VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

OPTIMALIZACE PROCESŮ V LOGISTICE S PODPOROU VIZUALIZACE

OPTIMIZATION OF PROCESSES IN LOGISTICS WITH VISUALIZATION SUPPORT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. MARTIN KRŠÁK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. ZBYNĚK KŘIVKA, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Kršák Martin, Bc.**
Program: Informační technologie Obor: Informační systémy
Název: **Optimalizace procesů v logistice s podporou vizualizace**
Optimization of Processes in Logistics with Visualization Support
Kategorie: Algoritmy a datové struktury

Zadání:

1. Seznamte se s problematikou plánování se zaměřením na využití v logistice a nastudujte různé přístupy k jejímu řešení (přesné, aproximační, heuristické (genetické)).
2. Specifikujte problém, na který se při plánování zaměříte a v jehož kontextu porovnáte vybrané algoritmy.
3. Navrhněte řešení optimalizované z pohledu zadané podúlohy. V úvahu berte i potřebu následné vizualizace výsledků.
4. Toto řešení implementujte a na základě dat ověřte přínos nového řešení oproti existujícím alternativám.
5. Vybraný algoritmus připravte pro reálné využití (např. ve formě dokumentované knihovny).

Literatura:

- S. Kaiafa, A. P. Chassiakos. A genetic algorithm for optimal resource-driven project scheduling. In: *Procedia Engineering* 123 (2015), pp. 260-267
- Y.C. Liu, H.M. Gao, S.M. Yang, C.Y. Chuang. Application of Genetic Algorithm and Fuzzy Gantt Chart to Project Scheduling with Resource Constraints. In: Huang DS., Jo KH., Wang L. (eds) *Intelligent Computing Methodologies* (ICIC 2014, LNCS 8589). Springer, Cham, 2014, pp. 241-252.
- W. Han, X. Zhang, H. Jiang, W. Li. An Ant Colony Optimization Algorithm for Software Project Management. In: 7th International Conference on Control and Automation, Haikou, 2014, pp. 19-23.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Křivka Zbyněk, Ing., Ph.D.**
Konzultant: Vopěnka Václav, Ing., SAP
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 28. října 2018

Abstrakt

Cieľom diplomovej práce je návrh, implementácia a porovnanie algoritmov, ktoré optimalizujú procesy v logistike, prevažne v plánovacej časti. Algoritmy pomocou heuristík a aproximačného genetického algoritmu nájdu takmer optimálne riešenie NP-ťažkého problému, podobného problému obchodného cestujúceho s oneskorením niekoľkých hodín. Úlohou týchto algoritmov je plánovanie efektívnej trasy smetiarskym vozidlám, ktoré zvážajú a rozvážajú veľkoobjemný odpad do zberných stredísk v konkrétnom meste. Cieľom optimalizácie je minimalizácia nákladov na dopravu.

Abstract

The master thesis aims to design, implement, and compare algorithms that optimize processes in logistics, mainly in the planning phase. Heuristics and approximation genetic algorithms will find an near-optimal solution to NP-hard problem, such as the traveling salesman problem, with a delay less than several hours. The role of this algorithm is to plan an efficient route for garbage trucks that collect and distribute large-scale waste to waste yards in a specific city. The goal of the optimization is to minimize the shipping costs.

Kľúčové slová

Optimalizácia, logistika, genetický algoritmus, hodnotiaca funkcia, metóda výberu, kríženie, mutácia, populácia, jedinec, chromozóm, NSGA-2, Ganttov diagram, smart city.

Keywords

Optimization, logistics, genetic algorithm, fitness function, selection method, crossover, mutation, population, individual, chromosome, NSGA-2, Gantt chart, smart city.

Citácia

KRŠÁK, Martin. *Optimalizace procesů v logistice s podporou vizualizace*. Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Zbyněk Křivka, Ph.D.

Optimalizace procesů v logistice s podporou vizualizace

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením Ing. Zbyňka Křivky Ph.D. Ďalšie informácie mi poskytli pán Ing. Michal Bidlo, Ph.D. a pán Ing. Václav Vopěnka. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Martin Kršák
22. mája 2019

Poďakovanie

Týmto ďakujem svojmu vedúcemu diplomovej práce Ing. Zbyňkovi Křivkovi, Ph.D., konzultantovi Ing. Michalovi Bidlovi, Ph.D. a externému konzultantovi Ing. Václavovi Vopěnkovi za ich rady, pripomienky, odbornú pomoc a čas, ktorý mi venovali pri vytváraní práce.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Kapitola 1

Úvod

Prvá časť diplomovej práce, ktorú tvoria kapitoly 2 až 4, informuje o problematike optimalizácie procesov v plánovaní.

V druhej kapitole je obecne rozpísaná teória zložitosti s niektorými známymi problémami, ktoré súvisia s problematikou plánovania. Sú definované metódy riešenia týchto problémov, ako aj ich negatíva. V oblasti optimalizácie sú najviac používaným prostriedkom k dosiahnutiu efektívneho riešenia genetické algoritmy.

Genetické algoritmy boli navrhnuté Johnom Hollandom v 70. rokoch minulého storočia a ich princíp je inšpirovaný Darwinovou evolučnou teóriou a Mendelovou teóriou dedičnosti. Napodobňujú prírodné princípy, ktoré fungujú už miliardy rokov.

Tretia kapitola má za cieľ predstaviť a definovať princíp genetických algoritmov a ich využitie v problematike optimalizácií. V tejto kapitole sú definované základné pojmy, vyskytujúce sa v genetických algoritmoch, ako napr. gén, chromozóm, mutácia a pod., ako aj rôzne metódy návrhu genetického algoritmu.

Kapitola 4 popisuje rôzne algoritmy, ktoré optimalizujú viackriteriálne problémy.

Druhá časť diplomovej práce sa zaoberá rozborom problému (v kapitole 5) a návrhom riešenia (v kapitole 6). Rozborom problému zistíme, že problém nie je triviálny a jeho riešenie je potrebné riešiť heuristickými alebo aproximačnými metódami. Problém, ktorý práca optimalizuje, je modifikácia problému TSP. Tento problém sa dá nazvať, ako hľadanie optimálnych trás pre smetiarske vozidlá, ktoré v konkrétnom meste zvážajú veľkoobjemný odpad, za účelom minimalizácie nákladov na dopravu.

Voľba optimalizačnej metódy tohto problému bola zvolená na základe informácií o najpoužívanejších metódach problému TSP (Traveling Salesman Problem) a jeho špeciálnych modifikáciách z triedy VRP (Vehicle Routing Problem). Na riešenie tohto problému bola vybraná optimalizačná metóda - genetický algoritmus.

Tretia časť diplomovej (kapitola 7) práce popisuje implementáciu navrhovaného algoritmu. Implementované sú tri metódy optimalizácie: 1. jednoduchá metóda, ktorá zodpovedá súčasnému plánovaniu, 2. algoritmus s agregovanými účelovými funkciami a 3. algoritmus NSGA-II. Posledné 2 algoritmy spadajú medzi algoritmy multikriteriálnej optimalizácie.

V poslednej časti diplomovej práce (kapitola 8) sú znázornené experimenty nad parametrami jednotlivých metód a vzájomné porovnanie výsledkov týchto algoritmov.

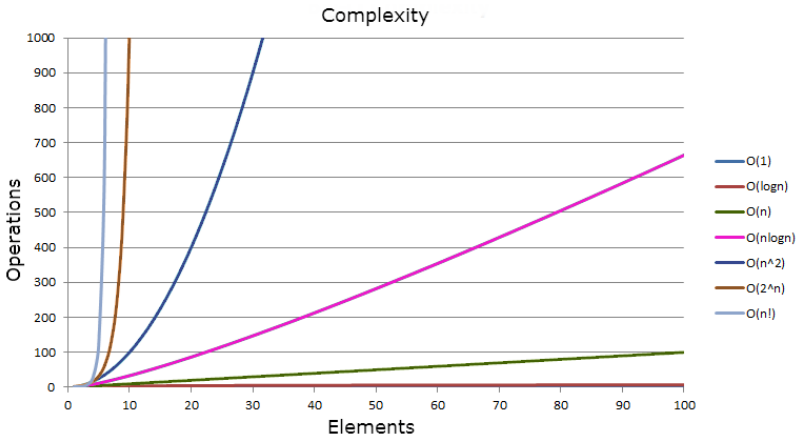
V závere diplomovej práce je zhodnotené využitie týchto metód na optimalizáciu plánovania veľkoobjemných kontajnerov.

Kapitola 2

Teória zložitosti

Teória zložitosti (Computational complexity theory) je časť teoretickej informatiky zaoberajúca sa množstvom požadovaných zdrojov pre vyriešenie daného problému. Najužívanejším zdrojom je čas (počet elementárných operací (krokov algoritmu), ktoré budú prevedené pri výpočte problému s danými vstupmi) a priestor (koľko pamete je potrebné alokovať pre vyriešenie problému) (doslovná citácia z [19]). Na základe týchto zdrojov klasifikujeme algoritmy do tried zložitostí (definované viz [13]).

Rozlišujeme, či má algoritmus *polynomiálnu* časovú zložitosť, ak jeho časová zložitosť je v tvare $O(n^k)$, o rozsahu n pre nejakú konštantu k alebo či má algoritmus *exponenciálnu* časovú zložitosť, napríklad algoritmy s časovou zložitosťou $O(n!)$, $O(2^n)$ alebo *konštantnú* časovú zložitosť v tvare $O(1)$ (doslovná citácia z [10]). Inými slovami výpočet algoritmu môže skončiť v konštantnom, polynomiálnom alebo exponenciálnom čase (viac viz [10]). Obrázok 2.1 (čerpaný z [17]) znázorňuje triedy zložitosti.



Obr. 2.1: Triedy zložitostí

Konštantnú časovú zložitosť $O(1)$ majú algoritmy, ktorých počet operácií je konštantný (napr. operácie push a pop na zásobníku). V opačnom prípade, polynomiálna a exponenciálna časová zložitosť je závislá od n, čo udáva veľkosť inštancie problému (napr. počet uzlov v grafe apod.). Výpočetne najnáročnejšie sú algoritmy s exponenciálnou časovou zložitosťou a ich výpočet na bežných počítačoch môže byť z hľadiska času nekonečný (viac viz [6]). V tabuľke 2.1 sú znázornené doby výpočtu algoritmu pri danej časovej zložitosti na počítači s frekvenciou pocesoru 3 GHz, ktorý zvládne $3 * 10^9$ operácií za sekundu.

	n=10	n=100	n=1000	n=10 000	n=100 000	n=1 000 000
$\log n$	1.1 ns	2.2 ns	3.3 ns	4.4 ns	5.5 ns	6.6 ns
n	3.3 ns	33.3 ns	0.3 μs	3.3 μs	33.3 μs	0.3 ms
n^2	33.3 ns	3.3 μs	0.3 ms	33.3 ms	3.3 s	5.5 min
n^3	0.3 μs	0.3 ms	0.3 s	5.5 min	92.5 hod	10.5 rokov
n^4	3.33 μs	33.3 ms	5.5 min	38.5 dní	1057 rokov	10^6 rokov
2^n	0.3 μs	10^{14} rokov	10^{286} rokov	$\approx \infty$	$\approx \infty$	$\approx \infty$

Tabuľka 2.1: Časová zložitosť na počítači s frekvenciou 3 GHz

Z tabuľky 2.1 môžeme pozorovať narastajúcu dobu výpočtu algoritmu prevažne v exponenciálnej časovej zložitosti $O(2^n)$ a s narastajúcim stupňom polynómu v polynomiálnej časovej zložitosti ($O(n^3)$, $O(n^4)$). Pre takéto úlohy, počítač s 3 GHz výpočetnou silou nie je schopný nájsť optimálne riešenie exaktnými metódami v rozumnom čase. Je potrebné zvoliť aproximačné a heuristické metódy, ako sú napríklad genetické algoritmy alebo simulované žíhanie, ktoré poskytujú riešenia v prijateľnej dobe.

Polynomiálne algoritmy obecne delíme na dve triedy problémov: P (polynomiálne problémy) a NP (nedeterministicky polynomiálne problémy). Triviálne platí inklúzia $P \subseteq NP$, ktorá vychádza z dokázaného tvrdenia $L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq NPSPACE \subseteq EXP$. (Ne)ekvivalencia P a NP je v súčastnosti najvýznamnejším otvoreným problémom, za ktorého vyriešenie je vypísaná odmena 1 000 000 amerických dolárov (viac viz [10]).

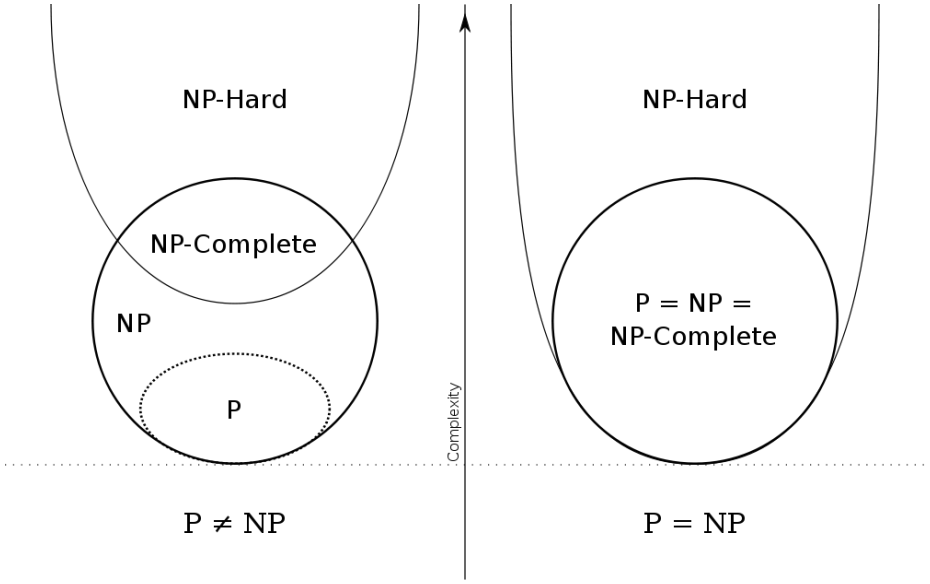
2.1 Trieda P

Do zložitostnej triedy P patria všetky (rozhodovacie) problémy, ktoré sa dajú riešiť v polynomiálnom čase na deterministickom Turingovom stroji (doslovná citácia z [10]). Trieda zahrňuje netriviálne problémy, ktoré na rovnaký vstup reagujujú vždy rovnakým výstupom a pri každom kroku algoritmu je vždy jednoznačne definovaný aj krok nasledujúci. P je trieda prakticky riešiteľných problémov ako sú napríklad, rozpoznávanie palindromu alebo hľadanie podreťazca v reťazci. Väčšinu úloh v zložitostnej triede P je možné vypočítať presnými algoritmami v prevažne krátkej dobe. S narastajúcim stupňom polynomu alebo narastajúcim rozsahom n, sa úloha z triedy P môže veľmi rýchlo dostať do superpolynomiálnej časovej zložitosti, kedy výpočet algoritmu môže trvať aj niekoľko rokov. V takomto prípade exaktné metódy zlyhavajú (viac viz [10]).

2.2 Trieda NP

Do zložitostnej triedy NP patria všetky (rozhodovacie) problémy, ktoré sa dajú riešiť v polynomiálnom čase na nedeterministickom Turinogovom stroji (doslovná citácia z [10]). Algoritmy NP problémov môžu na rovnaký vstup dať výsledok rôzny s rôznou postupnosťou elementárnych krokov algoritmu. Problém z triedy NP môže byť výnimočne predaný do triedy P, ak sa nájde polynomiálny algoritmus, ktorý by tento problém riešil (viac viz [10]).

V súvislosti s NP problémami definujeme *NP-úplné* (NP-complete) a *NP-ťažké* (NP-hard) problémy (definované viz [10]). Na obrázku 2.2 (čerpaný z [12]) sú znázornené jednotlivé triedy NP-hard a NP-complete, v prípade kedy by sa P $\neq$ NP a P = NP.



Obr. 2.2: Triedy zložitosti P a NP

NP-úplné problémy definujeme ako problémy, na ktoré sú polynomiálne redukovateľné všetky ostatné problémy z NP (doslovná citácia z [10]). Inými slovami, NP-úplné úlohy tvoria najťažšie úlohy z NP. Ak by sa našiel algoritmus, ktorý by dokázal príslušnosť NP-úplného problému do zložitostnej triedy P (bol by nájdený deterministický polynomiálny algoritmus pre NP-úplnú úlohu), znamenalo by to, že všetky nedeterministicky polynomiálne úlohy, sú riešiteľné v polynomiálnom čase a platilo by tvrdenie P=NP. Zatiaľ sa takýto algoritmus nevymyslel (viac viz [10]).

NP-ťažké úlohy v teórií výpočetnej zložitosti sú definované ako trieda problémov, ktoré sú neformálne "prinajmenšom tak ťazké ako najťažšie problémy z NP". Problém Q je NP-ťážký, keď každý problém H v NP môže byť redukovaný v polynomiálnom čase na Q (doslovná citácia z [10]). Z toho vyplýva, že keby bol vymyslený polynomiálny algoritmus, ktorý by riešil niektorý z množiny NP-ťazkých problémov, všetky problémy z NP by boli riešiteľné polynomiálnymi algoritmami (viac viz [10]).

V dnešnej dobe, na riešenie NP-úplných úloh sa používajú tieto techniky (definované viz [10]):

- **Aproximácia -** Namiesto hľadania optimálneho riešenia, hľadá riešenie, ktoré je blízko k optimálnemu riešeniu.
- **Randomizácia -** Používa náhodnosť pre získanie rýchlejšieho priemerného času behu a s malou pravdepodobnosťou dovoluje algoritmu zlýhať.
- **Obmedzenie -** Obmedzením štruktúry vstupu (napríklad na rovinné grafy) sa algoritmy prevažne zrýchlia.
- **Parametrizácia -** Často existujú rýchle algoritmy, keď sú určité parametre vstupu opravené.
- **Heuristiky -** Algoritmus, ktorý vo väčšine prípadov funguje "primerane dobre", ale pre ktorý neexistuje žiadny dôkaz.

2.3 Príklady NP-úplných problémov

V roku 1971 bol zavedený pojem úplnosti, kde prvým príkladom NP-úplného problému bol problém splniteľnosti (Boolean satisfiability problem - SAT). Ide o problém, kde pre nejaký booleovský výraz s premennými máme rozhodnúť, či existuje nejaké ohodnotenie premenných, pre ktoré má celý výraz platiť. Od tej doby boli nájdené tisíce ďalších NP-úplných problémov. V tejto sekcií sú vypísané niektoré známe a typické NP-úplné problémy.

- *Problém pokrytia grafu* - Je klasický problém teoretické informatiky z triedy NP-úplných problémov. Problém rieši otázku, či pre daný graf a nejaké číslo k, existuje pokrytie grafu o veľkosti k alebo menej. Pokrytím grafu rozumieme množinu vrcholov takú, že pre všetky hrany grafu potom platí, že aspoň jeden koniec hrany končí vo vrchole z množiny pokrytia. Formálne, vrcholové pokrytie grafu G je množina C jeho vrcholov taká, že každá hrana z G je incidentná aspoň s jedným vrcholom z C. Množina C se nazýva pokrytie hran grafu G.

- *Problém kliky* - Klika je taký podgraf nejakého grafu, ktorý je úplným grafom, tzn. jeho všetky vrcholy sú spojené hranou so všetkými ostatnými. Problém nájdenia najväčšej kliky v grafe je NP-ťažký. Problém, či daný graf má kliku o velikosti k je NP-úplný.

- *Problém batohu* - Nech je daných n závaží, z nich má každé jednoznačne určenú hmotnosť. Niektoré z nich vyberieme a dáme ich do uzavretého batohu, ktorý je nepriehľadný (a ktorý má sám nulovú hmotnosť). Potom batoh zvážime a určíme celkovú hmotnosť, z kterej se pokusíme určiť, ktoré závažia sú vnútri v batohu.

- *Problém dvoch lúpežníkov* - Je NP-úplný problém, ako rozdeliť korisť (ocenené veci), medzi dvoch lúpežníkov tak, aby mali korisť rozdelenú rovným dielom (súčet hodnôt v jednej skupine bol rovný súčtu hodnôt v druhej skupine).

- *Problém Hamiltonovej kružnice* - Pre daný graf rozhodnúť, či v tom grafe existuje Hamiltonova kružnica, čo je kružnica (cesta grafom, ktorá začína a končí v rovnakom uzle), ktorá prechádza všetkými uzlami grafu.

- *Izomorfismus grafov* - Rozhodovací problém, či pre dané dva grafy G_1 a G_2 platí, že G_1 je izomorfný s nejakým podgrafom G_2.

- *Subret sum problem* - Je dôležitý problém hlavne v kryptografií. Pre danú množinu celých čísiel chceme rozhodnúť, či existuje nejaká jej podmnožina taká, že súčet čísiel tejto podmnožiny je 0 (prípadne akékoľvek k). Napríklad pre množinu {1,5,3,10,-14,-5} je odpoveď ÁNO, pretože podmnožina {1,3,10,-14} má súčet 0.

- *Problém ofarbenia grafu* - Je rozhodovací problém, či sa dá daný graf ofarbiť s použitím k farieb alebo menej. Inými slovami, graf je korektne ofarbený, ak každému vrcholu priradíme číslo (farbu) a platí, že žiadne dva susedné vrcholy niesú ofarbené rovnakou farbou.

- *Problém štyroch farieb* - Rozhodovací problém, či stačia štyri farby na ofarbenie ľubovoľnej politickej mapy tak, aby žiadne dva susediace štáty neboli ofarbené rovnakou farbou. Za susedné štáty sú považované také, ktoré majú spoločnú hraničnú čiaru, tzn. nesusedia spolu iba v jednom bode. Problém rozhodnúť, či danú mapu je možné ofarbiť 3 farbami, je vzhľadom k počtu štátov (uzlov) NP-úplný. Problém rozhodnúť, či obecný (nerovinný) graf je možné ofarbiť 4 farbami, je NP-úplný.

- *Bin packing problém* - je problém, kde treba rozhodnúť, ako vložiť čo najviac objektov do čo najmenšieho počtu zásobníkov (bin) s konštantnou veľkosťou, resp. nech je daný zásobník s kapacitou C > 0 a zoznam objektov $\{p1, p2, p3, ..., p_i\}$. Aký je najmenší počet daných zásobníkov potrebných na uskladnenie všetkých objektov. Pre všetky p_i s veľkosťou s_i platí $0 \leq s_i \leq C$. Teda žiadny objekt nie je väčší ako veľkosť zásobníka. Formálnejšie ide o nájdenie rozdelenia a priradenia množiny objektov tak aby všetky obmedzenia boli dodržané a funkcia cieľa bola minimalizovaná (maximalizovaná).

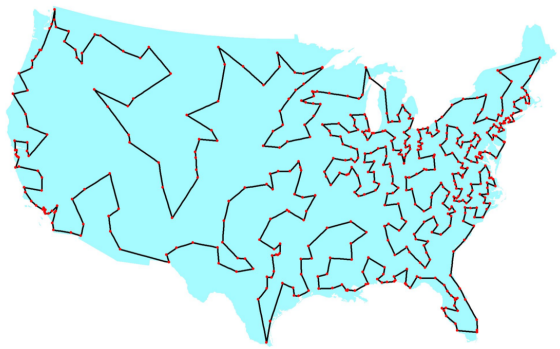
V nasledújúcich podsekciách sú detailne rozpísané NP-úplné problémy, ktoré sa objavujú v oblasti logistiky a ktorých princípy sú používané v tejto práci.

2.3.1 Problém obchodného cestujúceho

Problém obchodného cestujúceho (angl. traveling salesman problem - TSP), ktorý je otvorený už cez 50 rokov, je založený na princípe nájdenia nejkratšej cesty medzi N mestami. Pričom každé mesto má byť navštívené práve jeden krát a obchodný cestujúci by mal skončit v meste, kde začínal svoju cestu (doslovná citácia z [14]). Matematická formulácia problému znie: "V danom ohodnotenom úplnom grafe nájdite nejkratšiu hamiltonovskú kružnicu". Hamiltonovská kružnica je definovaná ako kružnica ktorá prechádza všetkými uzlami grafu (doslovná citácia z [3]).

Optimalizačná verzia problému obchodného cestujúceho patrí medzi NP-ťažké problémy. Rozhodovacia verzia problému TSP je NP-úplná, tzn. že existuje nedeterministický Turingov stroj (nedeterministický algoritmus), ktorý dodá odpoveď ÁNO alebo NIE, vždy po maximálne polynomiálnom počte krokov (doslovná citácia z [14]).

Príklad TSP, kde máme 30 miest, ktoré v grafe reprezentujú 30 uzlov, má približne $2.65 * 10^{32}$ možností naplánovania trasy pre obchodného cestujúceho. Pri rýchlosti spracovania, miliardu operácií za sekundu, by nám to trvalo asi 252 333 390 232 297 rokov. V praxi sa podobná úloha rieši iba približne a to prevažne genetickými algoritmami v kombinácií s tabu prehľadávaním. Týmto spôsobom sa dosahujú prakticky použiteľné časy výpočtu TSP, pričom riešenie nemusí byť optimálne. Napríklad pre 100 miest, výpočet TSP pomocou genetických algoritmov v kombinácií s tabu prehľadávaním dokáže do jednej minúty (viac viz [14]). Ukážka TSP trasy na obrázku 2.3 (čerpaný z [4]).



