítána celková váha trasy a výsledná trasa *pR* je zapsána do vektoru cest.

Takový návrh křížení se při měření ukázal jako velmi neefektivní a pomalý, jelikož obsahoval cykly vnořené do sebe a to klade velké výpočetní nároky. Navíc pro větší počet uzlů se rapidně snižovala pravděpodobnost, že by se vůbec ke křížení mohlo dojít a našla se shoda na nějakém indexu a zároveň uzly z druhé cesty nebyly obsaženy v cestě první. Proto byla navržena nová metoda křížení, která je vhodná pro jakoukoliv dvojici uzlů a její výpočetní náročnost není tak vysoká.

Optimalizovaná metoda křížení

Tato metoda, která je používaná také při měření a porovnávání nastavení parametrů algoritmu, je založena na náhodném vygenerování indexu (kromě nuly a dvou posledních indexů ve vektoru) na kterém dojde ke křížení mezi cestami. Po vygenerování indexu je do výsledné trasy opsána cesta *p1* od jejího začátku až po index (včetně), poté se přeskočí na cestu *p2* a začne se opisovat na indexu v *p2*, který obsahuje stejné číslo ID uzlu jako ve kterém skončilo opisování *p1*. Cesta *p2* se pak opisuje až do jejího konce, pokud je během opisování nalezeno číslo ID uzlu, který již je ve výsledné trase, tak se tento uzel přeskočí a opisuje se dál. Jakmile se narazí na konec cesty *p2*, tak se začne na cestě *p2* od nultého indexu a zapíší se zbývající uzly, které ještě do výsledné trasy zapsány nebyly.
Tuto metodu lze popsat následujícími body:

1. Vygeneruj na jakém indexu dojde k přeskoku.
2. Okopíruj *p1* po vygenerovaný index.
3. Okopíruj do výsledku uzly z *p2*, které začínají na čísle ID uzlu z vygenerovaného indexu a uzly se nevyskytovaly v *p1*.
4. Okopíruj do výsledku uzly z *p2*, se ještě ve výsledné trase nevyskytovaly.
5. Vypočítej celkovou váhu výsledné zkřížené trasy.

Příklad možnosti křížení:

```
vygenerovany index - 2
p1 = 0 1 5 2 3 4
p2 = 0 5 1 3 4 2
pR = 0 1 5 3 4 2
```

Tato metoda již neobsahuje vnořené cykly a je tak mnohem rychlejší, také se odstranil problém s nalezením vhodných dvojic. Metoda také zvyšuje šanci na nalezení lepšího výsledku, jelikož v ní dochází k většímu počtu změn. Mohlo by také být snadnější pro algoritmus dostat se mimo nalezené lokální optimum.

4.5.2 Mutace

Oproti křížení je mutace poměrně jednoduchá operace, ve které dojde na jedné trase k náhodném prohození dvou uzlů (kromě nulového). Nejdříve se vygenerují náhodné indexy uzlů které se následně přemístí jeden za druhého. Výhodou této operace je fakt, že je velmi málo náročná na výpočetní výkon a může se tak provést několikrát za stejný čas, který potřebuje například jedna operace křížení. Po zmutování je proveden výpočet celkové váhy zmutované trasy a je zapsána do vektoru cest.
Příklad mutace na dané cestě:

```
p1 = 0 1 5 2 3 4
pR = 0 4 5 2 3 1
```

V následující kapitole bylo prokázáno, že mutace skutečně ovlivňují schopnost algoritmu uniknout z lokálního extrému funkce a je tak nejúčinnějším způsobem jak vylepšit celkovou nalezenou váhu trasy. Křížení je tedy efektivní při počátečních fázích algoritmu, kdy dochází ke skokovým změnám směrem k nalezení optimální trasy, avšak čím více se algoritmus blíží k nalezení optimálního výsledku, tím více převažuje síla mutací nad křížením, kdy je více pravděpodobnější že se váha trasy vylepší mutací než křížením.

5 PARAMETRIZACE GENETICKÉHO ALGORITMU

5.1 Výchozí nastavení parametrů algoritmu

Měření budou probíhat s následující konfigurací parametrů genetického algoritmu a vstupním souborem, nebude-li uvedeno jinak:

Velikost populace, proměnná *PATHS* - 5000
Maximální poměr křížení, proměnná *CROSSOVER_MAX* - 0,4 (40%)
Poměr zahozených cest - 0,4 (40%)
Porovnávací rozsah ukončení běhu algoritmu, proměnná *condition* - 500
Vstupní SVG soubor - „graph80.svg“

Tyto parametry vycházely po řádném měření jako nejoptimálnější a proto byly zvoleny jako výchozí hodnoty se kterými se měří jednotlivé závislosti genetického algoritmu na jednotlivě nastavených parametrech. Při měření je jako výchozí SVG soubor zvolen graf s 80 uzly, které jsou spojeny v mřížce, reprezentuje tak mapu menší městské periferie. Při průběhu měření a testování byl na této mapě nalezen nejlepší výsledek o celkové váze 2380, tento údaj se bude považovat jako optimální nalezená trasa v daném grafu.

