Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Aplikace heuristiky SWARM pro úlohu VRP s časovými okny

Bc. Martin Klabeneš

Diplomová práce

2021

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2020/2021

ZADÁNÍ DIPLOMOVÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Martin Klabeneš**
Osobní číslo:	**I19282**
Studijní program:	**N0613A140007 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Aplikace heuristiky SWARM pro úlohu VRP s časovými okny**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

Při plánování rozvozu zboží z centrálních skladů vzniká logistická úloha VRP (Vehicle routing problem). Je třeba naplánovat nakládku zboží na vozidla a naplánovat rozvozní trasy s minimálními náklady. Tato úloha úzce souvisí s úlohou pro optimalizaci nakládky BPP (Bin Packing Problem) a okružní úlohou Travelling Salesman Problem (TSP).
Cílem práce je prostudovat různé přístupy pro řešení úlohy a naprogramovat řešení pro úlohu s více depy. Pro hledání řešení bude použita heuristika Particle Swarm Optimization. Důraz bude kladen na analýzu dopadů dělení zákazníků do sektorů na výslednou cenu dopravy a výpočetní náročnost. Pro dělení budou použity algoritmy Polar Region Partitioning, Rectangular Region Partitioning a metody shlukování. Zvolené optimalizační kritérium bude zahrnovat penalizaci nedodržení časových oken pro nakládku a vykládku zboží. Vytvořená aplikace umožní načítání dat s polohou zákazníků, výpočet matice vzdáleností a export výsledků (grafické znázornění linek, výčet vrcholů alokovaných na jednotlivé linky, doba jízdy).
V práci budou popsány veškeré aplikované metody a bude vysvětleno uživatelské prostředí aplikace.

Rozsah pracovní zprávy: **50-60**
Rozsah grafických prací: **5**
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

*BELMECHERI, Farah, Christian PRINS, Farouk YALAOUI, Lionel AMODEO. Particle swarm optimization algorithm for a vehicle routing problem with heterogeneous fleet, mixed backhauls, and time windows. J Intell Manuf 24, 2013, s. 775-789.
*SIMCHI-LEVI, David, Xin ChEN, Julien BRAMEL. The logic of logistics: theory, algorithms, and applications for logistics management. Third edition. New York: Springer, 2014. 447 s. ISBN 9781461491484.

Vedoucí diplomové práce: **Mgr. Jaroslav Marek, Ph.D.**
Katedra matematiky a fyziky

Datum zadání diplomové práce: **6. listopadu 2020**
Termín odevzdání diplomové práce: **15. května 2021**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

prof. Ing. Antonín Kavička, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 30. listopadu 2020

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 19.08.2021

Bc. Martin Klabeneš

PODĚKOVÁNÍ

Chtěl bych poděkovat Mgr. Jaroslavu Markovi, Ph.D. za příkladné vedení diplomové práce a poskytnuté materiály.

ANOTACE

Vehicle Routing Problem, do českého jazyka překládaný jako víceokruhový dopravní problém patří mezi skupinu logistických úloh, které se věnuje operační výzkum. Tyto úlohy jsou sice řešitelné metodami celočíselného programování, avšak jen pro úlohy malého rozsahu. Proto se místo tzv. exaktních metod používají heuristické algoritmy. V této práci bude cílem naprogramovat aplikací pro řešení VRP založenou na Particle Swarm optimalizaci. PSO je jednou z nejnovějších heuristik a její ideje jsou inspirovány chováním letících ptáků. Základem pro rozdělení zákazníků do sektorů obsluhovaných jedním vozidlem bude dělení do kruhových výsečí, tzv. Wheel algoritmus. Nejdůležitější části metodiky budou detailně popsány, resp. prezentovány jejich kódy a digramy tříd definované ve vývojovém prostředí IntelliJ IDEA a jazyku Java. K posouzení kvality vytvořené aplikace budou provedeny výpočty v Solomonových testovacích úlohách. Experimentální část umožní vybrat vhodné vstupní parametry algoritmu.

KLÍČOVÁ SLOVA

víceokruhový dopravní problém, problém obchodního cestujícího, optimalizace hejnem částic

TITLE

SWARM heuristic application for VRP with time windows

ANNOTATION

Vehicle Routing Problem belongs to the group of logistical problems devoted to operational research. Although these tasks are solvable by integer programming methods, they are only solvable for small-dimensional problems. Therefore, heuristic algorithms are used instead of so-called exact methods. In this work, the goal will be to programme an application for VRP solutions based on Particle Swarm optimization. PSO is one of the newest heuristics and its ideas are inspired by the behaviour of flying birds. The Wheel algorithm, which is based on dividing customers into arc slide sectors and serving these sector customers with one vehicle. The most important parts of the methodology will be described in detail or presented with their codes and class diagrams as defined in the IntelliJ IDEA development environment and Java language. To assess the quality of the application created, calculations will be performed on Solomon testing problems. This experimental part will also allow you to select the appropriate settings for the algorithm's input parameters.

KEYWORDS

Vehicle routing problem, Travelling Salesman Problem, Particle Swarm Optimization

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK

VRP	Vehicle Routing Problem
TSP	Travelling Salesman Problem
PSO	Particle Swarm Optimization
JRE	Java Runtime Environment
JDK	Java Development Kit
JVM	Java Virtual Machine