Obr. 2.3: TSP trasa pre 523 miest v USA v roku 1987

2.3.2 Problém kupujúceho cestujúceho

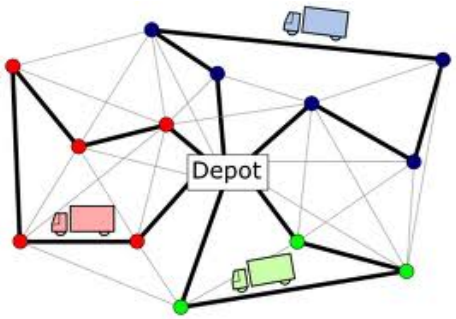
Problém kupujúceho cestujúceho (angl. traveling purchaser problem - TPP) je NP-ťažký problém, ktorý je špeciálny prípad TSP problému. Je daný zoznam tržníc, cena cesty medzi jednotlivými tržnicami, zoznam dostupného tovaru na jednotlivých tržniciach a ceny položiek tovaru na každej tržnici. TPP hľadá optimálnu trasu pre kupujúceho (ktorý má zoznam položiek, ktoré má kúpiť) s minimom kombinovaných nákladov za trasu a nakupovanie.

TSP je jednoduchšia verzia TPP. TPP je zobecnený TSP, keď berieme v úvahu, že každá kúpna položka je dostupná len na jednej tržnici a každý trh predáva len jednu položku. Keďže TSP je NP-ťažký, potom aj TPP je NP-ťažký. V praxi sa na vyriešenie TPP používa dynamické programovanie v kombinácií s tabu prehľadávaním.

2.3.3 Problém okružných jázd

Problém okružných jázd (angl. vehicle routing problem - VRP) je obecné meno pre celú triedu kombinatorických optimalizačných problémov. V tejto triede ide o nájdenie množiny ciest pre skupinu vozidiel (vehicle), sústredených v jednom alebo viacerých skladiskách (depot), ktoré zásobia určený počet miest alebo zákazníkov, geograficky rozložených na mape, podľa potreby. Cieľom je obslúžiť požiadavky celej množiny zákazníkov s minimalizáciou nákladov a času, na uskutočnenie všetkých ciest, ktoré začínajú aj končia v sklade.

VRP je dobre známy celočíselný optimalizačný problém spadajúci pod kategóriu NP-ťažkých problémov. Preto na hľadanie riešení daných problémov používame aproximačné metódy a heuristiky. Zložitosť VRP problému vo veľkej miere vyplýva z faktu, že VRP problém leží niekde na priesečníku dvoch dobre známych a študovaných problémov Bin Packing Problém (BPP) a Traveling Salesman Problém (TSP). Na vyriešenie VRP potrebujeme vyriešiť obidva tieto problémy. Treba podotknúť, že tieto problémy tiež patria do triedy NP-ťažkých problémov. A teda už nájdenie riešenia pre každý z nich osobitne vyžaduje veľkú námahu a je časovo náročné. Ukážka na obrázku 2.4 (čerpaný z [2]).

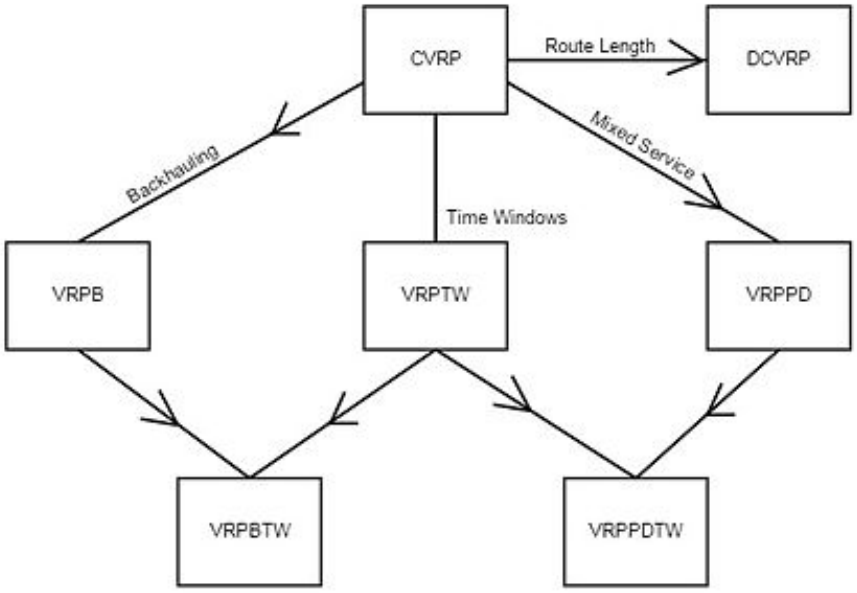


Obr. 2.4: Problém okružných jázd

Pod VRP spadá celá trieda problémov. V každom z nich síce ide o nájdenie ciest pre zásobenie zákazníkov tovarom zo skladu, ale postupným študovaním týchto problémov a hlavne skutočnými požiadavkami, ktoré prichádzali z prostredia praxe, vznikali nové prídavné obmedzenia. Pridávanie týchto obmedzení a rozširovaní dalo za vznik rôznym variantom VRP problému. Teraz si predstavíme niektoré najznámejšie VRP modifikácie.

- *Capacited Vehicle Routing Problem (CVRP)* - Obsluha jedným vozidlom s danou kapacitou.
- *Split Delivery VRP (SDVRP)* - Obsluha rozdelenou dodávkou (viac vozidlami).
- *VRP with Time Windows (VRPTW)* - S časovými oknami.
- *Multiple Depot VRP (MDVRP)* - S viacerými depami.
- *Stochastic VRP (SVRP)* - Stochastický VRP.
- *VRP with Backhauls (VRPB)* - Všetko vyložiť, následne naložiť.
- *VRP with Pick-Up and Delivering (VRPPD)* - Kedykoľvek je možné nakládať a vykládať.
- *Open Vehicle Routing Problem (OVRP)* - Vozidlá sa nemusia vrátiť do skladiska.
- *Vehicle routing problem with multiple trips (VRPMT)* - Vozidlá môžu prejsť viac ako jednu cestu.

Na obrázku 2.5 (čerpaný z [11]) sú znázornené VRP varianty a vzťahy medzi nimi.



Obr. 2.5: VRP varianty a ich vzťahy

Vzhľadom k obtiažnosti riešiť optimalizáciu rozsiahlych prípadov problému VRP, bolo venované veľké množstvo úsilia metaheuristikám, ako sú genetické algoritmy, tabu prehľadávanie a simulované žíhanie. Niektoré z najnovších a efektívnych metaheuristík pre VRP dosahujú riešenia v rozsahu 0,5 % až 1 % z optimálneho riešenia, pre náročné prípady počítania stoviek až tisíc dodacích bodov. Tieto metódy sú robustnejšie v tom zmysle, že je ich možné lepšie prizpôsobiť rôznym bočným obmedzeniam.

2.3.4 Problém čínského poštára

Problém čínského poštára (angl. chinese postman problem - CPP) hľadá najkratšiu cestu pre poštára, ktorý vyráža z pošty a musí prejsť všetkými ulicami mesta, pričom rozdáva poštu a musí sa vrátiť späť na poštu. V grafe G, ktorý reprezentuje mesto, sú hrany h_i reprezentujúce ulice a uzly u_i odpovedajúce križovatkám. Hrany sú ohodnotené kladnými číslami, ktoré odpovedajú dĺžkam ulíc. Pokiaľ je v grafu možné previesť Eulerovský ťah, potom je tento ťah optimálnym riešením úlohy (doslovná citácia z [3]). Poštár prejde všetkými ulicami práve jeden krát. Súčet ohodnotených hran udáva dĺžku cesty, ktorú prešiel. V opačnom prípade ak Eulerovský ťah neexistuje, potom musí poštár prejsť niektorými ulicami viackrát, tj. musíme minimalizovať súčet dĺžok opakovane prechádzaných ulíc (doslovná citácia z [3]).

Kapitola 3

Genetické algoritmy

Genetické algoritmy (angl. genetic algorithms - GA) sú malou ale dôležitou podmnožinou evolučných algoritmov. Patria medzi heuristické metody, ktoré sa snažia nájsť riešenie zložitých problémov, pre ktoré exaktné algoritmy zlyhavajú. Ich jednoduchosť a efektivita ich radí medzi obľúbené algoritmy, ktoré dokážu optimalizovať NP-úplné a NP-ťažké úlohy. GA sú inšpirované princípmi evolúcie v prírode, ktoré napodobujú biologické procesy, ako sú dedičnosť, prirodzený výber, mutácie a kríženia (doslovná citácia z [22]).

V roku 1858 Charles Darwin publikoval svoju evolučnú teóriu v knihe *On The Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life (O pôvode druhov (prírodným výberom, čiže uchovaním prospešných plemien v boji o život))* (viac viz [16]). V tej istej dobe slávny genetik George Mendel experimentoval s krížením hrachu a sledoval vzťahy medzi vlastnosťami rodičov a potomkov. Na základe zistení definoval tri Mendelove zákony dedičnosti.

3.1 Základné pojmy

Táto kapitola definuje jednotlivé základné pojmy, ktoré je potrebné poznať k pochopeniu genetických algoritmov.

3.1.1 Gén

Gén (angl. gene) je v informatike predstavovaný ako základná jednotka chromozómu jedinca. Hodnota génu sa nazýva *alela*. Význam génu je definovaný daným problémom a jeho kódovaním v rámci genetického algoritmu (viac viz [20]).

3.1.2 Chromozóm

Chromozóm (angl. chromosome) je vektor génov $v = (a_1, a_2, a_3, ..., a_n)$, kde n vyjadruje počet génov v chromozóme (doslovná citácia z [20]). Vo väčšine prípadov genetických algoritmov sa požívajú chromozómy rovnakej dĺžky, no nájdu sa prípady, kedy je potrebné počítať s premenlivou dĺžkou chromozómu. Jeden chromozóm predstavuje jedno riešenie daného problému.

3.1.3 Jedinec

Jedinec (angl. individual) je ekvivalencia chromozómu. Predstavuje jedno z možných riešení daného problému (doslovná citácia z [16]).

3.1.4 Populácia

Populácia (angl. population) je množina jedincov (chromozómov), ktorá sa mení každou iteráciu genetického algoritmu, kedy sa generujú nový jedinci. S každou generáciu sa vytvára nová populácia z predošlých vhodných jedincov, ktorí boli mutovaní a krížení (viac viz [15]).

3.1.5 Fitness funkcia

Fitness funkcia (angl. fitness function) má dôležitú úlohu v genetických algoritmoch. Hodnotí každého jedinca hodnotou *fitskóre*. fitskóre udáva vhodnosť jedinca pre daný problém (kvalitu riešenia). Fitness funkcia je definovaná pre každý problém zvlášť (doslovná citácia z [15]). Niektoré GA optimalizujú napríklad cenu za prejdené kilometre, prepočítanú na peniaze, kde v takom prípade jedinec s menším fitskóre je lepší od jedinca s väčším fitskóre. A naopak v inom prípade genetického algoritmu, hľadáme jedinca s najvyšším fitskóre.

3.2 Princíp genetických algoritmov

Genetické algoritmy nie je možné definovať obecne pre všetky problémy, ktoré majú riešiť. Využíva sa veľa rôznych techník a algoritmov, ktoré tvoria genetický algoritmus pre daný problém. Väčšinou majú spoločný základ, do ktorého pridávajú parametre vyžadujúce z daného problému. Chromozóm reprezentuje množinu parametrov daného problému. Gény predstavujú jednotlivé parametry a alely sú hodnoty týchto parametrov. Jedinec v GA je definovaný chromozómom a množina chromozómov definuje populáciu.

Genetické algoritmy sa snažia kombinovať a vyberať najlepších jedincov, za účelom konvergencie k optimálnemu riešeniu. Najlepší jedinec (chromozóm) s najlepším fitskóre odpovedá najlepšiemu získanému riešeniu pre daný problém. Základný postup, ktorý je vo väčšine genetických algoritmov rovnaký je v takomto poradí nasledovný:

1. *Inicializácia* - Vytvorenie prvej náhodnej populácie jedincov.
2. *Súťaž* - Nad každým jedincom populácie sa vypočíta fitskóre pomocou fitness funkcie.
3. *Selekcia* - Jednou zo selekčných metód sa vyberú rodičia pre novú generáciu.
4. *Reprodukcia* - Na vybraných rodičov aplikujeme genetické operácie, čím dostaneme novú generáciu.

Genetický algoritmus iteratívne prechádza jednotlivými krokmi až do doby, kedy nové generácie budú generovať rovnaké generácie.

3.3 Kódovanie

Voľba kódovania chromozómu je jeden zo základných problémov pri pracovaní s genetickými algoritmami (doslovná citácia z [16]). Kódovanie je závislé od typu problému, na ktorý hľadáme optimálne riešenie (viac viz [20]).

3.3.1 Binárne kódovanie

Je najpoužívanejšia metóda kódovania (doslovná citácia z [16]). Každý chromozóm je reprezentovaný reťazcom bitov 0 a 1. Binárné kódovanie poskytuje veľké množstvo kombinácií, aj pri malom množstve alel (doslovná citácia z [18]).

Chromozóm A	01101011101001010101010100101
Chromozóm B	10101010111011111011010000101

Tabuľka 3.1: Príklad binárneho kódovania chromozómu

3.3.2 Permutačné kódovanie

Chromozóm kódovaný permutačným kódovaním predstavuje reťazec dekadických čísiel, ktorý tvorý nejakú sekvenciu.

Chromozóm A	5 8 6 5 2 1 0 0 2 3 9 0 1 1 4 6 2
Chromozóm B	7 4 4 5 2 0 8 0 1 1 4 6 0 8 1 2 3

Tabuľka 3.2: Príklad permutačného kódovania chromozómu

3.3.3 Špeciálne kódovania

V prípadoch, kedy parametre GA nie je možné kódovať celými číslami alebo binárne, sa používa napríklad kódovanie pomocou znakov. Ak hľadáme riešenie problému, ktorý pracuje s reálnymi číslami, je možné použiť kódovanie pomocou reálnych čísiel.

Chromozóm A	1.342 5.223 7.359 1.321 4.432
Chromozóm B	H 2 5 R S T Q 9 8 5 7 T R B

Tabuľka 3.3: Špeciálne prípady kódovania chromozómu

3.4 Selekcia

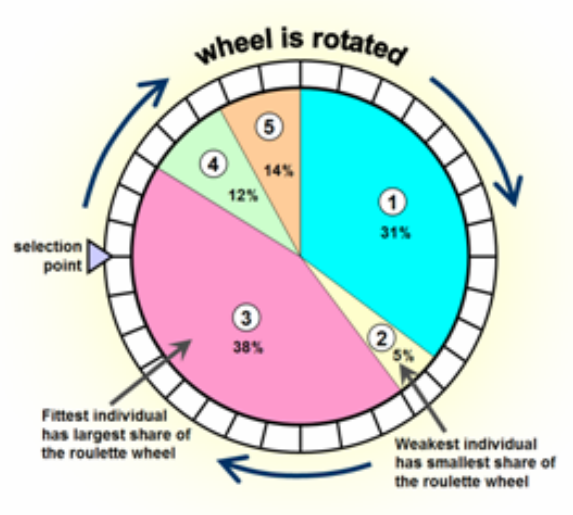
Selekcia je ďalší z postupov GA, ktorá zkvalitňuje nové populácie, tým že niektorou metódou vyberie jedincov, ktorých presunie do novej generácie (doslovná citácia z [20]). Rozlišujú sa dva spôsoby výberu jedincov podľa tzv. *selekčného tlaku* (definované viz [16]):

- **slabý výber** (soft alebo too-weak selection) umožňuje aj slabým jedincom, stať sa novými rodičmi do novej generácie,
- **silný výber** (hard alebo too-strong selection) umožňuje silným jedincom, stať sa novými rodičmi do novej generácie. Slabší jedinci majú v tejto variante malú šancu.

Selekčných metód založených na selekčnom tlaku je niekoľko. Detailne budú rozpísané v nasledujúcich podsekciách. Slabý výber je aplikovateľný v prípadoch, kedy hľadáme riešenia prevažne v neperspektívnych výsledkoch, čo môže veľkým spôsobom spomaliť zkvalitňovanie nových populácií a celkovo spomalí celý genetický algoritmus. V opačnom prípade selekcia, postavená na silnom výbere, má veľmi veľký ťah dopredu, kedy iba nadpriemerní jedinci majú šancu dostať sa do novej generácie.

3.4.1 Vážená ruleta

Vážená ruleta (angl. roulette wheel selection) - je selekčná metóda, ktorá je postavená na princípe, že kvalitnejší jedinec by mal produkovať viac potomkov, ako menej perspektivnejší. Na základe hodnôt fitness funkcií všetkých jedincov z populácie, je pridelená určitá časť kola rulety každému jedincovi podľa fitskóre (doslovná citácia z [16]). Jedincovi s nízkym fitskóre je pridelená menšia časť a naopak jedincovi s vyšším fitskóre je pridelená väčšia časť kola. Následne sa kolo roztočí toľko krát, koľko je jedincov v populácií. Ako rodič novej populácie sa vyberie jedinec, ktorému patrí výherné poličko z rulety. Na obrázku 3.1 (čerpaný z [7]) je znázornený princíp váženej rulety.

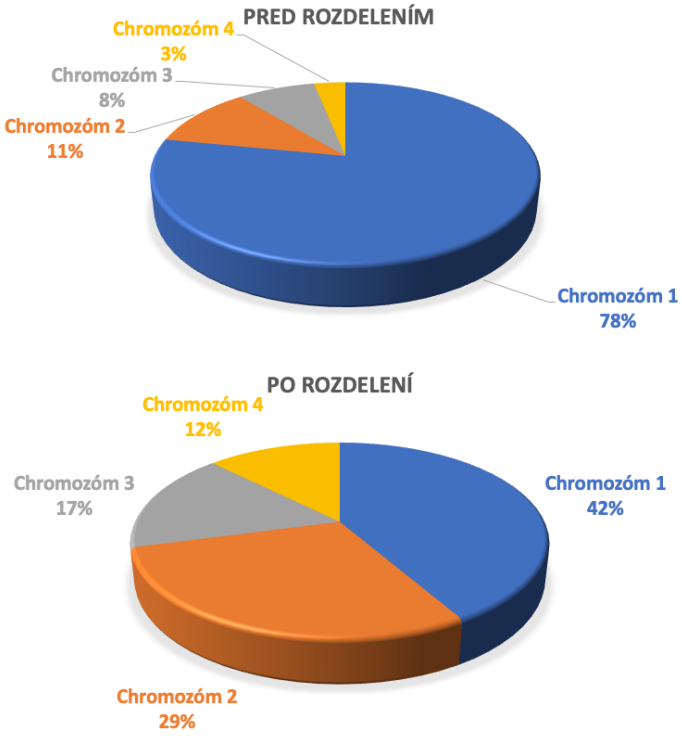


Obr. 3.1: Vážená ruleta

3.4.2 Selekcia pomocou poradia

Selekcia pomocou poradia (angl. rank selection) - je selekčná metóda, ktorá rieši prípad, kedy vo váženej rulete môže mať niektorý jedinec fitskóre 90 %. V takomto prípade jedinci s malým fitskóre majú veľmi malú šancu výberu. Rank selection ohodnotí v poradí jednotlivé chromozómy podľa fitskóre od najlepšieho po najhoršieho, kde najhorší jedinec bude mať hodnotu 1, druhý najhorší bude mať hodnotu 2 a najlepší jedinec bude mať hodnotu N, pri N jedincov populácie. Tieto hodnoty sú veľkosti častí kola rulety. Najlepší jedinec bude mať väčšiu časť ale nie v takom veľkom nepomere oproti slabšiemu, ako to je v metóde váženej rulety (viac viz [16]).

Nevýhoda tejto metódy je spomalenie zkvalitňovania celého algoritmu, pretože najlepší jedinec sa nebude veľmi odlišovať od ostatných a môže sa stať, že najlepší chromozóm nebude vybratý vôbec. Na obrázku 3.2 je znázornená metóda pred a po rozdelení podľa poradia (viac viz [15]).



Obr. 3.2: Selekcia pomocou poradia

3.4.3 Metóda Turnaj

Metóda Turnaj (angl. tournament selection) - je jedna z najpoužívanejších selekčných metód. Je založená na princípe turnaja. Celá populácia sa rozdelí na skupiny o veľkosti n a z každej skupiny je vybraný najlepší jedinec podľa fitskóre (víťaz). Skupina víťazov tvorí jedincov určených k reprodukcií. Výhoda metódy turnaja je, že sa do reprodukcie môže dostať jedinec s pomerne malým fitskóre. Najčastejšie používaná metóda turnaja je o veľkosti skupín $n = 2$. Ďalšou modifikáciou tejto metódy môže byť premenlivá veľkosť pre jednotlivé skupiny.

3.5 Elitizmus

Eltitizmus (angl. elitism) je metóda, ktorá ešte pred krížením a mutáciou zoberie n najkvalitnejších jedincov a presunie ich do novej generácie. Väčšinou po skončení selekcie, títo vybraní jedinci nahradia najhorších jedincov, ktorí prešli selekciou. Touto metódou je zaručená neklesajúca kvalita generácií, pretože ponecháva najlepších jedincov predchádzajúcej generácie (viac viz [20]).

3.6 Reprodukcia

Reprodukcia (angl. reproduction) je tvorená dvoma genetickými operátormi - kríženie a mutácia. Po výbere rodičovských jedincov z populácie pomocou selekčnej metódy sa títo jedinci reprodukujú. Jedinec sa môže stať rodičom aj viac krát (viac viz [20]).

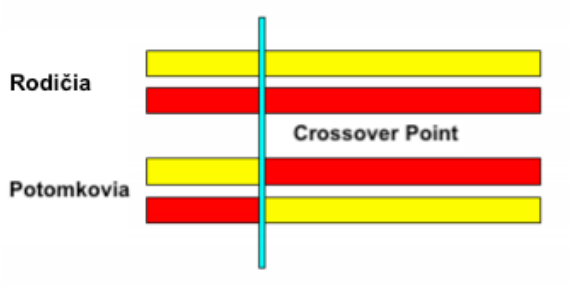
3.7 Genetické operátory

Genetické operátory (angl. genetic operators) tvoria kríženia a mutácie. Analógiu je možné vidieť v prírode, kde ich hlavným prínosom je reprodukovať a variovať jedincov do novej populácie. Existuje veľa variácií týchto operátorov. Okrem týchto dvoch operátorov existujú ďalšie, ktoré sa používajú špecificky iba na určitý druh problémov.

3.7.1 Kríženie

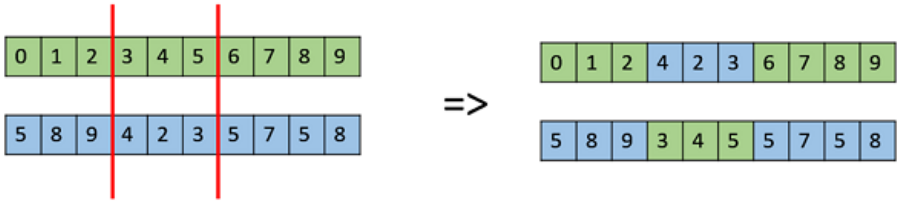
Kríženie (angl. crossover) je prvá operácia reprodukcie, kedy po zvolení rodičov krížime gény medzi sebou a vytvárame tak nových potomkov (doslovná citácia z [21]). Existuje niekoľko verzií krížení. Medzi základné patria jednobodové, dvojbodové a uniformné kríženia (viac viz [16]).

Jednobodové kríženie je kríženie, kedy sa vygeneruje náhodny bod z intervalu (0, dĺžka chromozómu-1). Tento bod sa nazýva *Crossover point*. Dva rodičovské chromozómy generujú dvoch potomkov. Začiatok chromozómu potomka 1 bude tvoriť časť z rodičovského chromozómu 1 v intervale (0, Crossover point-1), druhá časť chromozómu potomka 1 bude tvoriť časť z rodičovského chromozómu 2 v intervale (Crossover point, dĺžka chromozómu-1). Potomok 2 bude tvorený rovnako z častí rodičovských chromozómov 1 a 2 ale v opačnom poradí ako potomok 1. Na obrázku 3.3 je znázornené jednobodové kríženie.



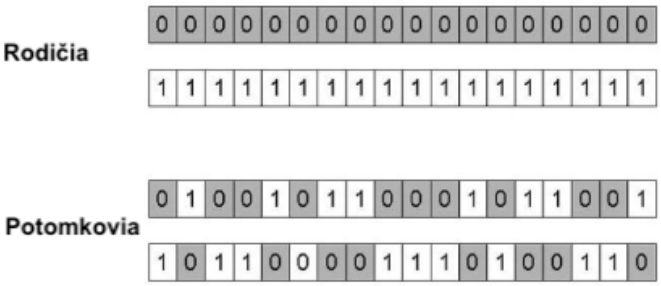
Obr. 3.3: Príklad jednobodového kríženia

Dvojbodové kríženie je postavené na rovnakom princípe ako jednobodové kríženie. Namiesto náhodného generovania jedného crossover bodu, dvojbodové kríženie vygeneruje dva crossover body. U obidvoch rodičovských chromozómov rozdelí chromozómy na 3 časti podľa crossover bodov. Nové dva potomky sú vytvorené z rodičovských chromozómov tak, že si navzájom rodičovské chromozómy vymenia strednú časť chromozómov vytvorených pomocou crossover bodov. Na obrázku 3.4 je znáznornené dvojbodové kríženie.