Měření probíhá pro každé nastavení jednotlivého parametru desetkrát a výsledky se zprůměrují, nalezne se tak přibližná hodnota výsledku při daném nastavení, jelikož výsledky se při každém běhu programu mohou lišit až o 20%. Tabulkové hodnoty *Max váha* a *Min váha* reprezentují největší a nejmenší naměřenou váhu trasy v daném nastavení při měření.

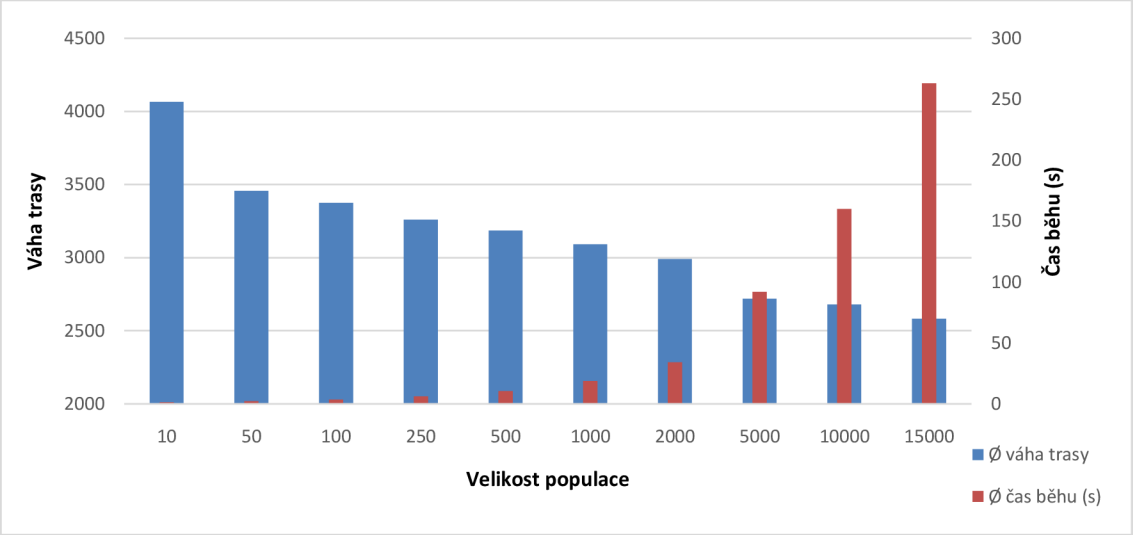
5.2 Vliv velikosti populace

Velikost populace, se kterou bude algoritmus počítat, v tomto případě počet cest, je jedním z jeho hlavních parametrů. Se zvyšujícím se počtem uzlů a tím také počtem hran dramaticky roste také počet kombinací cest kterými lze graf projít. Při měření a testování aplikace bylo zjištěno, že pro malý počet uzlů (<10) není nutné volit velkou populaci, protože populace pak může přesáhnout velikost všech možných cest a již při vygenerování první generace dojde k nalezení nejlepší cesty a velká část generace pak budou duplikáty již vygenerovaných cest. Pro malý počet uzlů je tak vytvořený genetický algoritmus nevhodný, né však z důvodu jeho výkonnosti, ale z časových nároků pro vytvoření takového genetického algoritmu. Pro nízký počet uzlů lze využít jiné metody výpočtu, například výpočet hrubou silou, který navíc vždy nalezne optimální řešení.
Avšak například pro měřených 80 uzlů je vhodná velikost populace už poměrně velká aby byla nalezena co nejoptimálnější trasa.

Z naměřených hodnot v tabulce 5.1 a grafu 5.1 je jasně patrné, že velikost populace má velký vliv na nalezení optimální trasy, avšak s rostoucí velikostí roste také exponenciálně čas potřebný k nalezení optimální trasy. Při použití malé populace je při možném počtu kombinací cest velmi nepravděpodobné, že křížením nebo mutací se v dané generaci nalezne lepší cesta, a to ani po velkém počtu generací, protože algoritmus nemá dostatečný prostor k vyváznutí z lokálního maxima.
Od velikosti populace 5000 je vidět že nalezený výsledek se s dále rostoucí populací již mění velmi málo k lepšímu a časová náročnost není nijak vysoká, budeme jej proto považovat za optimální pro danou velikost grafu. Zajímavostí je také sloupec s počtem iterací, kterými algoritmus projde než nalezne své řešení. S rostoucí populací se k optimálnímu výsledku dostane při méně iteracích, jelikož v rámci jedné iterace je zpracováváno a generováno velké množství tras, a je zde větší pravděpodobnost že cesta, která zmutuje a nebo se zkříží, bude mít lepší výslednou váhu trasy. Z těchto naměřených údajů pak lze zjistit jaký počet generací s danou velikostí populace a daný graf aby bylo nalezeno optimálního řešení. Počet iterací pak lze definovat ručně a tím se zkrátí čas potřebný k nalezení výsledku, jelikož pak algoritmus nebude projíždět ten počet iterací, po kterém se ukončí pokud nedojde ke změně ve váze nalezené trasy.