ÚVOD

Tato práce bude věnována heuristickému algoritmu Particle Swarm Optimization aplikovanému k řešení víceokruhové dopravní úlohy. Tento logistický problém je založen na rozdělení množiny zákazníků specifikovaných polohou a velikostí přepravovaného zboží tak, aby cena přepravy z depa jednotlivými vozidly byla nejmenší možná. V anglicky psané literatuře se úloha označuje jako Vehicle Routing Problem (VRP). Matematická úloha vede na celočíselné programování s omezujícími podmínkami, určenými kapacitou flotily vozidel atd. Další omezující podmínky vznikají v situaci, kdy musí být k zákazníkovi zboží dopraveno v předepsaném časovém okně. Po alokaci zákazníků na konkrétní vozidlo vzniká problém dopravního cestujícího, ve kterém je předmětem optimalizace délka trasy mezi všemi těmito zákazníky. Obě úlohy spadají do třídy superpolynomiálních algoritmů, kde složitost řešené úlohy exponenciálně roste s počtem zákazníků. Řešení těchto úloh založené na prohledání všech variant nelze dostatečně rychle získat v úlohách nad 15 zákazníků. Pro řešení úlohy se používají exaktní metody založené na lineárním programování, např. metoda Branch and Bound. Další možností je hledání řešení metodami umělé inteligence, např. genetické algoritmy, metoda mravenčí kolonie. V zadání práce byla specifikována metoda Particle Swarm Optimization, Tento heuristický algoritmus je inspirován chováním hejna ptáků. Při aplikaci PSO bude využita metoda Wheel, která s využitím polárních souřadnic zákazníky rozdělí do kruhových výsečí. Aplikace bude vytvořena v programovacím jazyku Java, ve vývojovém prostředí IntelliJ IDEA. V experimentální části bude využito 9 Solomonových testovacích úloh pro 25, 50 a 100 zákazníků. Sledován bude vliv počátečních parametrů algoritmu a počtu iterací na cenu a celkovou dobu výpočtu.

V první kapitole bude popsán studovaný dopravní problém a jeho různé varianty. Druhá kapitola bude zaměřena na popis základních myšlenek a pseudokódů heuristické metody Swarm. Tato metoda umožňuje získat přibližné řešení dané úlohy. Dále kapitola popisuje algoritmus Wheel – teoretický můstek pro propojení úlohy VRP a TSP – založenou na rozdělení zákazníků do sektorů. Je zde uvedena strategie prohazování tras. Ve třetí kapitole je popsána experimentální část. Tato část zjišťuje nejlepší hodnotu parametrů algoritmu PSO. Ve čtvrté kapitole je uveden popis vývojových technologií, programovacího jazyk, zálohování. Pátá kapitola prezentuje náhled na samotnou aplikaci a její strukturu. Závěrečné komentáře jsou uvedeny v šesté kapitole.

1 VEHICLE ROUTING PROBLEM (VRP)

Vehicle routing problem zahrnuje několik tříd logistických problémů. Poprvé byla úloha publikována v článku *The Truck Dispatching Problem* [8]. Předložený algoritmus byl aplikován při rozvozu benzínu.

Matematická formulace VRP je nejčastěji prezentována takto:

$$min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij}\, x_{ij}^{k} z_k.$$

za splnění omezujících podmínek uvedených v Tabulce 1,

Minimalizace se týká součtu ujetých vzdáleností c_{ij} se započtenými náklady na trasu. Některé symboly užité v Tabulce 1 jsou vysvětleny v Tabulce 2.

Tabulka 1: Matematická definice VRP [1]

$\sum_{\mathrm{k}\in\mathrm{K}} \sum_{\mathrm{j}:(\mathrm{i},\mathrm{j})\in\mathrm{A}} \mathrm{x}_{\mathrm{ij}}^{\mathrm{k}} = 1,$	$\forall i \in Dc \cup Pc$	Každá adresa je navštívena jedním vozem.
$\sum_{i:(0,i)\in A} x_{i0}^{k} = 1,$	$\forall k \in K$	Vozidlo začíná trasu v depu.
$\sum_{i:(i,n+1)\in A} x_{in+1}^{k} = 1,$	$\forall k \in K$	Vozidlo končí trasu v depu.
$\sum_{j:(j,i)\in A} x_{ji}^{k} = \sum_{j:(i,j)\in A} x_{ij}^{k},$	$\forall i \in Pc \cup Dc, \forall k \in K$	Zachování vzdáleností. Cesta z *i* do *j* je stejná jako z *j* do *i*.
$a_i \le T_i^k \le b_i,$	$\forall i \in V, \forall k \in K$	Splnění časových oken u vrcholu i.
$T_i^k + s_i + t_{ij} \le T_j^k + M\left(1 - x_{ij}^{k}\right),$	$\forall (i,j) \in A, \forall k \in K$	Kontrola dosažitelnosti časového okna následující zastávky.
$p_i \le L_i^k \le Q_k,$	$\forall i \in Pc, \forall k \in K$	První kapacitní omezení vozu a zásilky.

$0 \leq L_i^k \leq Q_k - d_i,$	$\forall i \in Dc, \forall k \in K$	Druhé kapacitní omezení vozu a zásilky.
$L_i^k + p_j - d_j \leq M(1 - x_{ij}^k),$	$\forall (i,j) \in A, \forall k \in K$	Kontrola, zda lze stihnou další zastávku i s dobou vykládky aktuální zastávky.
$L_0^k = \sum_{i \in Dc} d_i \sum_{j:(i,j)\in A} x_{ij}^k,$	$\forall k \in K$	Zaplnění vozu při výjezdu ze skladu musí odpovídat odběru zákazníků na trase.
$x_{ij}^k \in \{0,1\},$	$\forall (i,j) \in A, \forall k \in K$	Každá trasa smí být projeta pouze jednou.

Tabulka 2: Definice proměnných matematické definice VRP [1]

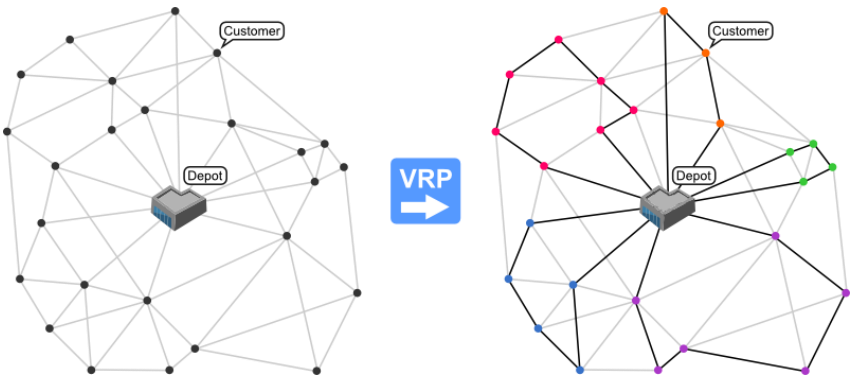
n	Počet adres.
$0, n+1$	Počáteční a koncová adresa (depo).
V	Seznam adres. $V = \{0, 1, 2, \dots, n, n+1\}.$
Dc	Seznam adres, u kterých dochází k doručení zásilky (linehaul).
Pc	Seznam adres, u kterých dochází k vyzvednutí zásilky (backhaul).
A	Seznam hran. $A = \{(i,j): i,j \in V, i \neq n+1, j \neq 0, i \neq j\}.$
a_i	Začátek časového okna na adrese i.
b_i	Konec časového okna na adrese i.
s_i	Trvání časového okna na adrese i.
d_i	Velikost doručované zásilky na adrese i. $i \in D_c, d_i = 0, \text{pokud } i \in P_c \cup \{0\} \cup \{n+1\}.$
p_i	Velikost vyzvedávané zásilky na adrese i.

