

ŽILINSKÁ UNIVERZITA V ŽILINE
FAKULTA RIADENIA A INFORMATIKY

DIPLOMOVÁ
PRÁCA

Bc. PETER RENDEK

**Aplikácia GIS pre monitoring separovaného odpadu
v meste Rajec s využitím Smart technológií**

Vedúci práce: doc. Ing. Peter Márton, PhD., KMMOA, FRI UNIZA

Registračné číslo: 592/2018

Žilina, 2021

ŽILINSKÁ UNIVERZITA V ŽILINE

FAKULTA RIADENIA A INFORMATIKY

DIPLOMOVÁ

PRÁCA

INTELIGETNÉ INFORMAČNÉ SYSTÉMY

Bc. PETER RENDEK

**Aplikácia GIS pre monitoring separovaného odpadu
v meste Rajec s využitím Smart technológií**

Žilinská univerzita v Žiline

Fakulta riadenia a informatiky

Školiace pracovisko: Žilina


Žilina, 2021

Čestné vyhlásenie

Čestne prehlasujem, že som diplomovú prácu vypracoval samostatne pod odborným dohľadom vedúceho diplomovej práce a v zozname použitých zdrojov som uviedol všetky použité zdroje a literatúru, ktoré som používal pri vypracovaní mojej diplomovej práce.

V Žiline 5.mája 2021

Bc. Peter Rendek

Podpis:  _____

Pod'akovanie

Týmto spôsobom by som chcel poďakovať vedúcemu diplomovej práce doc. Ing. Petrovi Mártonovi, PhD. za veľkú trpezlivosť, odborné vedenie, užitočné pripomienky a cenné rady, ktoré mi poskytoval v priebehu vypracovávaní diplomovej práce.

Ďalej by som chcel poďakovať doc. Ing. Michalovi Kohánimu, PhD. za odborné konzultácie o algoritmoch, ktoré som v rámci diplomovej práce implementoval, otestoval a nakoniec použil.

Najväčšia vďaka patrí mojim rodičom, ktorí ma v priebehu celého štúdia podporovali.

ABSTRAKT V ŠTÁTNOJ JAZYKU

RENDEK, Peter: *Aplikácia GIS pre monitoring separovaného odpadu v meste Rajec s využitím Smart technológií*. [Diplomová práca]. – Žilinská univerzita v Žiline. Fakulta riadenia a informatiky; Katedra matematických metód a operačnej analýzy. – Vedúci: doc. Ing. Peter. Márton, PhD., KMMOA, UNIZA – Stupeň odbornej kvalifikácie: Inžinier v odbore Inteligentné informačné systémy – Žilina: FRI UNIZA, 2021. Počet strán 79s.

Cieľom diplomovej práce je vytvoriť geografický informačný systém pre zber a monitoring separovaného odpadu v meste Rajec. Takýto informačný systém má využívať Smart technológiu Sensoneo (ultrazvukové senzory na meranie naplnenosti zberných nádob). Finálna verzia informačného systému je určená pre pracovníka odpadového hospodárstva mesta Rajec a má mu poskytovať informačnú podporu pri rozhodovaniach o plánovaní trás zberov jednotlivých druhov odpadu a rozmiestňovaní stojísk na zber odpadu. Informačný systém monitoruje naplnenosti zberných nádob a eviduje ho v štatistikách. Informácie o naplnenosti sú zberané na základe SMS s aktuálnym stavom zo senzorov Sensoneo, alebo od občanov, ktorí pomocou oskenovania QR kódu uvedeného na zbernej nádobe pošlú SMS s aktuálnym stavom nádoby. Informačný systém tiež umožňuje evidenciu uskutočnených (zberných aj servisných) jazd vozidiel odpadového hospodárstva, ktorú môže pracovník pri plánovaní využiť. Jednotlivé komponenty vytvoreného informačného systému ako napr.: rozhranie pre komunikáciu s SMS bránou, optimalizačné algoritmy a heuristiky majú v praxi široké využitie

Kľúčové slová: GIS, odpadové hospodárstvo, separácia odpadu, senzory Sensoneo, plánovanie trás, problém obchodného cestujúceho

ABSTRAKT V CUDZOM JAZYKU

RENDEK, Peter: *GIS Application for monitoring of separated waste in Rajec city by smart technology*. [Diploma thesis]. – The University of Žilina. Faculty of Management Science & Informatics; Department of Mathematical Methods and Operations Research.– Tutor: doc. Ing. Peter. Márton, PhD., KMMOA, ŽU –Qualification level: Master in field Intelligent information systems – Žilina: FRI UNIZA, 2021. -79p.

The aim of the master thesis is to create a geographic information system for the collection and monitoring of separated waste in the town of Rajec. Such information system is to use Smart technology Sensoneo (ultrasonic sensors for measuring the filling of collection containers). The final version of the information system is intended for the waste management employee of the city of Rajec and is intended to provide him with information support in decisions on the planning of collection routes for individual types of waste and the deployment of waste collection stands. The information system monitors the filling of collection containers and records it in statistics. Information on filling is collected on the basis of an SMS with the current status from Sensoneo sensors, or from citizens who send an SMS with the current status of the container by scanning the QR code present on the collection container. The information system also enables the registration of performed (collection and service) journeys of waste management vehicles, which can be used by the worker during planning. Individual components of this information system, such as interface for communication with the SMS gateway, optimization algorithms and heuristics are widely used in practice.

Key words: GIS, waste management, waste separation, sensors Sensoneo, route planning, travelling salesman problem

ZADANIE TÉMY DIPLOMOVEJ PRÁCE

Studijný program: Inteligentné informačné systémy

Meno a priezvisko

Peter Rendek

Osobné číslo

557499

Názov práce v slovenskom aj anglickom jazyku

Aplikácia GIS pre monitoring separovaného odpadu v meste Rajec s využitím Smart technológií

GIS Application for monitoring of separated waste in Rajec city by smart technology

Zadanie úlohy, ciele, pokyny pre vypracovanie

(Ak je málo miesta, použite opačnú stranu)

Cieľ diplomovej práce:

Mesto Rajec má záujem o využitie smart technológií SENSONEO (ultrazvukové senzory monitorujúce množstvo odpadu v zbernej nádobe) pre zber separovaného odpadu. Použitie tejto technológie by umožnilo aj zber, spravovanie, analýzu a vizualizáciu geografických informácií o zberných nádobách, ktoré spravuje mesto Rajec. V diplomovej práci by autor navrhol geografický informačný systém, vrátane sledovania premenlivých informácií o zaplnení zberných nádob. Súčasťou navrhnutého GIS by boli priestorové analýzy pre určenie miest, kde je zberných nádob nedostatok alebo prebytok a tiež sieťové analýzy (návrh optimálnych trás pre jazdy vozidiel pre zber odpadu zo zberných nádob). Nasadením vyvinutého GIS sa predpokladá úspora nákladov na odpadové hospodárstvo. Súčasťou GIS by bol aj prehľad úspor získaných vďaka zmene pri príprave trás vozidiel pre odvoz odpadu. Zmena by spočívala v tom, že nebudú vždy vyprázdňované všetky nádoby, ale iba tie, u ktorých je to potrebné.

Obsah:

Úvod

1. GIS ako nástroj pre podporu rozhodovania
2. Popis technológie SENSONEO a údajov, ktoré sa pomocou nej dajú získať o zberných nádobách
3. Návrh vlastného GIS pre manažment odpadového hospodárstva
4. Porovnanie návrhu so súčasným stavom - vyhodnotenie úspor

Meno a pracovisko vedúceho DP: doc. Ing. Peter Márton, PhD., KMMOA, ŽU

Meno a pracovisko tútora DP:

vedúci katedry
(dátum a podpis)

Zadanie zaregistrované dňa 24. 10. 2018 pod číslom 592/2018 podpis _____

Obsah

Zoznam obrázkov	10
Zoznam tabuliek	12
Zoznam skratiek	13
Úvod	14
1 Aktuálny stav	15
1.1 Aktuálny stav odpadového hospodárstva v Rajci	15
1.2 Aktuálny stav informačných systémov pre zber odpadu na trhu.	16
1.3 Technológie	16
1.3.1 O ₂ SMS Connector API [3]	16
1.3.2 Sensoneo Smart senzory	18
1.4 Problém obchodného cestujúceho	19
1.4.1 Matematický model TSP	19
1.4.2 Riešenia TSP	20
1.5 Úloha okružných jázd.....	26
1.5.1 Riešenia úlohy okružných jázd.....	27
1.6 Úloha o P-mediáne	29
1.6.1 Matematický model pre úlohu Kapacitného p -mediánu	30
1.6.2 Riešenia P-mediánu	31
2 Ciele práce	33
3 Riešenie	35
3.1 Získanie a spracovanie počiatočných dát	35
3.1.1 QGIS získanie a spracovanie dát o cestnej infraštruktúre	35
3.1.2 Implementácia grafu a načítanie vstupného GEOJSON súboru.....	38
3.2 Výber databázového systému a návrh databázového modelu	41
3.3 Optimalizácia zberných jázd	44
3.3.1 Implementácia modelu TSP v Gurobi optimizéri.....	45
3.3.2 Implementácia metódy vetiev a hraníc	47
3.3.3 Implementácia Ant colony optimization	48
3.3.4 Implementácia sekvenčnej Clark-Wright heuristiky	51
3.4 Optimalizácia zberných stojísk	54
3.5 Implementácia zobrazenia geografických dát	56
3.6 Implementácia Messengera	59
3.7 Implementácia Grafického užívateľského prostredia.....	60

3.8	Architektúra informačného systému pre potreby OH	60
4	Výsledky práce a diskusia	63
4.1	Výsledky práce	63
4.2	Užívateľská príručka	64
	Záver	75
	Zoznam použitých zdrojov	77
	Zoznam príloh	79
	Prílohy	79
	Príloha A: Obsah CD	79

Zoznam obrázkov

Obrázok 1. Vizualizácia všetkých stojísk v Rajci.....	15
Obrázok 2. Vytvorenie objektu SMS.....	17
Obrázok 3. Pripravenie http post požiadavky	17
Obrázok 4. Zaslanie požiadavky s nasledovným spracovaním odpovede.....	18
Obrázok 5. Smart senzor Quatro [11]	19
Obrázok 6. Zobrazenie stavového priestoru riešenia TSP	21
Obrázok 7. Koreň konštruovaného riešenia.....	22
Obrázok 8. Strom riešení po pridaní všetkých nasledovníkov koreňa	23
Obrázok 9. Konečný stav riešenia	24
Obrázok 10. Hľadanie potravy mravcov [12]	25
Obrázok 11. QGIS – počiatočné vrstvy	35
Obrázok 12. QGIS - Zelené – relevantné cesty	36
Obrázok 13. Hlavné problémy spracovania dát.....	36
Obrázok 14. Fungovanie funkcie „Join Attributes By Nearest“.....	37
Obrázok 15. Atribútová tabuľka výslednej vrstvy.....	38
Obrázok 16. Vkládanie uzlov a hrán do grafu	40
Obrázok 17. Schéma databázy informačného systému	43
Obrázok 18. Algoritmus optimalizácie zberných jázd.....	45
Obrázok 19. Zdrojový kód metódy callback	46
Obrázok 20. Model TSP v Gurobi solvéri	46
Obrázok 21. Algoritmus MVH	48
Obrázok 22. Výpočet hodnôt pre rozhodovaciu maticu	49
Obrázok 23. Algoritmus prechodov k ďalšiemu uzlu.....	50
Obrázok 24. Jadro algoritmu ACO	50
Obrázok 25. Porovnanie výsledkov ACO s optimálnym riešením.....	50
Obrázok 26. Sekvenčný algoritmus CW časť 1.....	52
Obrázok 27. Sekvenčný algoritmus CW časť 2.....	53
Obrázok 28. Sekvenčný algoritmus CW časť 3.....	53
Obrázok 29. Haversinov vzorec.....	55

Obrázok 30. Operácia mutácie.....	55
Obrázok 31. Metóda pre zobrazenie trás	57
Obrázok 32. Zobrazenie trás	58
Obrázok 33. Vykreslenie možných stojísk (červeno) a oblastí (zeleno) v optimizéri.....	58
Obrázok 34. Diagram najdôležitejších tried	62
Obrázok 35. Prihlasovacie okno do aplikácie.....	64
Obrázok 36. Základné okno – základné funkcionality	65
Obrázok 37. Modifikovanie údajov zbernej nádoby	66
Obrázok 38. Messenger	66
Obrázok 39. Pridanie novej zbernej nádoby	66
Obrázok 40. Vytvorenie zbernej jazdy	67
Obrázok 41. Editácia a vytvorenie jazdy	67
Obrázok 42. Záložka Aktuálne jazdy	69
Obrázok 43. Okná aktualizácie a vytvorenia vozidla	69
Obrázok 44. Zahájenie zbernej jazdy	70
Obrázok 45. SMS notifikácia pre zahájenie jazdy	70
Obrázok 46. Uzavretie jazdy – doplnenie potrebných informácií	70
Obrázok 47. Vytvorenie servisovej uzavretej jazdy	71
Obrázok 48. Kniha jazd – štatistiky OH	72
Obrázok 49. Zobrazenie zberných stojísk.....	72
Obrázok 50. Zobrazenie zberných trás	73
Obrázok 51. Optimizér stojísk	74
Obrázok 52. Fungovanie monitorovanie objemu odpadu pomocou SMS.....	74

Zoznam tabuliek

Tabuľka 1. Rajec zberné nádoby	15
--------------------------------------	----

Zoznam skratiek

DP	Diplomová práca
GIS	Geograficky informačný systém
IS	Informačný systém
OH	Odpadové hospodárstvo
HTTPS	Zabezpečený hypertextový prenosový protokol
JSON	JavaScriptový objektový zápis
API	Aplikačné programové rozhranie
TSP	Problém obchodného cestujúceho
API	Rozhranie na programovanie aplikácií
GPRS	Univerzálna paketová rádiová služba
IČO	Identifikačné číslo organizácie
DPH	Daň z pridanej hodnoty
ASCII	Americký štandardný kód pre výmenu informácií
MVH	Metóda vetiev a hraníc
ACO	Optimalizácia kolóniou mravcov
VRP	Úloha okružných jász
CW	Clark-Wright algoritmus
MsÚ	Mestský úrad

Úvod

Proces rozhodovania sa v dopravných logistických systémoch, ktorý sa zaoberá zabezpečovaním požiadaviek zákazníkov, je výhodné v praxi podporiť systémom na podporu rozhodovania, napr. informačným systémom (ďalej IS). Toto tvrdenie platí nielen pre zber triedeného odpadu, ale aj pre množstvo iných prípadov ako napr. distribúcia tovarov.

V mojej diplomovej práci (ďalej DP) najskôr analyzujem aktuálny zber triedeného odpadu v meste Rajec, následne navrhmem a vytvorím IS, ktorý zjednoduší pracovné procesy pracovníkom odpadového hospodárstva (ďalej OH) patriacej pod referát životného prostredia. Vytvorený IS pomôže optimalizovať zberné trasy a rozmiestnenia zberných miest. Aplikácia pre systém OH je založená na nasledujúcich pilieroch. Prvým pilierom sú optimalizačné heuristiky a algoritmy pre riešenie nasledujúcich úloh a problémov: problém obchodného cestujúceho, problém okružných jázd a úloha o p-mediáne

Druhým pilierom IS je zber dát o aktuálnej naplnenosti zberných nádob buď pomocou senzorov Sensoneo, alebo za pomoci občanov, prípadne pracovníkov OH Rajec. Tretím pilierom IS sú štatistiky uskutočnených jázd vozidiel (zberných aj servisných). Ďalšou evidovanou štatistikou historický vývoj naplnenosti nádob na jednotlivých stojiskách. Vzájomnou kombináciou vyššie uvedených pilierov, vzniká pomerne široko-využiteľný IS pre podporu rozhodovania sa nielen pre potreby OH ale pre iné podobné logistické systémy.

V teoretickej časti DP vysvetľujem základnú terminológiu pre pochopenie problematiky OH. Rozoberám aktuálnu situáciu a popisujem základne procesy OH Rajec. Ďalej v krátkosti predstavujem a zdôvodňujem použité nástroje, technológie, heuristiky a algoritmy. V praktickej časti DP sa zameriavam na vývoj aplikácie informačného systému a na vyriešenie problémov vzniknutých počas realizácie jeho tvorby. Hlavné funkcionality aplikácie sú:

- Navrhovanie: zberných trás a rozmiestnenia zberných stojísk
- Spravovanie: trás, vozidiel, odberných nádob
- Rozhranie pre komunikáciu s SMS bránou
- Vizualizácia polohy zberných stojísk a zberných trás

1 Aktuálny stav

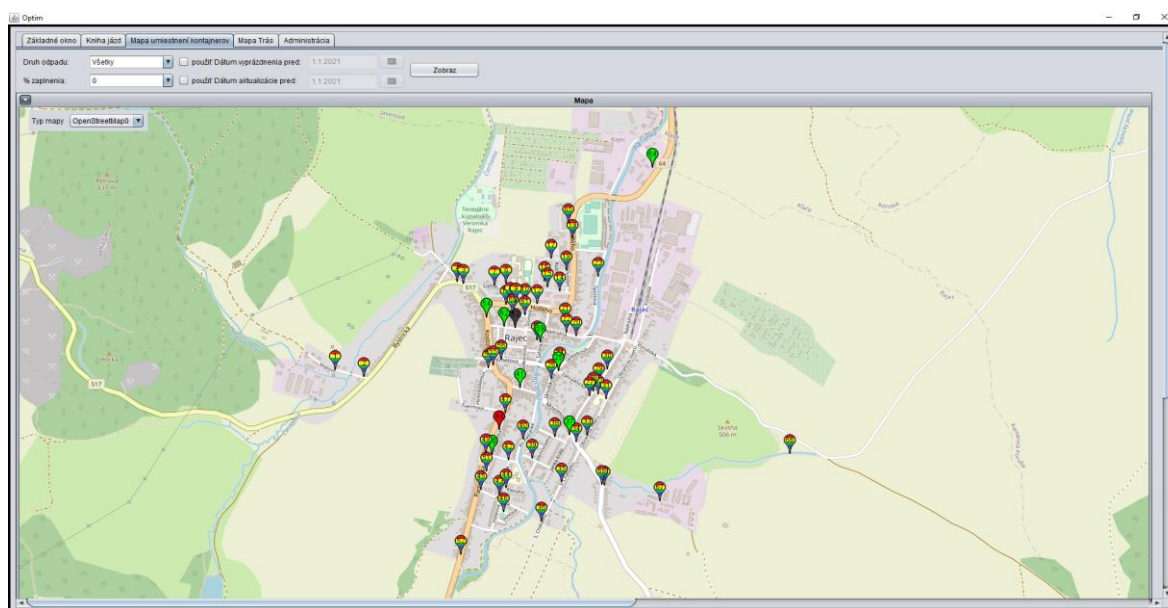
1.1 Aktuálny stav odpadového hospodárstva v Rajci

Rajec je malé mesto (32km²) s počtom obyvateľov 5800 v okrese Žilina [15]. OH mesta Rajec funguje rovnako ako vo väčšine Slovenských miest, ktoré sa snažia separovať odpad v rámci svojich možností a vybavenosti. Disponujú zbernými nádobami na rôzne druhy odpadu, ktoré odvážajú v pravidelných časových intervaloch. Proces zberu odpadu začína obdržaním rokmi overenému plánu jazdy a terénni pracovníci OH urobia podľa neho okružnú jazdu.

Aktuálne je v meste situovaných 65 stojísk, vid'. Obrázok 1, kde sa nachádzajú nasledujúce počty nádob na separovaný odpad.

Tabuľka 1. Rajec zberné nádoby

Druh odpadu:	Počet Nádob:	Druh odpadu:	Počet Nádob:
Sklo:	68	Kov. obaly:	17
Plast:	36	Textil:	11
Papier:	46	Olej:	2



Obrázok 1. Vizualizácia všetkých stojísk v Rajci

1.2 Aktuálny stav informačných systémov pre zber odpadu na trhu.

Aktuálne je na trhu na Slovenska a v Českej republiky nedostatok firiem, ktoré ponúkajú aplikácie pre manažovanie OH. Najväčšou spoločnosťou na trhu je medzinárodná spoločnosť Sensoneo, ktorá pôsobí vo viac ako 40 krajinách sveta, avšak jej služby využíva iba 30 miest, napr. Praha, Nitra, Trnava.

„Ich riešenie kombinuje unikátne ultrazvukové smart senzory, ktoré monitorujú odpad, so sofistikovaným softvérom, čo mestám a spoločnostiam umožňuje komplexne digitalizovať správu odpadu, robiť strategické rozhodnutia na základe reálnych dát a optimalizovať zvozovú logistiku.“ [10]

Na trhu sa ďalej nachádzajú aplikácie, cez ktoré si občania miest a obcí môžu zaregistrovať požiadavku na odvoz napr. nadrozmerného odpadu alebo aplikácie, ktoré zobrazujú plánované zbory jednotlivých komodít. Mnou navrhovaný IS, spočíva v kompromise oboch vyššie uvedených prístupov. S využitím meracích senzorov od spoločnosti Sensoneo, alebo bez nich, by môj informačný systém mohol, byť tak správnu voľbou pre malé a stredne veľké mestá, respektíve obce.

1.3 Technológie

V nasledujúcom texte popisujem technológie, ktoré navrhujem použiť pri riešení môjho zadania. Neskôr v texte popisujem aj optimalizačné úlohy a optimalizačné metódy pre ich riešenie. Podrobnejší popis použitých heuristik a algoritmov uvádzam najmä preto, aby ich podstata mohla byť vysvetlená aj čitateľovi bez znalostí v oblasti optimalizácie.

1.3.1 O₂ SMS Connector API [3]

Ide o internetové rozhranie s prehľadnou dokumentáciou [4], ktoré umožňuje hromadne posielať a prijímať SMS správy. Všetky volania API fungujú na protokole HTTPS a pre výmenu dát využívajú JSON formát. Zriadenie služby je bezplatné, ale je nutné sa zaregistrovať na web stránke [3]. Registrácia je primárne určená pre právnické osoby (je požadované IČO), ale na základe dohovoru s technickou podporou je registrácia možná aj pre fyzické osoby. Po úspešnom zaregistrovaní obdržíte kredit 2€, ktorý môžete využiť pre testovanie. Ak potrebujete vo Vašej aplikácii využívať prijímanie SMS, musíte si zriadiť

virtuálne číslo, ktoré je mesačne spoplatnené aktuálne sumou 6€ s DPH. Cena poslanej SMS je určená na základe aktuálnej ceny z bežného cenníka O₂. Pre potrebu posielania veľkého množstva SMS správ sú k dispozícii výhodnejšie balíčky predplatených správ a tiež sa dá za príplatok objednať krátke čísla. Základné volania API fungujú na základe HTTP POST alebo HTTP GET metódy. Príklad všeobecného volania: *https://api-tls12.smstools.sk/3/"názov funkcie"*, kde „názov funkcie“ môže byť napr.: *send_batch* – pre poslanie dávky SMS správ, *sms_get_received* – pre načítanie prijatých správ, *credit_remaining* – získanie informácií o zostatkovom kredite.