Velikost populace	Max váha	Min váha	Ø váha trasy	Ø čas běhu (s)	Počet iterací
10	4525	3645	4065	1,3	1683
50	3875	3025	3456	2,2	1694
100	3800	3020	3375	3,4	1352
250	3625	2935	3259	6,1	902
500	3435	2900	3185	10,5	684
1000	3420	2825	3091	18,8	484
2000	3305	2575	2990	34	343
5000	3025	2415	2720	92	418
10000	2970	2415	2680	160	268
15000	2730	2440	2583	263	334

Tab. 5.1: Tabulka závislosti naměřených hodnot na velikosti populace



Obr. 5.1: Graf váhové a časové závislosti na velikosti populace

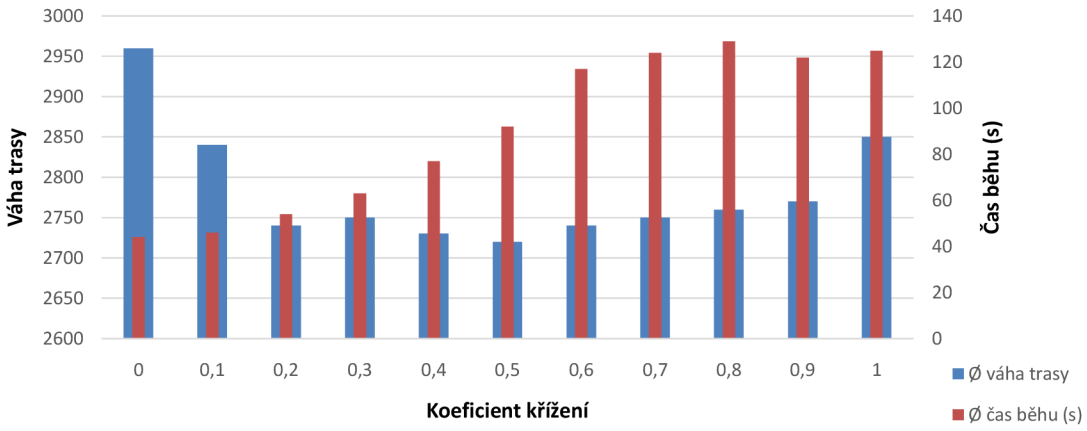
5.3 Vliv poměru křížení a mutací

Křížení a mutace jsou hlavní funkcí genetického algoritmu a zajišťují v každé iteraci generování nových potomků do další generace. V křížení dochází k celkem výrazným změnám v posloupnosti uzlů na trase, proto má velký potenciál nalézt vhodnější trasu která bude výrazně lepší než nejlepší doposud nalezená trasa. Hlavní slovo tak bude mít na začátku běhu algoritmu, kde se tak rychleji přiblíží hledanému optimu. Naopak mutace zajišťuje jen drobné změny na trase, jelikož prohazuje jen dva uzly mezi sebou. Mutace tak do jisté míry zajišťují, že algoritmus neuvázne v některém z lokálních optim. Jelikož křížení je oproti mutaci časově náročnější na výpočet, je tedy i vzhledem k nalezené nejlepší trase nutné nalézt vhodný poměr generování za pomoci mutace a za pomoci křížení.

Ze zobrazeného grafu 5.2 je vidět důkaz, že se zvyšujícím se poměrem křížení roste časová náročnost běhu algoritmu. Optimální nastavení koeficientu je tedy mezi hodnotami 0,2 - 0,5, kde nastavení vykazuje nejlepší výsledky nalezené trasy a časová náročnost není nijak vysoká. Při malé velikosti populace také může docházet k případům kdy nastavený koeficient může být například 0,5 (50%), avšak kvůli duplicitám bude možné křížením vygenerovat v iteraci pouze maximálně 35% a zbytek se dogeneruje za pomoci mutace. Poměr tedy není pevný, považuje se jen za horní hranici pro křížení. Poměr se tedy přizpůsobí aktuální situaci.
Z grafu a naměřených hodnot v tabulce 5.2 je také patrné, že generování nových cest pouze mutací není výhodné, jelikož mutace v trase provádí jen velmi malé změny. Trvá pak velký počet iterací než se přiblíží k nějakému optimu a výsledek nebude takový jaký se očekává. Naopak při nastavení kdy se má využít pouze křížení, tak systém sice za malý počet iterací se přiblíží k optimálnímu řešení, avšak kvůli absenci mutací nedojde už k výraznému zlepšení, jelikož při lokálním extrému je pro křížení obtížné se z něj dostat, protože změny ke kterým během křížení dochází jsou velké a je tak malá pravděpodobnost že dojde ke zlepšení výsledné trasy.