	$i \in P_c, p_i = 0$, pokud $i \in D_c \cup \{0\} \cup \{n+1\}$.
t_{ij}	Doba cesty z adresy i na adresu j.
c_{ij}	Hodnota hrany (i,j). Např. cena nebo jízdy mezi i a j.
K	Kardinalita seznamu vozů.
Q_k	Kapacita vozu k.
z_k	Cena za ujetou vzdálenost vozu k.
M	Velká kladná konstanta, která je přidána z důvodu minimalizace flotily vozů.
$T_i^k \geq 0$	Začátek obsluhy vozidla k u zákazníka i.
$L_i^k \geq 0$	Doba vykládky vozu k u zákazníka i.
$x_{ij}^k \in \{0,1\}$	$x_{ij}^k = 1$, pokud je trasa (i,j) projeta vozem k. Pokud trasa projeta není, $x_{ij}^k = 0$.

Varianty VRP se liší dle parametrů, kapacitních omezení, časem dodání atd. Viz sekce 1.1.

VRP je náročnosti typu NP-hard. Výpočetní náročnost se exponenciálně zvyšuje s velikostí množiny zastávek, které je třeba projet. [27]

Výsledky heuristických metod jsou proto pouze více či méně kvalitní aproximace skutečného minima. Pro ověření kvality algoritmů a jejich implementace byly vytvořeny tzv. testovací úlohy, např. Solomonovy testovací úlohy. Viz kapitola 3.2.



Obrázek 1: Hledání trasy VRP [47]

1.1 TSP

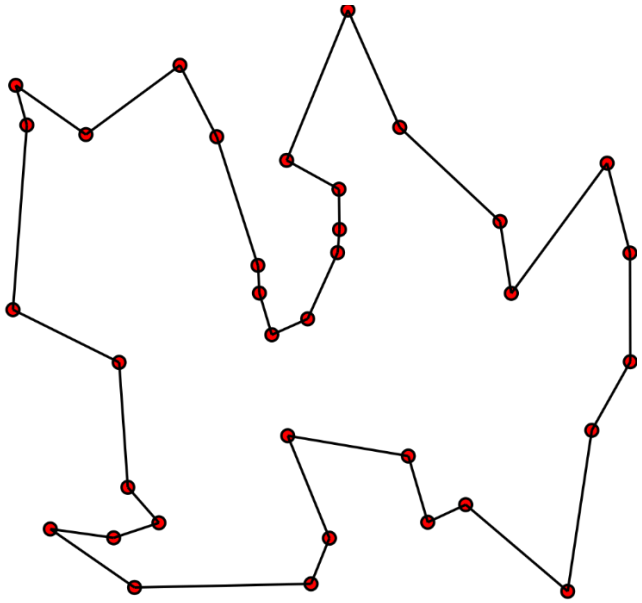
S úlohou VRP úzce souvisí problém obchodního cestujícího, anglicky TSP.

Cílem TSP je najít co možná nejkratší trasu přes množinu vrcholů. Každý z vrcholů této množiny smí být navštíven pouze jednou a zároveň žádný z vrcholů nesmí být vynechán. Žádný z vrcholů také nesmí být navštíven dvakrát. Po dosažení posledního nepřipojeného vrcholu je třeba vrátit se do vrcholu výchozího.

Pro řešení úlohy obchodního cestujícího (TSP) se užívají exaktní a heuristické metody. K nejznámějším metodám patří Clarke–Wrighteova metoda úspor (algoritmus saving), metoda Branch and bound, Branch nad Cut. Nejoblíbenější heuristické metody jsou:

- Metoda nejbližšího souseda (NNS),
- Simulované žíhání (SA),
- TABU search metoda (TS),
- Umělá neuronová sítě (NN),
- Optimalizace mravenčí kolonií (ACS),
- Genetické algoritmy (GA),
- a další [42].

Výsledné řešení by mělo vypadat například takto:



Obrázek 2: Vzor řešení TSP [50]

Problém TSP lze definovat jako graf $G = (V, E)$. Jedná o neorientovaný graf s vrcholy V a hranami E. Velikost množiny vrcholů je definována $|V| = n$.

Každá hrana z E má svou nenulovou délku. Délku hrany nejčastěji reprezentuje cenu dopravy mezi vrcholy s ní incidujícími. Velikost množiny hran tvořících trasu je definována jako:

$$|E| = \{(i,j): i,j \in V, i \neq j\},$$

přičemž:

- d_{ij} je vzdálenost z vrcholu i do vrcholu j,
- q_i je nezáporná kapacita vrcholu i,
- m počet vozů ve vozovém parku. [5]

1.2 Řešitelnost TSP

Řešení TSP je možné získat porovnáním ceny všech možných tras s různými pořadí míst, které jsou třeba navštívit. Množství tras, které je třeba projít, je definováno vzorcem:

$$\frac{(n-1)!}{2}.$$

Ze vzorce je patrné, že algoritmus je neefektivní již při malém množství vrcholů n. A při velkém počtu vrcholů je absolutně nepoužitelný v reálném čase. V případě, že bychom měli projet 20 měst, pak by počet všech možných řešení byl:

$$\frac{(n-1)!}{2} = 6.0822550204416 \times 10^{16}.$$

Pokud by generování všech možných tras bylo realizováno na stroji, který by měl vyhrazen pro výpočet 1GHz, tj. 1 000 000 000 operací za sekundu, pak by výpočet trval 703,96 dnů. [45]