Príklad poslanie dávky správ (HTTP Post metóda) krok po kroku:

1. Vytvorenie objektu SMS, ktorý sa následne prevedie v kroku 3. do JSON formátu (môže obsahovať len pôvodne ASCII znaky) :

```
public SMS(String text, String odosielatel, ArrayList<String> prijemcovia) {
    try {
        this.text = Normalizer.normalize(text, Normalizer.Form.NFD).replaceAll("[^\\p{ASCII}]", "");
        this.text = this.text.replace("\n", "");
        System.out.println(this.text);
        this.odosielatel = Normalizer.normalize(odosielatel, Normalizer.Form.NFD).replaceAll("[^\\p{ASCII}]", "");
        this.odosielatel = this.odosielatel.replace("\n", "");
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, e.getClass().getName());
    }
    this.odosielatel = odosielatel;
    this.prijemcovia = prijemcovia;
}
```

Obrázok 2. Vytvorenie objektu SMS

2. Pripravenie požiadavky:

```
String surl = "https://api-tls12.smstools.sk/3/send_batch";
public void prepareConnection(String surl) {
    URL url;
    try {
        url = new URL(surl);
        con = (HttpURLConnection) url.openConnection();
        con.setRequestMethod("GET");
        con.setRequestProperty("Content-Type", "application/json; utf-8");
        con.setRequestProperty("Accept", "application/json");
        con.setDoOutput(true);
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, e.getClass().getName() + ": " + e.getMessage());
    }
}
```

Obrázok 3. Pripravenie http post požiadavky

3. Poslanie požiadavky a následné spracovanie odpovede:

```
public String posliSMS(SMS sms) {
    try {
        sms.setApiKey(apiKey);
        String JSON_sms = sms.toString(); //vytvorenie JSON formátu
        prepareConnection(send); //prívenie požiadavky
        OutputStream os = con.getOutputStream();
        byte[] input = JSON_sms.getBytes("utf-8");
        os.write(input, 0, input.length); //odoslanie požiadavky
        BufferedReader br = new BufferedReader(
            new InputStreamReader(con.getInputStream(), "utf-8"));
        os.flush(); os.close();
        String s = ""; String responseLine = null; //spracovanie odpovede
        while ((responseLine = br.readLine()) != null) {
            s += responseLine.trim();
        }
        br.close();
        JSONParser parser = new JSONParser();
        Object obj = parser.parse(s);
        JSONObject jsonObject = (JSONObject) obj;
        return "" + jsonObject.get("id") + " " + jsonObject.get("note");
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, e.getClass().getName());
    }
    return "problem";
}
```

Obrázok 4. Zaslanie požiadavky s nasledovným spracovaním odpovede

1.3.2 Sensoneo Smart senzory

Smart senzory [11], vyrábané na Slovensku sú určené na monitorovanie naplnenosti zbernej nádoby pomocou ultrazvukových lúčov. Ich použitie je možné pri rôznych druhoch odpadu ako napr. plasty, sklo, tekutiny atď. v nádobách rôznych objemov a tvarov. Senzory sú tiež vodotesné (IP69), nárazu odolné a dajú sa používať celoročne v našich geografických podmienkach. Obal senzoru je vyrobený polyimidových optických vlákien, ktoré sú recyklovateľné a vďaka lokálnosti výroby a možnosti vymieňať nefunkčné respektíve opotrebované komponenty napr. batérie, je ich možné definovať ako produkt v súlade s ekodizajnom. Smart senzory je možné pripojiť na rôzne druhy sietí ako LoraWan, NB-IOT, Sigfox a GPRS čo im umožňuje rýchly dátový prenos. Senzory tiež poskytujú doplnkové užitočné funkcie ako napr. meranie teploty, požiarneho alarmu, hlásenie prevrátenia alebo akcelerometer, ktorý umožňuje detekciu vysypania nádoby.

Quatro senzor	
Technológia merania	Ultrazvuk
Min. a max. vzdialenosť merania	15 - 400 cm
Max. počet meraní denne	24x denne
Informácia	Percentuálne naplnenie nádoby
Prenos dát	Sigfox / LoRaWAN / NB-IoT / GPRS
Napájanie	Vymeniteľné baterie
Životnosť	7 - 10 rokov
Požiarový alarm	Áno
Nastavenie senzora	Mobilná aplikácia cez Bluetooth
Rozmery senzora	68 mm/200 mm/85 mm
Odolnosť	IP67

Obrázok 5. Smart senzor Quatro [11]

1.4 Problém obchodného cestujúceho

Travelling salesman problem – TSP, patrí medzi najbežnejšie a najdôležitejšie optimalizačné úlohy. TSP ako aj iné optimalizačné úlohy nájdu uplatnenie v rôznych oblastiach ako napr.: distribúcia tovarov a služieb, plánovanie jász, plánovanie pohybov robotov a pod..

Základná formulácia úlohy vyzerá nasledovne: je zadané n -miest a jednotlivé vzdialenosti medzi nimi. Úlohou obchodného cestujúceho, je navštíviť všetkých n -miest a následne sa vrátiť do východiskového miesta, tak aby absolvovaná trasa bola najkratšia a každé miesto bolo navštívené práve raz. Formálne ide o zostrojenie najlacnejšej Hamiltonovskej kružnice v úplne ohodnotenom grafe.

Pre lepšie pochopenie problematiky hľadania najkratšej cesty (v grafe) v nasledujúcich podkapitolách popíšem základnú terminológiu úlohy, rozoberiem problém s riešením rozsiahlych zadaní a načrtnem možnosti ako ich vyriešiť. Na záver popíšem heuristiky a algoritmy na riešenie TSP, ktoré som v DP na implementoval.

1.4.1 Matematický model TSP

[6][9] Označíme miesta 1 až n a definujeme bivalentné premenné:

$$x_{ij} = \begin{cases} 1, & \text{je zaradená cesta z miesta } i \text{ do } j \\ 0, & \text{inak} \end{cases}$$

Nech $C_{ij} \in \mathbb{Z}^{0+}$ je vzdialenosť z miesta i do j . Potom môžeme TSP považovať úlohu za problém celočíselného lineárneho programovania a účelová funkcia, vyjadruje celkovú dĺžku použitých hrán:

$$\min \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij}$$

Nasledujúce podmienky zabezpečia, že v každom vrchole práve jedna hrana končí a práve jedna hrana začína. To znamená, že do každého a z každého vrcholu sa pôjde práve raz.

$$\sum_{i=1, i \neq j}^n x_{ij} = 1, \forall j = 1, \dots, n$$

$$\sum_{j=1, i \neq j}^n x_{ij} = 1, \forall i = 1, \dots, n$$

Nasledujúce obmedzenie zaručí, že finálne riešenie bude zložené len z jedného cyklu a prípadne pod-cykly budú odstránené. Taktiež zabezpečí, že počet hrán, ktoré môžu byť zaradené do trasy sú definované množinou vrcholov Q a nemôžu presiahnuť $|Q| - 1$

$$\sum_{i \in Q} \sum_{j \neq i, j \in Q} x_{ij} \leq |Q| - 1, \forall Q \subset \{1, \dots, n\}, |Q| \geq 2$$

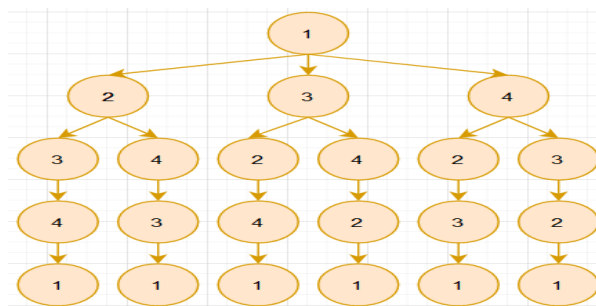
1.4.2 Riešenia TSP

Existuje viacero metód ako riešiť problematiku TSP. Tieto metódy sa dajú rozdeliť do nasledujúcich skupín. Prvú skupinu tvoria exaktné algoritmy, ktoré nájdu optimálne riešenie pokiaľ existuje. Exaktné metódy sú hlavne určené pre úlohy s menším počtom vrcholov z dôvodu, že TSP patrí medzi NP úplne úlohy, kde je náročné získať optimálny výsledok v polynomiálnom čase. Patria sem metódy ako Backtracking, metóda vetiev a hraníc a jej variácie a metódy lineárneho programovania. Ďalšia skupina metód sú heuristiky. Využívajú sa hlavne v prípadoch, ktoré sú ťažko riešiteľné exaktnými metódami z dôvodu časovej alebo pamäťovej zložitosti. Tieto metódy síce nezaručujú nájdenie optimálneho riešenia, ale sa k nemu môžu v rozumnom čase aspoň priblížiť. Heuristiky fungujú na princípe prechodu od prípustného (primárne heuristiky) alebo neprípustného (duálne heuristiky) riešenia k ďalšiemu riešeniu s lepšou hodnotou lokálneho kritéria. Patria sem napr. vkladacie, výmenné heuristiky alebo heuristiky s výhodnosťnými koeficientami.

Do poslednej skupiny metód na riešenie TSP patria Metaheuristiky. Taktiež nezaručujú nájdenie optimálneho riešenia, ale na rozdiel od heuristik, sú schopné opustiť lokálne minimum a tak prehľadať celý priestor možných riešení. Patria sem napríklad tieto metaheuristiky: Simulated annealing, Genetický algoritmus a Ant colony optimization.

Metóda vetiev a hraníc (angl. Branch and Bound method)

To, že TSP patrí medzi NP-úplne úlohy znamená, že s rastúcim počtom vrcholov rastie čas potrebný na vyriešenie úlohy exponenciálne. Pri TSP v symetrickom úplnom grafe vieme vypočítať počet všetkých ciest, ktoré treba preskúmať podľa vzorca $\frac{1}{2}(n-1)!$, kde n je počet vrcholov grafu. Pre úlohu so 4-mi vrcholmi je preto potrebné preskúmať $\frac{1}{2}(4-1)! = 3$ možné cesty. Pre 10 vrcholov to bude 18×10^4 a už pri 16 vrcholov $6,53 \times 10^{11}$ možných ciest. Prvou možnosťou ako exaktne vyriešiť TSP je postupné prehľadanie celého stavového priestoru všetkých riešení a následný výber toho najlepšieho. Takýto postup je však z vyššie uvedených dôvodov možný len pri malých príkladoch. Na obrázku nižšie je zobrazený stavový priestor riešení pomocou stromu konštrukcie riešenia pre TSP v úplnom 4-vrcholovom symetrickom grafe. Keďže ide o symetrický graf, je zrejmé, že nasledujúce vetvy ktoré reprezentujú cestu obchodného cestujúceho sú ekvivalentné $1-2-3-4-1 \leftrightarrow 1-4-3-2-1$, $1-2-4-3-1 \leftrightarrow 1-3-4-2-1$ a $1-3-2-4-1$ a $1-4-2-3-1$, preto je postačujúce preskúmať len 3 vetvy (cesty) stromu.



Obrázok 6. Zobrazenie stavového priestoru riešenia TSP

V reálnej praxi nie je vždy možné takto prehľadať celý stavový priestor riešení a preto je potrebné hľadať riešenia, ktoré dokážu stavový priestor riešení zredukovať.

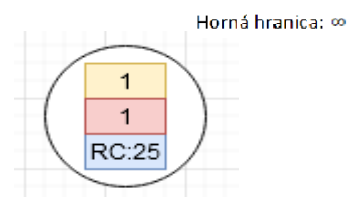
Metóda vetiev a hraníc je určená pre riešenie úloh celočíselného programovania. Využíva stratégiu postupného konštruovania stromu riešení do šírky, so snahou o čo najskoršie (najbližšie ku koreňu stromu) odrezávanie neperspektívnych vetiev.

Neperspektívne vetvy, sú vetvy, o ktorých vieme dokázať, že sa v nich nemôže nachádzať hľadané optimum a včasným odrezávaním takých vetiev sa znižuje exponenciálna zložitosť algoritmu. Fungovanie algoritmu vysvetlím na nasledujúcom príklade [1]. Máme zadaných 5 miest (uzlov) a maticu D vzdialenosti medzi nimi. Trasu budeme začínať v uzle 1 a na začiatok nastavíme hodnotu hornej hranice na ∞ .

$$D = \begin{pmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{pmatrix}$$

Operácia redukcie matice funguje nasledovne: Ak sa v riadku matice nenachádza hodnota 0, tak v riadku nájdeme minimum. Ak minimum $\neq \infty$ odčítame ho od nenulových hodnôt v danom riadku, inak ho považujeme za 0. Následne skontrolujeme stĺpce matice, či sa v nich nachádza aspoň jedna hodnota 0. Ak sa v stĺpci nenachádza hodnota 0, tak v stĺpci nájdeme minimum, Ak sa minimum $\neq \infty$, tak odčítame ho od nenulových hodnôt v danom stĺpci, inak ho považujeme za 0. Sčítaním nájdenej miním získame redukovanú cenu matice (RC). $RC = (10 + 2 + 2 + 3 + 4) + (1 + 0 + 3 + 0 + 0) = 21 + 4 = 25$. Koreň stromu riešenia, kde v žltom obdĺžniku je poradové číslo pridania vrcholu v strome, a v červenom obdĺžniku je index príslušného uzla z úlohy. Strom riešenia úlohy a redukovaná cena pre maticu v koreňovom vrchole bude vyzeráť nasledovne:

$$\begin{pmatrix} \infty & \mathbf{10} & \mathbf{17} & \mathbf{0} & \mathbf{1} \\ \mathbf{12} & \infty & \mathbf{11} & \mathbf{2} & \mathbf{0} \\ \mathbf{0} & \mathbf{3} & \infty & \mathbf{0} & \mathbf{2} \\ \mathbf{15} & \mathbf{3} & \mathbf{12} & \infty & \mathbf{0} \\ \mathbf{11} & \mathbf{0} & \mathbf{0} & \mathbf{12} & \infty \end{pmatrix}$$



Obrázok 7. Koreň konštruovaného riešenia

V nasledujúcich krokoch algoritmu budeme postupne pridávať so stratégiou konštruovania stromu do šírky nezaradené vrcholy do stromu riešenia. Pre každý takto pridaný vrchol vypočítame príslušnú redukovanú maticu a redukovanú cenu. Na nasledujúcom príklade pridaním 2.uzla do riešenia vysvetlím ako to funguje. Zoberieme maticu z vrcholu predchodcu. Nasledujúce hodnoty matíc nastavíme na nekonečno:

1. **hodnoty riadku** reprezentujúceho odchod z vrcholu predchodcu, lebo v tejto vetve už nie je možné znovu odchádzať z vrcholu predchodcu
2. **hodnoty stĺpca** reprezentujúceho príchod do aktuálneho vrcholu, lebo v tejto vetve už nie je možné znovu prísť do aktuálneho vrcholu
3. **hodnotu bunky**, reprezentujúci úsek od aktuálneho vrcholu do vrcholu predchodcu, lebo v tejto vetve už nemôže byť taký úsek.

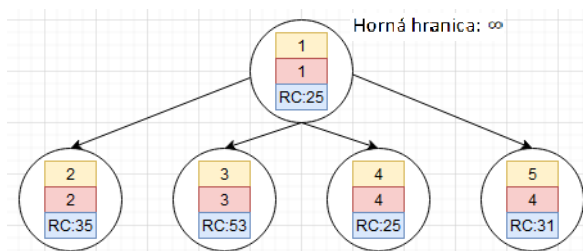
$$\begin{pmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{pmatrix} \Rightarrow \begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{pmatrix}$$

Na takto získanej matici, uskutočníme rovnakú operáciu redukovania matice ako bolo vysvetlené pri koreňovom vrchole. $RC_{akt_mat} = (0 + 0 + 0 + 0 + 0) + (0 + 0 + 0 + 0 + 0) = 0$. V tomto prípade maticu po nastavení príslušných hodnôt na nekonečná už nebolo potrebné ďalej redukovať. Redukovanú cenu pre aktuálny vrchol vypočítame podľa nasledujúceho vzťahu:

$$RC = RC_{akt_mat} + RC_{predchodcu} + M_{predchodcu}(predchodca, aktuálny)$$

$$RC = 25 + 0 + 10 = 35.$$

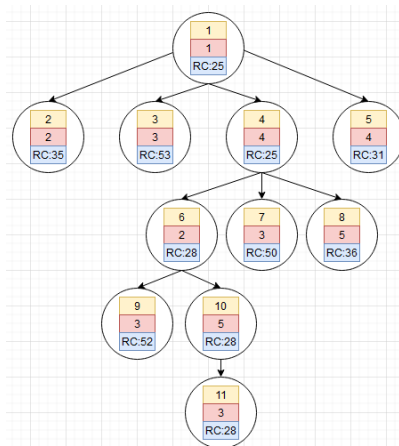
Rovnakým postup sa použije aj pre ostatné vrcholy rovnakého predchodcu. Na obrázku nižšie je zobrazený strom riešenia úlohy po pridaní všetkých možných ostatných vrcholov rovnakého predchodcu.



Obrázok 8. Strom riešení po pridaní všetkých nasledovníkov koreňa

Algoritmus následne pokračuje podobným spôsobom s tým, že sa pokračuje vždy v rozvíjaní vrcholu, ktorý je listom a má najmenšiu hodnotu RC. Po dosiahnutí kompletnej

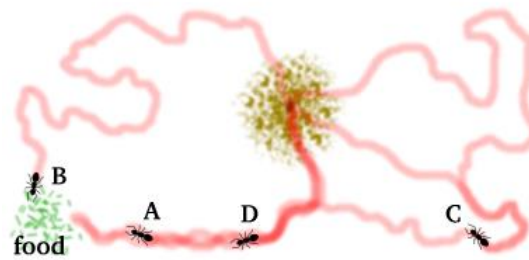
cesty v danej vetve (matica vrcholu bude obsahovať len 0 a ∞ , všetky uzly už budú v danej vetve použité), ak horná hranica $>$ RC v poslednom vrchole danej vetvy, tak aktualizujeme hornú hranicu na hodnotu RC a aktuálnu vetvu prehlásime za doteraz najlepšie nájdené riešenie. Ďalej tak pokračujeme v ďalšom rozvíjaní listových vrcholov. Ak všetky listové vrcholy majú $RC >$ hornú hranicu, algoritmus končí a doteraz nájdené najlepšie riešenie prehlásime za optimálne. Na obrázku nižšie je zobrazený prípad, keď prvým dosiahnutým riešením, sa našlo optimum, lebo ostatné listové vrcholy majú väčšie RC. Výsledná trasa je tvorená postupnosťou 1-4-2-5-3-1 a s nákladmi 28.



Obrázok 9. Konečný stav riešenia

Optimalizácia kolóniou mravcov (angl. Ant Colony Optimization, ACO)

[2] [12] Je metaheuristika používaná na riešenie komplexných problémov, ktorá sa obvyčajne používa na rýchle hľadanie dobrého riešenia v grafoch. Základnou myšlienkou optimalizácie je hľadanie obživy mravcami, ktorí navzájom vo svojom životnom prostredí nepriamo komunikujú pomocou vypúšťania chemickej pachovej stopy zvanej feromóny. Ostatné mravce na túto pachovú stopu pozitívne reagujú, a preto s väčšou pravdepodobnosťou preferujú cesty s väčšou koncentráciou feromónov, čím sa pachová stopa na ceste zosilňuje. Pokiaľ sa cesta nepoužíva, pachová stopa postupne slabne až úplne vyprchá. Čo umožňuje mravcom preskúmať aj nepoužívané cesty. Po určitom čase tak mravce nájdu optimálnu trasu k potrave. Potom používajú takto nájdenú cestu, pretože čím kratšia je táto cesta, tým viac krát ju stihnú použiť a feromónovú koncentráciu tak zosilniť.



Obrázok 10. Hľadanie potravy mravcov [12]

Optimalizácia TSP pomocou ACO funguje nasledovne. Do prostredia úlohy (grafu) náhodne rozmietnime určitý počet mravcov. Necháme ich vytvárať okružné trasy tak, že im zavedieme pravidlo pre použitie hrán a zakážeme im znova navštíviť už navštívené uzly s výnimkou počiatočného pri uzatvorení cesty. Po ukončení tvorby okružných ciest, ak sa našlo lepšie riešenie ako doteraz, tak doteraz najlepšie riešenie aktualizujeme. Následne aktualizujeme hladinu feromónov na hranách. Uvedený postup niekoľkokrát zapakujeme.

Pseudo kód metódy ACO :

1. Inicializujte prostredie;
2. Inicializujte počiatočnú feromónovú stopu; //napr. cestou najbližších susedov
3. Vytvorte mravce;
4. Pokiaľ *počet replikácií* < *požadovaný počet replikácií* opakujte :
 - Nechajte všetky mravce zostrojiť riešenie;
 - Aktualizujte feromónovú stopu;
 - Ak sa našlo lepšie riešenie tak ho uložte;
 - Pripravte mravce na ďalšie replikáciu;

Aktualizácia feromónovej stopy:

1. Vykonajte vyprchanie pre všetky úseky;
2. Aplikujte feromónovú stopu každého mravca;

Novú hladinu feromónu τ_{ij} pre hranu $i - j$ po vyprchaní vypočítate pomocou vzťahu:

$\tau_{ij} = (1 - \rho) \tau_{ij}$, kde $\rho \in (0,1)$ je miera vyprchania a označenie i pre začiatok a j koniec hrany.

Novú hladinu feromónu τ_{ij} pre hranu po ktorej prešiel mravec k vypočítate ako:

$\tau_{ij} = \tau_{ij} + \frac{Q}{L_k(t)}$, kde $L_k(t)$ je dĺžka celej trasy mravca k , a Q je konštanta.

Rozhodovacie pravidlo pre pravdepodobnosť prechodu mravca z miesta i do j :

$$p_{ij}(t) = \frac{(\tau_{ij}(t))^\alpha \left(\frac{1}{d_{ij}}\right)^\beta}{\sum_{j \in N} (\tau_{ij}(t))^\alpha \left(\frac{1}{d_{ij}}\right)^\beta}, \text{ kde } \alpha, \alpha \geq 0 \text{ je váha ohodnotenia hodnoty feromónov,}$$

$\beta, \beta \geq 1$ je váha ohodnotenia hodnoty dĺžky hrany $i - j$, d_{ij} je dĺžka hrany.

Pre pravdepodobnosti platí $p_{ij}(t)$ platí $\sum_{j \in N} p_{ij}(t) = 1$. Vygenerovaním pseudonáhodného čísla $r \in (0,1)$, pomocou ktorého určíte úsek, ktorý má mravec použiť.