Koeficient křížení	Max váha	Min váha	Ø váha trasy	Ø čas běhu (s)	Počet iterací
0	3115	2820	2960	44	1057
0,1	3140	2580	2840	46	574
0,2	2945	2600	2740	54	443
0,3	3050	2560	2750	63	411
0,4	2960	2545	2730	77	381
0,5	3025	2470	2720	92	418
0,6	3020	2440	2740	117	493
0,7	3040	2510	2750	124	425
0,8	2955	2540	2760	129	356
0,9	2875	2665	2770	122	306
1	2970	2775	2850	125	216

Tab. 5.2: Tabulka závislosti naměřených hodnot na koeficientu křížení



Obr. 5.2: Graf váhové a časové závislosti na koeficientu křížení

5.4 Vliv velikosti zahazování populace

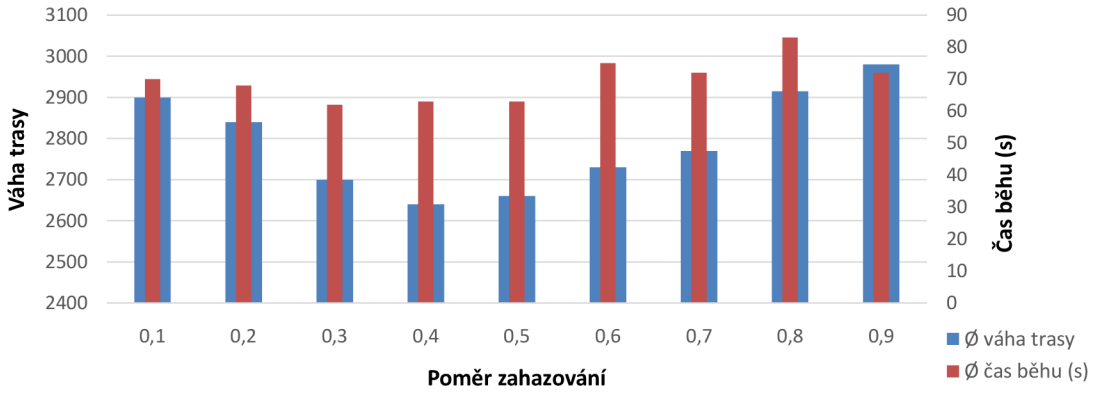
Při každé iteraci dojde na začátku k selekci cest, které postoupí do další fáze algoritmu. Selekce se provádí metodou ořezávání, kde určitá část dané populace se odstraní a zbytek přežije. Poměr odstraněných cest pak má vliv na výkonnost genetického algoritmu.

Z naměřených hodnot v tabulce 5.3 a grafu 5.3 lze posoudit, že zvolit malé procento zahazovaných cest, tedy méně jak 20%, vede ke zhoršení nalezeného výsledku, jelikož v generaci se vytvoří jen malý počet cest oproti zachovanému zbytku a algoritmus pak nemá dostatek prostoru zlepšit prozatímní nalezené optimální řešení. A také naopak při zahazování velké části generace, tedy více jak 80%, má za následek horší výsledky, jelikož není dostatečný rozsah cest pro mutaci a křížení ze kterého

se vytváří generace nová. Také může docházet k tomu, že zahodíme potenciálně dobrou trasu, která by po mutaci nebo křížení mohla výrazně snížit svou celkovou váhu trasy. Při zahazování velkého procenta cest také dojde k mírnému zvýšení časové náročnosti algoritmu, protože je náročnější vytvořit pro další generaci například 8000 nových cest než 2000. Jako optimální hodnota se volí 0,4 - 0,5, kde má algoritmus dostatek prostoru pro generování cest nových a zároveň má k dispozici dostatečný počet nejlepších cest z předchozí generace.

Poměr zahazování	Max váha	Min váha	Ø váha trasy	Ø čas běhu (s)	Počet iterací
0,1	2995	2745	2900	70	1945
0,2	3035	2585	2840	68	1296
0,3	2830	2535	2700	62	855
0,4	2960	2480	2640	63	672
0,5	2905	2460	2660	63	486
0,6	2800	2605	2730	75	471
0,7	3085	2550	2770	72	345
0,8	3055	2600	2915	83	257
0,9	3190	2700	2980	72	170