Z tohoto hlediska lze brát problém TSP jako algoritmicky efektivně neřešitelný. Efektivní algoritmus pro TSP neexistuje. Z tohoto důvodu se užívají heuristické algoritmy, které nám poskytnou řešení, které se alespoň blíží k řešení optimálnímu. [45]

Níže lze vidět graf reálné doby výpočtu trasy oproti teoretickému výpočtu doby trasy. Porovnání probíhá dle vzorce:

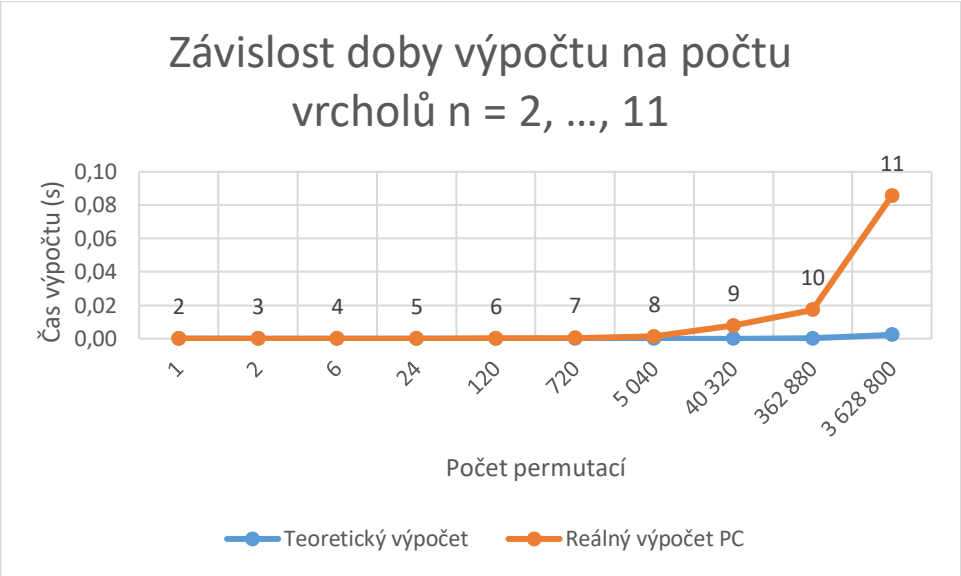
$$(n-1)!.$$

To znamená, že trasa je pro zjednodušení orientována. Z důvodů orientace tras je počet permutací dvojnásobný. Popisky nad body v grafu nám udává počet adres n.

Data ke grafu byla získána z aplikace vytvořené doc. Ing. Michaelem Bažantem, Ph.D. , kterou mi poskytl vedoucí této práce. Tento software provede všechny možné permutace množiny vrcholů a najde hrubou silou nejkratší možnou trasu v rámci TSP. Současně aplikace měří

dobu, kterou toto hledání trvá. Program jsem použil na notebooku značky Dell Vostro 5471 s následujícími parametry:

- procesor: Intel Core i5–8250U 1,6GHz,
- operační pamětí: 8 GB,
- operační systém: Windows 10 Pro – 64bit.



Graf 1: Porovnání teoretické a praktické doby výpočtu

Graf 1 znázorňuje rozdíly mezi teoretickou dobou výpočtu a dobou výpočtu na fyzickém stroji PC. Pro hodnoty, které byly získány výpočtem bylo dodatečně nutné započítat frekvenci procesoru aby bylo porovnání možné. Vzorec pro výpočet vypadal po započítání frekvence procesoru následovně:

$$((n-1)!)/\text{CPU\_Clock\_Rate}.$$

Výsledná hodnota je v jednotkách sekund. Data ke grafu jsou k dispozici v příloze D, viz str. 75.

1.2.1 Metoda nejbližšího souseda

Metoda nejbližšího souseda spočívá v tom, že z počátečního bodu vede trasa k nejbližšímu sousedovi. Tento krok je opakován, dokud nejsou navštíveny všechny vrcholy. Z posledního vrcholu následuje návrat do vrcholu výchozího. Popis návštěvy vrcholů lze definovat pomocí pseudokódu. [45]

Pseudokód:

Inicializace vrcholu v na náhodný vrchol.

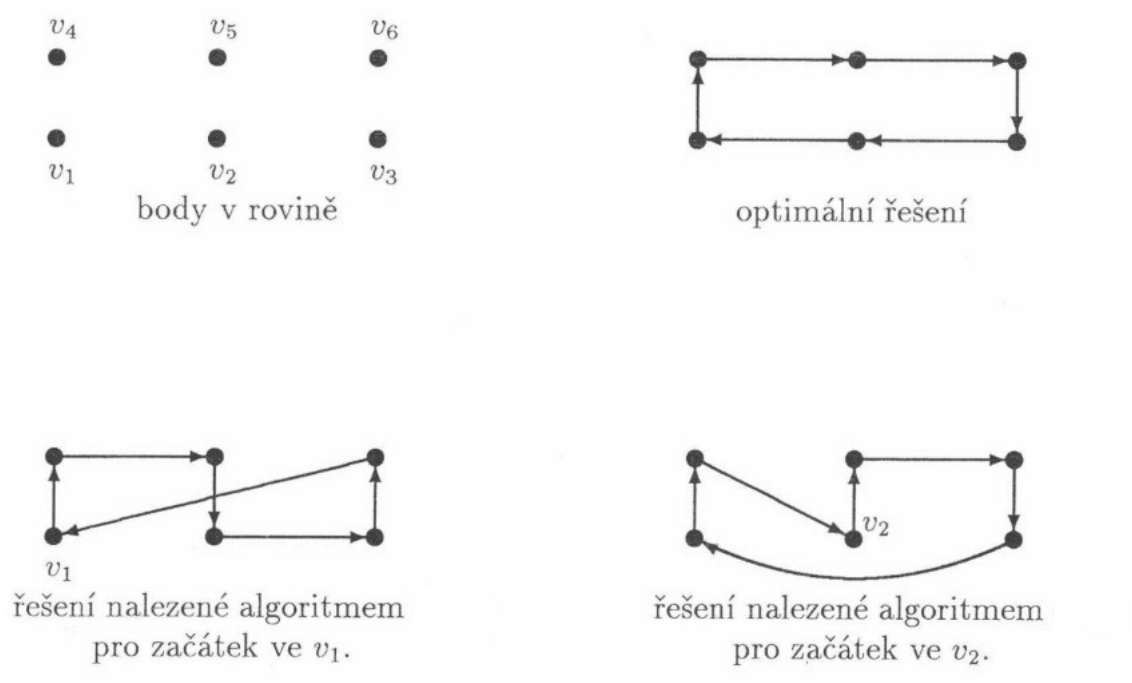
Inicializace seznamu vrcholů.