1.5 Úloha okružných jázd

Vehicle Routing Problem – VRP býva označovaná ako viacnásobný problém obchodného cestujúceho. Úlohou je určiť trasy vozidiel tak, aby boli uspokojené všetky požiadavky zákazníkov za minimálnych nákladov. Jednotlivé trasy začínajú a končia v depe, majú obslúžiť niekoľko zákazníkov, pričom v rámci trasy sa nesmie porušiť kapacita vozidla. VRP je preto zovšeobecnením TSP, ktorá môže byť doplnená o rôzne ďalšie podmienky a obmedzenia, ktoré úlohu viac približujú k reálnej praxi. Príkladom môže byť distribúcia komponentov zabezpečovaná z viacerých dep, s homogénnym alebo heterogénnym vozovým parkom a so zákazníckou požiadavkou na doručenie komponentov v určitom časovom intervale.

V praxi pri zbere odpadu sa stáva, že z kapacitných dôvodov vozidla (ale aj iných napr. časových) nie je možné pozbierať odpad jednou okružnou jazdou. Preto som sa rozhodol, že sa v mojej DP zameriam aj na úlohu okružných jázd s heterogénnym vozovým parkom.

1.5.1 Riešenia úlohy okružných jász

Podobne ako pri úlohe TSP existuje viacero metód na riešenie úlohy, ktoré rozdeľujeme do nasledujúcich kategórií viac sa môžete dočítať na [7. Janáček, 2016, s.193] :

1. Metódy primárneho zhľukovania (Cluster First–Route Second),
2. Metódy primárneho trasovania (First Route–Cluster Second),
3. Metódy výhodnostných koeficientov a vsúvania (Savings-Insertions),
4. Výmenné heuristiky (Improvement Exchange),
5. Metódy využívajúce matematické programovanie (Mathematical Programming Methods),
6. Interaktívne metódy (Man-Machine Approach),
7. Presné metódy (Exact Methods).

Heuristiky patriace do kategórie metód výhodnostných koeficientov a vsúvania, môžu byť postavené na primárnom aj duálnom princípe. Primárne heuristiky začínajú prípustným riešením, kde každý zákazník (prípadne skupina zákazníkov) má na začiatku vlastnú kyvadlovú jazdu, a postupným spájaním týchto jász sa získava nové prípustné riešenie. V duálnych heuristikách je na začiatku prázdna trasa (nepřípustné riešenie) do ktorej sa postupným vsúvaním pridávajú zákazníci. V oboch princípoch býva kritériom riešenia vloženie zákazníka (alebo spojenie jász), použitá hodnota výhodnostného koeficientu odhadujúceho dôsledky tohto výberu pre hodnotu účelovej funkcie. Do tejto kategórie metód patrí aj nasledujúci Clark-Wright algoritmus.

Sekvenčný algoritmus Clark-Wright (CW)

Pôvodný algoritmus [1964] Clarke-Wrightovej metódy [7. Janáček, 2016, s.208] začína vytvorením riešenia, kde sa pre každého zákazníka vytvorí samostatná kyvadlová jazda v tvare depo – zákazník - depo. Algoritmus prechádza od aktuálneho prípustného riešenia úlohy k ďalšiemu spojením dvoch jász. Pri jazdách sa rozlišujú zákazníci na dva nasledujúce typy. Zákazníci krajného typu sú zákazníci (prvý a posledný v jazde), ktorí sú priamo spojení s depom. Zákazníci vnútorného typu nie sú spojení priamo s depom. Spojenie dvoch jász funguje na základe spojenia cez dvoch krajných zákazníkov (i a j). Úsporu, ktorá vznikne týmto spojením možno vyjadriť nasledovne: $d_{is} + d_{sj} - d_{ij} = v_{ij}$. Kde d_{is} je

vzdialenosť medzi zákazníkom z prvej jazdy a depom, d_{sj} je vzdialenosť medzi depom a zákazníkom j z druhej jazdy, d_{ij} je vzdialenosť medzi zákazníkom i a j . Výhodnostný koeficient v_{ij} , vyjadruje ako moc sa oplatí spojiť dve jazdy do jedného celku. Jazdy sa následne spájajú na základe vypočítaných výhodnostných koeficientov, až po naplnenie maximálnej kapacity vozidla, ktoré má jazdu vykonať

Nevýhodou základného (paralelného) CW algoritmu je, že ho nie je možné aplikovať na VRP úlohu s heterogénnym vozovým parkom. Tento problém možno vyriešiť sekvenčnou verziou tohto algoritmu, ktorý je založený na zostupnom (od najväčšej po najmenšiu) zoradení áut podľa kapacít, pre ktoré sa postupne až do naplnenia kapacity auta vytvára okružná trasa. Sekvenčný algoritmus CW je možné popísať nasledujúcimi krokmi:.,

0. Inicializácia

Usporiadajte zoznam R doposiaľ nepoužitých vozidiel zostupne podľa ich kapacity. Pre všetky dvojice i, j zákazníkov zo zoznamu J zákazníkov s doposiaľ neuspokojenými požiadavkami vypočítajte výhodnostné koeficienty

*$v_{ij} = d_{is} + d_{sj} - d_{ij}$ a všetky **kladné** hodnoty vložte do zoznamu V .*

Chod'te na krok 1.

1. Inicializácia novej trasy

*Ak je zoznam J zákazníkov s doposiaľ neuspokojenými požiadavkami prázdny, **skončíte**, výsledné **prípustné** riešenie je vytvorené dosiaľ zostavenými trasami vozidiel.*

*Inak, ak je zoznam R doposiaľ nepoužitých vozidiel prázdny, **skončíte**, výsledné **neprípustné** riešenie je vytvorené dosiaľ zostavenými trasami s mierou neprípustnosti danou súčtom požiadaviek zákazníkov v zozname J .*

Inak z R vyberte vozidlo r a inicializujte mu súčasnú trasu $S_r = \emptyset$.

Chod'te na krok 2.

2. Inicializácia novej jazdy

Zo zoznamu J vyberte zákazníka j , ktorý má najväčšiu požiadavku b_j a ktorý spĺňa $b_j \leq K_r$.

*Ak taký zákazník neexistuje **Chod'te na krok 1.***

Inak inicializujte $B = b_j$ súčet požiadaviek spracovávanej jazdy vozidla r ,

inicializujte indexy krajných zákazníkov spracovávanej jazdy $k_1 = k_2 = j$.

*Aktualizujte aj trasu vozidla $S_r = S_r \cup \langle s, j, s \rangle$ pridaním kyvadlovej jazdy $s - j - s$. **Chod'te na krok 3.***

3. Zväčšenie spracovávanej jazdy

Vyberte zo zoznamu V najväčší koeficient v_{ij} , kde $i \in \{k_1, k_2\}$ a $j \notin \{k_1, k_2\}$.

*Ak sa taký koeficient v zozname V nenachádza, **chod'te na krok 2.***

Inak, ak sa $B + b_j \leq K_r$, tak vložte zákazníka j do spracovávanej jazdy vozidla r medzi zákazníka i a depo s .

Ak je $i = k_1$, položte $k_1 = j$ inak $k_2 = j$.

Ďalej aktualizujte $B = B + b_j$ a aktualizujte zoznam V vybraním výhodnostných koeficientov v_{kj} , resp. v_{jk} , kde $k \in \{k_1, k_2\}$ a pokiaľ zákazník i prestal byť krajným zákazníkom (tj. $i \notin \{k_1, k_2\}$), vyberte aj všetky v_{ih} a v_{hi} .

Opakujte krok 3. “ [7. Janáček, 2006, s.209-210]

1.6 Úloha o P-mediáne

Patrí medzi základné optimalizačné lokačné problémy. Úlohou daného problému je rozmiestniť zadaný počet obslužných centier na predom vybrané možné pozície tak, aby súhrnná vzdialenosť medzi zákazníkmi a k nim priradeným strediskom bola čo najmenšia. Najpoužívanejším rozšírením P-medián úlohy je rozšírenie na Kapacitný p-medián. V tomto rozšírení sú pridané kapacitné obmedzenia pre jednotlivé obslužné centrá a každému zákazníkovi je pridelená požiadavka, ktorá má byť uspokojená práve z jedného obslužného centra. Úlohy založené na p-mediáne majú v praxi široké využitie a ich aplikácie sa využívajú v mnohých odvetviach ako napr. v logistike, pri plánovaní alebo v priemysle. Konkrétnymi príkladmi môže byť umiestňovanie staníc hasičského zboru, rozmiestňovanie dobíjacích staníc, ale aj rozmiestnenie zberných miest pre triedený odpad.

1.6.1 Matematický model pre úlohu Kapacitného p -mediánu

Je zadané p – počet centier ktoré treba umiestniť, q - počet možných miest pre umiestnenie centra, n – počet zákazníkov, matica vzdialenosti D , kde d_{ij} vyjadruje vzdialenosť medzi centrom i a zákazníkom j , c_i – kapacita strediska i a b_j - požiadavka zákazníka j .

Rozhodovacie premenné vyzerajú nasledovne: $x_{ij} \in \{0,1\}$, vyjadruje či je zákazník j obsluhovaný z centra i . Ak áno, tak $x_{ij} = 1$, inak 0. Premenná $y_i \in \{0,1\}$, vyjadruje, či bude centrum vybudované na mieste i . Ak áno, tak $y_i = 1$, inak 0.

Účelová funkcia:

$$\min \sum_{i=1}^q \sum_{j=1}^n d_{ij} x_{ij}$$

Podmienka zabezpečujúca, že sa vybuduje práve p - centier:

$$\sum_{i=1}^q y_i = p$$

Podmienky zabezpečujúce, že zákazník je obsluhovaný práve z jedného centra:

$$\sum_{i=1}^q x_{ij} = 1 \quad \forall j \in 1..n$$

Podmienky zabezpečujúce, že sa nepresiahne kapacita centier:

$$\sum_{j=1}^n b_j x_{ij} < a_i \quad \forall i \in 1..q$$

Podmienky zabezpečujúce, aby zákazník nebol obsluhovaný z miesta, kde nie je umiestnené centrum.

$$x_{ij} \leq y_i \quad \forall i = 1..q, j = 1..n$$

1.6.2 Riešenia P-mediánu

Problém P-mediánu sa zaradzuje medzi NP-ťažké úlohy. Podobne ako predchádzajúce uvedené úlohy, je ho možné riešiť pomocou exaktných alebo heuristických metód. Exaktné algoritmy zaručujú nájdenie optimálneho riešenia ak existuje, ale je ich možné použiť len pre úlohy menšieho rozsahu. Pri rozsiahlejších úlohách zlyhávajú kvôli časovej zložitosti alebo nedostatku pamäti. Patria sem napríklad algoritmy založené na metóde vetiev a hraníc. Heuristické metódy síce nezaručujú nájdenie optimálneho riešenia, ale umožňujú sa k nemu aspoň priblížiť v rozumnom čase. Výhodou heuristík je, že ich je možné použiť na široké spektrum optimalizačných úloh a vynikajú svojou jednoduchosťou pre implementáciu a pochopenie. [14]

Genetický algoritmus

Patrí medzi evolučné metaheuristiky založené na princípoch evolučnej Darwinovej teórie. Základná terminológia genetického algoritmu vypadá nasledovne. Riešenie sa nazýva jedinec (alebo chromozóm), množina riešení (chromozómov) je populácia, nad ktorou sú definované operácie kríženia, mutácie a selekcie. Chromozóm je často reprezentovaný poľom znakov, ktoré sa volajú gény a hodnotu účelovej funkcie daného chromozómu voláme fitness hodnota. [14]

- **Selekcia** – je výber jedincov na operáciu kríženia. Je založená na rešpektovaní pravidla prirodzeného výberu Darwinovej teórie. To znamená, že jedinci s lepšou fitness hodnotou majú väčšiu šancu predať svoju genetickú informáciu. Nasledujúci vzorec vyjadruje pravdepodobnosť, že jedinec i bude vybraný do kríženia $p_i = \frac{f_i}{\sum_{j=1}^N f_j}$, kde f_i je fitness hodnota jedinca i a $\sum_{j=1}^N f_j$ je súčet všetkých fitness hodnôt celej populácie
- **Kríženie** – vybraní jedinci (väčšinou dvaja) vytvoria nového jedinca (potomka), pomocou predania svojich génov. Existuje viacero spôsobov ako modelovať operáciu kríženia. V princípe sú však založené na tom, že nový jedinec má 50% génov z rodiča 1. a 50% z rodiča 2.. Nový jedinec, však musí reprezentovať prípustné riešenie úlohy.

- **Mutácia** – operácia mutácie slúži na udržanie diverzifikovanej populácie, čo zamedzí algoritmu uviaznuť v lokálnom optime. Vybranému jedincovi sa zmení hodnota náhodne vybraným génom. Pravdepodobnosť uskutočnenia mutácie musí byť nízka (inak by sa to zdeformovalo len na jednoduché prehľadávanie riešení), býva v rozsahu od 0,1% do 5%. Mutáciu možno uskutočniť buď na novom jedincovi (po krížení), alebo na náhodne vybranom jedincovi z populácie. Zmutovaný jedinec samozrejme musí reprezentovať prípustné riešenie úlohy.

Pseudo kód genetického algoritmu:

1. Inicializujte počiatočnú populáciu;
2. V cykle opakujte:
 - Vykonajte selekciu jedincov;
 - Vykonajte kríženie vyselektovaných jedincov;
 - Vykonajte mutáciu;
 - Vypočítajte novým jedíncom fitness hodnoty a aktualizujte celkovú populáciu o nových jedincov;
3. Ak je splnená ukončovacia podmienka, ukončíte cyklus; //napr. vypršaný čas trvania algoritmu;
4. Jedinec s najlepšou fitness hodnotou reprezentuje najlepšie nájdené prípustné riešenie;

2 Ciele práce

Cieľom mojej diplomovej práce je zanalyzovať systém fungovania sekcie odpadového hospodárstva (OH) vybraného mesta a následne navrhnúť a vytvoriť informačný systém pre jeho bežné potreby. Výsledný informačný systém je určený pre potreby OH mesta Rajec, ale po zmenení inicializačných dát (napr. umiestnia stojísk, dát o cestnej sieti), ho môžeme využiť pre ľubovoľné mesto alebo obec nielen na Slovensku.

Výsledná aplikácia zjednodušuje a urýchľuje základné činnosti pracovníka sekcie OH, poskytuje mu vhodný nástroj pre navrhovanie a manažovanie zberných jazd separovaného odpadu. Pracovníkovi OH ďalej poskytuje monitoring aktuálnych údajov o jednotlivých zberných nádobách a vozidlách vozového parku. Monitoring zberných nádob je postavený na základe SMS komunikácie medzi smart senzormi Sensoneo (určených k meraniu naplnenosti) a aplikáciou. Ďalšou možnosťou je monitorovanie naplnenosti bežnými občanmi mesta, ktorí pomocou poslania SMS v správnom tvare o aktuálnom stave nádoby, sú nápomocní svojmu mestu s optimalizáciou nákladov OH.

V aplikácii sú k dispozícii rôzne štatistiky nielen o uskutočnených zberných jazdách, ale aj o jazdách vozidiel vozového parku iného charakteru. Štatistiky obsahujú historický vývoj naplnenosti zberných nádob. Doplnkovým nástrojom aplikácie pre OH je optimizér pre rozmiestňovanie zberných miest. Vyššie uvedené funkcionality a nástroje aplikácie sú nápomocné pri minimalizácii nákladov OH, čím by sa mohol zastaviť alebo aspoň spomaliť trend zvyšovania sa poplatkov za odvoz odpadu pre všetkých občanov mesta.

Potrebné informácie pre realizáciu projektu som čerpal z oficiálnej web stránky a priamo zo sekcie OH mesta Rajec. Dopĺňajúce informácie som získal z verejných databáz ako napr. OpenStreetMap a doplnil ich o informácie mnou vypozerované ako funguje zber odpadu v iných mestách a obciach na území SR.

Po získaní dát, informácií a požiadaviek potrebných pre tvorbu informačného systému, som ich zanalyzoval a spracoval. Prvou úlohou bolo zo získaných dát o cestnej infraštruktúre vyfiltrovať dôležité údaje a následne ich vložiť do vhodnej údajovej štruktúry reprezentujúcej graf. Z grafu vypočítavam maticu vzdialenosti, z ktorej sa vytvárajú submatice, pre jednotlivé optimalizačné úlohy.

Najzložitejšou a najdôležitejšiu výzvou je vhodne vybrať, naimplementovať a následne otestovať metódy riešenia pre optimalizačné úlohy, ktoré hotová aplikácia v praxi využíva. Po splnení predchádzajúcej výzvy pokračujem s vývojom aplikácie navrhnutím databázového modelu a štruktúry celého informačného systému. Postupným vytváraním a s následným spájaním komponentov aplikácia dostáva svoju finálnu funkčnosť a podobu. Hotovú aplikáciu je možné otestovať, odladiť a prípadné zistené problémy opraviť. Takto vytvorený informačný systém nasadzujem do skúšobnej prevádzky, čím sa ukáže, či je systém použiteľný v reálnej praxi alebo ho je treba ešte doplniť o nové funkcionality.

3 Riešenie

3.1 Získanie a spracovanie počiatkových dát

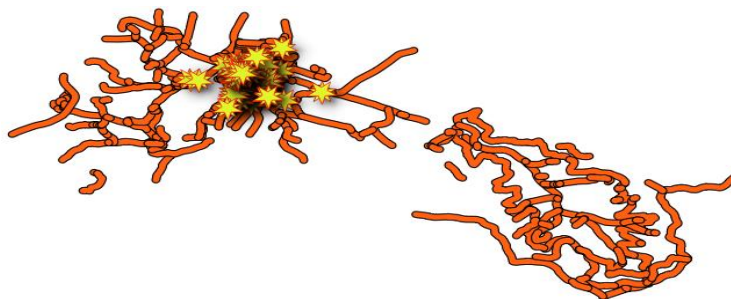
Na základe osobného stretnutia so zástupcami MsÚ Rajec zodpovednými za sekciu odpadového hospodárstva, som získal základné požiadavky pre informačný systém, ktoré som už podrobnejšie opísal v časti 1.1 Aktuálny stav odpadového hospodárstva v Rajci. Na stretnutí som obdržal dokument, v ktorom boli zapísané adresy jednotlivých zberných miest s príslušným počtom zberných nádob pre príslušný druh opadu. Uvedený dokument som následne spracoval v Exceli a doplnil som k stojiskám príslušné súradnice získané z uvedených adries. Získanie a následne spracovanie dát o cestnej infraštruktúre opíšem v nasledujúcej kapitole.

3.1.1 QGIS získanie a spracovanie dát o cestnej infraštruktúre

QGIS je multiplatformový open-source geograficky informačný systém, ktorý umožňuje analyzovať a spracovávať geografické dáta. Je známy svojou veľkou komunitou užívateľov a enormným počtom voľne dostupných zásuvných modulov, ktoré je možné využiť pri riešení rôznych problémov súvisiacich so spracovaním geografických dát.

V novo-vytvorenom projekte v programe QGIS som vytvoril novú vektorovú bodovú vrstvu „*RealneStojiska.shp*“ zo súboru so stojiskami s príslušnými súradnicami. Následne som stiahol pomocou doplnku „*QuickOSM*“ z OpenStreetMap voľne dostupnú vektorovú čiarovú vrstvu cestnej infraštruktúry a pomenoval ju „*RajecVsetkyCesty.shp*“.

Na obrázku 11. je vidieť vytvorené vektorové vrstvy, kde žlté hviezdy reprezentujú stojiská a oranžové čiary cestnú sieť, ktorá okrem bežných typov ciest (pre vozidlá) obsahuje aj chodníky, poľné cesty a pod., ktoré bolo treba odstrániť.



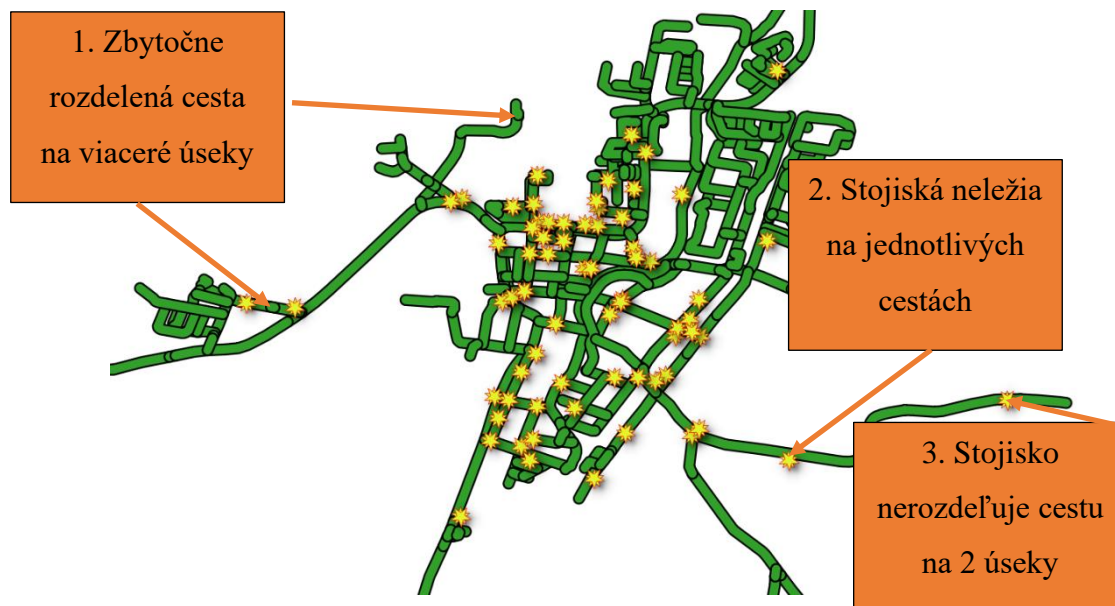
Obrázok 11. QGIS – počiatkové vrstvy

Nepotrebné čiary z vrstvy „RajecVsetkyCesty.shp“ som odstránil vyselektovaním cez SQL dopyt pomocou výrazu "highway" in ('living_street' , 'primary' , 'residential' , 'secondary' , 'service' , 'tertiary') a tým mi vznikla nová vektorová vrstva „RajecRelevantneCesty.shp“ (obrázok 12. – zelené cesty). Z obrázku cestnej siete vyplynulo, že vrstva „RajecRelevantneCesty.shp“ je nesúvislá a musel som nesúvislosti ručne odstrániť pomocou funkcie „Select Feature(s)“ čím mi vznikla nová vrstva „RajecRelevantneCestyOcistene.shp“ (obrázok 13).



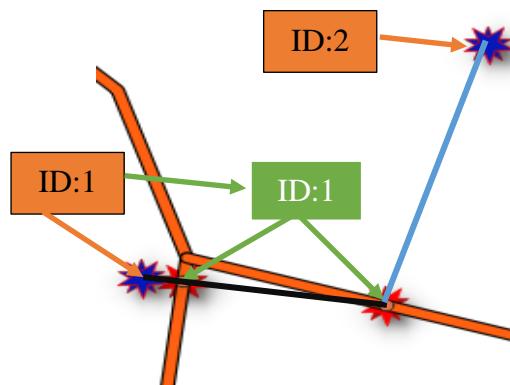
Obrázok 12. QGIS - Zelené – relevantné cesty

Na nasledujúcom obrázku sú zobrazené hlavné problémy, ktoré bolo treba vyriešiť, aby som získal dáta vhodné na import do grafovej údajovej štruktúry.