Tab. 5.3: Tabulka závislosti naměřených hodnot na poměru zahazování populace



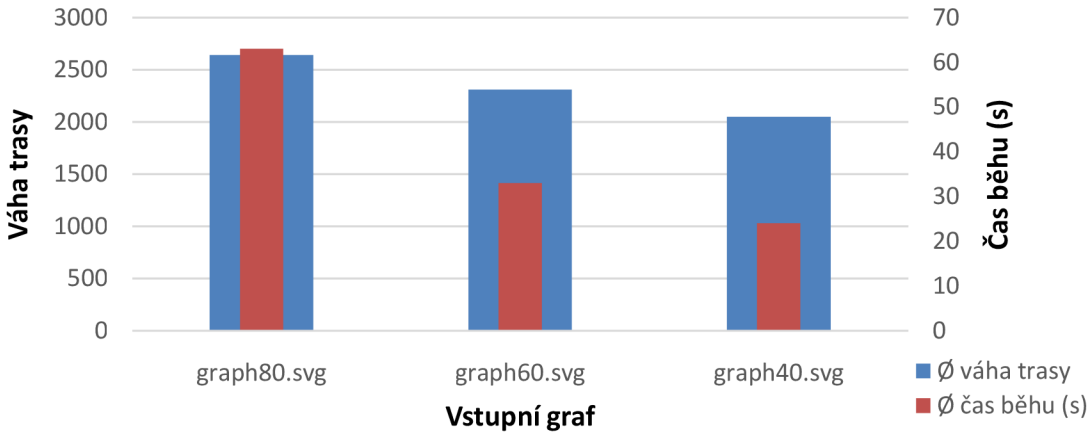
Obr. 5.3: Graf váhové a časové závislosti na poměru zahazování populace

5.5 Měření úspory nákladů svozu odpadu

Pro porovnání celkové váhy trasy byly vytvořeny dva grafy, ze kterého bylo z původního souboru „graph80.svg“ odstraněno 20, respektive 40 uzlů. Uzly byly odstraněny nahodile v celém grafu. Uzly se pak spojily zpět náležitě ohodnocenými hranami, takže celistvost a ohodnocení mapy zůstala zachována. Na takovém modelu pak lze odhadnout úspora nákladů na jeden cyklus svozu v dané oblasti pokud budou sváženy jen nutné kontejnery.
Po vypočítání nejlepší nalezené trasy pro všechny scénáře je i v tabulce 5.4 a grafu 5.4 vidět, že i když počet uzlů klesne o čtvrtinu, respektive o polovinu, náklady na svoz v cyklu klesnou jen o přibližně 13%, respektive 24%. To je dáno tím, že vůz stále musí jet i pro nejvzdálenější kontejnery a celková vzdálenost tedy nemůže být poloviční jak by se mohlo na první pohled zdát. Tyto procenta se také mohou měnit podle toho které kontejnery v různých částech grafu je nutné svážet. Úspora nákladu je však zřejmá. Ve skutečnosti by pak jeden svážecí vůz mohl pokrýt větší oblast a pro svoz odpadu v celém městě by nebylo nutné použít takový počet vozů jako bez optimalizace trasy a monitorování kontejnerů, což by mohlo ve výsledku být ještě efektivnější, jelikož by se redukoval vozový park a s ním spojené náklady na údržbu, pojištění, palivo apod.

Vstupní graf	Max váha	Min váha	Ø váha trasy	Ø čas běhu (s)	Počet iterací
graph80.svg	2960	2480	2640	63	672
graph60.svg	2530	2220	2310	33	305
graph40.svg	2365	1960	2050	24	162

Tab. 5.4: Tabulka porovnání vypočtených vah a časové náročnosti na počtu uzlů



Obr. 5.4: Graf váhové a časové závislosti na počtu uzlů v grafu

6 ZÁVĚR

V diplomové práci byla popsána problematika chytrých měst, některých jejich technologií a jejich výhody. Podrobněji byl popsán svoz odpadu v chytrých městech a byly vytyčeny hlavní cíle takového systému. Inteligentní svoz odpadu byl pak porovnán s konvenční metodou. Dále jsou v práci uvedeny používané senzory, které se umisťují v kontejnerech pro odečítání různých veličin a parametrů a je představen reálný model pro monitorování stavu kontejnerů a jeho komunikační principy. Byl také navržen vlastní model a komunikace v systému inteligentního svozu odpadu.

Modelem pro reprezentaci reálného prostředí pro svoz odpadu je využito ohodnoceného grafu. K nim byla popsána teorie a podrobněji popsaný Floyd-Warshallův algoritmus, který vytvořená aplikace využívá. Vlastní svoz odpadu byl pak popsán pomocí problému obchodního cestujícího. Pro samotný výpočet optimální trasy v grafu je zvolen genetický algoritmus. Jsou popsány jeho výhody a nevýhody a na jakém principu tento algoritmus pracuje.

Dále je navržena a naprogramována aplikace v jazyce C/C++, která zpracuje vstupní soubor ve formátu SVG pomocí parsovací knihovny TinyXML-2. Po analýze a přípravě dat proběhne samotný výpočet optimální trasy pomocí genetického algoritmu. Jsou naměřeny a popsány veškeré závislosti nastavení takového algoritmu a nakonec se porovnává úspora nákladů inteligentního svozu odpadu vůči konvenčnímu.