Obr. 3.4: Príklad dvojbodového kríženia

Uniformné kríženie náhodne vyberie gény z obidvoch rodičovských chromozómov a vymení ich. Tak vytvorí nových dvoch potomkov. Na obrázku 3.5 je znáznornené uniformné kríženie.



Obr. 3.5: Príklad uniformného kríženia

3.7.2 Mutácia

Mutácia (angl. mutation) je zmena hodnoty v náhodne vybranom géne chromozómu potomka. Zmyslom mutácie je zvýšiť rozmanitosť (diverzitu) potomkov a tým zabrániť predčasnej konvergencií genetického algoritmu (doslovná citácia z [16]). Mutácia môže byť jednobodová, dvojbodová alebo viacbodová. Napríklad, pri binárnom kódovaní a pri jednobodovej mutácií sa v chromozóme potomka zmení len jeden náhodne vybraný gén z nuly na jedna alebo naopak. Obdobne to funguje v ďalších variantách mutácie (viac viz [20]). Na obrázku 3.6 je príklad mutácie.

Obr. 3.6: Príklad trojbodovej mutácie v binárne kódovanom chromozóme

3.7.3 Inverzia

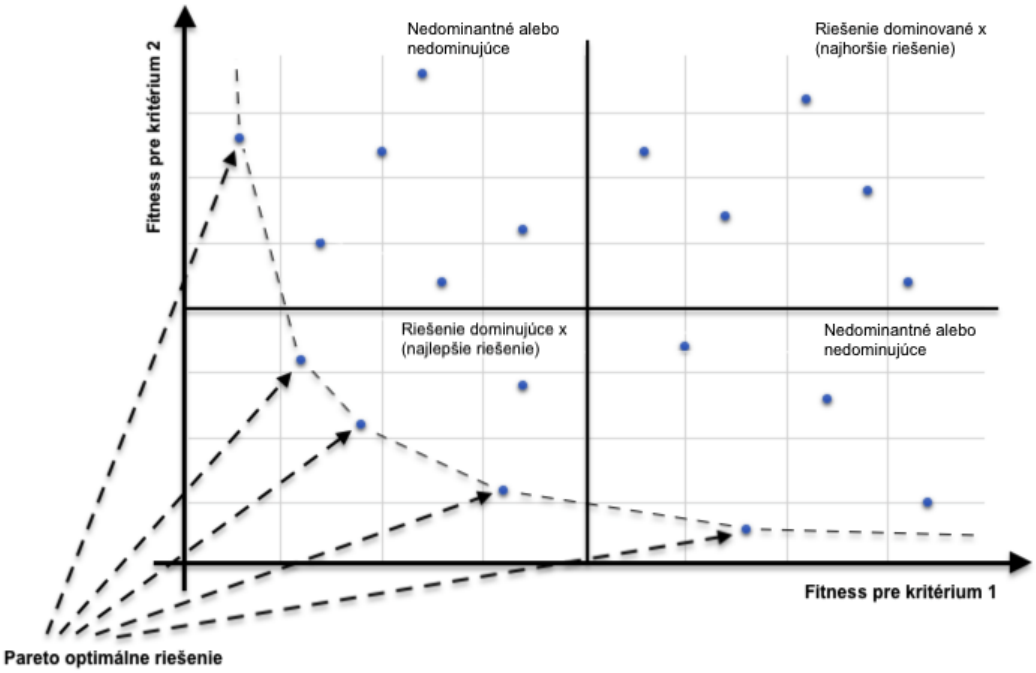
Inverzia (angl. inversion) je operácia nad jedným chromozómom, kedy náhodne vyberie dva body v rámci chromozómu. Medzi týmito dvomi bodami spraví výmenu hodnôt génov vnútri intervalu tak, že prvý gén sa vymení s posledným, druhý gén s predposledným, atď. Tento operátor je málo používaný a nie je obecne overený, či má v praxi kladný vplyv (viac viz [20]). Na obrázku 3.7 je príklad inverzie.

Obr. 3.7: Príklad inverzie v binárne kódovanom chromozóme

Kapitola 4

Multikriteriálna optimalizácia

Efektívne riešenie väčšiny problémov je založené na posúdení viacerých rôznych aspektov súčasne, pričom kritériá vychádzajúce z týchto aspektov sú veľakrát protichodné. Existujú problémy, ktoré zohľadňujú len jedno kritérium, na ktorých riešenie sa často navrhuje genetický algoritmus z kapitoly 3. V prípade, že problém musí zohľadňovať viac kritérií, využívajú sa techniky viackriteriálnej optimalizácie. V niektorých prípadoch je možné transformovať viackriteriálnu úlohu na problém optimalizácie funkcie s jedným kritériom. Tento prístup vykazuje často nízku efektivitu a typicky dokáže poskytnúť len jedno riešenie, čo má za následok nutnosť opakovaných výpočtov. Sú však aj problémy, ktoré nie je možné agregovať na jedno kritérium a u tých je potrebné použiť pokročilejšie techniky.



Obr. 4.1: Ukážka Pareto optimalizácie

Viackriteriálny prístup riešenia problému pomocou evolučných algoritmov je motivovaný hlavne snahou využiť populačnú povahu evolučných algoritmov, ktorá dovoľuje (za predpokladu zaistenia dobrej diverzity populácie) získať v jednom behu viac rôznych riešení, tzv. Pareto optimálne riešenia, ktoré nie sú dominované žiadnym iným riešením z pohľadu

daných kritérií. Cieľom multikriterálnych evolučných algoritmov je s rozumnou výpočtovou náročnosťou získať čo najviac Pareto optimálnych riešení.

Ako je znázornené na obrázku 4.1, riešenie môže byť najlepšie, najhoršie, ale aj neutrálne k iným riešeniam (ani dominujúce[1], ani dominantné[2]), z pohľadu dvoch kritérií. Najlepšie riešenie v tomto prípade je riešenie, ktoré nie je najhoršie v žiadnom z kritérií a je najmenej lepšie v jednom kritériu ako ostatné riešenia. Optimálne riešenie je riešenie, u ktorého nedominuje žiadne iné riešenie v prehľadávanom priestore. Množina týchto optimálnych riešení tvorí Pareto optimálne riešenie.

Algoritmy viackriteriálnej optimalizácie predstavujú širšiu triedu techník, ktoré sa stále vyvíjajú a vznikajú nové variácie pre riešenie rôznych problémov. V nasledujúcej kapitole sú vymenované niektoré najčastejšie používané prístupy riešenia viackriteriálnych úloh a bližšie definované algoritmy implementované v tejto práci.

4.1 Algoritmy multikriteriálnej optimalizácie

Multikriteriálne optimalizačné evolučné algoritmy (MOEA) sa v súčasnosti delia na MOEA prvej a druhej generácie.

MOEA prvej generácie sa delí na prístupy, ktoré využívajú Pareto dominanciu (napr. agregácia účelových funkcií, algoritmus VEGA, alebo lexikografické usporiadanie) a prístupy, ktoré nevyužívajú Pareto dominanciu (napr. algoritmus MOGA alebo NSGA). Algoritmy nevyužívajúce hodnotenie Pareto dominancie behom evolúcie vykazujú dobré vlastnosti pri riešení jednoduchších problémov (s nízkym počtom kritérií - veľmi často iba 2 kritériá). Nehodia sa však, pokiaľ požadujeme rozumné pokrytie Pareto množiny, tj. rôznorodé riešenia rovnomerne rozprestrená - kvalitná Pareto fronta. Z tohto pohľadu sú výhodnejšie pokročilejšie MOEA, ktoré priamo s Pareto dominanciou počítajú.

MOEA druhej generácie využívajú rôzne varianty Pareto dominancie k zaisteniu konvergencie algoritmu do oblasti Pareto množiny. V kombinácií s tým sú aplikované rôzne postupy na podporu diverzity populácie, špeciálne selekčné operátory apod., ktorých účelom je zvýšenie efektivity evolučných algoritmov (redukcie výpočetnej náročnosti), konvergencie a dosiahnutie čo najlepšieho pokrytia Pareto fronty. Najznámejšie algoritmy, ktoré patria do tejto triedy sú algoritmy micro-GA, SPEA, SPEA2, SPEA2+, NSGA-II, NSGA-III.

4.2 Agregácia účelových funkcií

Agregácia účelových funkcií rieši prevažne problémy viackriteriálnej optimalizácie (MOP), ktoré je možné redukovať na úlohu optimalizácie jediného kritéria a jeho výpočet je daný váhovým súčtom jednotlivých kritérií pôvodného MOP. Stanovením jednotlivých kritérií f_i pomocou vhodne stanovených váhových koeficientov w_i má fitness funkcia v prípade minimalizácie tvar:

$$fitness = min \sum_{i=1}^{k} w_i f_i(x)$$

[1] Dominujúce riešenie je riešenie, ktorého dominuje iné riešenie, tj. je v obidvoch kritériach horšie ako iné riešenie

[2] Dominantné riešenie je riešenie, ktorého nedominuje žiadne riešenie, tj. neexistuje riešenie, ktoré by malo obidve kritéria lepšie

4.3 Algoritmus NSGA-II

Klasické optimalizačné metódy navrhujú konverziu multikriteriálneho optimalizačného problému na jednokriteriálny zdôraznením jedného konkrétneho Pareto optimálneho riešenia. Ak sa má takýto prístup použiť na nájdenie viacerých Pareto optimálnych riešení, musí sa aplikovať viac krát. NSGA-II bol jedným z prvých evolučných algoritmov, ktorý je schopný nájsť viac Pareto optimálnych riešení v jednom behu algoritmu. V porovnaní s jeho nástupcom (NSGA-III) je stabilnejší a v niektorých problémoch dáva horšie výsledky ako algoritmus NSGA-II (viac viz [8]). Algortimus NSGA-II sa skladá z nasledujúcich kľúčových komponentov:

- **triedenie populácie podľa Pareto dominancie** (Fast Nondominated Sorting),
- **odhad hustoty populácie okolo jedincov** (Crowding distance),
- **operátor zhlukovania vzdialeností** (Crowded-Comparison Operator).

4.3.1 Rýchle nedominantné triedenie

Princíp spočíva v identifikácií úrovní vzájomne nedominovaných jedincov danej populácie a vytvorenie poradia týchto úrovní. Pre každé riešenie p z populácie o veľkosti N sa vypočíta dominantný počet n_p, ktorý definuje počet riešení, ktoré dominujú v riešení p a množinu riešení S_p, ktoré riešenie p dominujú. Zložitosť porovnaní je $O(MN^2)$. V prvej nedominantnej rade budú mať všetky riešenia dominantné číslo (rank) nula. Pre každé riešenie p, kde $n_p = 0$ bude navštívené každé riešenie q z množiny S_p. Každému riešeniu q sa zníži *rank* o jedna, až kým všetky riešenia q nebudú mať nulový rank. Tieto riešenia sú následne umiestnené do zoznamu Q, a tým sa vytvorí trietia rada. Tento proces pokračuje až kým sa nevytvoria všetky fronty (viac viz [8]). Na obrázku 4.6 (čerpaný z [8]) je znázornený algoritmus rýchleho nedominantného triedenia.

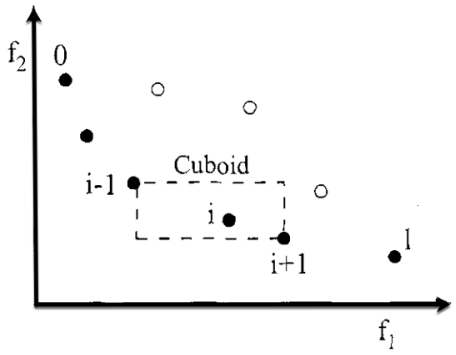
4.3.2 Odhad hustoty

Odhad hustoty populácie (Crowding Distance) je vykonaný pre každú úroveň nedominantných riešení a pre každého jedinca i v tejto úrovni (viac viz [8]). Na obrázku 4.2 je znázornený algoritmus prideľovania crowding distance.

$l = |\mathcal{I}|$ — Počet riešení v množine
for each i, set $\mathcal{I}[i]_{\text{distance}} = 0$ — Inicializácia vzdialenosti
for each objective m
  $\mathcal{I} = \text{sort}(\mathcal{I}, m)$ — Triedenie na základe každého kritéria
  $\mathcal{I}[1]_{\text{distance}} = \mathcal{I}[l]_{\text{distance}} = \infty$ — Vybrané sú vždy krajné body
  for $i = 2$ to $(l-1)$ — Pre všetky ostatné body
    $\mathcal{I}[i]_{\text{distance}} = \mathcal{I}[i]_{\text{distance}} + (\mathcal{I}[i+1].m - \mathcal{I}[i-1].m)/(f_m^{\max} - f_m^{\min})$

Obr. 4.2: Prideľovanie crowding distance

Na obrázku 4.3 (čerpaný z [8]) je znázornený výpočet vytesňovania vzdialeností. Vyplnené body sú riešenia rovnakých nedominantných front.



Obr. 4.3: Výpočet vytesňovania vzdialeností

4.3.3 Operátor zhlukovania vzdialeností

Relačný operátor zhlukovania vzdialeností ($\prec_n$) vyberá na základe dvoch atributov:

- **nedominantná kategória** (i_{rank}),
- **zhlukovanie vzdialeností** ($i_{distance}$).

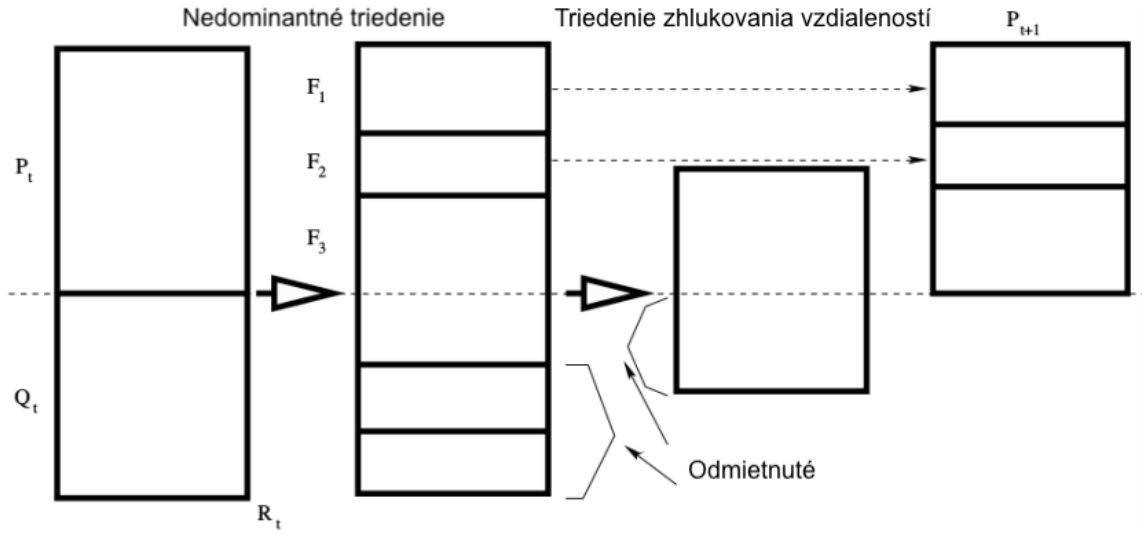
Čiastočné usporiadanie $\prec_n$ je definované ako (definované viz [8]):

$$i \prec_n j \quad if(i_{rank} < j_{rank})$$
$$alebo\ ((i_{rank} = j_{rank})\ a\ (i_{distance} > j_{distance}))$$

Výsledkom je, že z dvoch riešení s odlišnou kategóriou nedominantnosti operátor vyberie riešenie, ktoré je z nižšej (lepšej) kategórie. Ak budú v rovnakej kategórií, bude preferované riešenie s menším zhusťovacím prostredím.

4.3.4 Hlavný cyklus

Pre chod algoritmu NSGA-II sú potrebné všetky tri komponenty, ktoré sú definované v predchádzajúcich podsekciách. Ako prvý krok algoritmu sa vytvorí náhodná rodičovská populácia P_0 o veľkosti N. Použijú sa zvyčajné metódy ako kríženie, mutácia, binárna selekcia z rodičovskej populácie a vytvorí sa tak populácia potomkov Q_0 o veľkosti N. Je zformovaná nová populácia $R_t = P_t \cup Q_t$. Nová populácia Q_t má veľkosť $2N$. V ďalšom kroku algoritmus rozdelí populáciu R_t podľa nedominancie na množiny označované F_x, kde x označuje sadu riešení v jednej kategórií. Najlepšie riešenia sú v sade F_1. Ak je veľkosť F_1 menšia ako N, členovia F_1 sú definitívne vybraný do novej populácie R_{t+1}. Zvyšný chýbajúci členovia populácie R_{t+1} sú vybraný z nasledujúcich nedominantných kategórií F v poradí podľa kategórie do veľkosti populácie N. Následne vzniká ďalšia iterácia, doplnením populácie R_{t+1} o potomkov, ktorý boli vytvorený genetickými operáciami z nových rodičovských jedincov (viac viz [8]). Na obrázku 4.4 je znázornená schéma jednej iterácie algoritmu NSGA-II.



Obr. 4.4: Schéma iterácie NSGA-II

$R_t = P_t \cup Q_t$	Kombinuje rodičov a potomkov populácie
$\mathcal{F} = \texttt{fast-non-dominated-sort}(R_t)$	F = všetky nedominantné rady
$P_{t+1} = \emptyset$ and $i = 1$	
until $\lvert P_{t+1}\rvert + \lvert\mathcal{F}_i\rvert \leq N$	Dokým nie je populácie rodičov naplnená
$\quad\texttt{crowding-distance-assignment}(\mathcal{F}_i)$	Vypočítanie zhlukovania vzdialeností
$\quad P_{t+1} = P_{t+1} \cup \mathcal{F}_i$	Zahrnie i-tú nedominantnú radu do rodičovskej populácie
$\quad i = i + 1$	Kontroluje ďalšie rady pre započítanie
Sort$(\mathcal{F}_i, \prec_n)$	Pomocou operátora usporiada v zostupnom poradí
$P_{t+1} = P_{t+1} \cup \mathcal{F}_i[1 : (N - \lvert P_{t+1}\rvert)]$	Vyberie prvý prvok
$Q_{t+1} = \texttt{make-new-pop}(P_{t+1})$	Použije sa výber, kríženie a mutácia pre vytvorenie populácie Q
$t = t + 1$	Zvýši sa generačné počítadlo

Obr. 4.5: Algoritmus NSGA-II

Základné operácie a ich najhorší prípad zložitosti sú nasledujúce:

- nedominantné triedenie $O(M(2N)^2)$,
- priradenie zhlukovania vzdialeností $O(M(2N)log(2N))$,
- triedenie v $(\prec_n)$ je $O(2Nlog(2N))$.

Celková zložitosť algoritmu je $O(MN^2)$, čo je z hlavnej časti spôsobené nedominantnou triediacou časťou algoritmu (získané zložitosti viz [8]). Na obrázku 4.5 je znázornený algoritmus NSGA-II.

```
for each p ∈ P
   S_p = ∅
   n_p = 0
   for each q ∈ P
      if (p ≺ q) then                    Ak p dominuje q
         S_p = S_p ∪ {q}                 Pridá q do množiny riešení dominovaných p
      else if (q ≺ p) then
         n_p = n_p + 1                   Zväčšenie počtu nadvlády p o jedna
   if n_p = 0 then                       p patrí do prvej fronty
      p_rank = 1
      F_1 = F_1 ∪ {p}
i = 1                                    Inicializácia počítadla fronty
while F_i ≠ ∅
   Q = ∅                                 Používané na ukladanie členov prvej fronty
   for each p ∈ F_i
      for each q ∈ S_p
         n_q = n_q − 1
         if n_q = 0 then                 q patrí do nasledujúcej fronty
            q_rank = i + 1
            Q = Q ∪ {q}
   i = i + 1
   F_i = Q
```

Obr. 4.6: Rýchle nedominantné triedenie

Kapitola 5

Rozbor problému

V oblasti plánovania je veľa NP-ťažkých problémov, ktoré v reálnom čase nevieme vypočítať bežnými prístupmi. Cieľom tejto práce je vytvoriť algoritmus, ktorý bude efektívne plánovať denné trasy vozidlám, ktoré zvážajú a rozvážajú odpad vrámci jedného mesta. Konkrétny problém, ktorý práca rieši je zvoz a rozvoz veľkoobjemného alebo veľkokapacitného odpadu, tj. odpad, ktorý sa nezmestí do klasickej nádoby na zmesový komunálny odpad. Tieto veľkokapacitné kontajnery majú objem od 5,5 m^3 do 10 m^3 a sú určené na akýkoľvek odpad veľkých rozmerov a objemov. Bežne sa využívajú aj ako klasické kontajnery na priemyselný, komunálny, či menší stavebný odpad. Kontajner je potrebné po naplnení zaviezť na zberné stredisko aj s celou konštrukciou, na rozdiel od klasických nádob na zmesový komunálny odpad, kde nádoby stačí vyprázdniť do návesu vozidla.

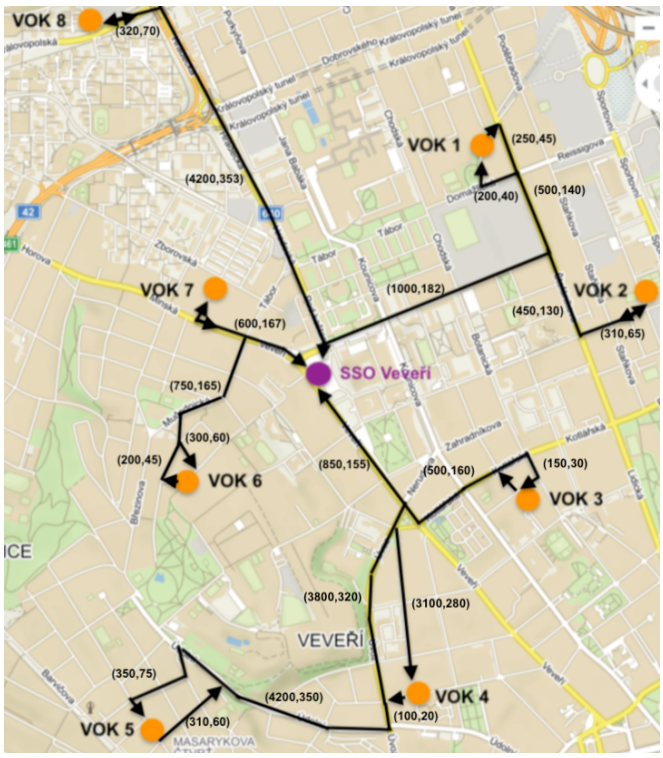
Na rozdiel od komunálneho odpadu, kontajnery na veľkoobjemný odpad majú premenlivé umiestnenie, ktoré sa môže meniť denne. Rovnako aj počet môže byť premenlivý podľa aktuálnej situácie v meste. V dobe veľkých úprav mesta môžu byť tieto kontajnery viac využívané ako počas kľudnejších týždňov. Počet kontajnerov na veľkoobjemný odpad je napr. v Brne okolo 500 a vozidiel približne 8 (záleží od dostupnosti). K dispozícií sú aj prívesy s hydraulickou rukou. V takom prípade jedno vozidlo je schopné odviezť dva veľkoobjemné kontajnery, no zvyčajne je s tým spojená veľká réžia, takže príves nie je bežne používaný.

Zvoz veľkoobjemných kontajnerov (ďalej VOK) nie je naplánovaný na mesiace dopredu ako zvoz komunálneho odpadu. V súčasnosti zvoz VOK-ov spoločnosti dopredu neplánujú, alebo plánujú jednoduchým spôsobom. Bežným prístupom je vyzdvihnutie naplneného kontajnera aktuálne dostupným vozidlom. Takýto prístup nie je efektívny z pohľadu nákladov za zvoz. V prípade väčšieho mesta s malým počtom vozidiel môže takéto plánovanie spôsobiť stratu zákazníkov. Dôvodom je preťaženie vozidiel.

Spoločnosti, ktoré plánujú zvoz veľkokapcitného odpadu postupujú tak, že dispečink počas dňa eviduje naplnené kontajnery a ich lokality a jednotlivým vozidlám naplánuje trasu na ďalší deň. Jednotlivé lokality naplnených VOK-ov pridelí najbližšiemu stredisku. Podľa počtu lokalít VOK-ov v jednotlivých zberných strediskách sa pomerovo pridelí počet vozidiel k týmto strediskám. V takomto prípade každé vozidlo zváža kontajnery len do jedného strediska. Takéto plánovanie sa opakuje každý deň. Tento prístup je efektívnejší a šetrnejší z pohľadu celkových nákladov za zvoz VOK-ov. No v prípade, že by sme chceli brať do úvahy viac faktorov ako je napr. počet najazdených kilometrov, alebo čas jednotlivých vozidiel, prípadne presúvať vozidlo medzi zbernými strediskami potom je potrebné využiť pokročilejšie prístupy, pretože z deterministickej úlohy sa stala NP-ťažká úloha.

Formálne, keď tento problém znázorníme graficky, dostaneme orientovaný graf (medzi dvoma uzlami môžu byť rôzne cesty z pohľadu jednosmeriek a obmedzení objemných vo-

zidiel) s ohodnotenými hranami (hodnota hrany je dĺžka cesty v kilometroch) a uzlami reprezentujúcimi VOK-y. Hrany tvoria najkratšie cesty medzi jednotlivými VOK-mi. Medzi dvoma uzlami môžu byť dve hrany s rozdielnou hodnotou (najkratšia cesta z uzla 1 do uzla 2 môže viesť rozdielne ako z uzla 2 do uzla 1). Do grafu sú pridané uzly reprezentujúce zberné dvory, kde vozidlá zvážajú a rozvážajú VOK-y. Na obrázku je znázornený orientovaný graf problému VOK v okolí zberného stredika SSO Veveří v Brne. Fialový uzol reprezentuje zberné stredisko a oranžový uzol odpovedá lokalitám s naplnenými veľkoobjemnými kontajnermi pripravenými na zvoz. Každá hrana grafu je ohodnotená dvojicou (X, Y), kde X je dĺžka trasy v metroch a Y je priemerný čas smetiarskeho vozidla v sekundách, za ktorý trasu prejde.