Obrázok 13. Hlavné problémy spracovania dát

Prvý problém som vyriešil pomocou nástroja „Join Multiple Lines“, ktorý spojil označené úseky do jednotného celku. Riešením problému dva bolo použiť funkciu „Find Closest Point For Each Feature“, ktorá vytvorila z vrstiev „RealneStojiska.shp“ a „RajecRelevantneCestyOcistene.shp“ novú bodovú vrstvu „RealneStojiskaNaHrane.shp“, ktorá zobrazuje najbližšie body na cestnej sieti od jednotlivých stojísk. K novej bodovej vrstve som pripojil z atribútovej tabuľky vrstvy „RealneStojiska.shp“ stĺpec „id_adresy“ (reprezentuje identifikátor stojiska) pomocou funkcie „Join Attributes By Nearest“. Následne som skontroloval, či sa mi v pripojenom stĺpci nevyskytli duplicity ktoré som opravil priamym prepísaním hodnoty v atribútovej tabuľke vrstvy. Duplicita by znamenala, že sa k bodu z „RealneStojiskaNaHrane.shp“ pripojil nesprávny bod z vrstvy „RealneStojiska.shp“, kvôli tomu, že funkcia hľadá najbližší bod na základe vzdušnej cesty (čierna úsečka je kratšia ako modrá). K vrstve „RealneStojiskaNaHrane.shp“ som pripojil cez stĺpec „id_adresy“ stĺpec „x“ a „y“ (reprezentácia súradnice) z vrstvy „RealneStojiska.shp“.



Obrázok 14. Fungovanie funkcie „Join Attributes By Nearest“.

Pomocou funkcie „Network/Connect Nodes To Lines“ som rozdelil vektorovú vrstvu „RajecRelevantneCestyOcistene.shp“ v miestach priesečníkov bodov z vrstvy „RealneStojiskaNaHrane.shp“. Následne som pridal do atribútovej tabuľky nové stĺpce pomocou atribútovej kalkulačky:

- „dlzka“ - reprezentujúci reálnu dĺžku čiary pomocou príkazu ($\$length$).
- „startPoint“ - reprezentujúci začiatkový bod cesty pomocou príkazu `'[||to_string(start_x)||','||to_string(start_y)||']'`

- „endPoint“ - reprezentujúci koncový bod cesty pomocou príkazu `'[|to_string(end_x)|','|to_string(end_y)|']'`
- a prázdne stĺpce „start_id_adresy“ a „end_id_adresy“

Ďalej bolo potrebné dvakrát spojiť vrstvu „RajecRelevantneCestyOcistene.shp“ cez stĺpce „startPoint“ a „endPoint“ s vrstvou „RealneStojiskaNaHrane.shp“ a do stĺpcov „start_id_adresy“ a „end_id_adresy“ vložiť príslušné „id_adresy“ z vrstvy „RealneStojiskaNaHrane.shp“. Odstránením zbytočných stĺpcov atribútovej tabuľky a uložením vrstvy do súboru vo formáte GEOJSON som získal vhodné dáta na ďalšie použitie. Na nasledujúcom obrázku je zobrazená atribútová tabuľka výslednej vrstvy. Najdôležitejšie stĺpce tabuľky sú „dlzka“ reprezentujúca dĺžku danej cesty v metroch, „oneway“- či sa jedná o jednosmernú alebo obojsmernú cestu a stĺpce „start_id_adresy“ a „end_id_adresy“, ktoré môžu nadobúdať hodnoty $\{NULL\} \cup \{0, \dots, n\}$, kde n je počet stojísk OH. Kde „NULL“ znamená, že cesta nezačína alebo nekončí v stojisku a v zbernom dvore.

	osm_type	abutters	highway	name	oneway	dlzka	fid	start_x	start_y	end_x	end_y	startPoint	endPoint	START_ID_adresy	end_ID_adresy
1	way	NULL	residential	Fuškáková	NULL	29,61307	347	18,644201	49,089706	18,644288	49,089957	(2:18.644201,49...	(2:18.644288,49...	0	NULL
2	way	NULL	residential	NULL	NULL	227,23194	349	18,622542	49,08702	18,624552	49,086666	(2:18.622542,49...	(2:18.624552,49...	1	2
3	way	NULL	residential	Holíčho	yes	100,08693	367	18,635862	49,090226	18,636759	49,090172	(2:18.635862,49...	(2:18.636759,49...	10	12
4	way	NULL	residential	M.R. L'etárika	NULL	42,7124	366	18,634556	49,090159	18,634535	49,089776	(2:18.634556,49...	(2:18.634535,49...	11	NULL
5	way	NULL	residential	Holíčho	yes	58,80496	368	18,636759	49,090172	18,637285	49,090119	(2:18.636759,49...	(2:18.637285,49...	12	13
6	way	NULL	residential	Holíčho	yes	35,64723	369	18,637285	49,090119	18,637603	49,090087	(2:18.637285,49...	(2:18.637603,49...	13	NULL
7	way	NULL	living_street	NULL	NULL	96,31177	371	18,638328	49,090789	18,637467	49,090869	(2:18.638328,49...	(2:18.637467,49...	14	NULL

Obrázok 15. Atribútová tabuľka výslednej vrstvy.

3.1.2 Implementácia grafu a načítanie vstupného GEOJSON súboru

Balíček graf je určený na reprezentáciu grafu (implementovaného pomocou množiny hrán a množiny uzlov), ktorého najdôležitejšie funkcie sú zostrojenie grafu zo vstupného GEOJSON súboru a výpočet matice vzdialenosti medzi uzlami grafu. Stručný popis tried a najdôležitejších atribútov metód vypadá nasledovne:

Trieda Suradnice - reprezentujúca suradnice:

Atribúty: *double x* – zemepisná šírka, *double y* – zemepisná dĺžka

Metódy: *static double vyratajVzdialenostKM(Suradnice s1, Suradnice s2)* – vyrátanie vzdialenosti v km medzi dvomi súradnicami pomocou Haversinovho vzorca.

Trieda Uzol - reprezentuje uzol grafu:

Atribúty: *Suradnice sur* – súradnice bodu, *int id* – identifikátor uzla.

Trieda Hrana -reprezentuje Hranu v grafe:

Atribúty: *String id* – identifikátor hrany, *Uzol srcUzol* – začiatkový uzol hrany, *Uzol destUzol* – koncový uzol hrany, *int dlzka* – dĺžka hrany v km, *LinkedList<Suradnice> suradnice* - súradnice začiatku, zlomových bodov a konca hrany.

Trieda Graf – reprezentácia grafu:

Atribúty: *int maxIdHrany* – hodnota maximálneho identifikátora hrany, *int maxIdUzla* – hodnota maximálneho identifikátora uzla, *List<Hrana> hrany* – zoznam hrán, *List<Uzol> uzly*- zoznam uzlov, *int[][] id_Dmatrix* – matica vzdialenosti, *String[][] s_Dmatrix* – matica najkratších ciest (napr. *s_Dmatrix[5][10]* = “5,6,8,9,10“ znamená, že najkratšia cesta z uzla 5 do uzla 10 je postupne cez 6-7-8-9)

Metódy: *Uzol vložUzolDoGrafu(double x_sur, double y_sur, int noveId)* – pridanie nového uzla do grafu, *void vložHranuDoGrafu(Uzol srcUzol, Uzol destUzol, int dlzka, boolean jednosmerka, LinkedList<Suradnice> suradnice)* – pridanie novej hrany do grafu, *void importGrafuZGEOJSON(String urlPath, int pocetStojisk)* – import GEOJSON súboru popísaného v prechádzajúcej kapitole, *int[][] getSubMatrixFromID(ArrayList<Integer> idecka)* – vracia subMaticu vzdialenosti pre vložené identifikátory uzlov, *LinkedList<Integer> vratKompletneCestu(LinkedList<Integer> subCesta)*- vráti celú cestu získanú z *s_Dmatrix* pre vloženú *subCestu*, *LinkedList<Hrana> vratHrany(LinkedList<Integer> cesta)* - vráti všetky hrany vlozenej cesty, *int vyratajUsporu(int vsuvaneMiesto_predchodca, int vyberaneMiesto_predchodca, int veberaneMiesto_nasledovnik, int presuvany, int vsuvaneMiesto_Nasledovnik)* – vyráta úsporu alebo predĺženie cesty pri výmene hrán, metódy a na načítanie a uloženie štruktúry do súboru a iné pomocné metódy.

Trieda Matice – pomocná trieda na prenášanie referencií na maticu vzdialenosti a maticu najkratších ciest.

Trieda Dijstra – implementácia Dijkstrovho algoritmu, pomocou ktorého sa vypočíta matica vzdialenosti a matica najkratších ciest.

Import grafu z GEOJSON súboru:

Keďže jeden riadok vstupného súboru reprezentuje jednu hranu grafu, tak som pomocou voľne dostupnej knižnice *json-simple-1.1.jar* vyparsoval nasledujúce dáta:

- *LinkedList<Suradnice> suradnice* – obsahuje súradnice začiatku, bodov zlomov a koniec hrany
- *boolean jednosmerka* – true, ak sa jedná o jednosmernú cestu
- *String start, String end* – informácia, či hrana začína alebo končí na stojisku, ak „null“ hrana nekončí na stojisku, inak reťazec obsahuje identifikátor stojiska
- *int dlzka* – dĺžka hrany v metroch.

Metóda *Uzol vlozUzolDoGrafu(double x_sur, double y_sur, int noveId)* kontroluje, či sa už v štruktúre nenachádza uzol zo zadanými parametrami, ak áno, tak ho vráti, inak vráti novo vytvorený uzol zo zadaných parametrov, ktorý sa pridal do štruktúry. Metóda *vlozHranuDoGrafu(Uzol srcUzol, Uzol destUzol, int dlzka, boolean jednosmerka, LinkedList<Suradnice> suradnice)* - kontroluje, či sa v grafe hrana s uvedenými parametrami nenachádza, ak nie, tak vytvorí novú hranu, ktorú následne pridá do štruktúry. Ak ide o obojsmernú hranu (*jednosmerka==false*), tak metóda pridá ďalšiu hranu s prehodeným poradím uzlov a LinkedList-om *suradnice*. Pomocou nasledovného kódu prebieha postupné vkladanie hrán a uzlov do grafu:

```
Uzol src;
if (start.compareTo("null") == 0) { // ak hrana nezacina v stojiskovom uzle
    src = vlozUzolDoGrafu(suradnice.getFirst().getX(), suradnice.getFirst().getY(), Integer.MAX_VALUE);
} else { //hrana zacina v stojiskovom uzle
    src = vlozUzolDoGrafu(suradnice.getFirst().getX(), suradnice.getFirst().getY(), Integer.parseInt(start));
}
Uzol dest;
if (end.compareTo("null") == 0) { //ak hrana nekonci v stojiskovom uzle
    dest = vlozUzolDoGrafu(suradnice.getLast().getX(), suradnice.getLast().getY(), Integer.MAX_VALUE);
} else { //ak hrana konci v stojiskovom uzle
    dest = vlozUzolDoGrafu(suradnice.getLast().getX(), suradnice.getLast().getY(), Integer.parseInt(end));
}
vlozHranuDoGrafu(src, dest, dlzka, jednosmerka, suradnice);
```

Obrázok 16. Vkladanie uzlov a hrán do grafu

3.2 Výber databázového systému a návrh databázového modelu

MySQL je veľmi populárny relačný databázový systém, ktorý je slobodný a otvorený. Patrí pod bezplatnú licenciu GPL. Vybral som si ho z dôvodu dostatočného výkonu, kvalitnej technickej podpory, bezplatnej dostupnosti a z dôvodu doplnenia si mojich znalostí o novú databázu. Databázový model som navrhol v Toad Data Modeler-i, ktorý som už poznal z prechádzajúceho štúdia databázových systémov. Databázu som rozširoval o nové funkcionality pomocou vývojového prostredia MySQL Workbench.

Opis databázového modelu:

1. **Tabuľka stav_vozidla** – je určená na evidenciu definovaných stavov vozidiel vozového parku:

Stĺpce tabuľky: *INT id_stavu_vozidla primárny kľúč (ďalej PK) Autoincrement (ďalej AI), VARCHAR(20) stav – popis stavu: { 'Pripravené', 'Na jazde', 'Mimo prevádzky', 'Vyradené' }.*

2. **Tabuľka stav_jazdy** – je určená na evidenciu definovaných stavov jazd:

Stĺpce tabuľky: *INT id_stav_jazdy PK AI, VARCHAR(45) stav – popis stavu jazdy: { 'nový', 'prebiehajúci', 'ukončený', 'uzavretý', 'servisová', 'servisová uzavretá' }.*

3. **Tabuľka druh_odpadu** – je určená na evidenciu definovaných druhov odpadu:

Stĺpce tabuľky: *INT id_odpadu PK AI, VARCHAR(20) nazov_odpadu – názvy odpadov: { 'Komunálny odpad', 'Kovy', 'Olej', 'Papier', 'Plast', 'Sklo', 'Tetra paky', 'Textil' }.*

4. **Tabuľka stojisko** – evidencia zberných stojísk:

Stĺpce tabuľky: *INT id_stojiska PK AI, POINT gps, VARCHAR(100) adresa.*

5. **Tabuľka kontajner** – evidencia zberných nádob:

Stĺpce tabuľky: *INT id_kontajnera PK AI, INT id_stojiska cudzí kľúč(d'alej FK), INT id_odpadu FK, INT aktualnyObjem, INT maxObjem, DATETIME poslednaAktualizacia, DATE posledneVyprazdnenie.*

6. **Tabuľka Historia** – evidencia historického stavu objemu zberných nádob:

Stĺpce tabuľky: *INT id PK AI, INT id_kontajnera FK, DATETIME datum, INT stav.*

7. **Tabuľka Vozidlo** – evidencia vozidiel OH:

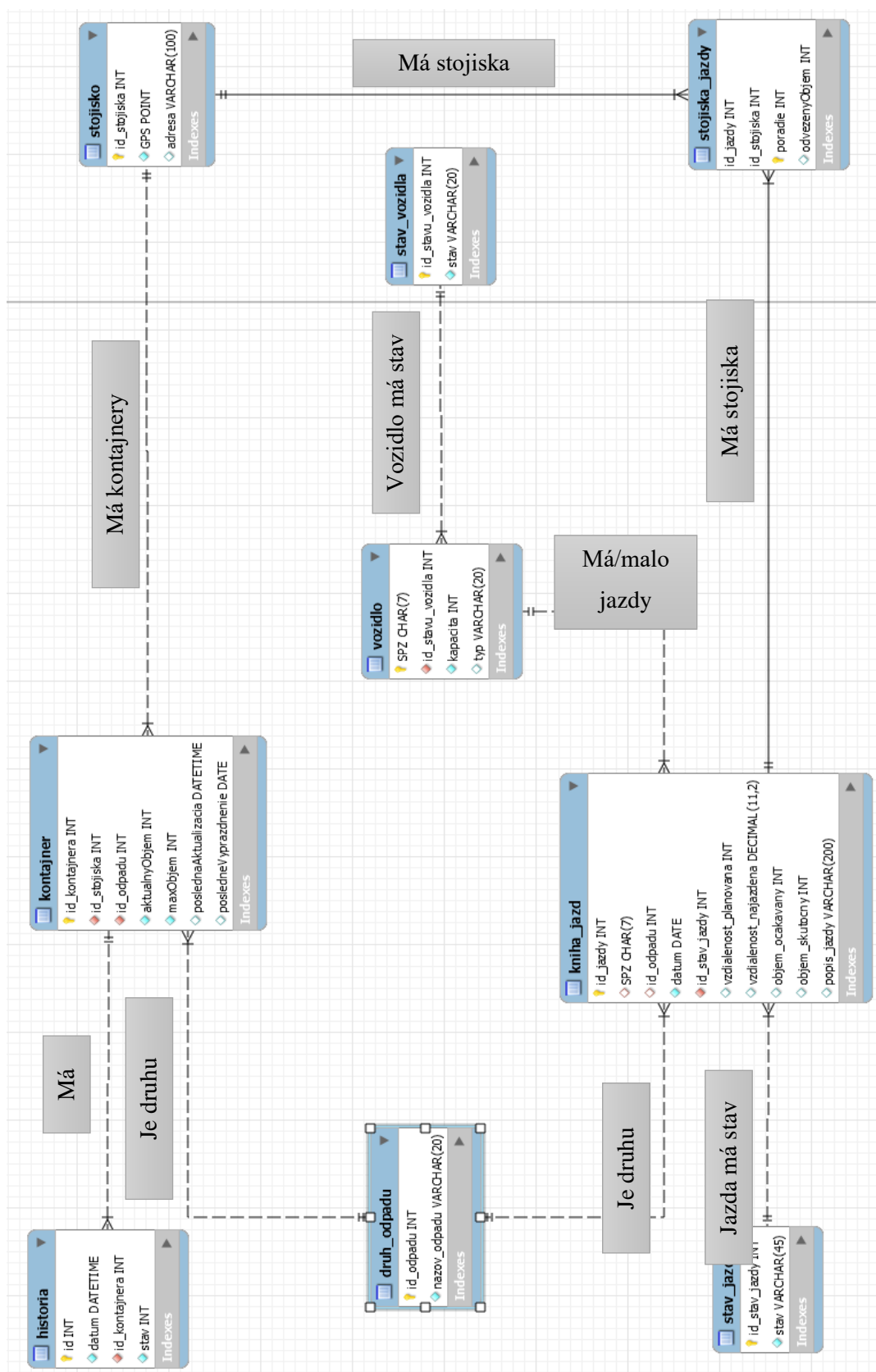
Stĺpce tabuľky: *CHAR(7) spz PK, INT id_stavu_vozidla, FK, INT kapacita*

8. **Tabuľka Kniha_jazd** – evidencia všetky jazdy OH:

Stĺpce tabuľky: *INT id_jazdy PK AI, CHAR(7) spz FK, INT id_odpadu FK, DATE datum, INT id_stav_jazdy FK, INT vzdialenost_planovana, decimal(11,2) vzdialenost_najazdena, INT objem_ocakavany, INT objem_skutocny, VARCHAR(200) popis_jazdy.*

9. **Tabuľka Stojiska_jazdy** – evidencia stojísk jednotlivých jazd OH:

Stĺpce tabuľky: *INT id_jazdy FK, INT id_stojiska FK, INT poradie, INT odvezený objem.*



Obrázok 17. Schéma databázy informačného systému

3.3 Optimalizácia zberných jázd

Prvou snahou ako vyriešiť úlohu obchodného cestujúceho v mojom IS bol pokus o nájdenie exaktného riešenia.

Prvou možnosťou bolo použiť solvér na riešenie úloh lineárneho matematického programovania. Táto možnosť zlyhala na tom, že sa mi nepodarilo nájsť voľne dostupný solvér, ktorý by sa dal importovať do vyvíjaného informačného systému. Pre testovacie potreby a pre porovnanie výsledkov mnou naimplementovaných heuristik a algoritmov som využil pod akademickou licenciou (časové obmedzenie použitia) komerčný solvér Gurobi Optimizer. Solvér sa dal bezproblémovo importovať a použiť, ale problémom by bola kúpa komerčnej licencie, ktorá by stála tisíce dolárov.

Druhou možnosťou bolo nájsť taký exaktný algoritmus, ktorý som následne naimplementoval a otestoval. Na riešenie úlohy som sa nakoniec rozhodol použiť metódu vetiev a hraníc, ktorá je podrobnejšie opísaná v kapitole 1.4.2 Riešenia TSP. Naimplementovanú metódu MVH som testoval pre maticu vzdialenosti stojísk v Rajci. Prekvapivým zistením bolo, že algoritmus zvládne v sekunde zistiť optimálne riešenie len pre matice vzdialenosti do veľkosti 15 x 15. Na maticiach väčšieho rozsahu algoritmus zlyhal z dôvodu prečerpania pamäti. Uvedené zistenie som sa neúspešne pokúsil potvrdiť alebo vyvrátiť pomocou odborných článkov uvedených na internete. Vo všetkých článkoch, kde bola rozoberaná problematika metódy MVH pre úlohu obchodného cestujúceho, bola metóda teoreticky rozoberaná na menších príkladoch a pre väčšie siete som aplikáciu MVH nenašiel. Následne som zistenie konzultoval s doc. Ing. M. Kohánim, PhD., ktorý mi moje zistenie potvrdil. Z toho dôvodu som sa rozhodol, že pre riešenie rozsiahlejších matíc, použijem metaheuristiku ACO, ktorú som následne naprogramoval a vyskúšal. Riešenia získané pomocou ACO a z Gurobi solvéru som porovnal a odkonzultoval s doc. Ing. M. Kohánim, PhD., ktorým mi odobril moje navrhované riešenie.

Ďalej som sa zamyslel ako by som mohol aplikáciu OH viac rozšíriť a napadlo mi, že by bolo vhodné pridať možnosť optimalizovať zber jedného druhu odpadu pre viaceré vozidlá (napr. aj rôznej kapacity) naraz. Preto som sa rozhodol pridať heuristiku Clark-

Wright založenú na sekvenčnom prístupe, čo umožní plánovať zber viacerým vozidlám s rôznou kapacitou alebo rôznou prioritou.