Cyklus, dokud existují nenavštívené vrcholy.

    Přidat vrchol v do seznamu vrcholů.

    Vrchol v je změněn na nejbližšího souseda aktuálního vrcholu.

Přidat vrchol v do seznamu vrcholů.



Obrázek 3: Příklad řešení TSP pomocí metody nejbližšího souseda [45]

Z Obrázku 3 je vidět, že metoda nejbližšího souseda není zcela efektivní. Důvodem může být:

- Špatně zvolený výchozí vrchol – obecně je relativně jedno, jaký výchozí vrchol si zvolíme.
- Problém exotických vrcholů – pokud máme více vzdálených vrcholů, tyto vrcholy zůstanou jako poslední. V tuto chvíli nastává problém dlouhých tras, které výrazně zvyšují celkovou délku trasy. [45]

1.3 Rozšíření TSP (GTSP)

Rozšíření TSP je velice jednoduché a zároveň velice praktické. Generalizované TSP, tedy GTSP, je rozšířen o rozdělení uzlů (vrcholů) do tzv. clusterů. Cílem je projít cluster s minimálními náklady průchodu. GTSP by bylo možné definovat jako jakýsi kombinatorický problém optimalizace. Tento pojem zavedli autoři Henry-Labordere, Saskena a Srivastava. [42]

Model GTSP nabízí přesnější model než klasický TSP. Obecně GTSP lze použít pro více skutečných problémů. Tento model byl navržen již v 60. letech 20. století. Avšak se moc neuchytil, a proto v rámci programovacích technik jsou problémy GTSP transformovány na TSP. Tato transformace na TSP je následně vyřešena pomocí již existujících algoritmů. Problémem transformace je výrazné zvýšení problematiky. [42]

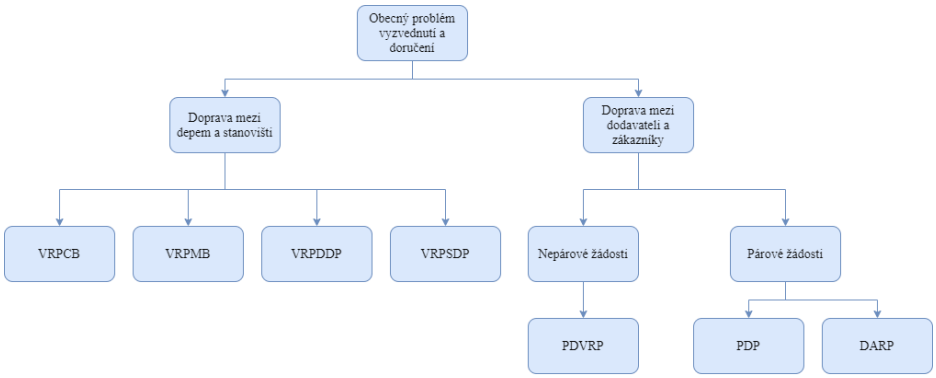
Tento problém se snažil řešit Wu a kol. navržením generalizovaného chromozomu (GC) a řešení GTSP pomocí generalizovaného genetického algoritmu založeném na chromozomech. Algoritmus GC lze využít pro sjednocení TSP a GTSP do jednoho modelu díky zavedení tzv. supervrcholu. Tento algoritmus obecně nezvětšuje řešený problém. [42]
Obecně lze problém VRP rozdělit na dvě podtřídy:

- doprava z depa k zákazníkům,
- doprava mezi dodavatelem a zákazníkem (bez depa)

Varianta dopravy z depa k zákazníkům je varianta, kdy je veškeré zboží naloženo v jednom nebo více skladech. Pokud by bylo zboží vyzvedáváno od zákazníků, tedy opačný proces k doručování, je třeba, aby veškeré zboží dopraveno do jednoho nebo více skladů. Nelze mít žádné zboží, které nebude doručeno. Problém této varianty je obvykle označována jako *Vehicle Routing Problem with Backhoul* (VRPB).

Ve variantě dopravy mezi dodavatelem a zákazníkem je úplně vypuštěna možnost depa. Jsou zde zahrnuty veškeré problémy, kdy zboží, případně cestující, je přepraveno mezi zákazníky. Problém této varianty je označován jako Vehicle Routing Problem s vyzvednutím a dodáním (VRPPD). [25], [38].

Souvislost problému VRP je naznačena v následujícím obrázku:



Obrázek 4: Rozdělení VRP [13]

Verze VRP pro dopravu mezi depem a stanovišti a verze pro dopravu mezi zákazníky jsou dále definovány dle modifikací.