LITERATURA

[1] WANG, Wen Qiang, Xiaoming ZHANG, Jiangwei ZHANG, Hock Beng LIM, Branislav MICUSIK a Sergio A. VELASTIN. *Smart Traffic Cloud: An Infrastructure for Traffic Applications.* 2012 IEEE 18th International Conference on Parallel and Distributed Systems. 2012, s. 254-275. DOI: 10.1111/(issn)1600-0854.

[2] CHOURABI, Hafedh, Taewoo NAM, Shawn WALKER, J. Ramon GIL-GARCIA, Sehl MELLOULI, Karine NAHON, Theresa A. PARDO a Hans Jochen SCHOLL. *Understanding Smart Cities: An Integrative Framework.* 2012 45th Hawaii International Conference on System Sciences. 2012. DOI: 10.3403/30277667.

[3] *Smart cities: magazín o chytrých technologiích pro efektivnější správu měst a obcí.* ISSN 2336-1786.

[4] SRIKANTH, S.V., Pramod P.J., Dileep K.P., Tapas S., Mahesh U. PATIL, Sarat Chandra Babu N., Theresa A. PARDO a Hans Jochen SCHOLL. *Design and Implementation of a Prototype Smart PARKing (SPARK) System Using Wireless Sensor Networks: An Integrative Framework.* 2009 International Conference on Advanced Information Networking and Applications Workshops. IEEE, 2009, s. 401-406. DOI: 10.1109/WAINA.2009.53. Dostupné z: <`http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5136681`>

[5] WANG, Hongwei, Wenbo HE, Wai Chong CHIA, Nami SALIMI, Mahesh U. PATIL, Sarat Chandra Babu N., Theresa A. PARDO a Hans Jochen SCHOLL. *A Reservation-based Smart Parking System: An Integrative Framework. A Reservation-based Smart Parking System.* IEEE, 2011, s. 91-101. DOI: 978-1-4577-0248-8. Dostupné z:<`http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5136681`>

[6] *International Journal of Innovative Research in Science, Engineering and Technology.* ISSN 2319 – 8753.

[7] ANDERSSON, Gunnar, Kamil DURSUN, Bjorn Gitle HAUGE, Bernt BREMDAL, Ghavameddin NOURBAKHSH, Marilyn A. BROWN, Shan ZHOU, Marilyn A. BROWN a Shan ZHOU. *Establishing sustainable and reliable smart grids.* 2013 IEEE International Workshop on Applied Measurements for Power Systems (AMPS). 2013, s. 271-318.

[8] YOSHIURA, Noriaki, Yusaku FUJII, Naoya OHTA, Rodrigo PANTONI, Cleber FONSECA a Dennis BRANDAO. *Smart street light system looking like*

*usual street lights based on sensor networks.*2013 13th International Symposium on Communications and Information Technologies (ISCIT). 2013. DOI: 10.5772/48718.

[9] CATANIA, Vincenzo a Daniela VENTURA. *An approch for monitoring and smart planning of urban solid waste management using smart-M3 platform.* Proceedings of 15th Conference of Open Innovations Association FRUCT. 2014. DOI: 10.1109/fruct.2014.6872422.

[10] ISSAC, Roshan a M AKSHAI. *SVASTHA: An effective solid waste management system for Thiruvalla Municipality in Android OS.* 2013 IEEE Global Humanitarian Technology Conference: South Asia Satellite (GHTC-SAS). 2013. DOI: 10.1109/ghtc-sas.2013.6629926.

[11] ZIEGLER, E. *Encyclopedia of environmental science and engineering.* 6th ed. Boca Raton, FL: Taylor, c2012, 2 v. (xxvi, 1386 p.). ISBN 97814398948282.

[12] MAMUN, Md. Abdulla Al, M. A. HANNAN, Aini HUSSAIN, Chen LI-WAN, Chen QIANG a Li HONG-BIN. *Real time solid waste bin monitoring system framework using wireless sensor network.* 2014 International Conference on Electronics, Information and Communications (ICEIC). 2014, s. 9-14.

[13] MAMUN, Md. Abdulla Al, M.A. HANNAN, Aini HUSSAIN, Hassan BASRI, Chen QIANG a Li HONG-BIN. *Wireless Sensor Network Prototype for Solid Waste Bin Monitoring with Energy Efficient Sensing Algorithm.* 2013 IEEE 16th International Conference on Computational Science and Engineering. 2013, s. 9-14. DOI: 10.1109/cse.2013.65.

[14] DIESTEL, Reinhard. *Graph theory.* 2nd ed. New York: Springer, c2000, xiv, 312 p. ISBN 03-879-8976-5.

[15] COMSA, Anamaria, Andrei Bogdan RUS a Virgil DOBROTA. Simulation of the Floyd-Warshall algorithm using OMNeT++ 4.1. In: 2012 9th International Conference on Communications (COMM) [online]. 2012 [cit. 2015-05-12]. DOI: 10.1109/iccomm.2012.6262572.