Algoritmus fungovania optimalizácie zberných jázd

Na nasledujúcom obrázku je zobrazený algoritmus fungovania zberných jázd, kde *pocetStojisk* – je počet stojísk ktoré treba navštíviť, „*pocetStojisk+1 <= 15*” – lebo v matici vzdialenosti musí byť aj zaradené depo (počiatočný a koncový uzol), *< 15* – lebo neimplementovaná metóda MVH nedokáže vypočítať úlohy, kde sa pracuje s väčšou maticou vzdialenosti (vysvetlené vyššie). Podmienka „*sumPoziadaviek < vozidla.get(0).kapacita*” – kontroluje, či sa zmestí všetok odpad do prvého vozidla v zozname usporiadaného podľa priority a kapacity.

```

if (pocetStojisk+1 <= 15 && sumPoziadaviek < vozidla.get(0).kapacita) {
    //optimalizacia s Metodou vetiev a hranic
} else if (stojiska.size() > 15 && sumPoziadaviek < vozidla.get(0).kapacita) {
    //optimalizacia s Koloniou mravcov
} else {
    //optimalizacia sekvenčnym ClarkWright
    for (Trasa trasa : cw.getTrasy()) {
        if (trasa.getZakazniciTrasy().size() > 15) {
            //doptimalizacia s Koloniou mravcov
        } else {
            //doptimalizaciou Metodou vetiev a hranic
        }
    }
}

```

Obrázok 18. Algoritmus optimalizácie zberných jázd

3.3.1 Implementácia modelu TSP v Gurobi optimizéri.

Na oficiálnej web stránke sa nachádza výukový príklad [5] modelu obchodného cestujúceho pre symetrickú cestnú sieť, získanú pomocou vyrátania euklidovskej vzdialenosti medzi náhodne vygenerovanými súradnicami bodov. Uvedený model som musel upraviť, aby som ho mohol použiť. Model v Gurobi vychádza z matematického modelu opísaného v kapitole 1.4.1 Matematický model TSP, ktorý je doplnený o metódu *callback* vďaka ktorej sa urýchľuje doba trvania výpočtu. Urýchlenie výpočtu je založené na tom, že obmedzenia, ktoré sa zriedkavo porušujú sa nastavujú do módu „lazy“ čo znamená,

že sa kontrolujú až v momente, ak súčasné riešenie porušuje niektoré z nich. V prípade keď sa pri vetvení nájde riešenie, ktorého $|S| < n$, riešenie obsahuje podcykly do modelu sa pridá obmedzenie na zamedzenie podcyklov, čím sa riešenie vylúči z prehľadávaného priestoru. Metóda *callback* a model zapísaný v Gurobi solvéri vypadá nasledovne:

```
int n = vars.length;
int[] tour = findsubtour(getSolution(vars));
if (tour.length < n) {
    // Obmedzenie na cykly
    GRBLinExpr expr = new GRBLinExpr();
    for (int i = 0; i < tour.length; i++) {
        for (int j = 0; j < tour.length; j++) {
            if (i != j) {
                expr.addTerm(1.0, vars[tour[i]][tour[j]]);
            }
        }
    }
    addLazy(expr, GRB.LESS_EQUAL, tour.length - 1);
}
```

$$\sum_{i \in Q} \sum_{j \neq i, j \in Q} x_{ij} \leq |Q| - 1$$

Obrázok 19. Zdrojový kód metódy callback

```
GRBEnv env = new GRBEnv();
GRBModel model = new GRBModel(env);
// Nastavenie modelu vyuzitie LazyConstraints
model.set(GRB.IntParam.LazyConstraints, 1);
// Vytvorenie premennych x, UF
GRBVar[][] x = new GRBVar[n][n];
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        x[i][j] = model.addVar(0.0, 1.0, c[i][j],
            GRB.BINARY,
            "x" + String.valueOf(i) + "_" + String.valueOf(j));
    }
}

// Zakazanie hran z uzla k samemu sebe
for (int i = 0; i < n; i++) {
    x[i][i].set(GRB.DoubleAttr.UB, 0.0);
}
//v kazdom vrchole pravejedna hrana konci a jedna zacina
for (int i = 0; i < n; i++) {
    GRBLinExpr expr = new GRBLinExpr();
    for (int j = 0; j < n; j++) {
        expr.addTerm(1.0, x[i][j]);
    }
    model.addConstr(expr, GRB.EQUAL, 1.0, "deg2_" + String.valueOf(i));
}

// z kazdeho vrcholu sa vychadza prave raz
for (int j = 0; j < n; j++) {
    GRBLinExpr expr = new GRBLinExpr();
    for (int i = 0; i < n; i++) {
        expr.addTerm(1.0, x[i][j]);
    }
    model.addConstr(expr, GRB.EQUAL, 1.0, "deg2_" + String.valueOf(j));
}

model.setCallback(new Tsp(x));
model.optimize();
```

$$\min \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij}$$

$$\sum_{i=1, i \neq j}^n x_{ij} = 1, \forall j = 1, \dots, n$$

$$\sum_{j=1, j \neq i}^n x_{ij} = 1, \forall i = 1, \dots, n$$

Obrázok 20. Model TSP v Gurobi solvéri

3.3.2 Implementácia metódy vetiev a hraníc

Implementovaná metóda, ktorá je podrobnejšie popísaná v kapitole 1.4.2 Riešenia TSP sa nachádza v balíčku MVH a obsahuje nasledujúce triedy:

Trieda Vrchol – reprezentuje vrchol stromu riešení:

Atribúty: *int id_uzla* – identifikátor uzla v grafe, *Vrchol predchodca* – referencia na predchádzajúci vrchol v strome, *int redukovanaCena* – redukovaná cena vrcholu, *int [][[] matrix* – matica vrcholu, *boolean[] navstiveneUzly* – kde *navstiveneUzly[i]==true* znamená, že uzol *i* už bol na navštívený vo vrchole, *int pocetNavstivenychUzlov*

Metódy: *Vrchol(int [][[] matrix)* – konštruktor pre koreň stromu riešení, *Vrchol(int id_uzla, Vrchol predchodca)* – konštruktor pre nekoreňový vrchol, ktorý sa vytvorí na základe predchodcu, *void upravMaticu()* – maticové operácie pre získanie novej matice pre vrchol z matice vrcholu predchodcu, *int compareTo(Vrchol vrchol)* – porovnanie vrcholov na základe ich redukovaných cien

Trieda MVH – reprezentuje algoritmus riešenia MVH

Atribúty: *PriorityQueue<Vrchol> PQ* – prioritný front vrcholov podľa redukovanej ceny, *int [][[] maticaVzdialenosti* – matica vzdialenosti, *int najmensieNaklady* – hodnota doteraz nájdeného najlepšieho riešenia, *Vrchol poslednyVrchol* – posledný vrchol doteraz nájdeného najlepšieho riešenia

Metódy: *MVH(int [][[] maticaVzdialenosti, int najmensieNaklady)* – konštruktor MVH, *int rataj()* – metóda vráti hodnotu najlepšieho riešenia, spustí algoritmus vložení koreňového vrcholu, *void metodaVetievHranic()* – algoritmus MVH, *Stack<Integer> vratNajlepsiuCestu()* – vracia najkratšiu hľadanú cestu

```

private void MetodaVetievHranic() {
    Vrchol vrchol = null;
    Vrchol novy = null;
    while (!PQ.isEmpty()) {
        vrchol = PQ.remove(); // redukovaná cena vybraného
        if (najmensieNaklady <= vrchol.redukovanaCena) {
            return; // koniec algoritmu, redukovaná cena > ako doteraz najlepšie riešenie
        }
        if (vrchol.redukovanaCena < najmensieNaklady // lepšia hodnota
            && vrchol.pocetNavstivenychUzlov == maticaVzdialenosti.length) { // je koreňový vrchol=>
            najmensieNaklady = vrchol.redukovanaCena; //aktual. naj.riešenia //navstivene všetky uzly
            poslednyVrchol = vrchol; //aktual. naj.riešenia
        }
        // pridanie nových vrcholov
        for (int j = 0; j < maticaVzdialenosti.length; j++) {
            if (maticaVzdialenosti[vrchol.id_uzla][j] > 0
                && vrchol.navstiveneUzly[j] == false) {
                novy = new Vrchol(j, vrchol);
                PQ.add(novy);
            }
        }
    }
}

```

Obrázok 21. Algoritmus MVH

3.3.3 Implementácia Ant colony optimization

Implementovaná metóda, ktorá je podrobnejšie popísaná v kapitole 1.4.2 Riešenia TSP sa nachádza v balíčku ACO a obsahuje nasledujúce triedy:

Trieda Prostredie – reprezentuje prostredie pre mravca

Konštanty: $ro = 0.5$, $alfa = 1.0$, $beta = 2.0$

Atribúty: $int[][] dMatrix$ – matica vzdialenosti, $double[][] fMatrix$ – matica feromónovej stopy, $double[][] rMatrix$ – matica pre výpočet pravdepodobnosti rozhodovania, $int[][] susedneUzly$ – matica susedných najbližších uzlov (najbližších n uzlov k uzlu, pre ktoré sa používa pravdepodobnostné pravidlo o prechode, po ich vyčerpaní využíva metódu prechodu k najbližšiemu nezaraďenému uzlu), $Mravec[] mravce$ – pole mravcov, $int pocetNajblizsichSusedov$ – počet susedných najbližších uzlov

Metódy: $Prostredie(int[][] dMatrix, long seed, int pocetMravcov)$ – konštruktor pre prostredie, $void pripravProstredie()$ – inicializácia matice feromónov, $void preratajRMatrix()$ – prerátanie matice pre výpočet pravdepodobnosti rozhodovania, $void pripravMravce(int pocetMravcov)$ – vytvorenie mravcov, $void pripravSusedov()$ – inicializácia matice $susedneUzly$, $void updateFMatrix()$ – obsahuje aktualizáciu $fMatrix$, prerátanie $rMatrix$ na základe vyprchania a následného pridania feromónových stop mravcov, $void urobReplikaciju()$ – jedna replikácia ACO

Trieda Mravec – reprezentuje mravca

Atribúty: *boolean[] navstivene* – ak *navstivene[i]==true* uzol *i* už bol navštívený, *int cesta[]* – *Prostredie* *prostredie* – reprezentácia prostredia

Metódy: *Mravec(Prostredie p)* – konštruktor mravca, *void chodKNajblizsiemu(int aktualnaPozicia)* – presun mravca na najbližší nezarađený uzol, *void vyratajCestu()* – výpočet nákladov za cestu, *int getCestaNajblizsichSusedov()* – vykoná cestu cez najbližšie nenavštívne susedné uzly a vráti jej hodnotu nákladov, *chodKNajlepsiemu(int aktualnaPozicia)* – prechod k najlepšiemu uzlu na základe rozhodovacích pravidiel, *chodPodlaPravidiel(int aktualnaPozicia)* – rozhodovacie pravidlá o prechodoch, *void vycistiNavstivene()* – reinitializácia pre ďalšiu replikáciu, *void setZaciatokMravca(int pozicia)* – postavenie mravca do počiatočného uzla, *void setKoniecMravca()* – uzavretie cesty mravca.

Trieda Výsledok – uchovávanie štatistík o najlepších výsledkoch v jednotlivých replikáciách

Atribúty: *int[] cesta* – najlepšia nájdená cesta, *int naklady* – náklady najlepšej cesty

Metódy: *Vysledok()* – konštruktor výsledku počiatočné náklady nastaví na *Integer.MAX_VALUE*, *void setVysledok(int naklady, int[] cesta)* – uloženie nového lepšieho výsledku, *int[] getSpravnePoradie(int id)* – vráti poradie cesty začínajúcej v *id*

Trieda ACO – implementácia ACO algoritmu

Atribúty: *Prostredie prostredie* – prostredie úlohy, *Vysledok vysledok* – ukladanie najlepšieho výsledku, *int pocetReplikacii* – počet replikácií

Metódy: *void spustiACO()* – spustenie algoritmu ACO

Na nasledujúcich obrázkoch sú zobrazené najdôležitejšie časti implementovaného ACO algoritmu:

```
public void preratajRMatrix() {
    for (int i = 0; i < dMatrix.length; i++) {
        for (int j = 0; j < dMatrix.length; j++) {
            rMatrix[i][j] = Math.pow(fMatrix[i][j], alfa) *
                Math.pow((1.0 / (dMatrix[i][j] + 0.1)), beta);
        }
    }
}
```

Obrázok 22. Výpočet hodnôt pre rozhodovaciu maticu

```

public void chodPodlaPravidiel(int aktualnaPozicia) {
    int aktualIndex = cesta[aktualnaPozicia - 1];
    double[] pravdepodobnosti = new double[prostredie.pocetNajblizsichSusedov + 1];
    double celkovaPP = 0;
    for (int i = 0; i < prostredie.pocetNajblizsichSusedov; i++) { // prehľadanie susedných uzlov
        if (navstivene[prostredie.susedneUzly[aktualIndex][i]]) { // bol navstivený
            pravdepodobnosti[i] = 0;
        } else { // nebol navstivený
            pravdepodobnosti[i] = prostredie.rMatrix[aktualIndex][prostredie.susedneUzly[aktualIndex][i]];
            celkovaPP += pravdepodobnosti[i];
        }
    }
    if (celkovaPP <= 0) { // všetky uzly zo susedných boli navstivene
        chodKNajblizsiemu(aktualnaPozicia); // prechod k najblizsiemu nenavstivenému
    } else { // prechod pomocou pravidiel rozhodovania
        int i = 0; // index pre vybraný vrchol
        double r = celkovaPP * rnd.nextDouble();
        double p = pravdepodobnosti[0];
        while (p <= r) {
            i++;
            p += pravdepodobnosti[i];
        }
        if (i == prostredie.pocetNajblizsichSusedov) { // ošetrenie zaokruhlenia
            chodKNajblizsiemu(aktualnaPozicia);
            return;
        }
        cesta[aktualnaPozicia] = prostredie.susedneUzly[aktualIndex][i];
        navstivene[cesta[aktualnaPozicia]] = true;
    }
}

```

Obrázok 23. Algoritmus prechodov k ďalšiemu uzlu

```

for (int i = 0; i < pocetReplikacii; i++) {
    prostredie.urobReplikaciju();
    prostredie.updateFMatrix();
    Mravec m = prostredie.getNajlepsi();
    if (vysledok.getNaklady() > m.naklady) {
        vysledok.setVysledok(m.naklady, m.cesta);
    }
}

```

Obrázok 24. Jadro algoritmu ACO

Naprogramovanú heuristiku som následne odladil tak, aby vrátila dostatočne dobrý výsledok v čo najkrajšom čase (do 2 sekúnd). Nastavenú heuristiku som otestoval pre rôzne veľké matice vzdialeností tak, že som porovnal jej výsledky oproti optimálnej hodnote riešenia získaného pomocou solvéra Gurubi. Výsledky testovania (obrázok 25.) som skonzultoval s vyučujúcim Metaheuristik a zhodli sme sa, že nastavenie heuristiky je vhodné na riešenie veľkosti úloh, ktoré sú informačným systémom požadované.

MATICA veľkosti: 18x18				MATICA veľkosti: 33x33			
ACO:	Replikácií	1000		ACO:	Replikácií	1000	
	Parametre replikácie				Parametre replikácie		
	Mravcov:	100			Mravcov:	100	
	Replikácie:	100			Replikácie:	100	
			rozdiel oproti optimu [%]				rozdiel oproti optimu [%]
	Najhorší výsledok:	12158	0,47		Najhorší výsledok:	14372	1,79
	Najlepší výsledok:	12101	0,00		Najlepší výsledok:	14140	0,15
	Priemerný výsledok:	12131	0,25		Priemerný výsledok:	14187	0,48
Gurobi:	Optimálny výsledok:	12101		Gurobi:	Optimálny výsledok:	14119	
MATICA veľkosti: 48x48				MATICA veľkosti: 65x65			
ACO:	Replikácií	1000		ACO:	Replikácií	1000	
	Parametre replikácie				Parametre replikácie		
	Mravcov:	100			Mravcov:	100	
	Replikácie:	100			Replikácie:	100	
			rozdiel oproti optimu [%]				rozdiel oproti optimu [%]
	Najhorší výsledok:	19177	8,39		Najhorší výsledok:	23760	12,29
	Najlepší výsledok:	17892	1,13		Najlepší výsledok:	22469	6,19
	Priemerný výsledok:	18527	4,72		Priemerný výsledok:	23207	9,67
Gurobi:	Optimálny výsledok:	17692		Gurobi:	Optimálny výsledok:	21160	

Obrázok 25. Porovnanie výsledkov ACO s optimálnym riešením

3.3.4 Implementácia sekvenčnej Clark-Wright heuristiky

Implementovaná metóda, ktorá je podrobnejšie popísaná v kapitole 1.5.1 Riešenia úlohy okružných jazd sa nachádza v balíčku CW a obsahuje nasledujúce triedy:

Pre lepšie pochopenie problematiky uvádzam, že Zákazník v IS pre OH reprezentuje Stojisko, a požiadavka zákazníka – je množstvo odpadu, ktoré treba zo stojiska odvieť.

Trieda Dvojica – reprezentácia dvojice zákazníkov i a j

Atribúty: *int index1* – index zákazníka i , *int index2* – index zákazníka j , *int uspora* – úspora zaradenia dvojice do jazdy

Metódy: *Dvojica(int index1, int index2)* – konštruktor, *int compareTo(Dvojica d)* – porovnanie dvojíc podľa úspory, *void vyratajUsporu(int dis, int dsj, int dij)* – vyrátanie úspory pre dvojicu pomocou vzorca $dis + dsj - dij$, kde dis je vziadelenosť medzi zákazníkom i a strediskom s , dsj je vziadelenosť medzi strediskom s a zákazníkom j , dij je vziadelenosť medzi zákazníkom i a zákazníkom j .

Trieda Zakaznik – reprezentácia zákazníka

Atribúty: *int id_zakaznika* – identifikátor zákazníka, *int poziadavka* – požiadavka zákazníka

Metódy: *Zakaznik(int id_zakaznika, int poziadavka)* – konštruktor, *int compareTo(Zakaznik z)* – porovnanie zákazníkov podľa požiadavky.

Trieda Vozidlo – reprezentácia vozidla, ktoré vykoná jazdu

Atribúty: *String SPZ* – identifikátor auta, *double kapacita* – maximálna kapacita auta, *int priorita* – priorita výberu auta

Metódy: *Vozidlo(String SPZ, double kapacita, int priorita)* – konštruktor, *int compareTo(Vozidlo t)* – porovnanie vozidiel podľa priority (čím menšie číslo, tým väčšia priorita).

Trieda Trasa – reprezentácia zbernej jazdy

Atribúty: *Vozidlo vozidlo* – vozidlo, ktoré vykoná jazdu, *int poziadavka* – požiadavka zákazníka
Metódy: *Trasa(Vozidlo vozidlo)* – konštruktor, *void pridajZakaznikaNaKoniec(Zakaznik z)* – pridanie zákazníka na koniec trasy, *void pridajZakaznikaNaZaciatok(Zakaznik z)* – pridanie zákazníka na začiatok trasy jazdy.

Trieda VRUloha – reprezentácia úlohy okružných jázd

Atribúty: `[[[] Dmatrix` – matica vzdialenosti, `ArrayList<Integer> poziadavky` – zoznam požiadaviek zákazníkov, `ArrayList<Vozidlo> vozidla` – zoznam vozidiel s ktorými sa majú jazdy zrealizovať

Metódy: `VRUloha(int[][] Dmatrix, ArrayList<Integer> poziadavky, ArrayList<Vozidlo> vozidla)` konštruktor.

Trieda ClarkWright – reprezentácia sekvenčného algoritmu Clark-Wright

Atribúty: `VRUloha uloha` – úloha okružných jázd, `ArrayList<Dvojica> dvojice` – zoznam dvojíc, `ArrayList<Zakaznik> zakaznici` – zoznam zákazníkov, `ArrayList<Trasa> trasy` – zoznam výsledných okružných jázd

Metódy: `ClarkWright(VRUloha uloha)` – konštruktor, kde sa zoznam vozidiel usporiada podľa priority, `ArrayList<Zakaznik> ratajCW()` – implementácia sekvenčného algoritmu CW, ktorá naplní `trasy` riešením a vráti zoznam zákazníkov, ktorých požiadavky sa nedali obslúžiť.

Obrázky zobrazujúce vytvorenú implementáciu:

```
public ArrayList<Zakaznik> ratajCW() {
    Dvojica d; Zakaznik pomZ; // 0 Inicializacia
    Collections.sort(uloha.getVozidla());
    for (int i = 0; i < uloha.getPoziadavky().size(); i++) {
        for (int j = 0; j < uloha.getPoziadavky().size(); j++) { //i+1
            if (i == j) {
                continue;
            } //vytvorenie dvojice s prislusnou usporou
            d = new Dvojica(i + 1, j + 1);
            int dis = uloha.getDmatrix()[i + 1][0];
            int dsj = uloha.getDmatrix()[0][j + 1];
            int dij = uloha.getDmatrix()[i + 1][j + 1];
            if (dis + dsj > dij) {
                d.vyratajUsporu(dis, dsj, dij);
                dvojice.add(d);
            }
        }
    }
    Collections.sort(dvojice); // zostupné uspor. dvojim dla. uspory
}
```

Obrázok 26. Sekvenčný algoritmus CW časť 1.

```

Trasa aktualnaTrasa = null; // 1 inicializacia novej trasy
krok1:
while (zakaznici.size() > 0) {
    if (uloha.getVozidla().isEmpty()) {
        return zakaznici; // koniec alg. neboli obsluzeny vsetci zak.
    }
    Vozidlo vozidlo = uloha.getVozidla().remove(0);
    aktualnaTrasa = new Trasa(vozidlo);
    trasy.add(aktualnaTrasa);
    pomZ = null; // 2 inicializacia novej jazdy
    for (Zakaznik zakaznik : zakaznici) {
        if (zakaznik.getPoziadavka() <= aktualnaTrasa.getVozidlo().getKapacita()) {
            pomZ = zakaznik; break; // najdenie zak. s max pozioadav., kt. je mozne obsluzit
        }
    }
    if (pomZ != null) {
        zakaznici.remove(pomZ); aktualnaTrasa.pridajZakaznikaNaKoniec(pomZ);
    } else { trasy.add(aktualnaTrasa); continue krok1; }
}

```

Obrázok 27. Sekvenčný algoritmus CW časť 2.

```

while (!dvojice.isEmpty()) { // krok3
    d = null;
    for (Dvojica dvojica : dvojice) {
        if (aktualnaTrasa.getZakazniciTrasy().size() == 1) { // hladanie najlepsiej dvojice
            if (dvojica.index1 == aktualnaTrasa.getZakazniciTrasy().get(0).getId_zakaznika()
                || dvojica.index2 == aktualnaTrasa.getZakazniciTrasy().get(0).getId_zakaznika()) {
                d = dvojica; break;
            }
        } else {
            if (aktualnaTrasa.getZakazniciTrasy().get(0).getId_zakaznika() == dvojica.index2) {
                d = dvojica; break;
            }
            if (aktualnaTrasa.getZakazniciTrasy().get(aktualnaTrasa.getZakazniciTrasy().size() - 1).getId_zakaznika() == dvojica.index1) {
                d = dvojica; break;
            }
        }
    }
    pomZ = null;
    if (d == null) { continue krok1; } // nenasia dvojica, koniec algoritmu
    else { // nasla dvojica
        dvojice.remove(d);
        for (Zakaznik zakaznik : zakaznici) {
            if (zakaznik.getId_zakaznika() == d.index1 || zakaznik.getId_zakaznika() == d.index2) {
                pomZ = zakaznik; break;
            }
        }
        if (pomZ.getPoziadavka() + aktualnaTrasa.getAktualnaKapacitaJazdy() <= aktualnaTrasa.getVozidlo().getKapacita()) {
            zakaznici.remove(pomZ); boolean spajanieCezPrveho = false;
            if (d.index2 == aktualnaTrasa.getZakazniciTrasy().get(0).getId_zakaznika()) {
                aktualnaTrasa.pridajZakaznikaNaZaciatok(pomZ); spajanieCezPrveho = true; // spajanie cez prveho
            } else if (d.index1 == aktualnaTrasa.getZakazniciTrasy().get(aktualnaTrasa.getZakazniciTrasy().size() - 1).getId_zakaznika()) {
                aktualnaTrasa.pridajZakaznikaNaKoniec(pomZ); // spajanie cez posledneho
            }
            ArrayList<Dvojica> dvojicePonechane = new ArrayList<>();
            for (Dvojica dvojical : dvojice) { // najdenie potrebných dvojic
                if (dvojical.index1 == pomZ.getId_zakaznika()) {
                    continue;
                }
                if (dvojical.index2 == d.index2) {
                    continue;
                }
                if (dvojical.index1 == d.index2) {
                    continue;
                }
                dvojicePonechane.add(dvojical); // nastavenie dvojic
            }
            dvojice = dvojicePonechane;
        }
    }
}
return null; // hotovo uspesne kazdy zakaznik bol obsluzeny

```

Obrázok 28. Sekvenčný algoritmus CW časť 3.