Obr. 5.1: Ukážka grafu problému VOK

Orientovaný graf, ktorý je definovaný v predchádzajúcom odseku, zjednodušíme na neorientovaný. Graf obsahuje uzly troch stredísk. Namiesto ôsmich vozidiel, berieme do úvahy len jedno. Tento zjednodušený model problému je ekvivalentný k problému TSP (obecný popis TSP v kapitole 2), ktorý je definovaný nad neúplným grafom. Takto obmedzený problém TSP môžme prirovnať k TSP nad grafom s maximálnym stupňom uzlov 3, ktorý je stále exponenciálne náročný (problém TSP, definovaný nad grafom s maximálnym stupňom uzlov 3 je vyriešený v čase $O(1.26^n)$ (dôkaz viz [9]). Pridaním väčšieho počtu vozidiel, stredísk, kritérií výpočtu a orientácie môžme predpokladať, že problém zvážania a rozvážania VOK-ov je NP-ťažký, keďže je ťažší ako obmedzený problém TSP, ktorý je exponenciálne zložitý. Pridávaním viacerých vozidiel do problému, sa problém približuje k problému VRP, ktorý je popísaný v kapitole 2. Princíp riešenia problému VRP bude využitý pri návrhu riešenia problému VOK-ov v nasledujúcej kapitole.

Kapitola 6

Návrh riešenia problému

Problém plánovania zvozu veľkoobjemného odpadu je možné zjednodušiť na problém VRP. Problém VRP pri veľkom počte uzlov grafu nie je možné v rozumnom čase riešiť exaktnými metódami. Najpoužívanejším algoritmom na riešenie problému VRP je genetický algoritmus a z toho dôvodu je navrhnutý na riešenie problému VOK. Keďže vieme, že problém má dve kritéria (celkový čas zvozu odpadu a najazdené kilometre), bude treba použiť algoritmy multikriteriálnej optimalizácie.

Prvým algoritmom na vyriešenie problému VOK bude genetický algoritmus s agregovanými účelovými funkciami, ktorý je definovaný v kapitole 4. Obidvom kritériám sa navrhne koeficient (váha) a pre výpočet výslednej fitness hodnoty sa vykoná váhový súčet jednotlivých kritérií. Váhou sa výsledok kritéria prevedie na cenu. Kritériu celkového času zvážania bude pridelená váha, ktorá bude reprezentovať cenu za jednotku času, v ktorej je výsledok tohto kritéria. Kritériu celkových najazdených kilometrov bude pridelená váha, ktorá bude reprezentovať cenu za jednotku vzdialenosti. Po váhovom súčte týchto kritérií dostaneme výslednú cenu nákladov celého riešenia.

Pre porovnanie výsledkov genetického algoritmu bude implementovaná jednoduchá metóda plánovania, ktorú využívajú niektoré spoločnosti zvozu odpadu. Bude obsahovať jednu úpravu oproti metóde, ktorá bola definovaná v kapitole 4. Namiesto rozdelenia vozidiel jednotlivým zberným strediskám pomerovo na základe počtu VOK-ov, bude rozdelenie pomerovo na základe celkových vzdialeností všetkých trás pre jednotlivé zberné strediská a im priradené lokality VOK-ov. Tento algoritmus je deterministický a bude zastávať metódu súčasného plánovania zvozu odpadu.

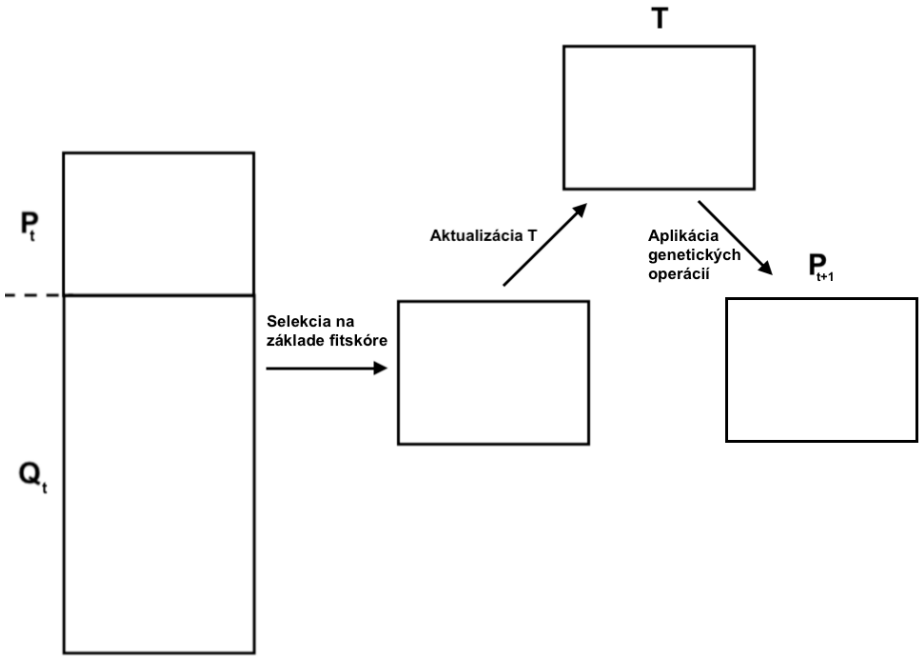
Druhým multikriteriálnym algoritmom pre riešenie problému VOK je algoritmus NSGA-II, ktorý je popísaný v kapitole 4. NSGA-II na rozdiel od genetického algoritmu s agregovanými účelovými funkciami prehľadáva a vyberá riešenia zo širšieho spektra. Nevyberá riešenia len jedným smerom. Vyberá riešenia, ktoré sú vzájomne najlepšie medzi obidvoma kritériami. NSGA-II nemá definované váhy a preto dáva viac možností používateľovi, ktorý si môže vybrať, či preferuje riešenie s menšou dobou trvania alebo s menším počtom najazdených kilometrov alebo ich kombináciu. Táto metóda je vhodná v prípade, keď nemáme k dispozícií cenu nákladov za celkový čas alebo celkovú vzdialenosť riešenia.

Všetky algoritmy plánovania sú navrhnuté tak, že pred vyzdvihnutím naplneného kontajnera sa dovezie prázdny kontajner zo zberného strediska. V takom prípade, vozidlá vždy pri výklade plného kontajnera v zbernom stredisku, naložia prázdny kontajner a pokračujú trasou k ďalšiemu naplnenému kontajneru, ktorý je v poradí daného vozidla. V nasledujúcich sekciách budú popísané návrhy genetických algoritmov. Definovaný bude zápis a sémantika jednotlivých génov chromozómu, ako aj genetické operácie nad nimi.

6.1 Návrh genetických algoritmov

Návrh genetických algoritmov problému plánovania VOK-ov, bude vychádzať z obecnej definície genetického algoritmu z kapitoly 3.

Genetický algoritmus s agregovanými účelovými funkciami na začiatku náhodne vygeneruje populáciu veľkosti N. Pridá ďalšiu náhodne vygenerovanú populáciu veľkosti M, takže výsledná populácia bude mať veľkosť $N+M$. Aplikuje genetické operácie a vyberie z tejto populácie M najlepších jedincov podľa výsledku z agregovaných účelových funkcií (fitskóre). Uloží si ich do populácie veľkosti M, ktorá reprezentuje najlepších jedincov. Nová generácia je tvorená z populácie náhodne vygenerovaných jedincov veľkosti N spojená s populáciou veľkosti M, ktorá je vytvorená potomkami z populácie najlepších jedincov prostredníctvom genetických operácií. Tento cyklus sa opakuje, až kým najlepší jedinci nekonvergujú k jednému riešeniu a odchylka medzi jednotlivými najlepšími riešeniami nebude príliš malá. Na obrázku 6.1 je znázornená schéma tohto algoritmu. Populácia T je výsledná najlepšia populácia, ktorá sa každou generáciou aktualizuje, ak je nájdený lepší jedinec. Populácia P_t je každú generáciu tvorená potomkami z populácie T, po genetických operáciach.

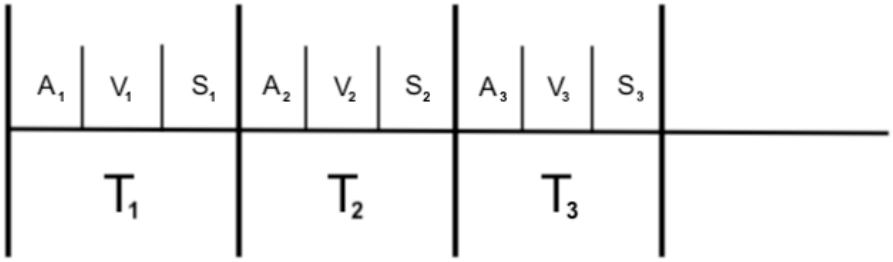


Obr. 6.1: Schéma GA s agregovanými účelovými funkciami

Algoritmus NSGA-II bude využívať rovnaké genetické operácie a rovnakú reprezentáciu chromozómu ako GA s agregovanými účelovými funkciami. Bude využívať všetky komponenty obecného algoritmu NSGA-II definovaného v kapitole 4. Pre tento algoritmus platí rovnaká schéma, ako je znázornená v podkapitole 4.3. Výsledkom tohto algoritmu bude Pareto fronta obsahujúca Pareto optimálne riešenia z prehľadávaných riešení. Počet generácií bude stanovený používateľom. Pareto fronta bude znázornená na grafe podobne ako graf znázorňujúci Pareto frontu v kapitole 4.

6.1.1 Reprezentácia chromozómu

Reprezentácia chromozómu v obidvoch genetických algoritmoch je rovnaká. Chromozóm sa skladá z trojíc T_x veľkosti N, kde N je počet VOK-ov a x je číslo poradia trojice v chromozóme. Každá trojica obsahuje tri gény. Prvý gén určuje číslo smetiarskeho auta A_x v intervale <1,*počet smetiarských vozidiel*>. Druhý gén odpovedá číslu veľkoobjemného kontajnera V_x v intervale <1, *počet veľkoobjemných kontajnerov*>. Tretí gén je číslo zberného strediska S_x v intervale <1, *počet zberných stredísk*>. Poradie trojíc v chromozóme určuje poradie odbavenia jednotlivých kontajnerov. Na obrázku 6.2 je znázornené zloženie chromozómu.



Obr. 6.2: Reprezentácia chromozómu

Pre znázornenie použijeme príklad jednoduchého chromozómu (1, 3, 2, 2, 1, 1, 1, 2, 3). Rozdelíme chromozóm na trojice, kde prvá trojica T_1 je (1, 3, 2). Sémanticky táto trojica znamená, že vozidlo s číslom 1 odbaví kontajner číslo 3 do zberného strediska číslo 2. Braním do úvahy poradie trojíc v chromozóme, vozidlo číslo 1 odbaví najprv kontajner číslo 3 do strediska s číslom 2 a následne potom odbaví kontajner číslo 2 do strediska s číslom 3.

6.1.2 Účelové funkcie

Algoritmy navrhnuté na riešenie problému plánovania VOK-ov budú využívať rovnaké účelové funkcie (fitness funkcie). Rozdiel bude v manipulácií s výsledkami z týchto účelových funkcií. Účelové funkcie v tomto prípade sú dve. Jedna bude počítať celkovú dobu zvozu a rozvozu kontajnerov, od výjazdu prvého vozidla, do príchodu posledného vozidla do zberného strediska. Výsledok je v sekundách. Druhá účelová funkcia počíta celkový počet najazdených metrov všetkých vozidiel.

Pre znázornenie výpočtu účelových funkcií zoberieme jednoduchý príklad chromozómu definovaný v predchádzajúcej sekcií. Chromozóm (1, 3, 2, 2, 1, 1, 1, 2, 3) rozdelíme na trojice. Postup účelových funkcií je zhodný, s rozdielom v tom, že pri hľadaní najkratších trás v jednej funkcií hľadáme najkratšiu dobu a v druhej najkratšiu vzdialenosť. Vypočíta sa najkratšia trasa pre smetiarske vozidlo 1 tak, že najprv nájde najkratšiu cestu z počiatočnéhu uzla do uzla, v ktorom je VOK 3. Následne najkratšiu cestu do zberného dvora 2. Zo zberného dvora 2 najkratšiu trasu k uzlu VOK 2 a odtiaľ najkratšiu cestu do zberného dvora 3. Pre vozidlo 2 nájde najkratšiu cestu z počiatočnéhu uzla do uzla, kde sa nachádza VOK 1 a odtiaľ na zberný dvor 1. Funkcia vzdialenosti spočíta všetky vzdialenosti všetkých vozidiel. Funkcia času spočíta čas najdlhšie trvájúcemu vozidlu.

Genetický algoritmus s agregovanými účelovými funkciami pridá k výsledkom účelových funkcií odpovedajúce koeficienty (váhy). Vykoná váhový súčet, ktorého výsledok je fitskóre

daného riešenia. Algoritmus NSGA-II pracuje s obidvoma výsledkami účelových funkcií. NSGA-II v tomto prípade rieši optimalizáciu s dvoma kritériami a riešenia prehľadáva v 2D priestore.

6.1.3 Genetické operácie

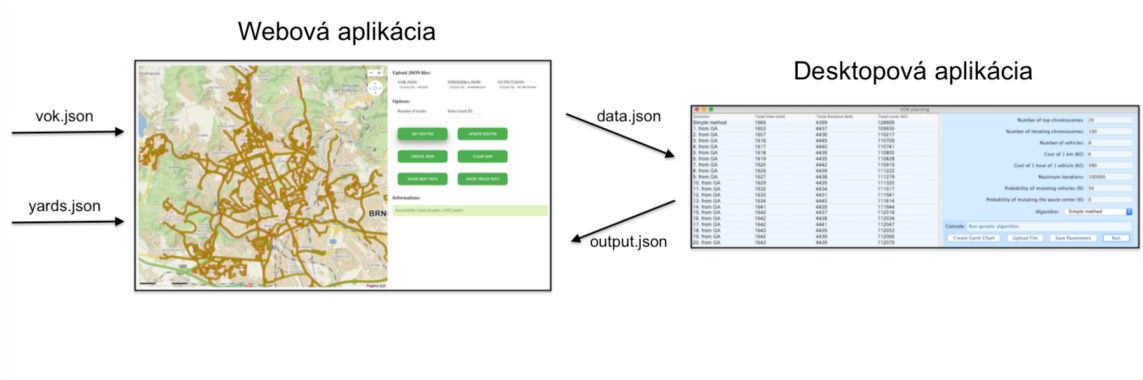
Algoritmy využívajú len jednu genetickú operáciu, ktorou je viacbodová mutácia. Náhodne sa vyberie n trojíc, kde $n <$ počet VOK-ov. Na vybrané trojice je aplikovaná mutácia tak, že s určenou pravdepodobnosťou je mutovaný gén vozidla a gén strediska. V poslednom rade je mutované aj poradie trojíc náhodným prehodením trojíc v chromozóme. Pravdepodobnosť mutácie génov vozidla a strediska si určí používateľ.

Kapitola 7

Implementácia

Implementácia výsledného aplikačného systému, ktorý rieši problém plánovania veľkoobjemných kontajnerov vychádza z požiadaviek na používanie. V prvej fáze plánovania sú na vstupe systému dáta všetkých získaných lokalít aktuálne naplnených kontajnerov, ktoré dispečink evidoval počas stanovenej doby, ako aj informácie o zberných strediskách. V ďalšom kroku je potrebné získať hodnoty všetkých hrán grafu problému (najkratšie trasy medzi kontajnermi a strediskami). Zvolené bolo nelicencované a nijako neobmedzené webové rozhranie Mapy.cz API. Z toho dôvodu bola vytvorená webová aplikácia v interpretovanom jazyku Javascript, ktorý umožňoval asynchrónne volania služby, zobrazenie mapy a jej interakciu.

Ďalšia fáza pozostáva z možnosti voľby algoritmu na nájdenie riešenia. Implementovaná bola metóda jednoduchého plánovania (odpovedá súčasnému plánovaniu), genetický algoritmus s agregovanými účelovými funkciami a algoritmus NSGA-II. Efektivita a rýchlosť výpočtu genetických algoritmov sa odvýja aj od architektúry aplikácie. Preto bola zvolená desktopová aplikácia implementovaná v programovacom jazyku Java, bežiaca na virtuálnom stroji, ale s podporou JIT (Just in Time) prekladu. Pre účely výpočtu a behu genetických algoritmov je efektívnejšia v porovnaní s webovou aplikáciou implementovanou v interpretovanom jazyku.



Obr. 7.1: Schéma aplikačného systému

Na obrázku 7.1 je znázornená schéma aplikačného systému. Tvoria ju dve komponenty (webová a desktopová aplikácia). Na vstupe webovej aplikácie sú dva súbory v JSON formáte, ktoré používateľ obdrží z dispečinku. Súbor `vok.json` obsahuje informácie o aktuálne

naplnených kontajneroch (GPS súradnice a názvy adries). Súbor `yards.json` obsahuje informácie o aktuálne dostupných zberných dvoroch (GPS súradnice a názvy adries). Následne webová aplikácia získa najkratšie trasy medzi kontajnermi a strediskami na základe vstupných dát. Vygeneruje súbor data.json, v ktorom sú informácie o zberných strediskách (GPS súradnice a názvy adries) a informácie o kontajneroch (GPS súradnice a názvy adries) rozšírené o informácie o trasách k jednotlivým strediskám. V prípade, že súbor `yards.json` obsahuje 3 prvky (strediská), informácie o kontajneroch v súbore `data.json` budú vyzerať ako na ukážke 7.1. Atribút *time1* definuje čas, za ktorý prejde smetiarske vozidlo zo strediska 1 k lokalite kontajnera v sekúndách. *dist1* je vzdialenosť tejto cesty v metroch. Atribút *preferredYard* definuje najbližšie stredisko ku kontajneru (na ukážke 7.1 je to stredisko číslo 1).

```
{
   "lat": "49.211961258705",
   "lng": "16.589790149195",
   "address": "Pod Kastany 26",
   "time1": 39,
   "dist1": 327,
   "time2": 422,
   "dist2": 4350,
   "time3": 497,
   "dist3": 4213,
   "preferredYard": 1
}
```

Výpis 7.1: Ukážka atribútov jedného kontajnera v súbore `data.json`

Desktopová aplikácia po obdržaní vstupných informácií o kontajneroch a dvoroch nájde riešenie podľa zvolenej metódy plánovania. Zobrazí Ganttov diagram s naplánovanými trasami vozidiel a vygeneruje súbor `solution.json` s výsledným riešením. Tento súbor obsahuje pole všetkých ciest všetkých vozidiel. Na ukážke 7.2 sú znázornené atribúty naplánovanej cesty kontajneru z ukážky 7.1. Atribút *truck* vyznačuje id vozidla, ktoré tento kontajner odbavuje.

```
{
   "latVok": "49.211961258705",
   "lngVok": "16.589790149195",
   "nameVok": "Pod Kastany 26",
   "latYard": "49.209458",
   "lngYard": "16.590127",
   "nameYard": "SSO Veveri",
   "truck": 6,
   "preferredYard": 1
}
```

Výpis 7.2: Ukážka atribútov jednej cesty v súbore `output.json`

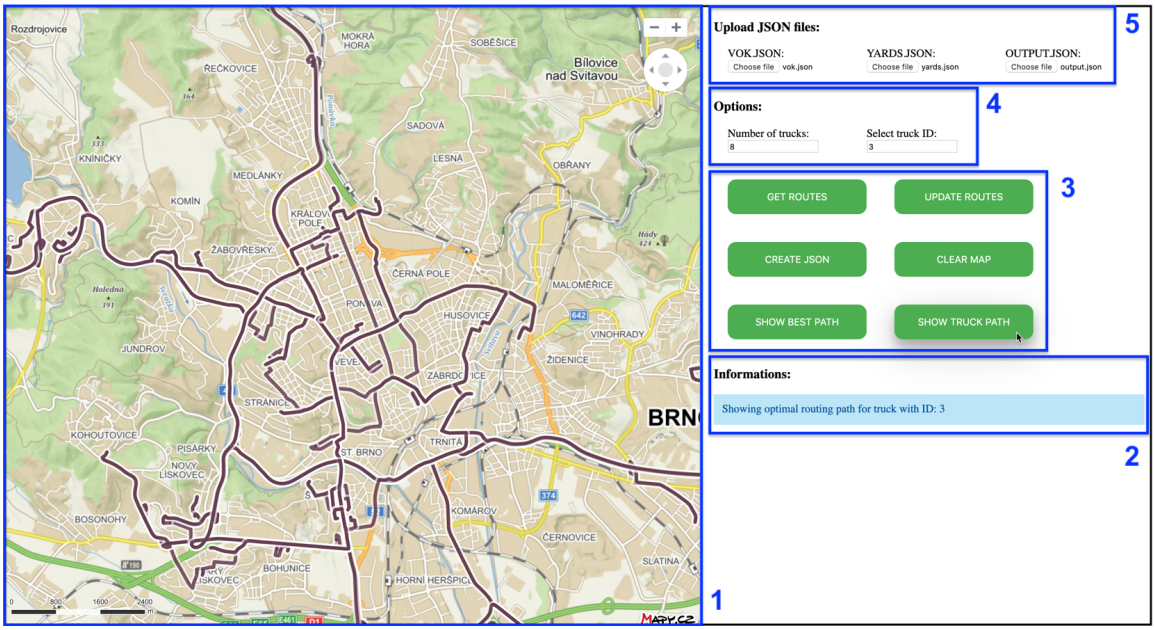
Webová aplikácia obdrží súbor `output.json` a zobrazí výsledné plánovanie na interaktívnej mape. V nasledujúcich sekciách sú detailne opísané jednotlivé aplikácie aplikačného systému.

7.1 Webová aplikácia

Webová aplikácia zastáva funkciu vstup-výstupnej komponenty celého aplikačného systému. Importuje vstupné dáta a zároveň má funkciu zobrazovania výsledného riešenia. Úlohou aplikácie je spracovať vstupné informácie o kontajneroch a zberných dvoroch, získať najkratšie vzdialenosti medzi kontajnermi a dvormi, generovať súbor s nájdenými cestami a zobrazovať výsledné riešenie.

Na získanie najkratších vzdialeností medzi kontajnermi a zbernými strediskami bolo využité webové rozhranie Mapy API prevádzkované spoločnosťou Seznam.cz. Vo výbere bola aj služba Google Maps Platform od spoločnosti Google, ktorá však mala obmedzenia, napr. v počte volaní za deň. Vybrané rozhranie Mapy API je kompletne zadarmo a je možné ho využiť aj pre komerčné účely. Pre väčšinu činností nie je používanie nijako obmedzené. Rozhranie je využité pri získavaní najkratších vzdialeností a na zobrazenie a manipuláciu interaktívnej mapy od služby Mapy.cz.

Grafické uživateľské rozhranie (ďalej len GUI) bolo navrhnuté ako jednoduché, prehľadné a užívateľsky prívetivé. Na obrázku 7.2 je znázornená webová aplikácia. Na obrázku sú rozdelené časti modrými čiarami pre lepší popis jednotlivých komponentov GUI aplikácie. Interaktívna mapa vyznačená číslom 1 má úlohu zobrazovať výsledky hľadaní najkratších trás v prvých fázach používania systému a zobrazovať výsledné naplánované trasy riešenia generovaného desktopovou aplikáciou. Oblasť vyznačená pod číslom 5 slúži k importu jednotlivých json súborov (`vok.json`, `yards.json` a `output.json`). Oblasť číslom 4 udáva parametre výsledného zobrazenia riešenia. Prvé políčko definuje počet vozidiel s ktorými pracoval optimalizačný algoritmus v desktopovej aplikácií. Políčko *Select truck ID* slúži k vybratiu konkrétneho vozidla, ktorého naplánované trasy budú zobrazené na mape v časti 1.