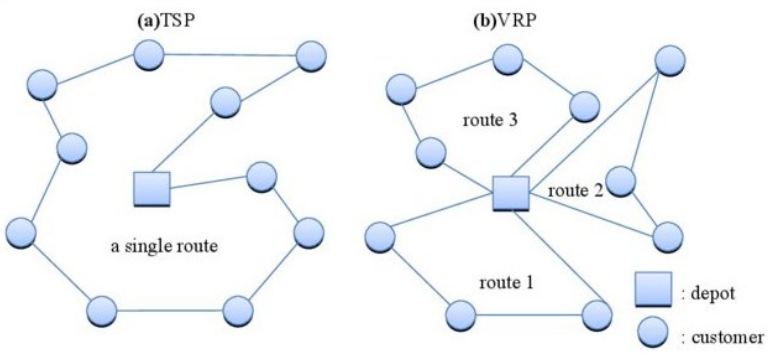
- Možnost VRP s depem,
  - VRPCB – Vehicle Routing Problem Clustered Backhauls,
  - VRPMB – VRP with Mixed linehauls and Backhauls,
  - VRPDDP – VRP with Divisible Delivery and Pick-up,
  - VRPSDP – VRP with Simultaneous Delivery and Pick-up.
- Možnost VRP mezi zákazníky,
  - PDVRP – Pick-up and Delivery Vehicle Routing Problem,
  - PDP – Pick-up and Delivery Problem,
    - SPDP – Single Pick-up and Delivery Problem (stejné jako PDP, pouze obsluha je jen jedním vozem).
  - DARP – Dial-A-Ride Problem,
    - SDARP – Single Dial-A-Ride Problem (stejné jako DARP, pouze obsluha je jen jedním vozem).

Další možnosti VRP jsou odvozeny od těchto, již existujících. Za zmínku určitě stojí:

- CVRP – Capacitated Vehicle Routing Problem – v podstatě stejný jako VRPCB,
- OVRP – Open Vehicle Routing Problem,
- SDVRP – Site-dependent Vehicle Routing Problem,
- MDVRP – Multi-depot Vehicle Routing Problem. [46]

1.4 Rozdíl mezi VRP a TSP

Obecně, pokud je v rámci VRP problém rozdělen na menší clustery, tyto clustery jsou řešeny jako samostatné problémy typu TSP.



Obrázek 5: Porovnání TSP a VRP [32]

1.5 Definice VRP

VRP lze popsat jako graf:

$$G = (V, E),$$

kde V je sada vrcholů $\{0, \dots, n\}$, n je počet zákazníků, které je třeba obsloužit. Depo se nachází na vrcholu 0, pokud uvažujeme variantu s depem. E je množina hran, které jsou neorientované. Každá hrana je definována jako:

$$(i, j) \in E, kde\ i \neq j,$$

její hodnota je nezáporná. [10]

Tato hodnota může být například délka mezi zákazníky i a j, nebo také rychlost projetí. [29]

1.6 VRP s depem

Pro řešení problému VRP s depem je nutné definovat depo. V tomto depu je více než 1 vozidlo. Dispečer vozového parku rozhoduje o tom, jaké vozidlo kam pojede, jaký náklad a kolik zásilek poveze. Základním problémem je tedy nalezení souborů vrcholů, které jsou následně propojeny s co nejkratší trasou. Vozidlo vyjíždí z depa, projede trasu a zase se vrátí zpět do depa. [43].

VRP s depem lze rozdělit do dalších 4. podtříd. První dvě podtřídy obsahují zákazníky buď pro dodání, nebo pro vyzvednutí zásilky, ale ne oba typy. Ve 3. a 4. podtřídě jsou definováni zákazníci, kteří při převzetí zásilky následně odesílají další svou zásilku. [25].

1.6.1 VRPCB

Tato modifikace je založena na kapacitním rozvozu, přičemž vozy mají heterogenní kapacitu. Je však rozšířena o možnost odběru od zákazníků a odvoz do depa. Jedná se tedy jak o dovoz, tak zároveň o odvoz zásilek. V odborně literatuře je tato modifikace označována jako *Vehicle Routing Problem with Clustered Backhauls*. Celková trasa je zde rozdělena na menší části – clustery – a ty jsou následně řešeny jako TSP. [12]

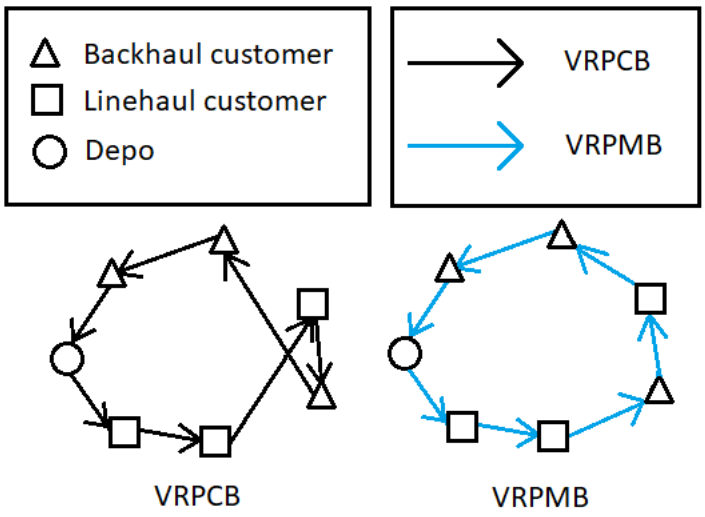
Tento problém lze definovat jako soubor zákazníků s deterministický požadavky na službu (vyzvednutí nebo dodání). Vozy sice mají heterogenní kapacitu, ale tato kapacita je fixní. Vozy začínají a končí v depu. Každý zákazník musí být přiřazen ke každému vozidlu právě jednou. Zákazníci jsou řazení do hlavních dvou skupin:

- zákazníci linehaulu – zákazníci, kteří čekají na dodání zásilky,
- zákazníci backhaulu – zákazníci, kteří čekají na vyzvednutí zásilky.

Cílem je minimalizovat celkovou trasovou vzdálenost.