3.4 Optimalizácia zberných stojísk

Optimalizácia zberných miest je postavená na implementácii genetického algoritmu pre úlohu o P-mediáne. Vychádzam z materiálu z predmetu Metaheuristiky, ktorý som v rámci laboratórnych cvičení inžinierskeho štúdia doplnil, a pre aktuálnu potrebu DP čiastočne prispôbil. Pre lepšie pochopenie problematiky úlohy o P-mediáne uvádzam, že ďalej používané centrá sú zberné stojiská, zákazníci sú oblasti (napr. štvrť, ulica, budova atď.), ktorí majú svoje požiadavky, čo môže reprezentovať napr. množstvo ľudí produkujúcich odpad. Balíček GenAlg obsahuje nasledujúce triedy:

Trieda GenDisUnif – generátor diskrétného normálneho rozdelenia

Atribúty: *int aMin* – minimálna hodnota, *int Max* – maximálna hodnota

Metódy: *GenDisUnif(int paMin, int paMax)* – konštruktor, *int Generate()* – generovanie hodnoty.

Trieda PMedian – trieda reprezentuje zadania P-medián úlohy

Atribúty: *int pocetUmiestnenychCentier* – počet centier ktoré treba umiestniť, *LinkedList<Integer> poziadavky* – požiadavka danej oblasti, *int[][] dmatrix* – matica vzdialenosti medzi jednotlivými miestami na umiestnenie centra a zákazníkmi

Metódy: *PMedian(int pocetCentier, LinkedList<Suradnice> mozneUmiestnenia, LinkedList<Suradnice> zakaznici, LinkedList<Integer> poziadavky)* – konštruktor, v ktorom sa pomocou Haversinovho vzorca vypočíta matica vzdialenosti zo súradníc možných umiestnení centier a súradníc zákazníkov.

Trieda GenAlg – implementácia genetického algoritmu

Atribúty: *PMedian pMedian* – zadanie úlohy, *int[] vybraneUzly* – indexy umiestnených centier, *int[] nevybraneUzly* – indexy neumiestnených centier, *int velkostPopulacie* – počet jedincov v populácii, *long[] hodnotyUceloviek* – fitness hodnoty pre jednotlivých jedincov, *int[][] populacia* – pole jedincov, *int[] poradie* – pole pre určenie poradia na základe fitness hodnôt, *int pocetCentier* – počet možných pozícií pre umiestnenie centra, *long najlepsiaUF* – fitness hodnota najlepšieho riešenia, *int dlzkaTrvaniavMin* – minimálne trvanie algoritmu, *int pocetPopulacnychVymen* – požadovaný počet výmen populácie, *int*

pocetJedincovPreKrizenie – potrebný počet jedincov pre kríženie, *double pravdepodobnostMutacie* – pravdepodobnosť mutácie a všetky potrebné generátory.

Metódy: *GenAlg(PMedian pMedian)* – konštruktor, nastavenie generátorov, vytvorenie počiatočného riešenia, *void vytvorPopulaciu()* – vygenerovanie počiatočnej populácie, *void vygenerujJedince(int[] jedinec)* – náhodné vygenerovanie nového jedinca, *long vyratajUF(int[] jedinec)* – výpočet fitness hodnoty daného jedinca, *void aktualizujPoradia()* – aktualizácia poľa poradia, podľa príslušných fitness hodnôt jedincov, *int getRank()* – vráti poradie jedinca, ktorý sa má vybrať na kríženie, *void mutaciacia(int[] jedinec)* – vykonanie mutácie na jedincovi, *void krizenie(int[] rodic1, int[] rodic2, int[] potomok1, int[] potomok2)* – metóda kríženia, *void startGenAlg()* – spustenie algoritmu.

Na nasledujúcich obrázkoch je zobrazená implementácia Haversinovho vzorca na výpočet vzdialenosti medzi dvomi súradnicami a operácia mutácie jedinca.

```
public static double vyratajVzdialenostKM(Suradnice s1, Suradnice s2) {
    double latDistance = Math.toRadians(s2.getX()-s1.getX()); //x=lat, y = long
    double lngDistance = Math.toRadians(s2.getY()-s1.getY());
    double a = Math.sin(latDistance / 2) * Math.sin(latDistance / 2)
        + Math.cos(Math.toRadians(s1.getX())) * Math.cos(Math.toRadians(s2.getX()))
        * Math.sin(lngDistance / 2) * Math.sin(lngDistance / 2);
    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    return POLOMER_ZEME_V_KM * c; //POLOMER_ZEME_V_KM = 6371
}
```

Obrázok 29. Haversinov vzorec

```
private void mutaciacia(int[] jedinec) {
    int[] geny = new int[pocetCentier];
    if (rnd_budeMutacia.nextDouble() < pravdepodobnostMutacie) { //je mutacia
        System.arraycopy(jedinec, 0, geny, 0, pMedian.pocetUmiestnenychCentier);
        HashSet<Integer> uzPouziteCentra = new HashSet<>();
        for (int i = 0; i < geny.length; i++) {uzPouziteCentra.add(geny[i]);}
        GenDisUnif rnd_intMutacia = new GenDisUnif(0, pMedian.pocetUmiestnenychCentier - 1);
        GenDisUnif rnd_celyRozsah = new GenDisUnif(0, jedinec.length - 1);
        HashSet<Integer> pouziteIndexyKdeUzBoliNahrady = new HashSet<>();
        int index; int indexcelyrozsah; int i = 0;
        int pocetMutovanychGenov = (int) rnd_intMutacia.Generate(); //gener. poctu genov na mutaciu
        while (i < pocetMutovanychGenov) {
            index = rnd_nagen.nextInt(geny.length); // index pre centrum ktore bude nahradene
            if (!pouziteIndexyKdeUzBoliNahrady.contains(index)) { //aby sa nemutoval viac krat rovnaky gen
                pouziteIndexyKdeUzBoliNahrady.add(index);
                while (true) { //generovanie náhrady
                    indexcelyrozsah = (int) rnd_celyRozsah.Generate();
                    if (!uzPouziteCentra.contains(indexcelyrozsah)) {
                        uzPouziteCentra.add(indexcelyrozsah);
                        geny[index] = indexcelyrozsah; //nahradenie mutovaneho genu
                        break;
                    }
                }
            }
            i++;
        }
    }
}
```

Obrázok 30. Operácia mutácie

3.5 Implementácia zobrazenia geografických dát

Balíček Mapy určený na vizualizáciu geografických dát, je založený na voľne dostupnej knižnici JXMapView2 [8], ktorá umožňuje jednoduchú integráciu OSM mapy do Java aplikácie. Pri tvorbe balíčka som vychádzam z tutoriálu pripojenom ku knižnici [13]. Balíček obsahuje nasledovné triedy:

Trieda StojiskoWaypoint – dedí triedu DefaultWaypoint určenú na zobrazovanie bodov v mape zo spomenutej knižnice, zabezpečuje nastavenie farieb, popisného textu k ikonám bodov-Stojísk a tiež má za úlohu definovať parametre ďalej spomenutých tlačidiel.

Trieda StojiskoRender – je trieda určená na vykreslenie ikony stojiska v mape. Ak sa na stojisku nachádzajú viaceré nádoby určené pre rôzne druhy odpadu ikona sa vykresľuje viacfarebne, inak sa ikona vyfarbí podľa typu odpadu napr. pre plast na žlt.

Trieda MultiplyComposite – pomocná trieda pre zafarbovanie ikon.

Trieda StojiskoOverlayPainter – je trieda, ktorá všetky ikony stojísk v mape prekryje tlačidlom, ktoré po stlačení zobrazí informáciu o zberných nádobách na stojisku.

Trieda UzolWaypoint – dedí triedu DefaultWaypoint určenú na zobrazovanie bodov v mape zo spomenutej knižnice, zabezpečuje nastavenie farieb, popisného textu k ikonám bodov - (stojiskových alebo križovatky) a tiež má za úlohu definovať parametre ďalej spomenutých tlačidiel.

Trieda UzolRender – je trieda, ktorá pri zobrazovaní trás vykresľuje ikony uzla v mape pre križovatku – zeleno a stojiskové uzly – žlt.

Trieda UzolOverlayPainter – je trieda, ktorá všetky ikony uzlov v mape prekryje tlačidlom, ktoré po stlačení zobrazí dodatočnú informáciu o uzle.

Trieda RoutePainter – je trieda určená na vykresľovanie úsečiek medzi súradnicami (zobrazenie trás) zo zoznamu súradníc.

Trieda DvojicaTras – je trieda určená k preneseniu dát o pôvodnej trase a jej zlepšenej trase (získaná optimalizáciou pôvodnej trasy) pre zobrazenie na mape.

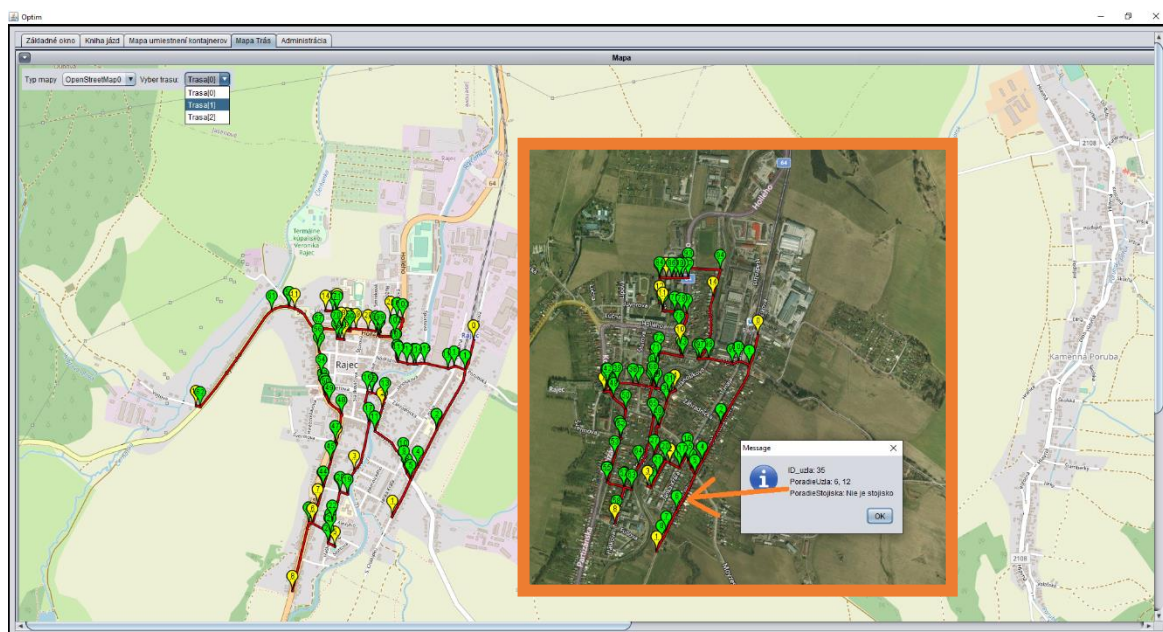
Trieda MapPoint - dedí triedu DefaultWaypoint určenú na zobrazovanie bodov v mape zo spomenutej knižnice, zabezpečuje nastavenie farieb a popisného textu k ikonám bodov.

Trieda MapPointRender - je trieda, ktorá pri zobrazovaní optimizéra stojísk vykresľuje ikony miesta umiestnenia stojiska – červeno a ikony reprezentujúce centroid oblasti - zeleno.

Trieda Mapa – zabezpečuje vytvorenie objektu mapy, v konštruktoe sa nastaví Combobox, ktorý slúži na prepínanie medzi rôznymi typmi mapového podkladu. Má 4 základne metódy: *void zobrazRozmiestnenieStojisk(JInternalFrame frame)* v okne zobrazí mapu stojísk, *void zobrazTrasy(JInternalFrame frame, ArrayList<Trasa> trasy)* v okne zobrazí jednotlivé trasy zo zoznamu, medzi ktorými sa dá prepínať pomocou Comboboxu, *zobrazTrasu(JInternalFrame frame, LinkedList<Integer> idecka)* v okne zobrazí trasu poskladanú zo zoznamu *idecka*, *optimalizaciaStojisk(JInternalFrame frame)* v okne optimizéra stojísk zobrazuje polohy možných umiestnení stojísk (červené body), umiestnenia oblastí (zelené polygóny) obrázok 33.

```
public void zobrazTrasy(JInternalFrame frame, ArrayList<Trasa> trasy) {
    ArrayList<DvojicaTras> mapy = spracujTrasy(trasy); //spracovanie tras do struktury
    JLabel label = new JLabel("Vyber trasu: "); //Vytvorenie comboboxu na prepínanie
    String[] tfLabels = new String[mapy.size()]; //zobrazovaných tras
    for (int i = 0; i < mapy.size(); i++) {
        tfLabels[i] = "Trasa[" + i + "]";
    }
    JComboBox cmb = new JComboBox(tfLabels);
    cmb.addItemListener(new ItemListener() {}); //akcia na prepísanie tras
    List<Painter<JXMapView>> painters = new ArrayList<Painter<JXMapView>>();
    JXMapView mapView = new JXMapView();
    mapView.setTileFactory(factories.get(0));
    comboFactories.addItemListener(new ItemListener() {}); // akcia na prepínanie podkladu
    JPanel panel = new JPanel();
    panel.add(this.label);
    panel.add(comboFactories);
    panel.add(label);
    panel.add(cmb);
    mapView.add(panel, BorderLayout.NORTH);
    //ovladanie mapy
    MouseInputListener mia = new PanMouseListener(mapView);
    mapView.addMouseListener(mia);
    mapView.addMouseMotionListener(mia);
    mapView.addMouseListener(new CenterMouseListener(mapView));
    mapView.addMouseWheelListener(new ZoomMouseWheelListenerCenter(mapView));
    mapView.addKeyListener(new PanKeyListener(mapView));
    //zamierenie mapy
    mapView.zoomToBestFit(new HashSet<>(mapy.get(0).getTrasa(), 0.7);
    mapView.setZoom(7); // priblizenie
    RoutePainter routePainter = new RoutePainter(mapy.get(0).getTrasa());
    painters.add(routePainter); //vykreslenie tras
    WaypointPainter<MyWaypoint> swingWaypointPainter = new UzolOverlayPainter();
    swingWaypointPainter.setWaypoints(mapy.get(0).getWaypointsIdecka());
    painters.add(swingWaypointPainter); // pridanie tlacidiel na ikony
    WaypointPainter<MyWaypoint> waypointPainter = new WaypointPainter<>();
    waypointPainter.setWaypoints(mapy.get(0).getWaypointsIdecka());
    waypointPainter.setRenderer(new UzolRender()); // zobrazenie ikon uzlov
    painters.add(waypointPainter);
    CompoundPainter<JXMapView> painter = new CompoundPainter<>(painters);
    mapView.setOverlayPainter(painter);
    for (MyWaypoint w : mapy.get(0).getWaypointsIdecka()) {
        mapView.add(w.getButton());
    }
    frame.setContentPane(mapView);
    frame.setTitle("Mapa");
}
```

Obrázok 31. Metóda pre zobrazenie trás



Obrázok 32. Zobrazenie trás



Obrázok 33. Vykreslenie možných stojísk (červeno) a oblastí (zeleno) v optimizéri

3.6 Implementácia Messenger

Pre komunikáciu s O₂ SMS Connector API som vytvoril balíček Messenger, ktorý obsahuje nasledovné triedy:

Trieda PrijataSprava – reprezentácia prijatej SMS správy

Atribúty: *String tel* – kontakt na odosielateľa, *String text* – telo správy, *Date datum* – dátum prijatia správy

Metódy: *PrijataSprava(String tel, String text, String datum)* - konštruktor, *gettre na tel, text, datum, Date parseDate(String d)*.

Trieda PrijataDavka – reprezentácia prijatej dávky SMS-iek

Atribúty: *LinkedList<PrijataSprava> prijateSpravy* – zoznam prijatých SMS, *String id_stav* – {OK, ZIADNE_NOVE_SPRAVY}

Metódy: *PrijataDavka(String id)* - konštruktor, *pridajSpravu(PrijataSprava sprava)* - pridanie prijatej správy do dávky, *gettre na stav a zoznam prijatých správ*.

Trieda SMS – pomocná trieda na vytvorenie SMS

Atribúty: *String apiKey* – kľúč k SMS API, *String text* – telo SMS správy, *String odosielatel* – kontakt na odosielateľa SMS napr. „OH Rajec“, *ArrayList<String> prijemcovia* – kontakty na prijímateľov SMS-iek

Metódy: *SMS(String text, String odosielatel, ArrayList<String> prijemcovia)* - konštruktor, kde sa znaky jednotlivých atribútov normalizujú (prepísanie NO-ASCII znakov na príslušné ASCII znaky), *setApiKey(String apiKey)* – nastavenie API kľúča, *String toString()* – prepísanie objektu SMS na reťazec v požadovanom JSON formáte.

Trieda Messenger – trieda zabezpečujúca komunikáciu s SMS bránou

Atribúty: *String apiKey* – kľúč k SMS API, *String VLN* – tel. číslo vašej SMS brány, *String send* – url adresa pre odosielanie požiadavky na poslanie dávky SMS, *String remaining* – url adresa pre odosielanie požiadavky na stav kreditu, *String receiving* – url adresa pre odosielanie požiadavky na prijatie dávky SMS, *HttpURLConnection con* – objekt na poslanie požiadavky

Metódy: *Messenger(String apiKey, String vln)* - konštruktor, *prepareConection (String surl)* - vytvorenie pripojenia na funkciu servera na základe url adresy, *String posliSMS(SMS sms)* – poslanie dávky SMS, *String zistiKredit()* – zistenie hladiny kreditu, *PrijataDavka stiahniSMS()* – stiahnutie prijatých SMS.

3.7 Implementácia Grafického užívateľského prostredia

Implementácia grafického užívateľského prostredia sa nachádza v balíčku GUI a obsahuje nasledujúce triedy:

Trieda Refresher – je určená na aktualizovanie komponentov okna.

Triedy Combobox editor a IconRender – pomocné triedy pre použitie príslušných komponentov v JTable.

Trieda JFPridajKontajner – reprezentuje okno pre pridanie novej zbernej nádoby k stojisku.

Trieda FUpdateKontajner – okno pre modifikovanie zbernej nádoby.

Trieda JFUpdate_Create_Vozidlo - okno pre pridanie nového vozidla alebo modifikovanie existujúceho vozidla.

Trieda JF_Messenger – okno pre SMS Messenger.

Trieda JF_Login – prihlasovacie okno do aplikácie.

Trieda JFMainFrame – hlavné okno aplikácie.

3.8 Architektúra informačného systému pre potreby OH

Základnou súčasťou informačného systému je databáza, ktorá je popísaná v kapitole 3.2 Výber databázového systému a návrh databázového modelu. Hlavné funkcionality aplikácie IS pre OH sa nachádzajú v balíčku DP. Balíček obsahuje nasledujúce triedy:

Trieda DBConnection – je trieda určená na vytvorenie prepojenia pomocou JDBC ovládača medzi databázou a aplikáciou.

Trieda Login – je trieda určená na načítanie a ukladanie inicializačných dát (pripojenie k DB a pripojenie k SMS bráne) aplikácie.

Trieda Kontajner – reprezentuje zbernú nádobu.

Atribúty: *int id_kontajnera* – identifikátor zbernej nádoby, *String nazovOdpadu* – názov druhu odpadu pre ktorého je určená, *String adresa* – adresa umiestnenia, *int aktualnyObjem* – percentuálna naplnenosť nádoby, *int maxObjem* – maximálna kapacita nádoby, *Date Datum_poslednehoVyprazdnenia* – dátum posledného vyprázdnenia, *Date Datum_poslednejAktualizacie* – dátum poslednej aktualizácie objemu v nádobe.

Metódy: konštruktory a príslušné gettre a settre.

Trieda VysledokOptimalizacie – trieda určená na ukladanie výsledkov optimalizácie

Atribúty: *String nazovRiesenia* – názov optimalizácie {"MVaH", "ACO", "CW"}, *ArrayList<Trasa> trasy* – zoznam výsledných trás, *ArrayList<Zakaznik> neobsluzeny* – zoznam neobslužených stojísk, *double sumPoziadaviek* – odhadované množstvo objemu odpadu, *double naklady* – náklady v km pre zrealizovanie jazdy

Metódy: *String nazovRiesenia, ArrayList<Trasa> trasy, double sumPoziadaviek, double naklady*) – konštruktor pre jednu okružnú jazdu, *VysledokOptimalizacie(String nazovRiesenia, ArrayList<Trasa> trasy, double sumPoziadaviek, double naklady, ArrayList<Zakaznik> neobsluzeny)* – konštruktor pre úlohu okružných jázd (algoritmus CW), *String toString()* - reťazcová reprezentácia výsledku optimalizácie, *ArrayList<Trasa> getTrasy()* – getter na trasy, *odoberTrasu(int poradie)* - odobratie trasy z výsledku podľa indexu.