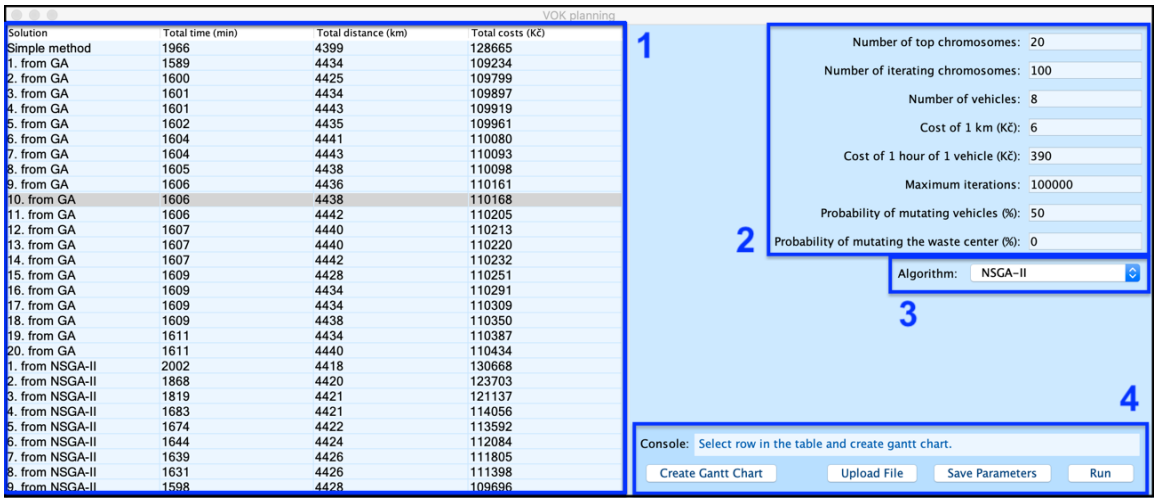


Obr. 7.2: Ukážka webovej aplikácie

Časť 3 obsahuje tlačítka, ktoré získajú najkratšie trasy (tlačítko *GET ROUTES*), aktualizujú informácie o kontajneroch pridaním atribútu *preferredYard* (tlačítko *UPDATE ROUTES*), generujú JSON súbor `data.json` spomínaný v predchádzajúcich odsekoch (tlačítko *CREATE JSON*), vyčistia mapu od zobrazených ciest (tlačítko *CLEAR MAP*), zobrazia naplánované cesty všetkých vozidiel oddelených farbou zo súboru `output.json` (tlačítko *SHOW BEST PATH*) a zobrazia naplánované cesty jedného vozidla definovaného v časti 4 obrázka 7.2. Panel v časti 2 zastáva informatívnu rolu, ktorou oznamuje jednotlivé kroky, ktoré webová aplikácia aktuálne vykonáva.

7.2 Desktopová aplikácia

Desktopová aplikácia je predovšetkým výpočtová jednotka aplikačného systému. Implementuje tri metódy plánovania veľkoobjemných kontajnerov, ktorých výsledky je možné navzájom porovnať. Aplikácia je implementovaná v kompilovanom objektovo-orientovanom jazyku Java. Vlastnosti tohto jazyka boli využité pri návrhu zápisu chromozómu genetických algoritmov, kedy sa využil objektový prístup. Na obrázku 7.3 je ukážka desktopovej aplikácie rozdelená na 4 časti. Prvá časť predstavuje tabuľku výsledkov jednotlivých algoritmov. Každý riadok reprezentuje jedno riešenie. Riadok obsahuje celkový čas zvozu, celkovú vzdialenosť a hodnotu nákladov zvozu tohto riešenia v českých korunách. Druhá časť parametrizuje algoritmy a pridáva tak používateľovi rôzne možnosti behu algoritmov. Kombobox z časti 3 určuje optimalizačnú metódu. Časť 4 obsahuje informatívnu konzolu, ktorá informuje používateľa o aktuálnom stave aplikácie. Tlačítka v časti 4 umožňujú uloženie paremetrov z časti 2, spustenie vybraného algoritmu, vloženie súboru `data.json` a tlačítko na vytvorenie Ganttovho diagramu riešenia vybraného v tabuľke z časti 1.



Obr. 7.3: Ukážka desktopovej aplikácie

V nasledujúcich podsekciách bude opísaná implementácia jednotlivých metód plánovania veľkoobjemných kontajnerov.

7.2.1 Jednoduchá metóda

Jednoduchá metóda zastáva metódu súčasného plánovania zvozu kontajnerov. Jej princíp a implementácia je triviálna v porovnaní s ostatnými metódami. Na základe atribútu *preferredYard* kontajnera zo súboru `data.json` (znázornené na ukážke 7.1) boli jednotlivým zberným dvorom pridelené kontajnery podľa preferencie. Následne boli sčítané vzdialenosti jednotlivých kontajnerov podľa atribútu *timeX* kde *X* je číslo atribútu *preferredYard* daného kontajnera. V druhom kroku algoritmus priradí vozidlá jednotlivým strediskám pomerovo podľa súčtov vzdialeností kontajnerov priradených k týmto strediskám.

Názorná ukážka tohto algoritmu s 485 naplnenými kontajnermi, 8 vozidlami a 3 zbernými strediskami najprv priradí kontajnery k stediskám a spočíta všetky vzdialenosti. Zberné stredisko 1 bude mať súčet vzdialeností v tomto prípade 2 737 580 metrov. Stredisko 2 má súčet 921 134 metrov a stredisko 3 má 727 730 metrov. Pomerovo algoritmus rozdelí vozidlá jednotlivým strediskám. Stredisko 1 obsluhujú 4 vozidlá, stredisko 2 obsluhujú 2 vozidlá a stredisko 3 obsluhujú 2 vozidlá. Jednotlivým vozidlám sa priradia kontajnery náhodne tak, že vozidlám strediska 1 bude priradených 66, 66, 66 a 64 veľkoobjemných kontajnerov. Vozidlám strediska 2 bude priradených 62 a 60 kontajnerov a stredisko 3 bude mať priradených 51 a 50 kontajnerov. Výsledné naplánované riešenie sa pridá do riadka *Simple method* tabuľky aplikácie v časti 1 na obrázku 7.3. Rovnako ako ostatné riešenia v tejto tabuľke, aj z riešenia jednoduchej metódy je možné vytvoriť Ganttov diagram, ako aj zobrazenie vo webovej aplikácií.

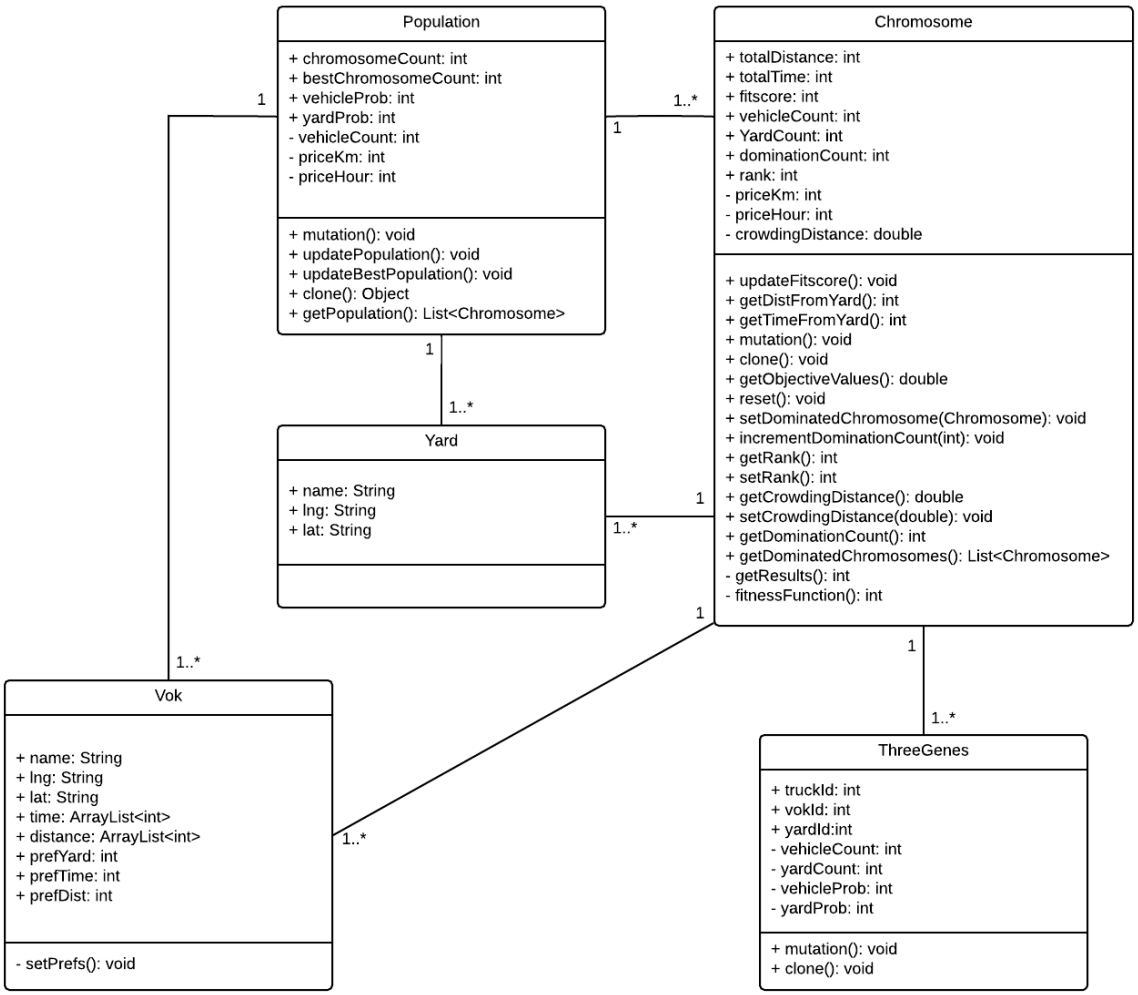
7.2.2 Genetický algoritmus s agregovanými účelovými funkciami

Implementácia genetického algoritmus s agregovanými účelovými funkciami vychádza z obecného návrhu genetického algoritmu z kapitoly 3. Úprava hodnoty fitskóre vychádza z definície obecného použitia princípu agregovaných účelových funkcií z kapitoly 4. Napojenie týchto obecných postupov vychádza z návrhu riešenia problému plánovania VOK-ov prostredníctvom algoritmu agregovaných účelových funkcií z kapitoly 6. Návrh definuje reprezentáciu chromozómu, ktorý je zložený z N po sebe nasledujúcich trojíc, kde každá trojica reprezentuje konkrétny veľkoobjemný kontajner a číslo N je počet veľkoobjemných kontajnerov pripravených na zvoz.

Objektová vlastnosť jazyka Java, v ktorej bol implementovaný tento genetický algoritmus, umožnila každý genetický prvok definovať vo vlastnej triede. Definované boli triedy pre populáciu, jedinca, trojicu, vok a stredisko. Genetické operácie ako napr. mutácia patrili k metódam týchto tried. Trieda Population (zodpovedá populácií) obsahuje atribúty parametrov genetického algoritmu, ktoré sú zvolené v časti 2 z ukážky 7.3. Atribútom tejto triedy je aj kolekcia objektov triedy Chromosome (zodpovedá jedincovi populácie) veľkosti M, kde M je počet zvolených chromozómov algoritmu, definovaný v parametri. Trieda Chromosome obsahuje kolekciu objektov triedy ThreeGenes (zodpovedá trojici génov) veľkosti N, kde N je počet veľkoobjemných kontajnerov získaný zo vstupného súbora aplikácie `data.json`.

Trieda ThreeGenes, je zložená z prvkov definovaných v návrhu genetického algoritmu v kapitole 6. Obsahuje číslo vozidla, číslo kontajnera a číslo strediska. Tieto prvky patria medzi gény chromozómu. Jednou z metód triedy ThreeGenes je mutácia, ktorá na základe pravdepodobnosti určí súhlas k mutácií tejto trojice. Mutované sú len atribúty čísla

vozidla a čísla strediska. Pravdepodobnosti mutácie týchto atribútov sú zvolené v parametroch desktopovej aplikácie v časti 2 z ukážky 7.3. Objekty triedy Vok a Yard reprezentujú konkrétne veľkoobjemné kontajnery a zberné dvory. Atribúty tvoria informácie o lokalite, adrese, vzdialenostiach a časoch k strediskám rovnako ako v súbore `data.json`. Na obrázku 7.4 je znázornený diagram tried genetického algoritmu.



Obr. 7.4: Diagram tried genetického algoritmu

Využitím objektového prístupu bolo zvolené špeciálne kódovanie (z definíc kódovania v kapitole 3). Genetický kód je tvorený prvkami objektov triedy ThreeGenes. Mutácia chromozómu prebieha v troch fázach. Najprv náhodne vyberie trojice chromozómu, na ktoré aplikuje metódu mutácie (náhodne zvolí objekty triedy ThreeGenes, ktoré zavolajú metódu mutácie). V druhej fáze po aplikácií metódy mutácie, objekty tried ThreeGenes mutujú hodnotu atribútu čísla vozidla a čísla strediska, ako je opísané v predchádzajúcom odseku. Genetická operácia kríženia sa neaplikuje v tejto práci z dôvodu nízkej efektivity a spomalenia algoritmu.

Aplikácia pri behu genetického algoritmu každou generáciou ukladá najlepších jedincov do kolekcie triedy Chromosome veľkosti S, kde S je počet najlepších jedincov definovaných v parametroch aplikácie, ktorí sa zobrazujú vo výslednej tabuľke v časti 1 ukážky 7.3.

Selekčná metóda výberu nových jedincov do novej generácie je jednoduchá. Vyberú sa jedinci z populácie najlepších jedincov veľkosti S, na ktorých sa aplikuje mutácia. K tejto populácií sa pridá populácia náhodne vygenerovaných jedincov veľkosti M, kde M je počet zvolených chromozómov algoritmu, definovaný v parametri.

Agregovaná účelová funkcia pracuje s váhami, ktoré sú zvolené v aplikacií v časti 2 ukážky 7.3. Tieto váhy sú priradené k výpočtu celkového času riešenia a celkovej vzdialenosti riešenia a následne je vypočítaný výsledok váhového súčtu. Celková vzdialenosť riešenia sa vypočíta na základe súčtu všetkých vzdialeností všetkých vozidiel smerom zo stredísk k veľkoobjemných kontajnerom a opačne. Celkový čas je vypočítaný pre každé vozidlo zvlášť sčítaním času zo strediska k jednotlivým kontajnerom a opačne. K týmto časom sa pričíta čas strávený výkladom a nákladom kontajnera, ktorý je konštantnej hodnoty 10 minút. Zo všetkých časov vozidiel sa vyberie najdlhší čas, ktorý sa považuje za výsledný čas riešenia.

Riešenia tohto algoritmu sú zobrazované v tabuľke aplikácie aktualizované každú sekundu. Vizualizácia riešenia z tohto algoritmu, ako aj ostatných algoritmov, je vo forme Ganttovho diagramu, ktorý je opísaný v sekcií 7.3.

7.2.3 Algoritmus NSGA-II

Implementácia základných metód algoritmu NSGA-II bola využitá zo zdroja [1]. Zdroj odkazuje na originálny kód algoritmu NSGA-II implementovaného v jazyku C od autorov K. Deb, A. Pratap, S. Agarwal a T. Meyarivan. Tento algoritmus bol upravený tak, aby reprezentácie jednotlivých genetických prvkov boli rovnaké, ako v genetickom algoritme s agregovanými účelovými funkciami.

Genetické kódovanie, implementácia chromozómu a genetické operácie sú pre algoritmus NSGA-II rovnaké, ako v genetickom algoritme s agregovanými účelovými funkciami. Trieda Jedinec bola rozšírená o metódy a atribúty algoritmu NSGA-II. Definície atribúty *crowdingDistance, dominationCount, rank* a kolekcia objektov triedy Jedinec *dominatedChromosome* sú definované v kapitole 4 v sekcií NSGA-II.

```
Population parent = new Population();
Population child = new Population();
Population combinedPopulation;

for(generation = 0; generation < maxIterations; generation++) {
   combinedPopulation = NSGAII.preparePopulation(
       Synthesis.createCombinedPopulation(parent, child));
   parent = NSGAII.getChildFromCombinedPopulation(combinedPopulation);
   child = Synthesis.synthesizeChild(parent);
}
```

Výpis 7.3: Hlavný cyklus algoritmu NSGA-II

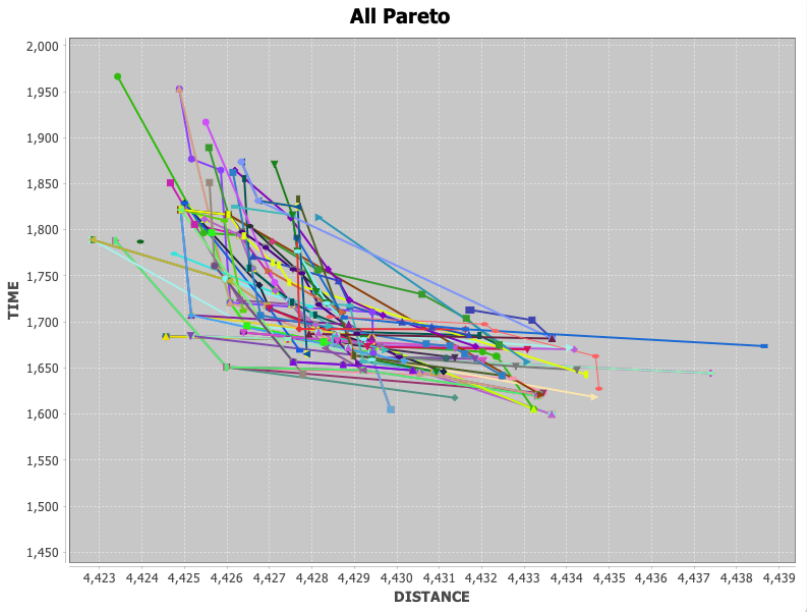
Hlavný cyklus algoritmu NSGA-II, ktorý využíva základné komponenty obecného algoritmu NSGA-II je znázornený na ukážke 7.3. Vytvoria sa dve populácie *parent* a *child*, každá veľkosti N. Metódou *Synthesis.createCombinedPopulation(parent, child)* spojí algoritmus obidve populácie do jednej. Metóda *preparePopulation(population)* (znázornená na ukážke 7.4) volá základné komponenty algoritmu NSGA-II ako *fastNonDominatedSort,*

crowdingDistanceAssignment a *randomizedQuickSortForRank* definované v kapitole 4 v sekcií NSGA-II. Následne je volaná funkcia *getChildFromCombinedPopulation(combinedPopulation)*, ktorá vyberá jedincov do novej generácie podľa atribútu *rank* a *crowdingDistance* ako je to definované v kapitole 4 v sekcií NSGA-II. Metóda synthesizeChild(parent) dopĺňa novú generáciu o potomkov geneticky upravených z rodičovských jedincov *parent*. V tejto metóde sa aplikujú genetické operácie, v tomto prípade len mutácia.

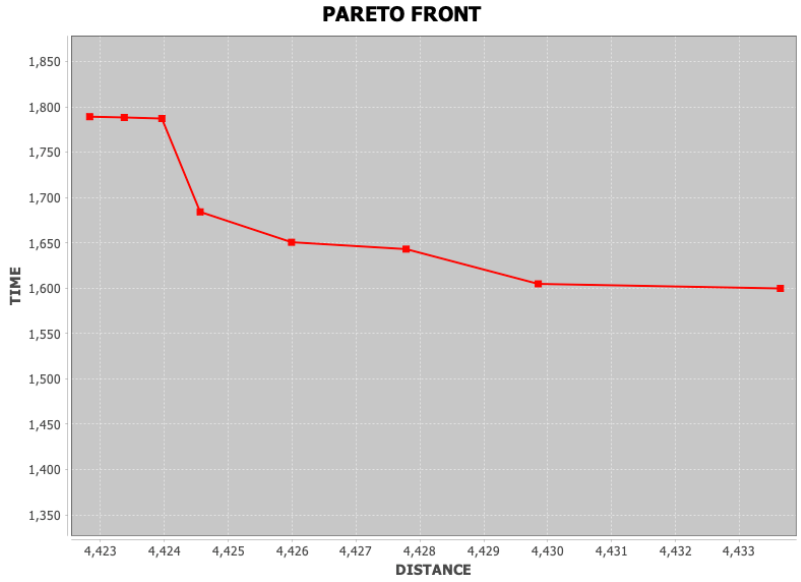
```
public Population preparePopulation(Population population){
   NSGAII.fastNonDominatedSort(population);
   NSGAII.crowdingDistanceAssignment(population);
   Service.randomizedQuickSortForRank(population);
   return population;
}
```

Výpis 7.4: Metóda preparePopulation

Výsledné hodnoty Pareto fronty sú vypísané do tabuľky aplikácie a je možné z nich vygenerovať Ganttov diagram. Po skončení algoritmu NSGA-II sú výsledne Pareto optimálne riešenia vizualizované grafmi znázornenými na obrázkach 7.5 a 7.6. Graf na obr. 7.5 vykresluje všetky pareto optimálne riešenia pre jednotlivé generácie (generácie sú odlíšené farbou). Graf 7.6 vykresluje Pareto frontu odpovedajúcu ku grafu 7.5, kde vytvára Pareto optimálne riešenia zo všetkých generácií. Grafy boli implementované prostredníctvom knižníc *JFreeChart(1.5.0)* a *JCommon(1.0.24)*.



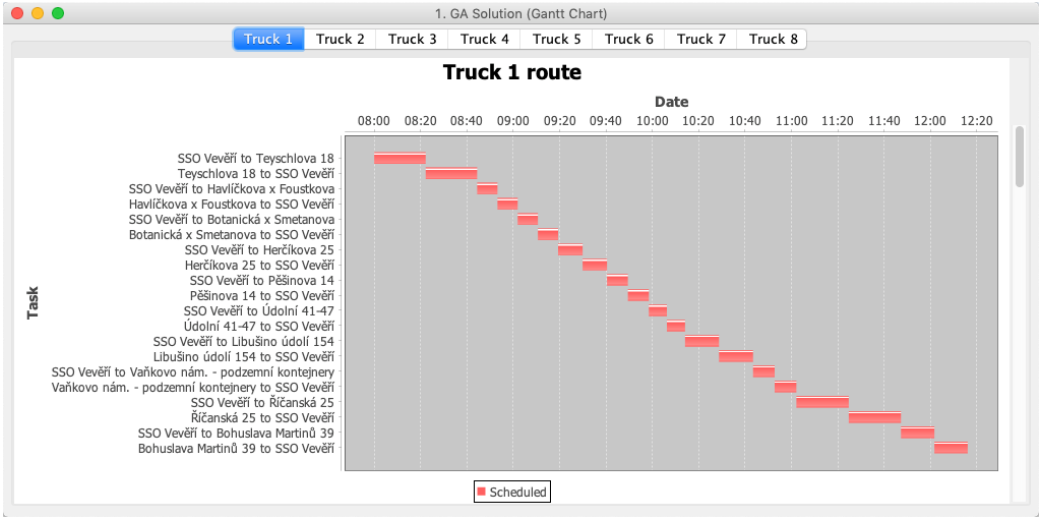
Obr. 7.5: Graf Pareto optimálnych riešení jednotlivých generácií



Obr. 7.6: Graf Pareto fronty algoritmu NSGA-II

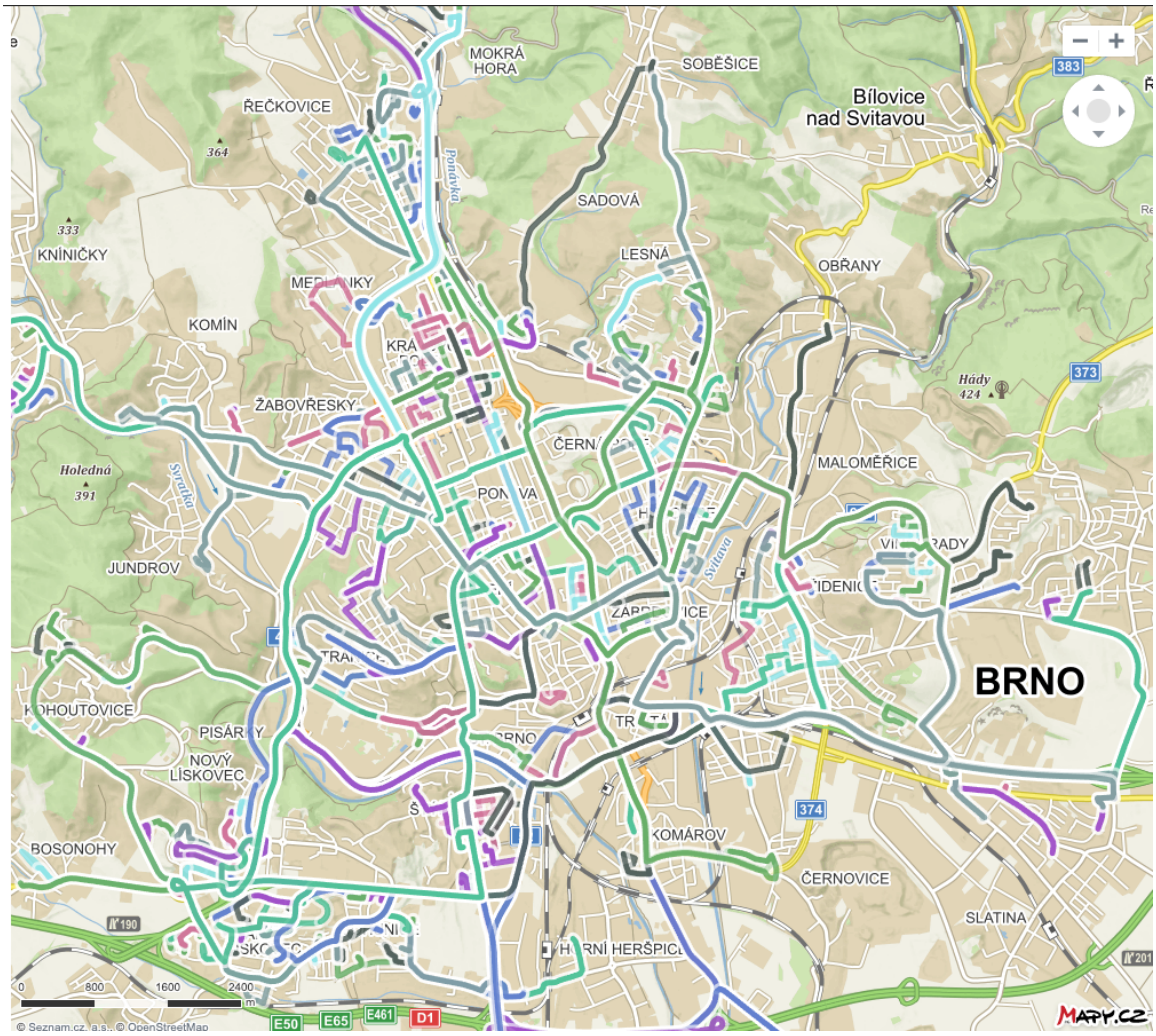
7.3 Vizualizácia riešenia

Riešenia z tabuľky desktopovej aplikácie sú vizualizované prostredníctvom Ganttovho diagramu. Implementácia Ganttovho diagramu využíva knižnice *JFreeChart(1.5.0)* a *JCommon(1.0.24)*. Na obrázku 7.7 je znázornené riešenie v Ganttovom diagrame. Diagram zobrazuje plán trasy pre jedno vozidlo v smere od zberného strediska k lokalite veľkoobjemného kontajnera a naopak. Diagram podporuje približovanie a selekciu vybranej časti grafu. Každá úloha tohto diagramu je vyznačená názvom, ktorý určuje odkiaľ aktuálne vozidlo štartuje a kde má cieľ. To znamená, že pre úlohu s názvom *SSO Vevěří to Herčíkova 25*, je štartovacia lokalita v zbernom dvore SSO Vevěří a destinácia na adrese Herčíkova 25. Prostredníctvom horného panelu je možné prepínať medzi jednotlivými plánmi vozidiel.