[16] POKORNÁ, Petra. 2008. Problém obchodního cestujícího pomocí metody Mravenčí kolonie. Pardubice. Bakalářská práce. Univerzita Pardubice Fakulta ekonomicko-správní.

[17] WINSTON, W. L. – VENKATARAMANAN, M. Introduction to Mathematical Programming: Operations Research. Brooks/Cole-Thomson Leasing. California 2003.

[18] KUDELOVÁ, Blanka. 2007. Srovnání efektivnosti algoritmů pro řešení úloh obchodního cestujícího. Praha. Diplomová práce. Vysoká škola ekonomická v Praze.

[19] MITCHELL, Melanie. An introduction to genetic algorithms. 1st ed. Cambridge: The MIT Press, 1998, viii, 209 s. ISBN 0262133164.

[20] KACHAN, Dmitry. *Integration of ns-3 with MATLAB/Simulink.* Lulea University of Technology, Sweden, 2010. ISSN: 1653-0187. Dostupné z: <`http://epubl.ltu.se/1653-0187/2010/062/LTU-PB-EX-10062-SE.pdf`>. Master Thesis.

[21] ns-3 project. *ns-3 Manual, Release ns-3-dev.* Dostupné z: <`http://www.nsnam.org/docs/release/3.13/manual/ns-3-manual.pdf`>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

CEPT	Conférence européenne des administrations des postes et des télécommunications - Konference evropských správ pošt a telekomunikací
CCTV	Closed Circuit Television - Uzavřený televizní okruh
GPS	Global Positioning System - Globální polohovací systém
GIS	Geographic Information Systems - Geografický informační systém
GSM	Groupe Spécial Mobile - Globální Systém pro Mobilní komunikaci
LED	Light-Emitting Diode - Dioda emitující světlo
RFID	Radio Frequency Identification - Identifikace na rádiové frekvenci
SRD	Short Range Devices - Zařízení na krátkou vzdálenost
TCP/IP	Transmission Control Protocol/Internet Protocol
SVG	Scalable Vector Graphics - Škálovatelná vektorová grafika
XML	Extensible Markup Language - Rozšiřitelný značkovací jazyk