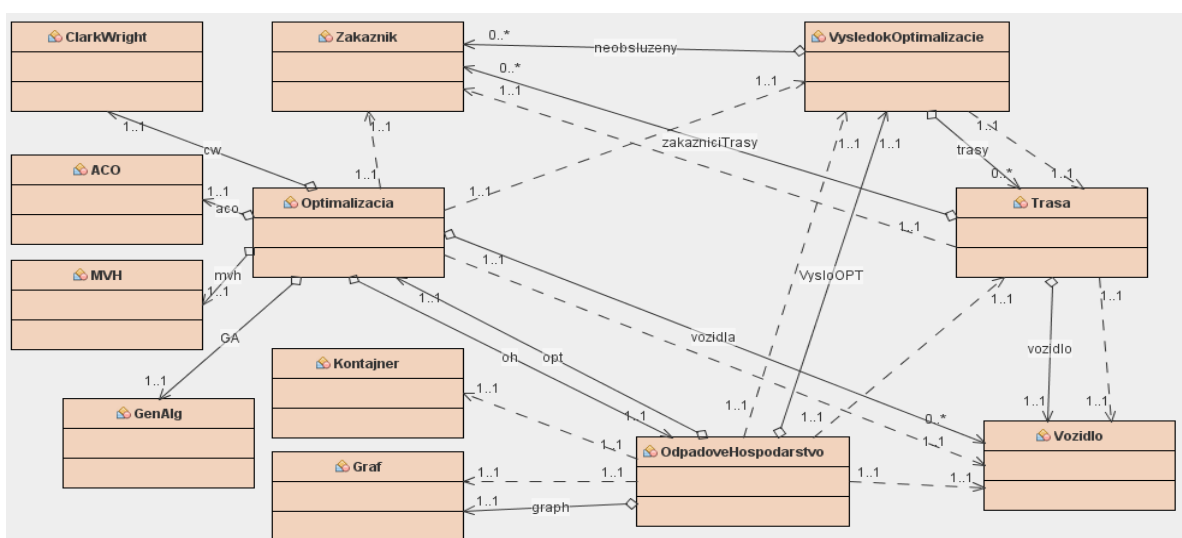
Trieda Optimalizacia – Trieda spravujúca naimplementované optimalizačné metódy

Atribúty: *OdpadoveHospodarstvo oh* – referencia na hlavnú triedu aplikácie IS, *MVH mvh* – objekt na optimalizáciu metódou vetiev a hraníc, *ACO aco* - objekt na optimalizáciu kolóniou mravcov, *ClarkWright cw* - objekt na optimalizáciu metódou Clark-Wright, *GenAlg genAlg* - objekt na optimalizáciu Genetickým algoritmom, *ArrayList<Integer> stojiska* – identifikátory stojísk pre optimalizáciu, *ArrayList<Integer> poziadavky* – odhadované množstvo odpadu na stojiskách, *ArrayList<String> adresy* – adresy jednotlivých stojísk, *ArrayList<Vozidlo> vozidla* – zoznam vozidiel pre optimalizáciu, *int*

sumPoziadaviek – odhadované množstvo odpadu na odvoz, *int[][] dMatrix* – matica vzdialenosti pre optimalizáciu.

Metódy: *Optimalizacia(OdpadoveHospodarstvo oh)* – konštruktor optimalizácie, *setOptimalizacia(ArrayList<Integer> stojiska, ArrayList<Integer> poziadavky, ArrayList<String> adresy, ArrayList<Vozidlo> vozidla, int sum, int[][] dMatrix)* – nastavenie atribútov pre optimalizáciu, *VysledokOptimalizacie optimalizuj(String nazovOpadu)* – algoritmus celkovej optimalizácie popísaný v kapitole 3.3 Optimalizácia zberných jázd, *int nakladyTrasy(ArrayList<Zakaznik> zakaznici)* – výpočet nákladov pre jazdu. *int[] optimalizujPMedian(LinkedList<Suradnice> umiestneniaCentier, LinkedList<Suradnice> zakaznici, LinkedList<Integer> poziadavky, int pocetCentier)* – optimalizácia zberných miest pomocou genetického algoritmu, kde *umiestneniaCentier* – zoznam súradníc pre možné umiestnenia stojísk, *zakaznici* – zoznam súradníc reprezentujúci centroid danej oblasti (napr. sídlisko, panelák), *poziadavky* – požiadavka jednotlivých oblastí, *pocetCentier* – počet stojísk, ktorý sa má umiestniť.

Trieda OdpadoveHospodarstvo – je trieda, ktorá spája všetky vyššie popísané komponenty do jedného funkčného celku. Najdôležitejšie funkcie triedy sú zabezpečenie výmeny dát medzi aplikačnou a databázou časťou informačného systému, podpora funkcionalít pre grafické užívateľské prostredie a zabezpečenie fungovania aplikácie od vytvorenia návrhu zberných jázd cez monitorovanie a riadenie aktuálnych jázd až po uloženie jázd do štatistík, ktoré si môže používateľ prehľadávať a vyhodnocovať.



Obrázok 34. Diagram najdôležitejších tried

4 Výsledky práce a diskusia

4.1 Výsledky práce

Po ukončení implementácie geografického informačného systému pre potreby odpadového hospodárstva som vytvorený informačný systém začal testovať. Počas testovania som zistil, že aplikácia obsahovala drobné chyby, ktoré som postupnými krokmi odstránil. Takto pripravenú aplikáciu som odprezentoval prednostovi MsÚ Rajec. Prezentácia IS sa uskutočnila prostredníctvom platformy Microsoft Teams. Odhalila menšie technické nedostatky, ktoré som napravil. Zároveň môžem skonštatovať, že prezentácia priniesla pozitívnu reakciu a prísľub prednostu MsÚ Rajec, k zavedeniu IS do testovacej praxe. Z diskusie o prezentácii vzišli i nové užitočné informácie a námety, o ktoré IS rozšírim rámci testovacej prevádzky. Napríklad o kalkulačku prevodov medzi objemom zlisovaným (objem odpadu v zbernom vozidle) aj nezlisovaným (objem odpadu v zbernej nádobe) a hmotnosťou jednotlivých zberaných komodít.

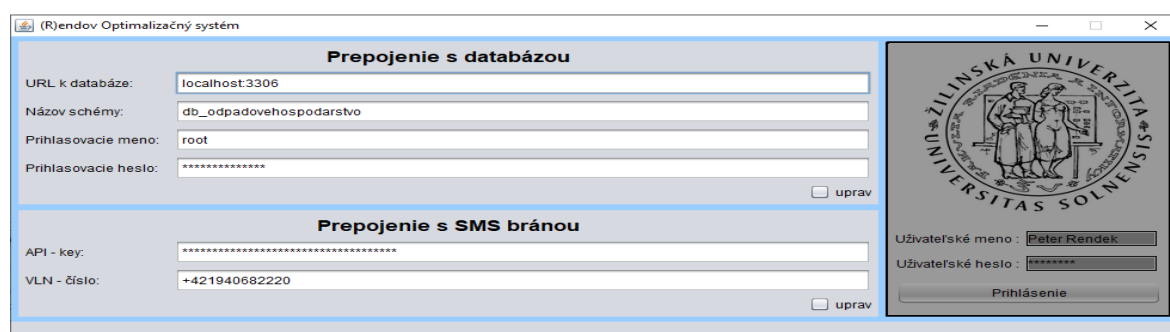
Hlavnou výhodou môjho informačného systému oproti aktuálnemu stavu považujem zavedenie sledovania podrobnejších štatistík odpadového hospodárstva. Prvým a možno najdôležitejším ukazovateľom je online monitoring stavu objemu odpadu na jednotlivých stojiskách pomocou smart senzorov alebo prostredníctvom SMS reakcií o aktuálnom stave objemu nádoby od obyvateľstva. Druhým ukazovateľom v rámci evidencie uskutočnených zberných jász je monitorovanie aktuálneho, percentuálneho stavu naplnenosti jednotlivých stojísk pri zbere a celkového vyzbieraného množstva odpadu. Pomocou ďalšieho analyzovania takto zozbieraných dát sa dá zber odpadu zefektívniť, čo môže priniesť nielen finančné, časové, a v neposlednom rade aj ekologické a environmentálne benefity.

Ďalšou funkcionalitou IS je, že užívateľovi poskytuje vhodný nástroj pre navrhovanie, plánovanie a manažovanie čo najlepších zberných trás so zameraním sa na minimalizáciu najazdených vzdialeností. Za užitočný doplnok IS možno považovať optimizér rozmiestnenia zberných miest, ktorý umožní zlepšiť separovanie odpadu obyvateľstva, kvôli lepšej dostupnosti zberných miest

4.2 Užívateľská príručka

Hlavným úmyslom tvorby užívateľského prostredia pre informačný systém OH, bolo vytvoriť čo najintuitívnejšie, najjednoduchšie a najrýchlejšie ovládania aplikácie.

Pred prvým použitím aplikácie treba v prihlasovacom okne obrázok 35. vyplniť údaje potrebné pre pripojenie k SMS bráne a k databáze. Následne sa užívateľ môže prihlásiť pomocou svojho prihlasovacieho mena a hesla do aplikácie.



Obrázok 35. Prihlasovacie okno do aplikácie

Po úspešnom prihlásení sa zobrazí základné okno, ktoré obsahuje 6 hlavných záložiek. Prvá záložka je „Základné okno“ (obrázok 36.) a obsahuje hlavné funkcionality IS od monitorovania a aktualizovania stavov odpadu na jednotlivých stojiskách, cez plánovanie zberných jazd až po samostatné manažovanie aktuálne prebiehajúcich jazd.

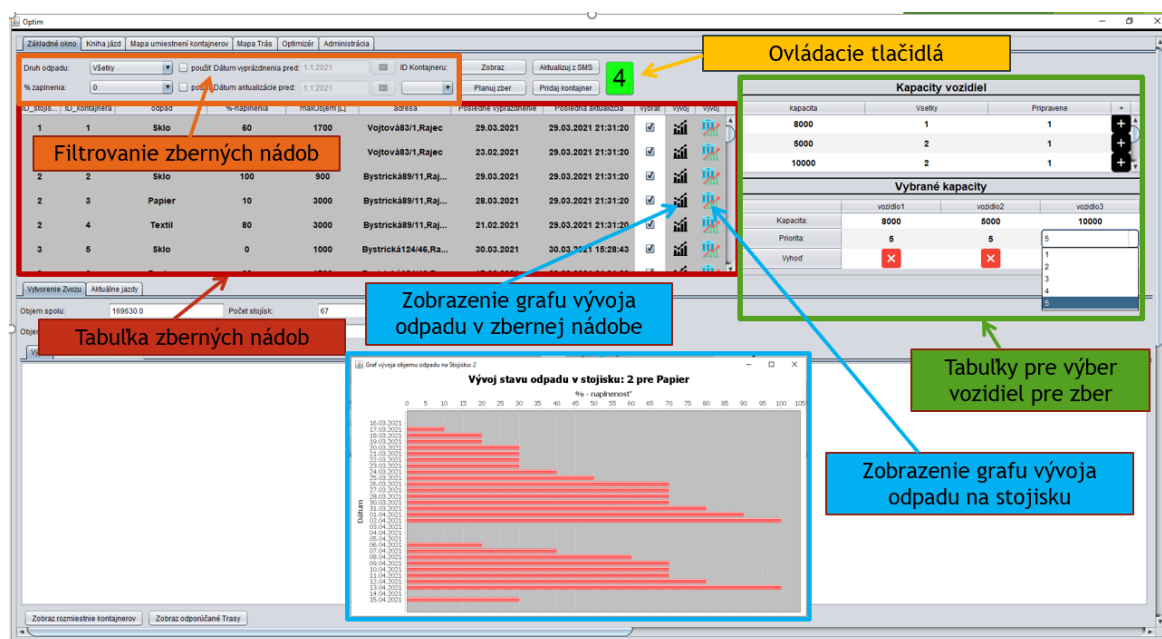
Tabuľka zberných nádob zobrazuje aktuálne informácie k jednotlivým zberným nádobám, ktoré je možno vyfiltrovať pomocou komponentov nachádzajúcich sa v časti Filtrovanie zberných nádob. Tabuľka obsahuje nasledujúce funkcionality:

- Dvoj-klik ľavým tlačidlom myši na bunku tabuľky s ikonou grafu zobrazí graf vývoja naplnenosti stojiska alebo zbernej nádoby
- Zakliknutím / od kliknutím v stĺpci „Vybrať“ znamená, či nádoba bude zaradená do zbernej jazdy alebo nie
- Dvoj-klik ľavým tlačidlom myši na ostatné bunky zobrazí okno na modifikovanie údajov danej zbernej nádoby (obrázok 37.)
- Dvoj-klik pravým tlačidlom myši na ostatné bunky vykoná akciu vyprázdnenia nádoby.

Tabuľky pre výber vozidiel pre zber slúžia na vybratie typov vozidiel (kapacity) s ktorými sa má rátať pre optimalizáciu. Kliknutím na čiernu ikonu + v tabuľke Kapacity vozidiel, sa pridá vozidlo do tabuľky Vybrané kapacity, kde užívateľ môže priradiť prioritu 1 až 5 (kde 1 je najväčšia priorita) k vybraným vozidlám. Kliknutím na červenú ikonu x sa odoberie vozidlo z vybraných.

Funkcie ovládacích tlačidiel:

- „Zobraz“ - aktualizovanie tabuľky zberných nádob
- „Aktualizuj z SMS“ – stiahne správy z SMS brány a aktualizuje príslušné naplnenosti jednotlivých nádob, správy ktoré prišli v nesprávnom formáte si užívateľ môže zobrazit' v Messengeri obrázok 38.
- „Plánuj zber“ - je tlačidlo, ktorého stlačením sa spustí optimalizácia zbernej jazdy pre vybrané nádoby v tabuľke zberných nádob
- „Pridaj kontajner“ – stlačením tlačidla sa zobrazí okno pre pridanie novej zbernej nádoby obrázok 39.
- „4“ - je tlačidlo, ktoré zobrazuje počet správ, ktoré systém nedokázal spracovať z dôvodu, že prišli v zlom formáte. Stlačením tlačidla sa zobrazí Messenger obrázok 38.



Obrázok 36. Základné okno – základné funkcionality

Obrázok 37. Modifikovanie údajov zbernej nádoby

Dátum prijatia	Odosielateľ	Správa
01.04.2021 20:00:00	0944000000	SMS v nespravnom formate
02.04.2021 21:00:00	0944111111	Treba ocistiť stojisko 5
03.04.2021 22:00:00	0944222222	Na stojisku 63 sa nachádza m...
04.04.2021 23:00:00	0944333333	Pri cintorine je prevratena nad...

Dátum: 04.04.2021 23:00:00 Odosielateľ: 0944333333

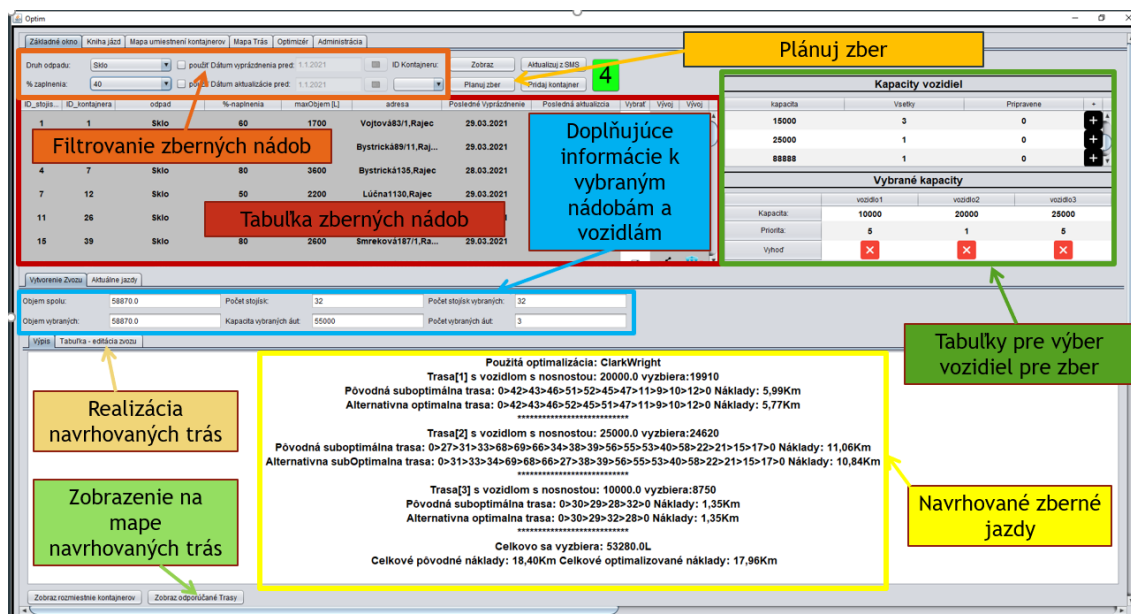
Pri cintorine je prevratena nadoba na plast

Obrázok 38. Messenger

Obrázok 39. Pridanie novej zbernej nádoby

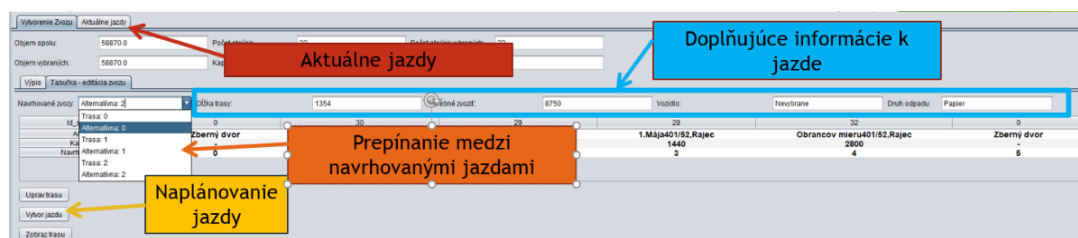
Naplánovanie zvozu jedného druhu odpadu (obrázok 40.) začína vyfiltrovaním zberných nádob, ktoré sa majú zozbierať. Filtrovanie funguje pomocou komponentov v časti filtrovanie zberných nádob, kde sa zvolí aký typ odpadu sa má zozbierať. Ďalej sa dá použiť pri filtrácii napr. minimálna %-na naplnenosť nádoby, alebo použiť dátum posledného vyprázdnenia, alebo dátum posledného aktualizovania objemu prípadne ich rôzna vzájomná kombinácia. Z vyfiltrovaných zberných nádob je pomocou odškrtnutia v stĺpci „Vybrať“ možnosť odobrať zbernú nádobu z optimalizácie. Pokračovaním plánovania jazdy je

vybranie vozidiel, prípadne určenie ich priority pre optimalizáciu pomocou tabuliek pre výber vozidiel pre zber. Pod tabuľkou zberných nádob sú zobrazené doplňujúce informácie ako napr. celkový objem odpadu na zozbieranie, počet stojísk na obsluhu a kapacita vozidiel vybraných na zber. Stlačením tlačidla „Plánuj zber“ sa spustí optimalizácia.



Obrázok 40. Vytvorenie zbernej jazdy

Navrhované trasy užívateľ ďalej môže zrealizovať na záložke Tabuľka – editácia zvozu obrázok 41. Užívateľ si tu môže prepínať a vyberať medzi navrhovanými trasami, ktoré si môže ďalej prezerať na mape, modifikovať alebo definitívne vytvoriť zbernú jazdu. Po kliknutí na tlačidlo „Uprav trasu“ užívateľ môže editovať jazdu pomocou výmeny poradie stojísk, kliknutím na hlavičku stĺpca reprezentujúceho stojisko a presunutím ho pred alebo za iný stĺpec reprezentujúci iné stojisko. Užívateľ tiež môže stojisko vyradiť z jazdy pomocou kliknutia na červenú ikonu x v tabuľke. Potvrdenie zmeny sa uskutočňuje pomocou tlačidla „Ulož trasu“. Použitím tlačidla „Vytvor jazdu“ sa jazda naplánuje a zobrazí v aktuálnych jazdách na záložke „Aktuálne jazdy“.



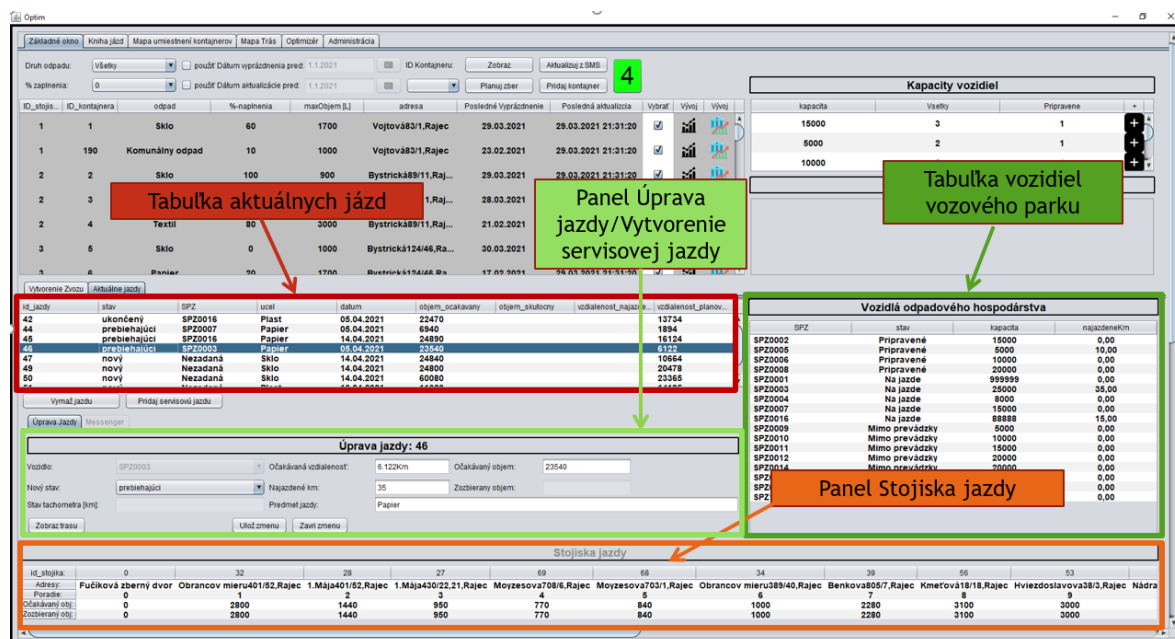
Obrázok 41. Editácia a vytvorenie jazdy

Záložka „Aktuálne jazdy“ obrázok 42. je určená na manažovanie nie len zberných jazd. Obsahuje tabuľku Vozidlá odpadového hospodárstva, ktorá zobrazuje všetky vozidlá vozového parku OH a ich stav, kapacitu a najazdenú vzdialenosť, má nasledujúce funkcionality:

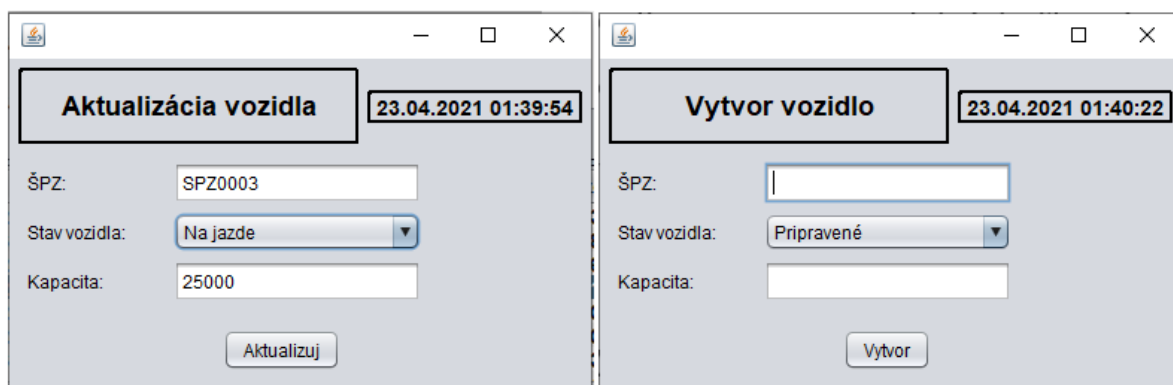
- Dvoj-klik ľavým tlačidlom myši na riadok tabuľky otvorí okno určené na modifikáciu vozidla obrázok 43.
- Dvoj-klik pravým tlačidlom myši na tabuľku otvorí okno na pridanie nového vozidla obrázok 43.