Obr. 7.7: Riešenie v Ganttovom diagrame

Druhou vizualizáciou je interaktívna mapa vo webovej aplikácií, v ktorej je možné zobraziť trasy vozidiel. Každé vozidlo má odlišnú farbu a je možné vykresliť všetky vozidlá naraz, alebo vybrať konkrétne vozidlá na zobrazenie. V tejto vizualizácií, na rozdiel od Ganttovho diagramu, chýba informácia o čase jednotlivých úloh, čo môže byť jedno z ďalších rozšírení vývoja tejto aplikácie. Na obrázku 7.8 je ukážka interaktívnej mapy.



Obr. 7.8: Ukážka mapy naplánovaných trás 8 vozidiel

Kapitola 8

Experimenty

Úlohou kapitoly Experimenty je predovšetkým porovnanie získaných výsledkov medzi jednotlivými metódami optimalizácie a znázornenie behu týchto metód. Dátová sada, na ktorej boli prevedené experimenty, bola čerpaná z verejne dostupného zdroja [5]. Online dostupný zdroj znázorňuje na mape lokality jednotlivých nádob na komunálny odpad a lokality zberných dvorov v Brne. Lokality týchto nádob na komunálny odpad, pre účely experimentov, reprezentovali veľkoobjemné kontajnery. Z celkového počtu 1 200 nádob boli na mape vybrané len niektoré, od seba dostatočne vzdialené lokality, ktoré by mohli odpovedať približnému rozmiestneniu veľkoobjemných kontajnerov. Dátová sada je tvorená lokalitami veľkoobjemných kontajnerov veľkosti 50, 150, 300 a 485. Algoritmy pracujú s troma zbernými strediskami. V nasledujúcich sekciách sú popísané experimenty jednotlivých metód a ich vzájomné porovnanie.

8.1 Jednoduchá metóda

Algoritmus jednoduchej metódy je deterministický, takže pre každý beh s rovnakou vstupnou datovou sadou a parametrami dostaneme vždy rovnaký výsledok. Experimenty tejto metódy sú preto porovnávané na základe rôznych parametrov algoritmu. Datová sada, na ktorej budeme testovať rôzne hodnoty parametrov algoritmu pozostáva zo 485 lokalít veľkoobjemných kontajnerov a troch zberných stredísk.

V prvom experimente ukážeme závislosť počtu vozidiel k celkovej cene riešenia. V tabuľke 8.1 sú znázornené výsledky experimentu jednoduchej metódy pri spustení s rôznym počtom vozidiel. Parameter ceny jednej hodiny jedného vozidla je 390 Kč a parameter ceny jedného kilometra je 6 Kč. Riešenie s troma vozidlami je najhoršie v porovnaní s ostatnými, pretože každému stredisku je priradené práve jedno vozidlo. V tomto prípade je celkový čas plánu oproti ostatným riešeniam vysoký, čo spôsobí nárast v cene celého riešenia.

Od riešenia s počtom vozidiel 12, je veľkým prídavným faktorom aj počet vozidiel, ktorý upravuje celkovú cenu riešenia. Riešenie s počtom vozidiel 12 má menší celkový čas a menší počet najazdených kilometrov, ako riešenie s počtom vozidiel 13, no má lepšiu celkovú cenu. Spôsobené to je rozdielnym počtom vozidiel, kde cena hodiny zvozu a rozvozu je násobená každým vozidlom. Čím viac vozidiel, tým väčšia cena za hodinu. Od riešenia s počtom vozidiel 12, celková cena už len stúpa. Najlepšie riešenie jednoduchej metódy pri 485 kontajneroch a 3 strediskách je riešenia s 12 vozidlami.

Počet vozidiel	Celkový čas (v minútach)	Celková vzdialenosť (v km)	Celková cena (v Kč)
3	7 119	4 391	165 187
4	3 713	4 391	122 897
5	3 713	4 395	147 055
6	2 610	4 395	128 180
7	2 610	4 399	145 173
8	1 966	4 399	128 665
9	1 733	4 399	127 812
10	1 733	4 404	139 107
11	1 468	4 404	131 421
12	1 222	4 404	121 769
13	1 146	4 403	123 274
14	1 202	4 403	135 817
15	1 202	4 403	143 632

Tabuľka 8.1: Výsledky experimentu jednoduchej metódy pri spustení s rôznym počtom vozidiel

So stúpajúcim počtom vozidiel klesá celkový čas riešenia a stúpa počet najazdených kilometrov. Výsledná cena nemá lineárny pokles, alebo stúpanie. Veľkú rolu vo výpočte celkovej ceny hrá rozdelenie vozidiel medzi jednotlivé zberné dvory. Na ukážke 8.1 je znázornený výpis výpočtu algoritmu najlepšieho riešenia, kedy algoritmus počíta s 12 vozidlami. Ukážka 8.2 znázorňuje výpis výpočtu algoritmu druhého najhoršieho riešenia, v ktorom je zvolených 5 vozidiel.

```
All distances for the 1. waste yard: 2 737 580 m
All distances for the 2. waste yard: 921 134 m
All distances for the 3. waste yard: 727 730 m
Number of vehicles added to the 1. waste yard: 7
Number of vehicles added to the 2. waste yard: 3
Number of vehicles added to the 3. waste yard: 2
Number of VOKs for the 1. vehicle: 38
Number of VOKs for the 2. vehicle: 38
Number of VOKs for the 3. vehicle: 38
Number of VOKs for the 4. vehicle: 38
Number of VOKs for the 5. vehicle: 38
Number of VOKs for the 6. vehicle: 38
Number of VOKs for the 7. vehicle: 34
Number of VOKs for the 8. vehicle: 41
Number of VOKs for the 9. vehicle: 41
Number of VOKs for the 10. vehicle: 40
Number of VOKs for the 11. vehicle: 51
Number of VOKs for the 12. vehicle: 50
```

Výpis 8.1: Výpis algoritmu jednoduchej metódy (485 VOK-ov, 3 strediská a 12 vozidiel)

```
All distances for the 1. waste yard: 2 737 580 m
All distances for the 2. waste yard: 921 134 m
All distances for the 3. waste yard: 727 730 m
Number of vehicles added to the 1. waste yard: 2
Number of vehicles added to the 2. waste yard: 2
Number of vehicles added to the 3. waste yard: 1
Number of VOKs for the 1. vehicle: 132
Number of VOKs for the 2. vehicle: 130
Number of VOKs for the 3. vehicle: 62
Number of VOKs for the 4. vehicle: 60
Number of VOKs for the 5. vehicle: 101
```

Výpis 8.2: Výpis algoritmu jednoduchej metódy (485 VOK-ov, 3 strediská a 5 vozidiel)

Vo výpise 8.1 je efektívne pridelené množstvo vozidiel podľa všetkých vzdialeností jednotlivých stredísk, následkom čoho nestúpa cena za čas vozidlám, ktoré skončili zvoz a rozvoz. Každé vozidlo má pridelený pomerne rovnaký počet VOK-ov od 38 do 51 kontajnerov. Naopak vo výpise 8.2 je k stredisku 1 a 2 pridelený rovnaký počet vozidiel, pričom stredisko 1 má pridelenú celkovú vzdialenosť 2 737 580 metrov a stredisko 2 má 921 134 metrov. Veľký nepomer je v priradení VOK-ov k vozidlám strediska 1 (132 a 130 kontajnerov) a strediska 2 (62 a 60 kontajnerov). Dôsledkom tohto riešenia je vysoký celkový čas a vysoká celková cena nákladov za zvoz a rozvoz.

Druhým experimentom jednoduchej metódy je znázornenie závislostí parametrov ceny času vozidla a ceny vzdialenosti k celkovej cene riešenia. Experiment počíta s 485 kontajnermi a s troma strediskami. Výsledky tohto experimentu sú znázornené v tabuľke 8.2. Celková cena je počítaná váhovým súčtom dvoch kritérií (čas a vzdialenosť). K týmto kritériam sa pridávajú váhy (cena času a cena vzdialenosti). Výsledkom toho je lineárna závislosť medzi týmito parametrami ako je vidieť v tabuľke 8.2.

Cena za jeden km (v Kč)	**Cena za jednu hodinu jedného vozidla (v Kč)**	**Celková cena riešenia (v Kč)**
6	390	128 665
7	390	133 065
6	408	133 385
8	390	137 464
6	424	137 581
9	390	141 864
6	441	142 038
10	390	146 264
6	458	146 496

Tabuľka 8.2: Výsledky experimentu jednoduchej metódy pri spustení s rôznymi parametrami cien

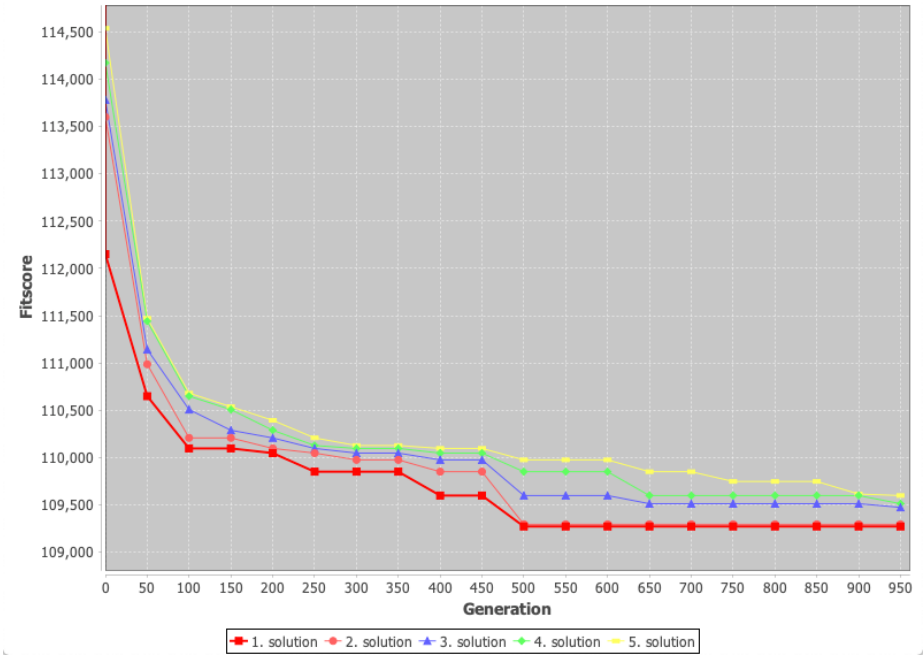
Posledný experiment jednoduchej metódy v tabuľke 8.3 zobrazuje najlepšie riešenia pre rôzne dátové sady (50, 150, 300 a 485 kontajnerov) z pohľadu celkových nákladov na dopravu. Tieto výsledky využijeme v nasledujúcich experimentoch ostatných algoritmov pri porovnávaní metód.

Počet VOK-ov	**Počet vozidiel**	**Celkový čas (v minútach)**	**Celková vzdialenosť (v km)**	**Celková cena (v Kč)**
50	16	118	544	15 581
150	7	762	1 449	43 403
300	4	2 528	2 691	81 893
485	12	1 222	4 404	121 769

Tabuľka 8.3: Najlepšie riešenia jednoduchej metódy pre rôzne datové sady

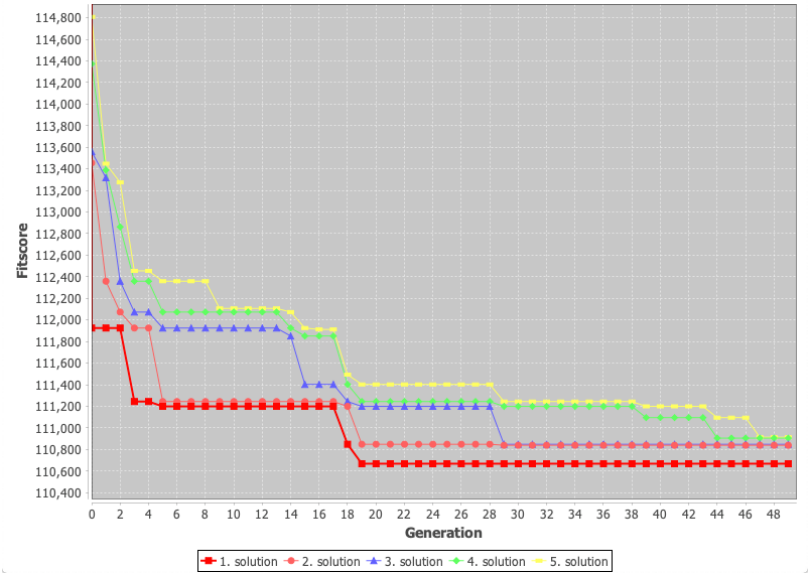
8.2 Genetický algoritmus s agregovanými účelovými funkciami

Najlepšie riešenia genetického algoritmu s agregovanými účelovými funkciami sa menia narastajúcim číslom generácie. Na grafe obrázku 8.1 je znázornených 5 najlepších riešení jedného behu tohto algoritmu, ktorých hodnoty sa menia narastajúcou generáciou. Experiment bol spustený nad štandardnou datovou sadou (485 kontajnerov a 3 zberné dvory) so štandardnými parametrami (cena jednej hodiny na jedno vozidlo je 6 Kč, cena jedného kilometra je 390 Kč, počet vozidiel je 8).



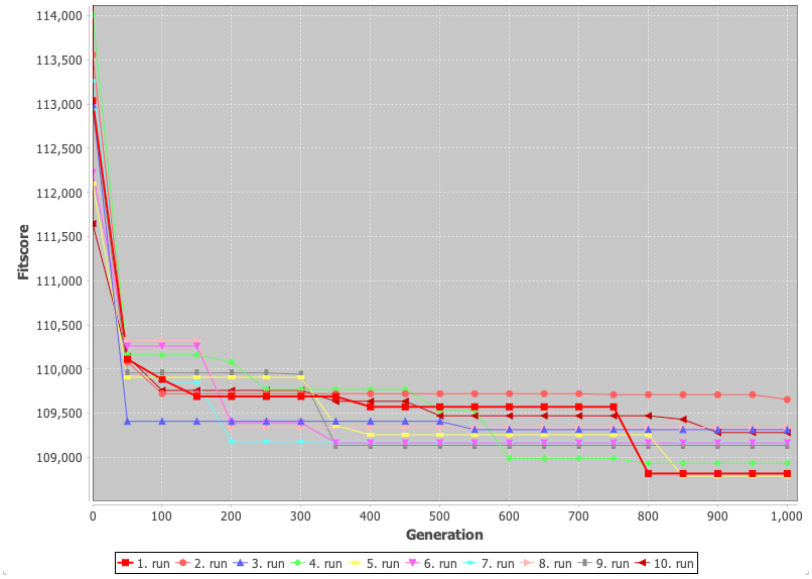
Obr. 8.1: Ukážka jedného behu genetického algoritmu (1 000 generácií)

V grafe na obrázku 8.1 je vidieť veľké zmeny riešení do prvých 500 generácií. Od generácie 500 sa výsledky riešení stabilizujú. Najväčšie zmeny riešení tohto algoritmu sú medzi prvými 50 generáciami. Na grafe obrázku 8.2 je zobrazených 5 najlepších riešení jedného behu algoritmu pri prvých 50 generácií. Týchto 50 generácií odpovedá behu algoritmu znázornenom na obrázku 8.1.



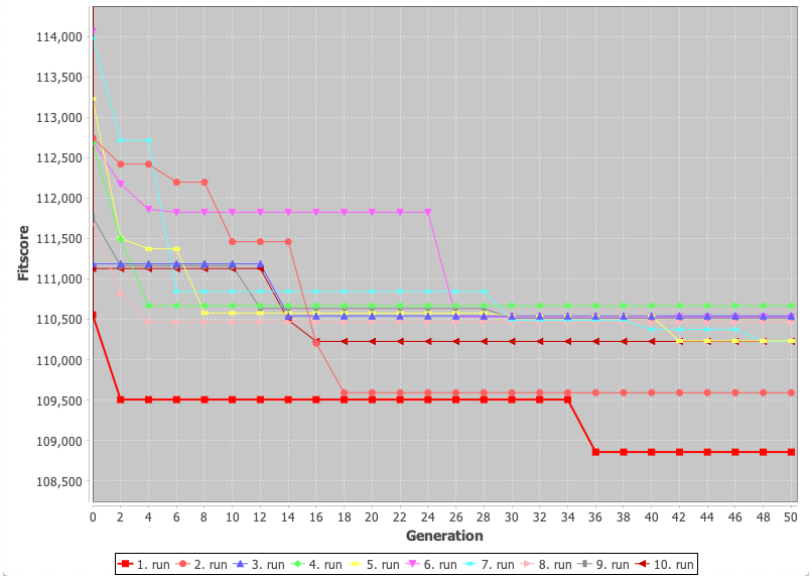
Obr. 8.2: Ukážka jedného behu genetického algoritmu (50 generácií)

Porovnanie najlepších riešení rôznych behov genetického algoritmu je znázornené na grafe v obrázku 8.3. Na grafe je súčasne spustených 10 behov genetického algoritmu, ktoré sú oddelené farebne. Zobrazené sú výsledky najlepších riešení jednotlivých behov. Veľké zmeny nastávajú do generácie 400. Od generácie 400 sú výsledky riešení relatívne stabilné.



Obr. 8.3: Ukážka 10 behov genetického algoritmu (1 000 generácií)

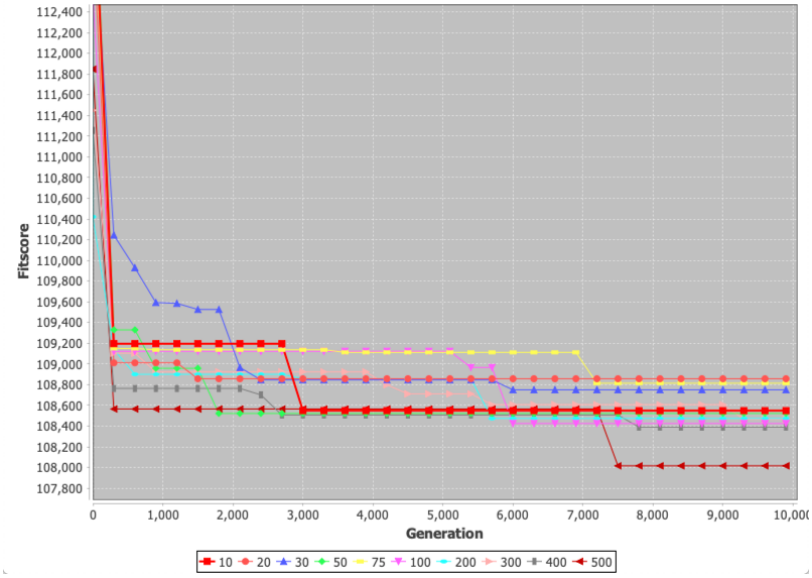
Na grafe v obrázku 8.4 sú znázornené najlepšie výsledky 10 rôznych behov genetického algoritmu v počiatku behu algoritmu (prvých 50 generácií). Podľa grafu vykazujú najväčšie zmeny v porovnaní s neskoršími generáciami, ako na obrázku 8.3. V nasledujúcich podsekciách je znázornená experimentácia rôznych nastavení parametrov genetického algoritmu.



Obr. 8.4: Ukážka 10 behov genetického algoritmu (50 generácií)

8.2.1 Parameter počtu chromozómov

Aby bolo možné vybrať efektívny počet chromozómov, boli zvolené dlhodobé behy algoritmov s 10 000 generáciami. Na grafe v obrázku 8.5 sú znázornené behy algoritmov s rôznym počtom iterujúcich chromozómov.



Obr. 8.5: 10 behov genetického algoritmu s rôznymi počtami chromozómov (1 000 generácií)

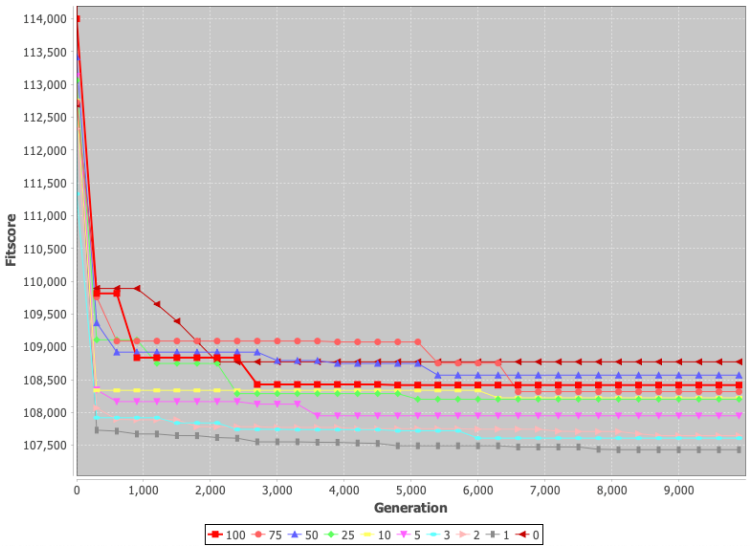
Zvyšovaním počtu chromozómov dosiahneme lepšie riešenia v skorších generáciach. V tabuľke 8.4 je znázornené značné spomalenie algoritmu pri veľkom počte chromozómov. Do úvahy je preto potrebné brať obidva faktory (čas a počet chromozómov). Na základe grafu z obrázka 8.5 a tabuľky 8.4, bolo zvolených 200 chromozómov, ako efektívny počet chromozómov na riešenie tohto problému. Nasledujúce experimenty pracujú s týmto počtom. Rýchlosť výpočtov v tabuľke bola testovaná na počítači s procesorom AMD Phenom II X4 945 s frekvenciou 3 GHz.

Počet chromozómov	**Čas behu 1 000 generácií (v sekundách)**	**Čas behu 1 600 000 generácií (v hodinách)**
500	80	35
400	60	27
300	50	22
200	20	9
100	18	8
75	15	7
50	10	5
30	8	3,5
20	6	2,5
10	4	2

Tabuľka 8.4: Časová závislosť počtu chromozómov algoritmu

8.2.2 Parameter mutácie

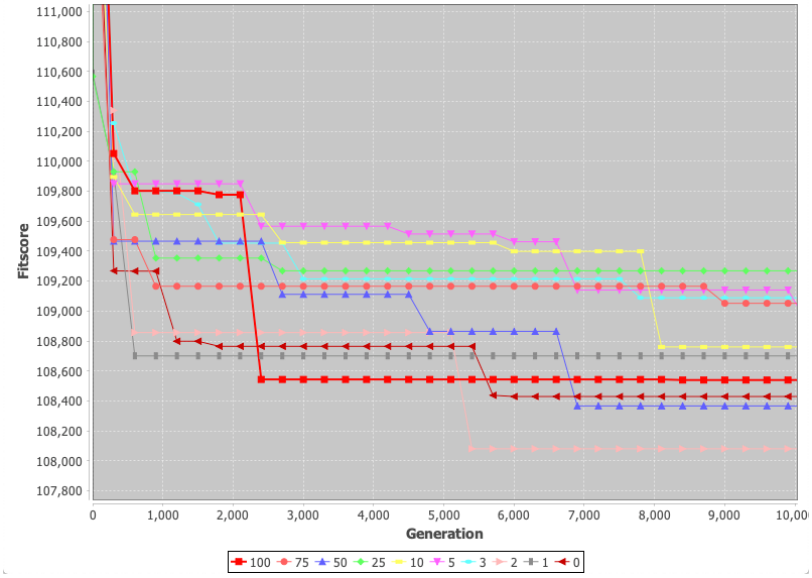
Obidva parametre mutácie je možné zvoliť v desktopovej aplikácií v intervale od 0 do 100 v percentách. Nasledujúce experimenty nastavení týchto parametrov boli spustené na 10 000 generáciách (predstavuje dlhodobý beh), aby bolo možné efektívne určiť najlepšie riešenia.