SEZNAM PŘÍLOH

A Komunikační modely systémů pro svoz odpadu **57**

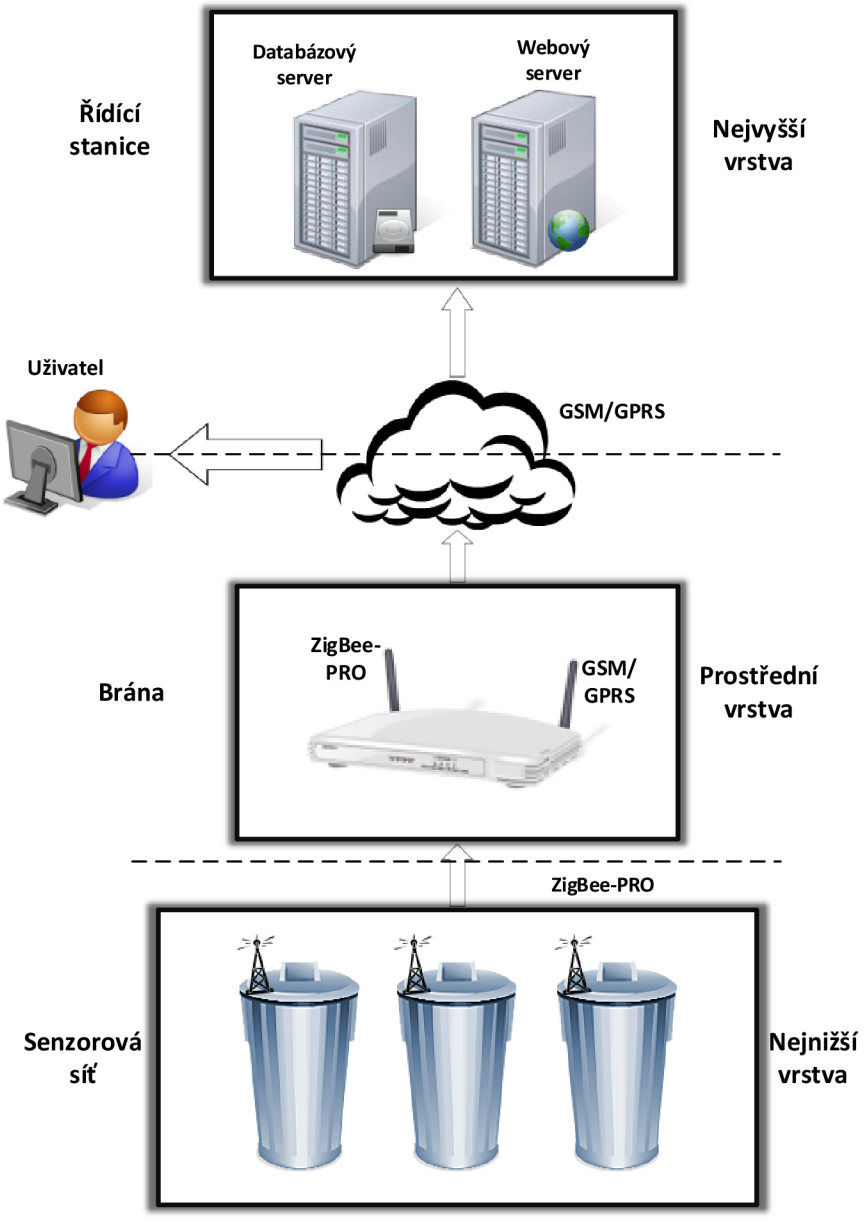
A.1 Příklad modelu pro svoz odpadu 57

A.2 Navržený model pro monotorování kontejnerů 58

B Obsah přiloženého CD **59**

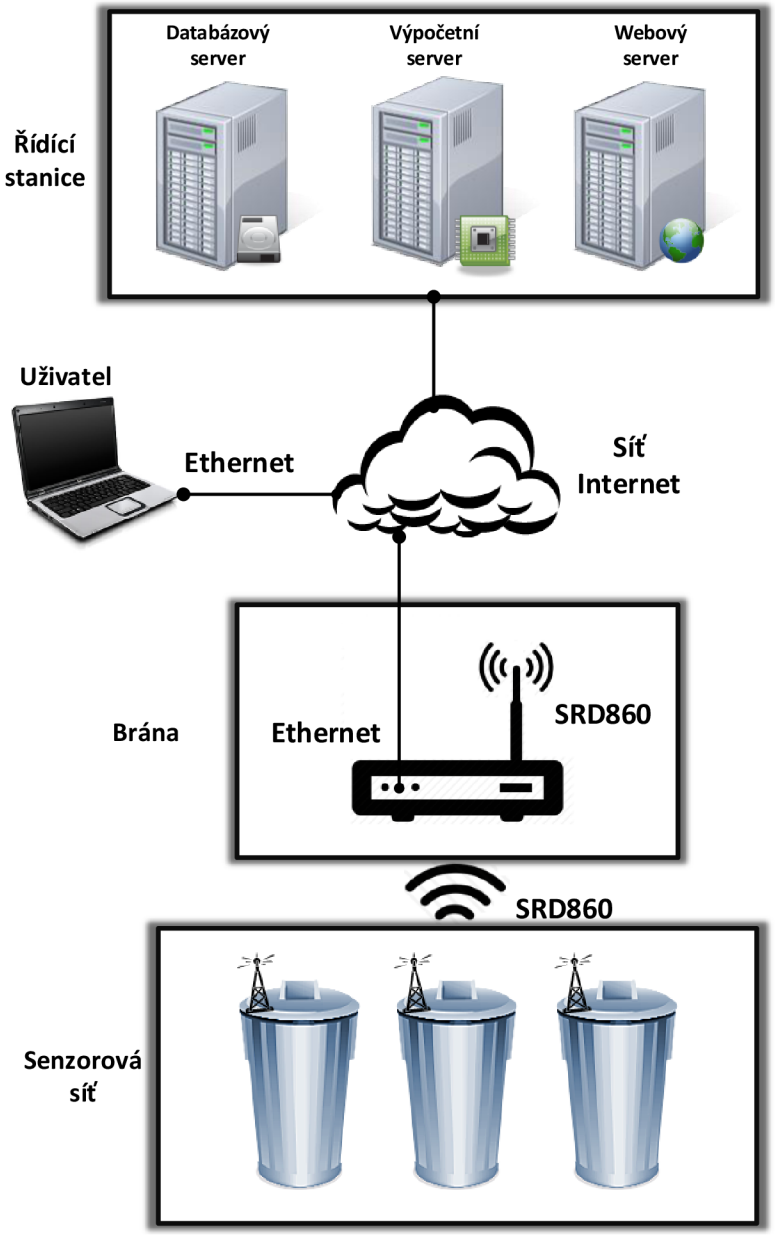
A KOMUNIKAČNÍ MODELY SYSTÉMŮ PRO SVOZ ODPADU

A.1 Příklad modelu pro svoz odpadu



Obr. A.1: Komunikační model systému pro monitorování kontejnerů.

A.2 Navržený model pro monotorování kontejnerů



Obr. A.2: Navržený model pro monitorování stavu kontejnerů.

B OBSAH PŘILOŽENÉHO CD

Jednotlivé soubory jsou samostatně v odpovídajících složkách.
Obsahem přiloženého CD jsou tyto soubory:

- Hlavní dokument-PDF soubor s diplomovou prací
- Spustitelný soubor aplikace
- Vstupní SVG soubory
- Kompletní projekt aplikace