Záložka ďalej obsahuje:

- Tabuľku aktuálnych jazd – ktorá zobrazuje aktuálne prebiehajúce jazdy, a kliknutím na príslušný riadok reprezentujúci jazdu sa aktivujú komponenty na manažovanie takto vybranej jazdy
- tlačidlo „Vymaž jazdu“ – slúži na vymazanie jazdy, ktorá sa ešte neuskutočnila z IS
- tlačidlo „Pridaj servisovú jazdu“ – vytvorenie jazdy iného charakteru ako zberná
- panel Úprava jazdy/ Vytvorenie servisovej jazdy – manažovanie vybranej jazdy
- panel Stojiska jazdy – zobrazuje stojiská zbernej jazdy a je určený na doplnenie informácií o množstve odpadu odvezeného z daného stojiska.



Obrázok 42. Záložka Aktuálne jazdy



Obrázok 43. Okná aktualizácie a vytvorenia vozidla

Životný cyklus riadenia zbernej jazdy vyzerá nasledovne:

1. Novo vytvorená jazda sa objaví tabuľke aktuálnych jazd a zobrazuje sa v stave „nový“. Užívateľ musí priradiť vozidlo ktoré ju vykoná. V prípade, ak sa jazda má začať vykonávať, užívateľ zmení stav z nový na prebiehajúci a skutočnosť potvrdí kliknutím na tlačidlo „Ulož zmenu“. Ak užívateľ ponechá zaškrtnutú možnosť poslať Jazdu ako SMS, aplikácia mu ponúkne možnosť poslať SMS notifikáciu pracovníkovi, ktorý ma jazdu vykonať.

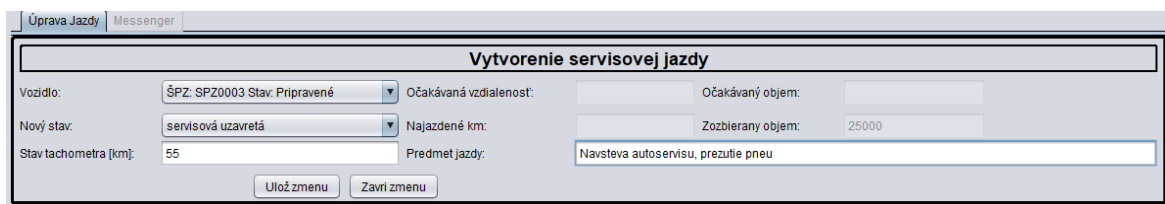
Obrázok 44. Zahájenie zbernej jazdy

Obrázok 45. SMS notifikácia pre zahájenie jazdy

2. V momente, keď sa zberná jazda ukončí, respektíve vozidlo s posádkou je pripravené na ďalšiu činnosť, pracovník zmení stav z „prebiehajúci“ na „ukončený“ a zmenu potvrdí.
3. Pre uzavretie jazdy užívateľ doplní do aplikácie celkové množstvo zozbieraného odpadu, množstvo odpadu zozbierané na jednotlivých stojiskách a aktuálny stav najazdených km vozidla. Následne zmení stav „ukončený“ na „uzavretý“ čím jazda zmizne z tabuľky aktuálnych jazd a zapíše sa do štatistík.

Obrázok 46. Uzavretie jazdy – doplnenie potrebných informácií

4. Pre vytvorenie servisovej jazdy užívateľ klikne na tlačidlo „Pridaj servisovú jazdu“ a v paneli Vytvorenia servisovej jazdy vyberie vozidlo, s ktorým sa má alebo mala jazda vykonať. Keď sa jazda má aktuálne vykonať, užívateľ nastaví stav na „servisová“ a vyplní predmet jazdy. Ak sa jazda už uskutočnila, užívateľ má možnosť vytvoriť jazdu so stavom „servisová uzavretá“, kde doplní aktuálny stav najazdených km vozidla a predmet jazdy. Takto vytvorená jazda sa priamo zapíše do štatistík.

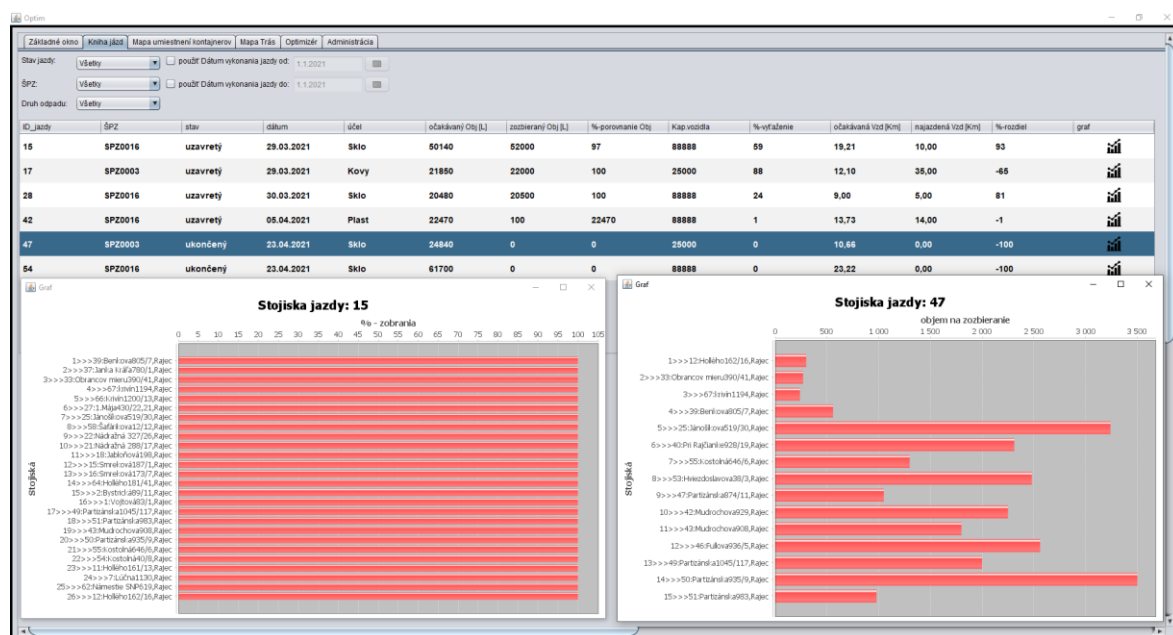


The screenshot shows a web application window titled "Vytvorenie servisovej jazdy". It contains several input fields and buttons. The "Vozidlo:" field shows "ŠPZ: SPZ0003 Stav: Pripravené". The "Nový stav:" dropdown is set to "servisová uzavretá". The "Stav tachometra [km]:" field contains the value "55". The "Predmet jazdy:" text area contains "Navšteva autoservisu, prezutie pneu". There are also buttons for "Ulož zmenu" and "Zavri zmenu".

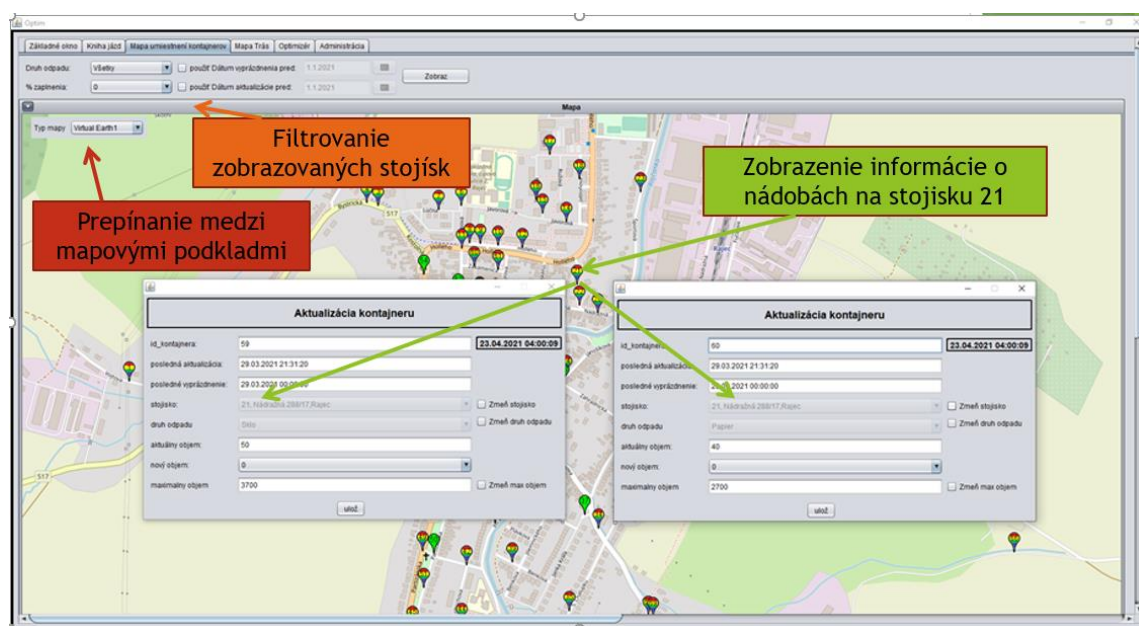
Obrázok 47. Vytvorenie servisovej uzavretej jazdy

5. Uzavretie servisovej jazdy funguje podobne ako uzavretie zbernej jazdy, kde užívateľ zmení stav na „servisová uzavretá“, vyplní nový stav kilometrov, prípadne môže doplniť predmet jazdy.

Druhou hlavnou záložkou je „Kniha jász“ obrázok 48., ktorá je určená na zobrazovanie všetkých štatistík IS pre OH s možnosťou vyfiltrovaní údajov podľa rôznych parametrov. Ďalšími záložkami sú „Mapa umiestnení kontajnerov“ obrázok 49. a „Mapa trász“ obrázok 50., ktoré sú určené na vizualizáciu geografických dát. Mapy okrem štandardných funkcií ako posúvanie (kliknutie a posunutie), približovanie (Dvoj-klik ľavým tlačidlom myši alebo použitie kolieska) menenie mapových podkladov majú integrované aj doplnkové funkcionality. Pri mape rozmiestnení stojísk kliknutím na (jednofarebné stojiská len pre jeden druh odpadu, multifarebné stojiská pre viacero druhov odpadu) ikony bodov sa zobrazia okna pre zobrazenie a modifikovanie zberných nádob nachádzajúcich sa na stojisku. Pri mape trász je možnosť prepínania medzi jednotlivými navrhovanými trasami. Kliknutím na ikonu bodu sa otvorí doplnkové informačné okno, v ktorom sa nachádza informácia, aký má uzol identifikátor, koľko krát sa cez uzol prešlo, prípadne či sa jedná o stojisko.



Obrázok 48. Kniha jász – štatistiky OH



Obrázok 49. Zobrazenie zberných stojísk



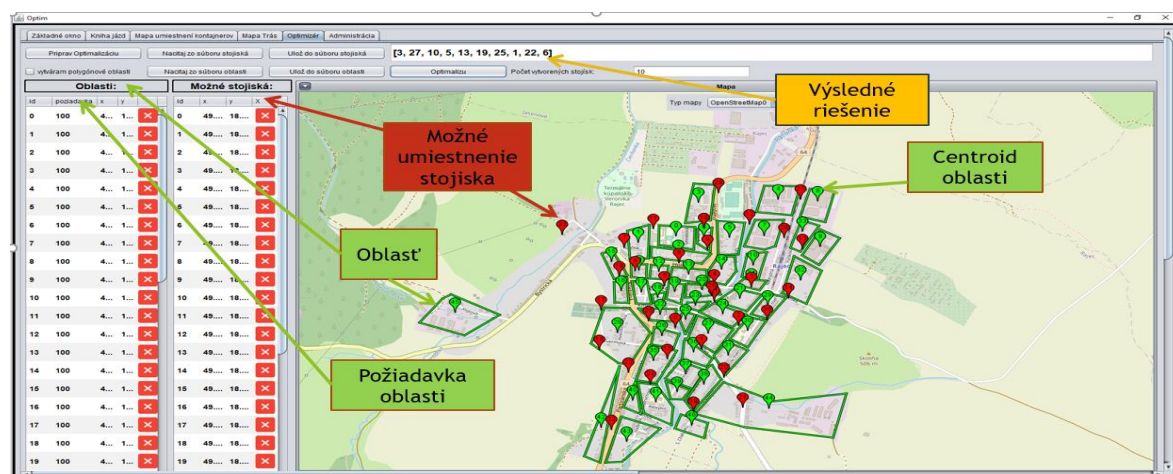
Obrázok 50. Zobrazenie zberných trás

Ďalšou hlavnou záložkou je „Optimizér“, ktorý je určený na optimalizovanie rozmiestnenia zberných nádob. Obsahuje nasledujúce komponenty:

- „Priprav optimalizáciu“ - tlačidlo, pripraví nástroj na optimalizáciu
- „Načítaj zo súboru stojiská“ – načíta zo súboru možné umiestnenia stojísk a vloží ich do tabuľky Možné stojiská
- „Načítaj zo súboru oblastí“ – načíta zo súboru oblasti a vloží ich do tabuľky Oblasti
- „Ulož do súboru stojiská“ – uloží dáta do súboru o možných stojiskách
- „Ulož do súboru oblastí“ – uloží dáta do súboru o oblastiach
- „Optimalizuj“ – spustí sa optimalizácia.

Okrem načítania možných umiestnení stojísk a oblastí zo súboru je možné ich pridať aj manuálne. Vytvorenie možného stojiska sa robí kliknutím pravého tlačidla myši na mapu, kde chcete možno umiestniť stojisko. Pre pridávanie oblastí je potrebné zaškrtnúť „vytváram polygónové oblasti“ a postupným klikaním pravého tlačidla myši na mapu možno pridať krajné body oblasti. Posledný krajný bod oblasti, ktorý sa automaticky spojí s prvým možno pridať dvoj-klikom pravým tlačidlom myši. Po uzavretí oblasti sa systém spýta na

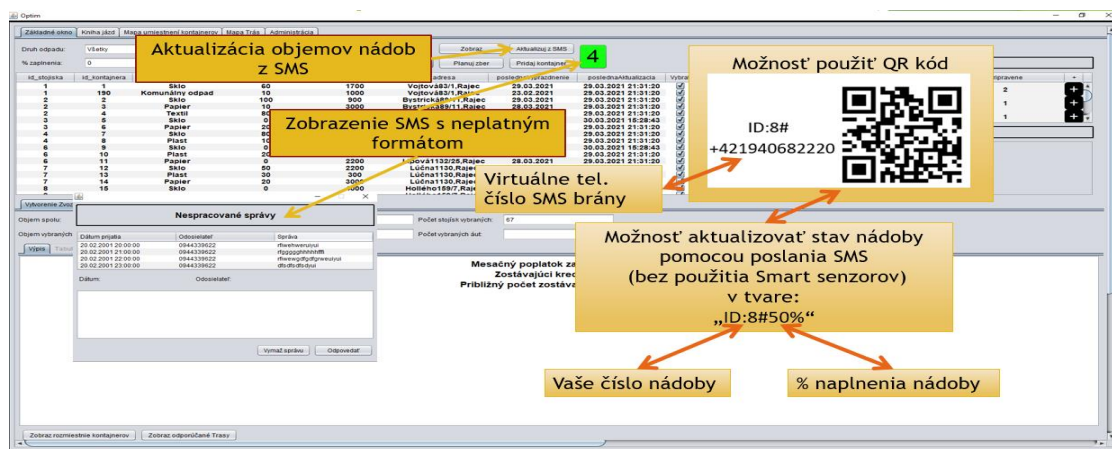
požiadavku danej oblasti ktorú treba vyplniť. Vybrané možné umiestnenia stojiska alebo oblasti možno vymazať kliknutím na ikonu červené X v príslušnej tabuľke.



Obrázok 51. Optimizér stojísk

Poslednou hlavnou záložkou je „Administrácia“, ktorá je bežnému užívateľovi zablokovaná. Je vytvorená pre spracovanie vstupného GEOJASON súboru potrebného na naplnenie grafovej údajovej štruktúry a pre vypočítanie matice vzdialenosti.

Vzhľadom k tomu, že zavedenie Smart senzorov pre monitorovanie naplnenosti zberných nádob je ekonomicky veľmi náročné, rozhodol som sa vymyslieť lacnejší spôsob vyriešenia problému s meraním naplnenosti. Navrhujem, že sa na zberné nádoby nakreslí ryska, po ktorej prekročení, by aktívni obyvatelia mesta, ktorí by mohli byť za to odmeňovaní poslali SMS pomocou oskenovania QR kódu umiestneného na zbernej nádobe, ktorý by obsahoval požadovaný formát SMS pre môj informačný systém.



Obrázok 52. Fungovanie monitorovanie objemu odpadu pomocou SMS

Záver

Počas vypracovávania diplomovej práce som využil rôzne poznatky, ktoré som získal v priebehu bakalárskeho a inžinierskeho štúdia na Fakulte riadenia a informatiky Žilinskej univerzity. V priebehu realizácie diplomovej práce som získal veľa nových poznatkov a zručností. Rozšíril som znalosti o ďalší druh relačnej databázy MySQL, zlepšil som si znalosti o geografickom informačnom systéme QGIS, čiastočne som si naštudoval problematiku odpadového hospodárstva, zlepšil som si komunikačné schopnosti s klientami aj s dodávateľmi IT technológií. Mnohé takto nadobudnuté znalosti a zručnosti v mojej ďalšej kariére s určitosťou využijem.

Hlavnou úlohou mojej diplomovej práce bolo zanalyzovať, navrhnúť a následne vytvoriť vhodný informačný systém určený pre základné potreby pracovníka sekcie odpadového hospodárstva mesta Rajec. Hotová aplikácia uľahčuje a urýchľuje rutinné činnosti pracovníka a poskytuje mu ideálny nástroj pre plánovanie, navrhovanie a triedenie zberných jazd separovaného odpadu. Pracovníkovi tiež poskytuje výborný prehľad o plánovaných, prebiehajúcich alebo ukončených jazdách, o stave vozového parku ale hlavne o stave naplnenosti jednotlivých zberných nádob OH. Monitoring zberných nádob pomocou smart senzorov. Pre lepšie separovanie odpadu IS poskytuje optimizér umiestňovania stojísk, ktorým užívateľ môže zabezpečiť občanom čo najlepšiu dostupnosť zberných nádob.

Prvým krokom vypracovávania praktickej časti DP bolo zoznámiť sa s problematikou odpadového hospodárstva mesta Rajec, jeho organizáciou a riadením. Následne som zozbieral informácie, požiadavky a geografické dáta, ktoré som analyzoval a spracoval. Z geografickým dát som si pomocou nástroja QGIS vytvoril vstupný JSON súbor, ktorý reprezentoval cestnú infraštruktúru mesta Rajec. Pre takto získaný súbor, som si vytvoril nástroj (balíček graf) pre vytvorenie a naplnenie grafovej štruktúry a získanie matice vzdialenosti pre optimalizačné úlohy. Najdôležitejšou a najťažšou výzvou bolo vhodne si zvoliť čo najlepší výber metód potrebných pre riešenie daných optimalizačných úloh. Vybrané metódy som sa pokúsil naimplementovať čo najefektívnejšie. Po naprogramovaní optimalizačných heuristik a algoritmov som pristúpil k fáze navrhnutia štruktúry celého informačného systému, ktorého súčasťou bolo aj vytvorenie

databázového modelu. Vývojom a spájaním jednotlivých komponentov som IS doviedol až do funkčnej podoby. IS som začal testovať a opravovať chyby zistené testovaním. Aplikáciu informačného systému som týmto pripravil na prezentáciu pre zástupcov mesta Rajec. Na základe zapracovania pripomienok, je IS pre odpadové hospodárstvo pripravený na zavedenie do testovacej praxe.

Z uvedeného možno konštatovať, že mnou vytvorený informačný systém určený pre OH je možno uplatniť aj pre iné obce i mestá v SR po upravení inicializačných dát.

Zoznam použitých zdrojov

1. Bari, A., 2018. *7 Branch and Bound Introduction*. [online] Dostupné z: https://www.youtube.com/results?search_query=branch+and+bound.
2. Ciba, M., 2015. *Návrh optimalizačných algoritmov na báze simulovania kolónie mravcov*. [online] Dostupné z: https://www.fei.stuba.sk/buxus/docs/2015/autoreferaty/Autoref_Ciba.pdf.
3. COM-TRADE s.r.o., *SMStools*. [Online] Dostupné z: <https://www.smstools.sk>.
4. COM-TRADE s.r.o., *O2 SMS Connector API dokumentácia*. [online] Dostupné z: <https://smstools.sk/downloads/SMSTOOLS-API-dokumentacia.pdf>.
5. Gurobi Optimization, LLC.. *Documentation*. [online] Dostupné z: https://www.gurobi.com/documentation/9.1/examples/tsp_java.html.
6. Hanzálek, Zdeněk, a kol., 2020. *ČVUT FEL - Combinatorial Optimization*. [online] Dostupné z: https://cw.fel.cvut.cz/b192/_media/courses/ko/10_tsp.pdf.
7. Janáček, J., *OPTIMALIZACE na dopravních sítích*. 2.prepracované vyd. Žilina : EDIS-vydavateľstvo ŽU, 2006. 248 s. ISBN 80-8070-586-0.
8. OSM. *JXMapView2*. [online] Dostupné z: <https://wiki.openstreetmap.org/wiki/JXMapView2>.
9. Roberti, R., Toth, P., 2012. *Models and algorithms for the Asymmetric Traveling Salesman Problem: an experimental comparison* [online] Dostupné z: <https://link.springer.com/article/10.1007/s13676-012-0010-0#Sec2>.
10. SENSONEO j. s. a., *Webová stránka spoločnosti*. [online] Dostupné z: <https://sensoneo.com/sk/>.
11. SENSONEO j. s. a., *Smart senzory*. [online] Dostupné z: <https://sensoneo.com/sk/product/smart-senzory/>.
12. Schmitt, J. P., 2017. *Ant Colony Optimization to solve Travelling Salesman Problem*. [online] Dostupné z: <https://github.com/schmittjoaopedro/aco-tsp-java/blob/master/README.md>.

13. Steiger, M., 2021, *xmapviewer2*. [online] Dostupné z: <https://github.com/msteiger/jxmapviewer2>.
14. Vasilovský, P., 2016. *Skupinový genetický algoritmus pre kapacitný p-medián*. Žilina. 51s.
15. Wikipedia.sk, *Rajec*. [Online] Dostupné z: <https://sk.wikipedia.org/wiki/Rajec>.

Zoznam príloh

Príloha A

Prílohy

Príloha A: Obsah CD

Priložené CD obsahuje:

- Práca v elektronickej podobe (formát PDF)
- Zdrojové kódy aplikácie
- Použité knižnice
- Technickú dokumentáciu
- Export databázy
- Obrázky z aplikácie
- Diagramy