Obr. 8.6: Ukážka závislosti behov algoritmov na parametri mutácie génu vozidla (10 000 generácií)

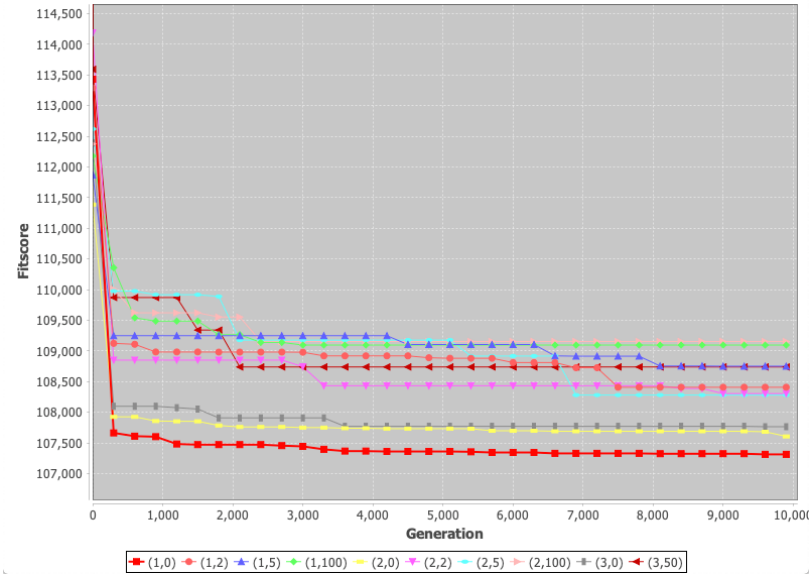
Na grafe v obrázku 8.6 je znázornených 10 behov genetického algoritmu s rôzne nastavenou pravdepodobnosťou mutácie génu (pravdepodobnosť je uvádzaná v percentách), ktorý reprezentuje vozidlo priradené k trojici (trojica reprezentuje kontajner). Z tohto grafu je zreteľné, že s klesajúcou pravdepodobnosťou dostaneme lepšie výsledky v skorších generáciach. Najlepšie riešenie má nastavenie parametra mutácie 1 %. Riešenie s 0 %, tj. neaplikuje sa mutácia na gény reprezentujúce číslo vozidla, vykazuje pomerne slabšie výsledky.

Druhým parametrom mutácie je pravdepodobnosť mutácie génov reprezentujúcich stredisko. Na grafe v obrázku 8.7 sú znázornené behy algoritmu pri rôznych nastaveniach tohto parametra. Hodnoty parametra sú uvádzané v percentách.



Obr. 8.7: Ukážka závislosti behov algoritmov na parametri mutácie génu strediska (10 000 generácií)

Na grafe v obrázku 8.7 nie je možné určiť, ktoré nastavenie mutácie je efektívnejšie, ako je to v prípade nastavenia mutácie vozidla. Na grafe v obrázku 8.8 sú znázornené behy s kombinovanými nastaveniami parametrov mutácie. Jeden beh algoritmu je značený dvojicou (X, Y), kde X je pravdepodobnosť mutácie génu vozidla a Y je pravdepodobnosť mutácie génu strediska. Obidve pravdepodobnosti sú vyjadrené v precentách. Parametre mutácie vozidla boli vybrané podľa výsledkov grafu z obrázka 8.6. Parametre mutácie strediska boli zvolené na základe grafu z obrázka 8.7. Zvolené boli najlepšie nastavenia, ako aj nastavenia, u ktorých dochádzalo k častej zmene riešenia (napr. beh s nastavením 50 % mutácie strediska).



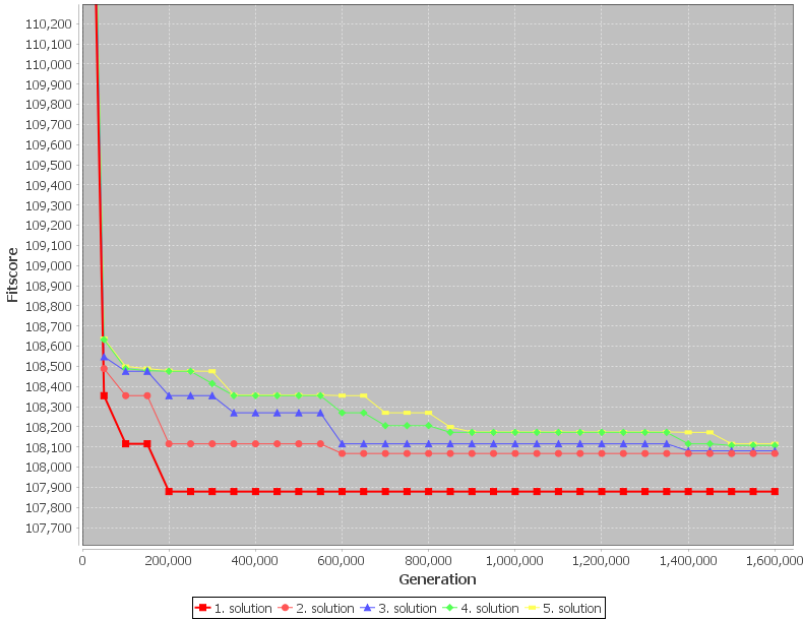
Obr. 8.8: 10 behov genetického algoritmu s rôznymi kombináciami parametrov mutácie (10 000 generácií)

Z grafu na obrázku 8.8 je možné určiť efektívne nastavenie parametrov mutácie. Hodnota efektívneho parametra mutácie génu vozidla je 1 a hodnota parametra mutácie génu strediska je 0. Nasledujúce experimenty budú počítať s týmto nastavením parametrov mutácie.

8.2.3 Modelový príklad nasadenia v praxi

Ako ukážka modelového príkladu nasadenia genetického algoritmu na naplánovanie veľkoobjemných kontajnerov je zvolená dátová sada s 485 kontajnermi, troma strediskami a ôsmimi vozidlami. Počet chromozómov algoritmu je 200. Hodnoty parametrov boli zvolené náhodne. Hodnota parametra mutácie génu vozidla je 50 % a hodnota parametra mutácie génu strediska je 20 %. Experiment bol testovaný na počítači s procesorom AMD Phenom II X4 945 s frekvenciou 3 GHz a operačným systémom Windows 7.

Ukážka príkladu má odpovedať reálnemu využitiu aplikácie, kde počas dňa dispečink nazbiera informácie o aktuálne naplnených kontajneroch a počas noci je spustený genetický algoritmus, ktorý hľadá najlepšie trasy vozidlám na zvoz a rozvoz kontajnerov na nasledujúci deň. Celková doba trvania behu tohto algoritmu bola 9 hodín, čo zodpovedá reálnemu použitiu aplikácie po nasadení. Na obrázku 8.9 je znázornený jeden beh tohto experimentu. Zobrazuje 5 najlepších riešení, ktoré algoritmus počas 9 hodín vypočítal. V tabuľke 8.5 je znázornených 20 najlepších riešení, po skončení algoritmu.



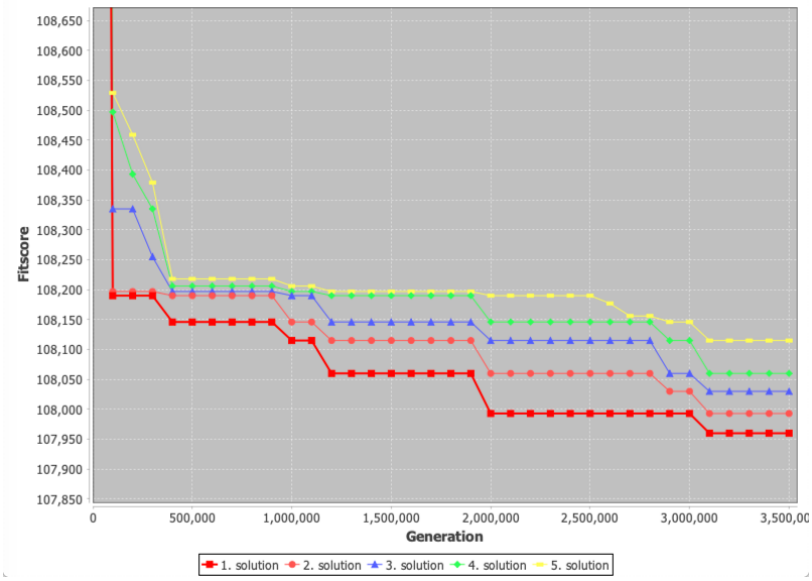
Obr. 8.9: Modelový príklad behu genetického algoritmu v praxi (1 600 000 generácií)

Top solutions	Time (minutes)	Distance (km)	Total costs (Kč)
1. from GA	1563	4433	107879
2. from GA	1565	4442	108068
3. from GA	1566	4436	108081
4. from GA	1567	4430	108108
5. from GA	1567	4432	108116
6. from GA	1568	4436	108173
7. from GA	1568	4434	108175
8. from GA	1568	4440	108199
9. from GA	1569	4435	108206
10. from GA	1569	4436	108208
11. from GA	1570	4431	108244
12. from GA	1570	4434	108250
13. from GA	1570	4429	108261
14. from GA	1570	4437	108270
15. from GA	1570	4440	108284
16. from GA	1570	4437	108310
17. from GA	1571	4437	108319
18. from GA	1571	4430	108323
19. from GA	1572	4430	108331
20. from GA	1571	4437	108347

Tabuľka 8.5: Najlepších riešenia modelového príkladu (po 1 600 000 generáciach)

Z grafu na obrázku 8.9 sa dá pozorovať, že najlepšie riešenie sa od generácie 200 000 nemení. Zmeny v stovkách korún však nastávajú v ostatných riešeniach.

Druhým experimentom modelového príkladu nasadenia je beh genetického algoritmu na rovnakej dátovej sade s rovnakými parametrami, ako v predchádzajúcom príklade, pri 3 500 000 generáciách (zodpovedá 20 hodinám výpočtového času). Na grafe v obrázku 8.10 je znázornených päť najlepších riešení jedného behu algoritmu v priebehu generácií. Odpovedajúca tabuľka 8.6 k tomuto príkladu zobrazuje najlepších 20 riešení po skončení algoritmu. Úlohou tohto experimentu bolo ukázať zmeny riešení pri dlhodobom behu s týmito parametrami.

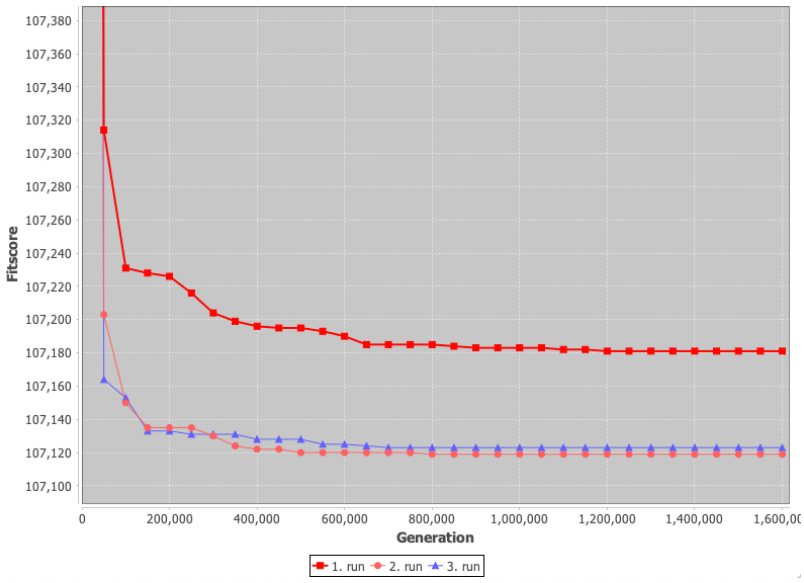


Obr. 8.10: Modelový príklad behu genetického algoritmu v praxi (3 500 000 generácií)

Top solutions	Time (minutes)	Distance (km)	Total costs (Kč)
1. from GA	1564	4436	107960
2. from GA	1565	4433	107993
3. from GA	1565	4436	108030
4. from GA	1566	4437	108060
5. from GA	1566	4440	108115
6. from GA	1568	4427	108136
7. from GA	1567	4437	108146
8. from GA	1568	4430	108154
9. from GA	1568	4434	108156
10. from GA	1568	4435	108163
11. from GA	1568	4437	108165
12. from GA	1568	4433	108177
13. from GA	1568	4433	108181
14. from GA	1569	4430	108190
15. from GA	1568	4437	108197
16. from GA	1569	4432	108206
17. from GA	1569	4432	108209
18. from GA	1569	4437	108218
19. from GA	1569	4436	108227
20. from GA	1570	4431	108239

Tabuľka 8.6: Najlepšie riešenia modelového príkladu (po 3 500 000 generáciach)

Posledný experiment modelového príkladu nasadenia genetického algoritmu využíva efektívne nastavené parametre mutácie a počtu chromozómov z predchádzajúcich experimentov. Počet chromozómov je 200. Hodnota parametru mutácie génu vozidla je 1 %. Hodnota parametru mutácie génu strediska je 0 %. Experiment porovnáva najlepšie riešenia troch behov algoritmu pri rovnakej dĺžke výpočtu (9 hodín), ako v predchádzajúcom experimente. Na obrázku 8.9 je znázornený graf tohto modelového prikladu pri troch behoch algoritmu.



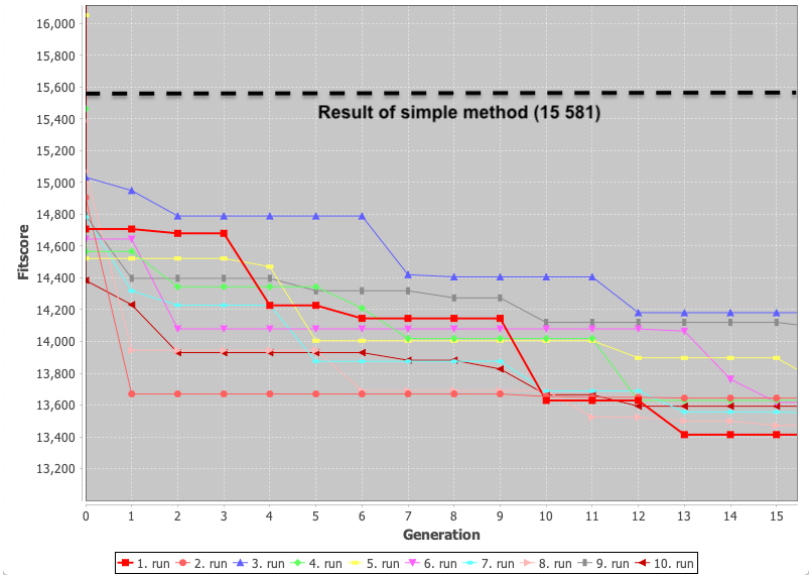
Obr. 8.11: Modelový príklad troch behov genetického algoritmu v praxi (1 600 000 generácií)

Z obrázku 8.11 je vidieť, že správnym nastavením parametrov mutácie, algoritmus rýchlo konverguje k jednému výsledku. Už od generácie 100 000 na obrázku 8.11 algoritmus dostáva vo všetkých behoch lepšie výsledky, ako beh algoritmu na obrázku 8.9. Od generácie 600 000 na obrázku 8.11 sa výsledna hodnota výsledku riešenia nemení (maximálne v jednotkách korún). Efektívny výsledok algoritmu dosiahneme, ak spustíme viac behov algoritmu na približne 600 000 generácií (zodpovedá 2,5 hodiny na bežnom počítači). Najlepší výsledok sady s 485 kontajnermi, ktorý dosiahla metóda genetického algoritmu s agregovanými účelovými funkciami je riešenie s 107 119 Kč celkových nákladov, 1 551 hodín celkového času zvážania odpadu a 4 411 najazdených kilometrov všetkých vozidiel.

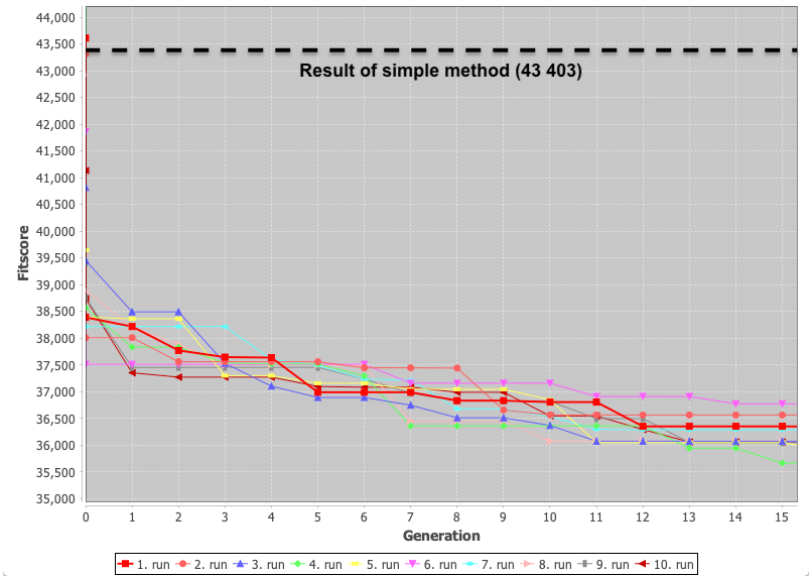
V nasledujúcich podsekciách sú porovnané výsledky behov genetického algoritmu s výsledkami jednoduchej metódy na rôznych dátových sadách.

8.2.4 Porovnanie s jednoduchou metódou

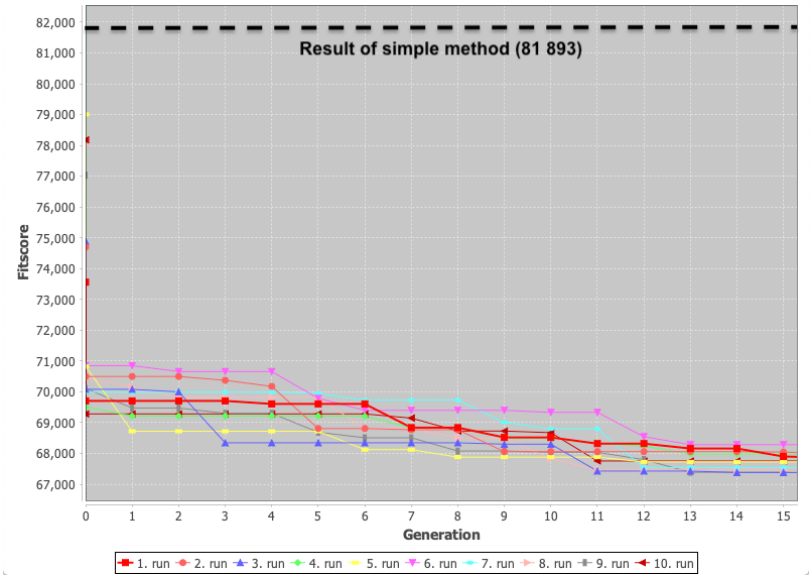
Výsledky najlepších riešení rôznych datových sád jednoduchej metódy z tabuľky 8.3 sú využité pri porovnávaní výsledkov behu genetického algoritmu s agregovanými účelovými funkciami. Pre porovnanie bolo zvolených 10 behov genetického algoritmu na všetkých štyroch datových sadách. Parametre týchto behov boli zvolené z predchádzajúcich experimentov. Zvolených bolo 8 vozidiel, hodnota parametra mutácie génu vozidla bola 1 %, hodnota parametra mutácie génu strediska bola 0 % a počet chromozómov bol 200. Na grafoch v obrázkoch 8.12, 8.13, 8.14, 8.15 sú znázornené tieto experimenty.



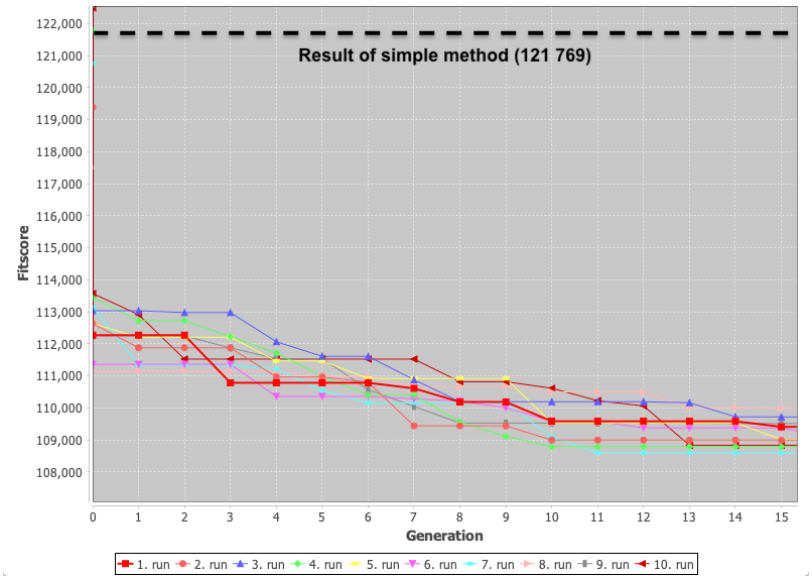
Obr. 8.12: Porovnanie jednoduchej metódy s genetickým algoritmom (50 VOK-ov)



Obr. 8.13: Porovnanie jednoduchej metódy s genetickým algoritmom (150 VOK-ov)



Obr. 8.14: Porovnanie jednoduchej metódy s genetickým algoritmom (300 VOK-ov)



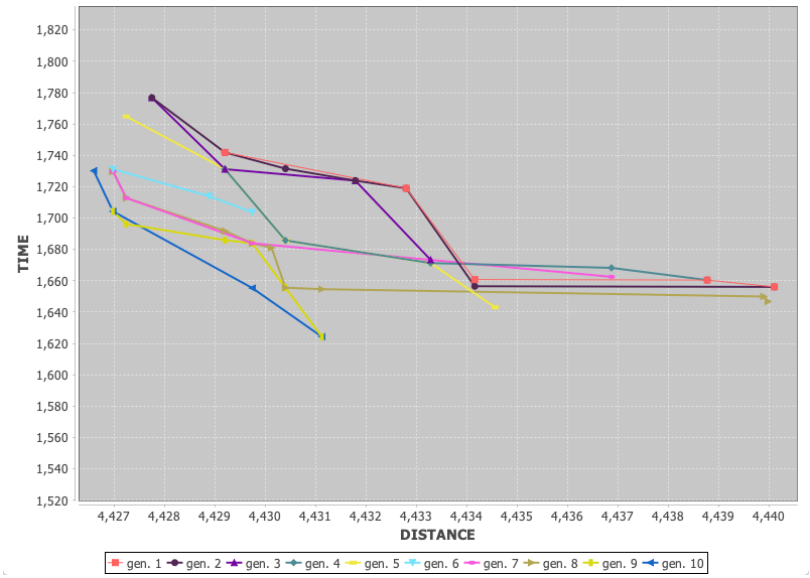
Obr. 8.15: Porovnanie jednoduchej metódy s genetickým algoritmom (485 VOK-ov)

Zo znázornených grafov je možné vidieť, že už v prvej generácií sú riešenia všetkých behov genetického algoritmu lepšie, ako výsledky z jednoduchej metódy. Genetický algoritmus s agregovanými učelovými funkciami je preto efektívnejší, ako jednoduchá metóda (zodpovedajúca súčasnému plánovaniu), keďže dokáže vygenerovať lepší výsledok za 20 milisekúnd (zodpovedá rýchlosti výpočtu jednej generácie bežným počítačom).

8.3 Algoritmus NSGA-II

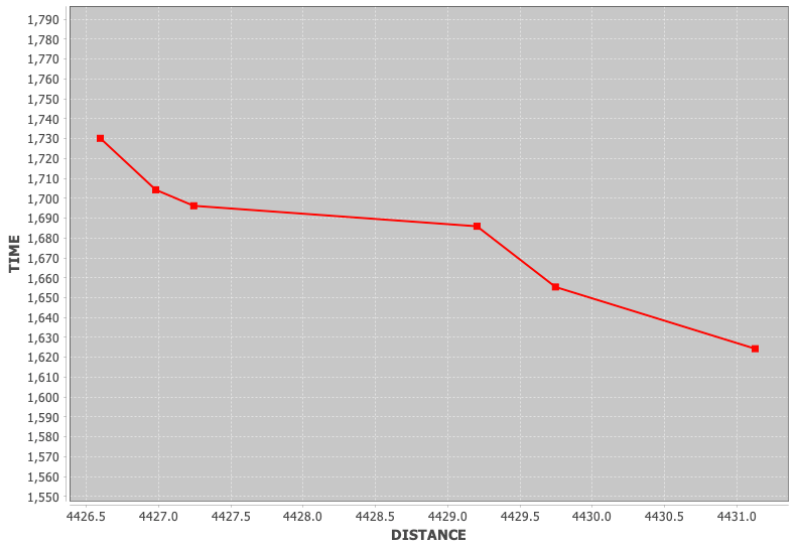
Zloženie genetického algoritmu NSGA-II je z väčšej časti rovnaké, ako genetický algoritmus s agregovanými účelovými funkciami. Obsahuje rovnaký genetický operátor mutácie, pracuje s rovnakou reprezentáciou chromozómu a rovnako tak počet chromozómov ovlyvňuje rýchlosť behu algoritmu NSGA-II. Zvyšovaním počtu chromozómov algoritmu sa zmenšuje doba výpočtu jednej generácie. Genetický algoritmus s agregovanými účelovými funkciami počíta 1 600 000 generácií 8 hodín, pri 200 vozidlách. Algoritmu NSGA-II to trvá 20 hodín, čo je skoro dvojnásobný čas. Keďže mutácia algoritmu NSGA-II je rovnaká, nie je potrebné experimentovať s nastavením parametrov mutácie. Tieto parametre budú nastavené podľa experimentov genetického algoritmu s agregovanými účelovými funkciami. Hodnota parametra mutácie génu vozidla bude 1 %. Hodnota parametra mutácie génu strediska bude 0 %. Počet chromozómov bude 200. Počet vozidiel bude 8.

Prvý experiment algoritmu NSGA-II bol spustený nad dátovou sadou so 485 kontajnermi a 3 strediskami. Na grafe v obrázku 8.16 sú znázornené Pareto optimálne riešenia jednotlivých generácií. Experimentoval sa beh algoritmu NSGA-II pri 10 generáciach.



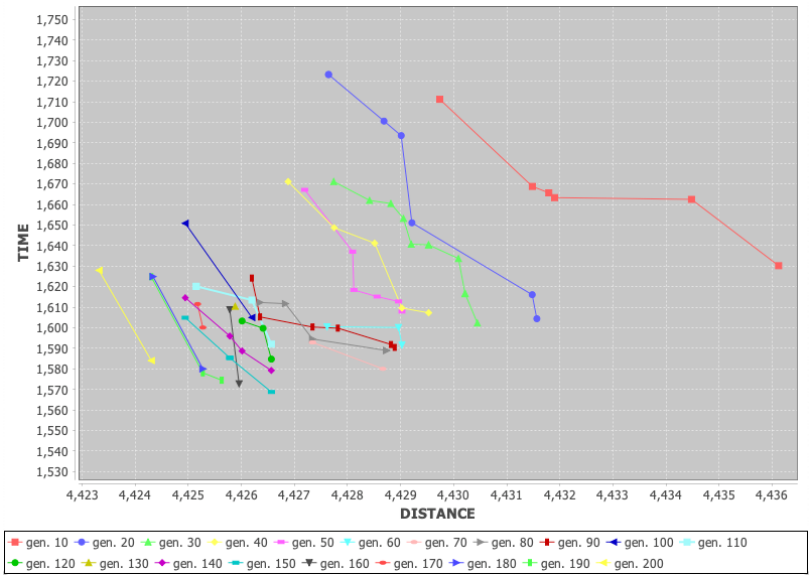
Obr. 8.16: Pareto optimálne riešenia prvých 10 generácií algoritmu NSGA-II

Z grafu na obrázku 8.16 je vidieť správne smerovanie algoritmu NSGA-II v priebehu generácií k počiatku súradnicovej osi (0,0). Pareto optimálne riešenia generácie 1 sú najďalej v porovnaní s riešeniami ostatných generácií. Do výslednej pareto fronty sú pridané riešenia zo všetkých pareto optimálnych riešení jednotlivých generácií, ktoré najlepšie odpovedali kritériám Pareto fronty. Výsledná pareto fronta je znázornená na grafe v obrázku 8.17.

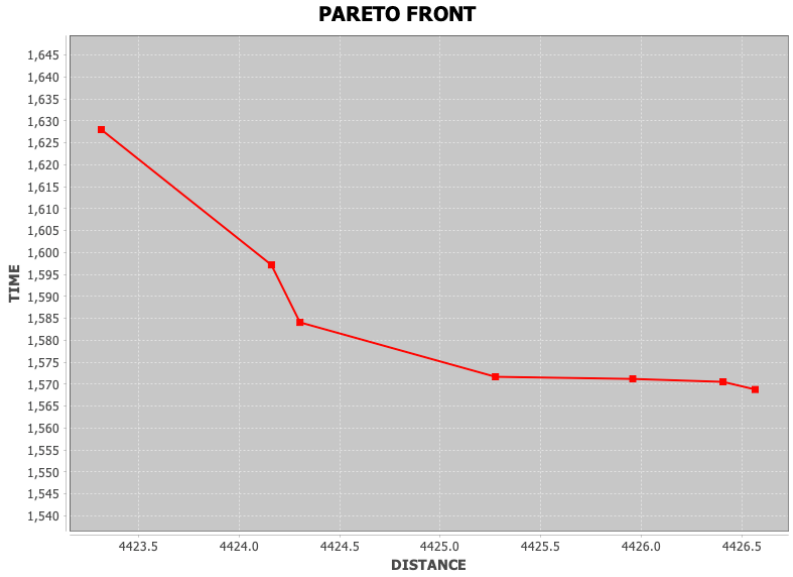


Obr. 8.17: Pareto fronta prvých 10 generácií algoritmu NSGA-II

Druhý experiment lepšie znázorňuje priebeh smerovania výsledkov jednotlivých generácií. Znázornený je na obrázku 8.18. Tento experiment bol spustený na rovnakej dátove sade, ako predchádzajúci experiment. Algoritmus počítal 200 generacií. Na grafe v obrázku 8.18 sú vyznačené Pareto optimálne riešenia každej desiatej generácie. Odpovedajúca Pareto fronta tohto experimentu je na obrázku 8.19. Táto fronta je počítaná zo všetkých generácií, nie len z vybratých generácií, ktoré sú zobrazené na obrázku 8.18.



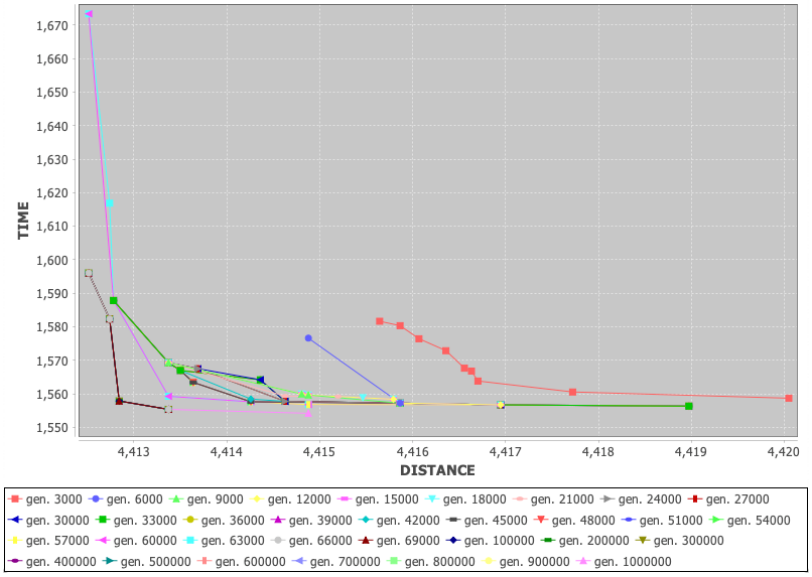
Obr. 8.18: Pareto optimálne riešenia algoritmu NSGA-II (200 generacií)



Obr. 8.19: Pareto fronta algoritmu NSGA-II (200 generacií)

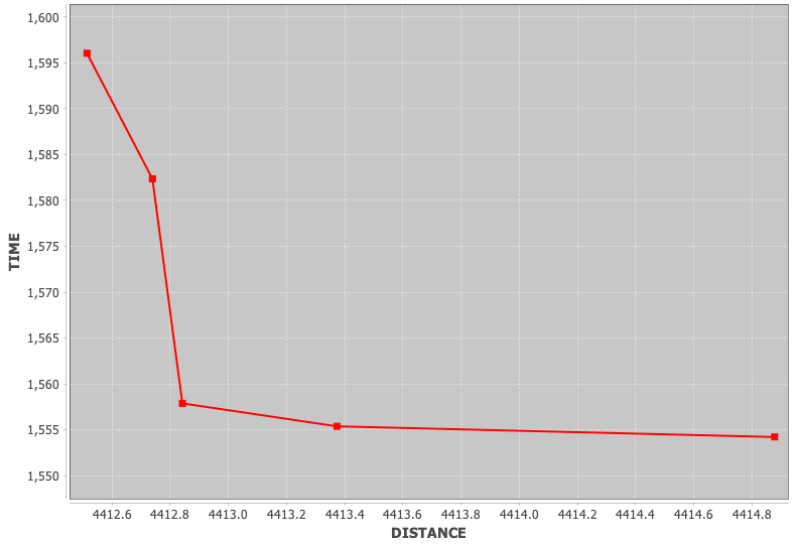
8.3.1 Modelový príklad nasadenia v praxi

Ukážka modelového príkladu nasadenia algoritmu NSGA-II je spustená na dátovej sade so 485 kontajnermi a 3 strediskami. Parametre sú zvolené podľa minulých experimentov genetického algoritmu, tj. parameter mutácie génu vozidla je 1 %, parameter mutácie génu strediska je 0 %, počet vozidiel je 8 a počet chromozómov je 200. Na obrázku 8.20 je graf znázorňujúci beh algoritmu NSGA-II pri 1 000 000 generáciach. Experiment bol testovaný na počítači s procesorom AMD Phenom II X4 945 s frekvenciou 3 GHz. Trval 10 hodín.



Obr. 8.20: Pareto fronty generácií algoritmu NSGA-II (1 000 000 generacií)

Graf na obrázku 8.20 zobrazuje pareto fronty vytvorené zo všetkých predchádzajúcich generácií, na rozdiel od predchádzajúcich experimentov algoritmu NSGA-II, kde sú zobrazované pareto optimálne riešenia jednotlivých generácií. Pareto frontu sa mení do generácie 66 000. Od generácie 66 000 sa táto fronta stabilizuje a negenerujú sa nové riešenia. Na grafe v obrázku 8.25 je znázornená výsledná Pareto fronta po 1 000 000 generáciach tohto experimentu.



Obr. 8.21: Pareto fronta algoritmu NSGA-II (1 000 000 generacií)

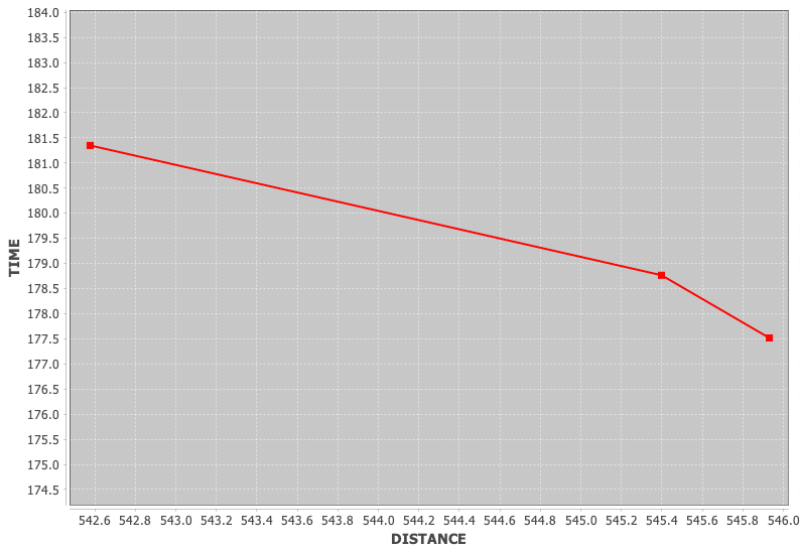
8.3.2 Porovnanie s ostatnými metódami

Porovnávané sú výsledky všetkých metód na všetkých dátových sadách. V tabuľke 8.7 sú zobrazené najlepšie dosiahnuté výsledky jednoduchej metódy a metódy genetického algoritmu s agregovanými účelovými funkciami. Výsledky jednoduchej metódy sú čerpané z tabuľky 8.3. Každý z behov genetického algoritmu s agregovanými účelovými funkciami trval 9 hodín, ktorých výsledky sú pridané do tejto tabuľky. Obidva algoritmy počítali s rovnakými hodnotami parametrov ceny času a vzdialenosti. Hodnota ceny jednej hodiny jedného vozidla je 390 Kč a cena jedného kilometra je 6 Kč. Tieto hodnoty sú stanovené na základe priemernej spotreby veľkoobjemného vozidla a priemerných platov smetiarov. Genetický algoritmus s agregovanými účelovými funkciami a algoritmus NSGA-II počítajú s 8 vozidlami na zvoz a rozvoz kontajnerov.

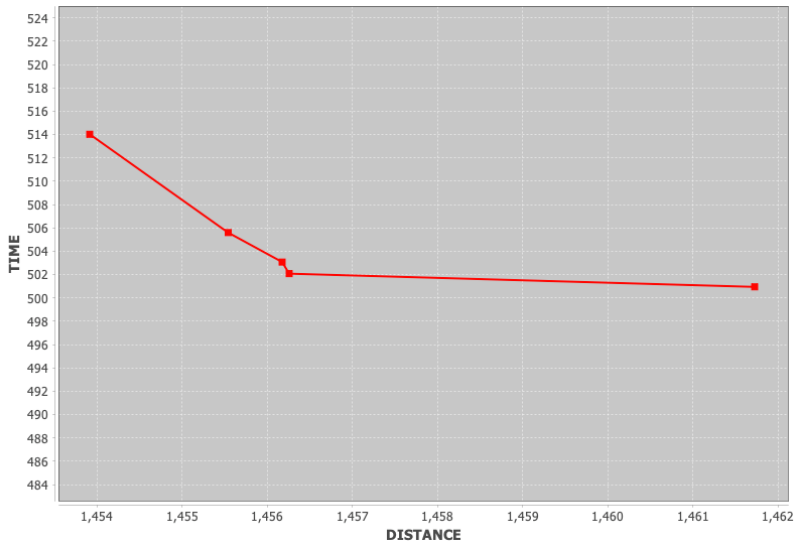
Počet VOK-ov	Optimalizačná metóda	Počet vozidiel	Celkový čas (v minútach)	Celková vzdialenosť (v km)	Celková cena (v Kč)
50	Jednoduchá metóda	10	118	544	15 581
50	Gen. alg. s ag. úč. fun.	8	117	551	12 544
150	Jednoduchá metóda	7	762	1 449	43 403
150	Gen. alg. s ag. úč. fun.	8	501	1 482	34 991
300	Jednoduchá metóda	4	2 528	2 691	81 893
300	Gen. alg. s ag. úč. fun.	8	959	2 734	66 277
485	Jednoduchá metóda	12	1 222	4 404	121 769
485	Gen. alg. s ag. úč. fun.	8	1 551	4 411	107 119

Tabuľka 8.7: Výsledky metód optimalizácie

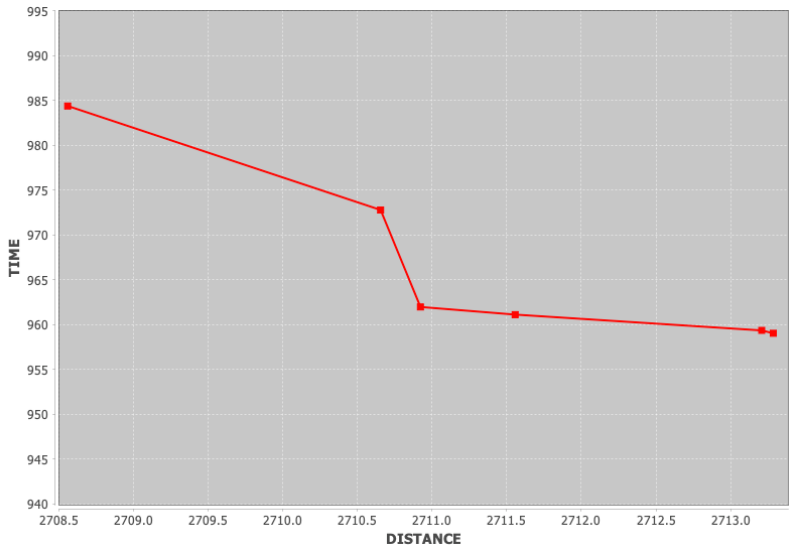
Výsledky z tabuľky 8.7 porovnáme s výsledkami algoritmu NSGA-II na všetkých dátových sadách. Algoritmus NSGA-II bol spustený 10 hodín nad každou dátovou sadou. Nastavené mal rovnaké parametre mutácie a počtu chromozómov, ako v predchádzajúcich experimentoch algoritmu NSGA-II. Na obrázkoch 8.22, 8.23, 8.24 a 8.25 sú znázornené grafy s pareto frontami behov algoritmu NSGA-II, na jednotlivých dátových sadách.



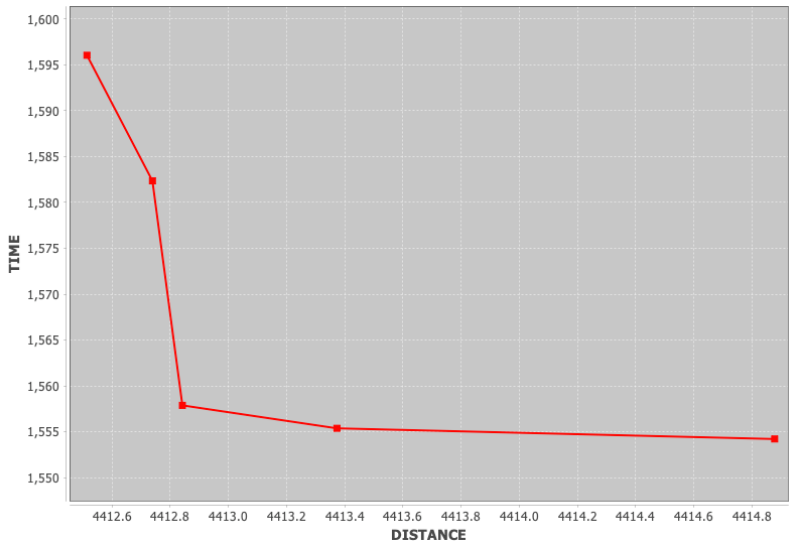
Obr. 8.22: Pareto fronta algoritmu NSGA-II (50 VOK-ov, 3 000 000 generacií)



Obr. 8.23: Pareto fronta algoritmu NSGA-II (150 VOK-ov, 2 250 000 generacií)



Obr. 8.24: Pareto fronta algoritmu NSGA-II (300 VOK-ov, 1 400 000 generacií)



Obr. 8.25: Pareto fronta algoritmu NSGA-II (485 VOK-ov, 1 000 000 generacií)

Jednoduchú metódu nebudeme porovnávať s výsledkom algoritmu NSGA-II, pretože je spúšťaná s rôznym počtom vozidiel ako algoritmus NSGA-II. Z tabuľky 8.7 vidíme, že pre všetky dátové sady, genetický algoritmus našiel lepší výsledok, ako jednoduchá metóda. Algoritmy NSGA-II a genetický algoritmus s agregovanými účelovými funkciami sú spustené s 8 vozidlami.

Porovnaním výsledkov z tabuľky 8.7 a grafov riešení pre jednotlivé sady sme zistili, že pri určených cenách času a vzdialenosti, genetický algoritmus s agregovanými účelovými funkciami pracuje lepšie, ako algoritmus NSGA-II. Efektívnejší však je algoritmus NSGA-II, v prípade že tieto ceny nemáme určené. Algoritmus NSGA-II dáva v takom prípade používateľovi viac možností ako naplánovať zvoz a rozvoz veľkoobjemných kontajnerov z pohľadu dvoch kritérií. Tento algoritmus prehľadáva riešenia s cieľom dosiahnúť rovnovážneho stavu medzi obidvoma kritériami. Používateľ sa tak môže prikloniť k jednému kritériu na úkor zhoršenia toho druhého.

Kapitola 9

Záver

Riešenie problému plánovania zvozu a rozvozu veľkoobjemných kontajnerov v jednom meste nie je vôbec triviálne. Naštastie, dokážeme vyvinúť spôsoby, ktorými nájdeme efektívne trasy vozidlám, ktoré zvážajú a rozvážajú tieto kontajnery. Aplikačný systém, ktorý je výstupom tejto práce, dokáže v rozumnom čase naplánovať zvolené vozidlá s cieľom minimalizácie celkových nákladov.

Práca sa zaoberá rozborom, návrhom a implementáciou optimalizácie tohto problému. Určil som zložitosť tohto problému, popísal podobné problémy a z teoretického hľadiska ukázal, že sa jedná o NP-ťažký problém, v ktorom je potrebné využiť heuristické a aproximačné metódy. Preštudoval som preto problematiku genetických algoritmov. Vyznačuje sa ako overená a primeraná metóda podobných problémov problému plánovania VOK-ov. Aby som docielil lepšiu efektivitu, využil som algoritmy multikriteriálnej optimalizácie.

Po dôkladnej príprave návrhu genetických algoritmov som v implementácií ukázal rozdelenie aplikačného systému na dve entity (aplikácie):

- *webová aplikácia*, ktorá získava najkratšie trasy medzi všetkými kontajnermi a strediskami a vizualizuje riešenia na interaktívnej mape,
- *desktopová aplikácia*, ktorá spúšta metódy optimalizáciea vizualizuje riešenie Ganttovým diagramom.

Následne som ukázal implementáciu troch optimalizačných metód:

- *jednoduchá metóda*, ktorá zastávala funkciu súčasného plánovania,
- *genetický algoritmus s agregovanými účelovými funkciami*, ktorý optimalizoval z pohľadu minimalizácie ceny riešenia,
- *algoritmus NSGA-II*, ktorý dával používateľovi viac možností výberu ideálneho riešenia z pohľadu dvoch kritérií.

Po implementácií som ukázal funkčnosť jednotlivých metód na základe experimentov. Experimenty ukazujú, že sa podarilo navrhnúť a úspešne implementovať systém, ktorý plánuje trasy vozidlám, ktoré zvážajú a rozvážajú veľkoobjemné kontajnery.

Rozšírenie tohto systému by mohlo byť spojením aplikacií systému do jednej, tj. webovú aplikáciu transformovať na desktopovú. Ďalšie rozšírenie by vzájomne ovládalo Ganttov diagram s mapou, kde zvolenú cestu v Ganttovom diagrame by aplikácia znázornila na mape. Tieto rozšírenia však neboli súčasťou požiadaviek na aplikačný systém.

Literatúra

[1] *An implementation of NSGA-II in Java.* [Online; navštívené 9.01.2019].
URL https://github.com/onclave/NSGA-II

[2] *Any R packages to solve Vehicle Routing Problem?* [Online; navštívené 9.01.2019].
URL https:
//www.r-bloggers.com/any-r-packages-to-solve-vehicle-routing-problem

[3] *Eulerovské a hamiltonovské grafy.* [Online; navštívené 9.01.2019].
URL https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=23498

[4] *Previous TSP Tours of the 48 States.* [Online; navštívené 9.01.2019].
URL
http://heraqi.blogspot.com/2014/12/np-problems-tutorial-with-coding.html

[5] *Vyhledávač sběrných středisek a kontejnerů.* [Online; navštívené 9.01.2019].
URL https://www.sako.cz/sberna-strediska-a-kontejnery/cz/

[6] Cormen, T.: *Introduction to algorithms.* Cambridge, Mass.: MIT Press, 2009, ISBN 978-0-262-03384-8.

[7] Dalton, J.: *Genetic Algorithms.* [Online; navštívené 9.01.2019].
URL http://www.edc.ncl.ac.uk/highlight/rhjanuary2007.php

[8] Deb, K.: *A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II.* [Online; navštívené 9.01.2019].
URL https://www.iitk.ac.in/kangal/Deb_NSGA-II.pdf

[9] Eppstein, D.: *The Traveling Salesman Problem for Cubic Graphs.* [Online; navštívené 9.01.2019].
URL http://jgaa.info/accepted/2007/Eppstein2007.11.1.pdf

[10] Erlebach, P.: *Descriptive Complexity.* FIT VUT v Brně, 2004, [Online; navštívené 9.01.2019].
URL http://www.fit.vutbr.cz/~meduna/mti/2004_05/erlebach.pdf

[11] Gencer, C.; Aydogan, E. K.: *Simultaneous Pick-up and Delivery Decision Support Systems.* [Online; navštívené 9.01.2019].
URL https://www.researchgate.net/figure/The-basic-problems-of-the-VRP-Mixed-pickups-and-deliveries-VRPPD-There-is-not_fig1_221908319

[12] Hesham, E.: *NP Problems Tutorial with Coding Examples.* [Online; navštívené 9.01.2019].

URL
http://heraqi.blogspot.com/2014/12/np-problems-tutorial-with-coding.html

[13] Immerman, N.: *Descriptive complexity*. Springer-Verlag New York, 1999, ISBN 03-879-8600-6.

[14] Kirk, J.: *Traveling Salesman Problem - Genetic Algorithm*. 2008, [Online; navštívené 9.01.2019].
URL https://www.mathworks.com/matlabcentral/fileexchange/13680-traveling-salesman-problem-genetic-algorithm

[15] Koza, J.: *Genetic programming: on the programming of computers by means of natural selection*. Cambridge, Mass.: MIT Press, 1992, ISBN 0262111705.

[16] Mitchel, M.: *An Introduction to Genetic Algorithms*. Cambridge: MIT Press, 2002, ISBN 02-626-3185-7.

[17] Newton, D.: *Learning Big O Notation With O(n) Complexity*. [Online; navštívené 9.01.2019].
URL
https://dzone.com/articles/learning-big-o-notation-with-on-complexity

[18] Popelka, O.: *Použití evolučních a genetických algoritmů v ekonomických aplikacích*. Dizertačná práca, Mendelova Univerzita, Brno, 2009.

[19] Reed, T.: *An introduction to algorithm design and structured programming*. New York: Prentice Hall, 1988, ISBN 0134777204.

[20] Sekaj, I.: *Evolučné výpočty a ich využitie v praxi*. Bratislava: Iris, 2005, ISBN 80-89018-87-4.

[21] Zelinka, I.; Oplátková, Z.; Šéda, M.; aj.: *Evoluční výpočetní techniky: principy a aplikace*. Praha: BEN, 2009, ISBN 978-80-7300-218-3.

[22] Zelinka, I.; Snášel, V.; Abraham, A.: *Handbook of Optimization. From Classical to Modern Approach*. Berlin, Springer-Verlag, 2013, ISBN 978-3-642-30-503-0.