VRPB lze ještě následně rozdělit na dvě podskupiny:

- VRPB s přesností zákazníka – zákazník linehaulu musí být v každé trase obsloužen přednostně oproti zákazníkům backhaulu,
- VRPB bez přednosti zákazníka – zákazníci linehaulu a backhaulu mohou být libovolně prokládáni. [44]



Obrázek 6: Rozdíl mezi VRPCB a VRPMB [31]

1.6.2 VRPMB

Modifikace VRPMB je varianta VRP, kdy je kombinováno doručování zásilek k zákazníkům z centrálního skladu (tzv. linehaulu zákazníci) a zákazníky, kteří čekají na vyzvednutí zásilky a dovezení této zásilky na centrální depo (tzv. backhaulu zákazníci). Tato modifikace je často využívána ještě s přidanou modifikací časových oken (Time Windows). V reálném světě se s tímto způsobem doručování často setkáváme u doručovacích firem, např. PPL. [2]

1.6.3 VRPDDP

Tato modifikace je v odborné literatuře pojmenována jako *VRP with Divisible Deliveries and Pickups*. Tato varianta není příliš známá. Ve této variantě VRP je skupina zákazníků, kteří žádají o dodávku a/nebo vyzvednutí. K dispozici je park homogenních vozidel. Tyto vozidla jsou umístěna pouze v jednom depu. Veškeré zboží, které by bylo dopravováno pouze ze skladu k zákazníkům nebo naopak od zákazníků do skladu. Každé vozidlo má omezenou kapacitu, stejně tak jako předchozí modifikace. Zásilka nesmí být rozdělena, tj. pokud je zákazníkovy doručována zásilka, musí být vždy doručena celá najednou. V případě, že jeden zákazník chce zásilku odeslat a zároveň mu má zásilka přijít, může být obsloužen dvěma na sobě nezávislými vozy. Cílem je najít soubor tras, které uspokojí poptávku zákazníků, nepřekročí kapacitní omezení automobilů a zároveň je snaha zajistit minimální celkovou ujetou vzdálenost. [36]

1.6.4 VRPSDP

V odborné literatuře je tato varianta VRP označována jako *VRP with Simultaneous Deliveries and Pickups.* Ve této variantě VRP je skupina zákazníků, kteří žádají o dodávku a/nebo vyzvednutí zásilky. K dispozici je park homogenních vozidel a pouze jedno depo. Veškeré zboží, které by bylo dopravováno pouze ze skladu k zákazníkům nebo naopak od zákazníků do skladu. Každé vozidlo má omezenou kapacitu, stejně tak jako předchozí varianty VRP. Zásilka nesmí být rozdělena, tj. pokud je zákazníkovy doručována zásilka, musí být vždy doručena celá najednou. Oproti předchozí variante VRP se tato varianta liší v tom, že jeden zákazník chce zásilku odeslat a zároveň mu má zásilka přijít, nesmí být obsloužen dvěma na sobě nezávislými vozy. Musí být obsloužen pouze jedním vozem. Cílem je najít soubor tras, které uspokojí poptávku zákazníků, nepřekročí kapacitní omezení automobilů a zároveň je snaha zajistit minimální celkovou ujetou vzdálenost. [36]

1.7 VRP bez depa

Tato část se zaměřuje na přepravu zboží mezi místy vyzvednutí a místy dodání. Tento způsob dopravy se zcela vyhne využití centrálního skladu, depa. Problémy tohoto typu jsou označovány jako VRP s vyzvednutím a dodáním (VRPPD). [38]

1.7.1 PDVRP

Jedná se typ VRP, u kterého se nenachází depo. V této modifikaci dochází k doručování vzájemně mezi zákazníky, tj. každý vrchol v grafu může být obsloužen jiným vrcholem. Každý vrchol smí být projet pouze jednou. Všechny zásilky jsou stejné, tudíž velikost zásilky není pro PDVRP důležitá. Zásilky jsou vyzvedávány cestou od zákazníků a následným pokračováním v trase doručovány. [11]

PDVRP naráží na tři problémy při doručování:

- každá dvojice, tj. odesilatel i adresát musejí být přiřazeni ke stejnému vozidlu,
- nelze zásilku doručit dříve, než by byla vyzvednuta doručovatelem (nelze navštívit dříve adresáta než odesilatele),
- hmotnost vozidla se dynamicky mění průjezdem zastávek (během celé trasy nelze porušit kapacitní omezení vozu). [35]

V odborné literatuře je tato modifikace pojmenována jako *Pick-up and Delivery Vehicle Routing Problem.* [11], [34]

1.7.2 PDP

Stejně tak jako u předchozí modifikace i tato nepoužívá centrální depo. Doručení probíhá výhradně mezi zákazníky (vrcholy v grafu). Doručování probíhá přímo. Dochází zde ke spárování odesílatele a adresáta. Následně jsou tyto zastávky projety po těchto párech. TSP pro tuto modifikaci lze popsat jako standardní TSP, kde každý pár je brán jako samostatný vrchol TSP. Mezi obsluhou každého odesilatele a adresáta nelze cestou obsloužit další zastávky mimo tento pár. [38]

PDP je založeno na rozdělení dvou služeb, vyzvednutí a doručení. PDP může být dále rozšířeno o další parametry, jako je:

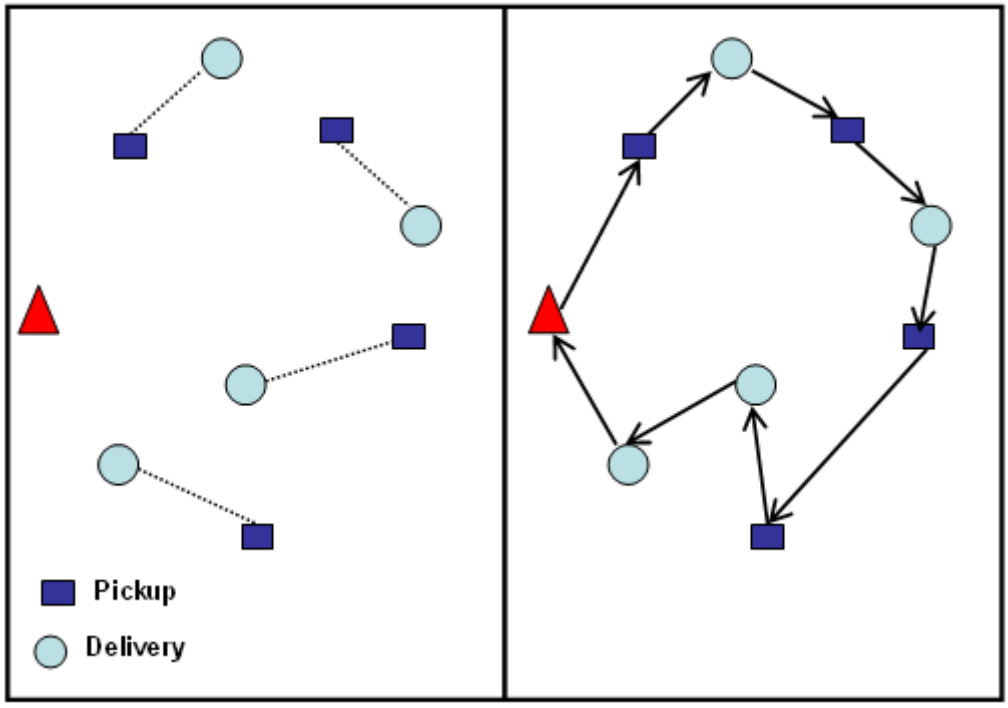
- místo původu/určení,

- převoz jedné nebo více zásilek,
- možnost párování odesilatele a zákazníka,
- převoz zboží nebo lidí.

Dále je potom možné uplatnit další omezení, stejně jako u původního VRP:

- kapacita vozu,
- délka provozu.

V odborné literatuře lze tento problém nalézt pod názvem *Pick-up and Delivery Problem.*



Obrázek 7: Zobrazení PDP [15]

1.7.3 DARP

Tato modifikace nevyužívá depo. Doručování probíhá mezi zákazníky. Tato metoda spočívá v navržení trasy pro *n* zákazníků, kteří specifikují požadavky na vyzvednutí a doručení mezi místem naložení a místem vyložení. Tuto variantu VRP je možné přirovnat k dopravě taxi. Často se stává, že jeden uživatel má za jeden den dvě žádosti:

- Odjezd z místa bydliště do námi zvolené destinace.
- Odjezd z destinace do místa bydliště (tedy zpět).

Cílem je naplánovat všechny trasy s co možná nejmenšími náklady. Zároveň je třeba vyhovět co největšímu počtu žádostí v rámci souboru omezení. Při přepravě cestujících je třeba myslet na snížené pohodlí cestujících kvůli minimalizaci nákladů na přepravu. Kapacita vozidel v rámci DARP bývá velice omezena, zatímco v rámci PDVRP je naopak nadbytečná. Modifikace DARP modelově použitelná spíše pro přepravu lidí, zatímco modifikace PDVRP je spíše užita ke sběru listovních zásilek nebo menších balíčků. [6]

1.8 Kapacitní VRP

Jeden ze základních rozšíření problému VRP je přidání kapacity vozu. Kapacita všech vozů ve vozovém parku je stejná, ale omezená. Vozidla jsou umístěna v depu.

1.8.1 Kapacitní VRP se stejnými kapacitními požadavky

Cílem je zase projet podskupinu zastávek a zároveň neporušit kapacitní omezení vozidla, tj. vozidlo nesmí být přetíženo. V případě, že by byla zásilka příliš velká, nebo se do vozu z nějakého důvodu nevejde, lze uvažovat nad rozdělením zásilky na menší části. Pokud je však zásilka rozdělena na menší části, je třeba aby tato zastávka byla projeta více vozidly a zásilka u adresáta zkompletována. Tento model nazývá CVRP s rozdělitelnými požadavky nebo ECVRP. Pokud bychom depo označili jako x_0 a množinu zákazníků $N = \{x_1, x_2, \dots, x_n\}$, potom soubor $N_0 \doteq N \cup \{x_0\}$ nám představuje sjednocené množiny depa a zákazníků. Zákazníci a depo jsou součástí ohodnoceného grafu jako jeho vrcholy. Graf zapisujeme takto:

$$G = (N_0, E).$$

Následně je zde definována vzdálenost od depa k zákazníkovy. Tuto vzdálenost zapisujeme jako d_i. Vzdálenost k nejvzdálenějšímu zákazníkovy zapíšeme jako:

$$d_{max} \doteq max_{i \in N}\, d_i.$$

Vzdálenost mezi dvěma zákazníky (zákazníkem i a zákazníkem j) bychom tedy zapsali jako d_{ij}. Matice vzdáleností takového grafu $\{\mathrm{d}_{ij}\}$ splňuje trojúhelníkovou nerovnost a zároveň je symetrická. Platí že $d_{ij} = d_{ji}$ a také platí $d_{ij} \leq d_{ik} + d_{jk}$ pro všechna i, j, k. [43]

1.8.2 Kapacitní VRP s rozdílnými kapacitními požadavky

Tato modifikace VRP je zapisována jako UCVRP. V této modifikaci má každý zákazník i balíček o velikosti w_i. Kapacitní omezení nám stanovuje, že maximální dodané množství zboží

jedním vozem nesmí překročit Q. Nechť Z_u^* je označení optimální hodnotu řešení UCVRP, tedy minimální ujetá vzdálenost všech vozů. V této verzi VRP balíček od zákazníka nelze rozdělit na více vozidel, to znamená, že každému zákazníkovi bude přiděleno pouze jedno vozidlo. Rozdělení balíčků je dost často velice složité a mnohdy i nereálné. Problematice řešení tohoto problému bylo vyvinuto několik heuristik. Dle Christofidese [30] jsou tyto heuristiky zařazeny do 4 kategorií:

- konstruktivní metody:

jedna z nejrannějších heuristik. Pokud máme depo a n zákazníků, přidělíme každému zákazníkovi jedno vozidlo a sledujeme vzdálenosti jednotlivých cest, tj. di. Tyto cesty je nutno projet dvakrát, do depa a zpět, projetá vzdálenost je tedy $2d_i$. Celková projetá vzdálenost v grafu je zapsána jako:

$$2\sum_{i=1}^{n} d_i.$$

Následné spojování cest. Cesty se spojují dle vzorce:

$$s_{ij} = 2d_i + 2d_j - (d_i + d_j + d_{ij}) = d_i + d_j - d_{ij}.$$

Čím menší je hodnota s_{ij}, tím více je pravděpodobné, že tyto dva vrcholy budou spojeny.

- route-first–cluster-second metoda:

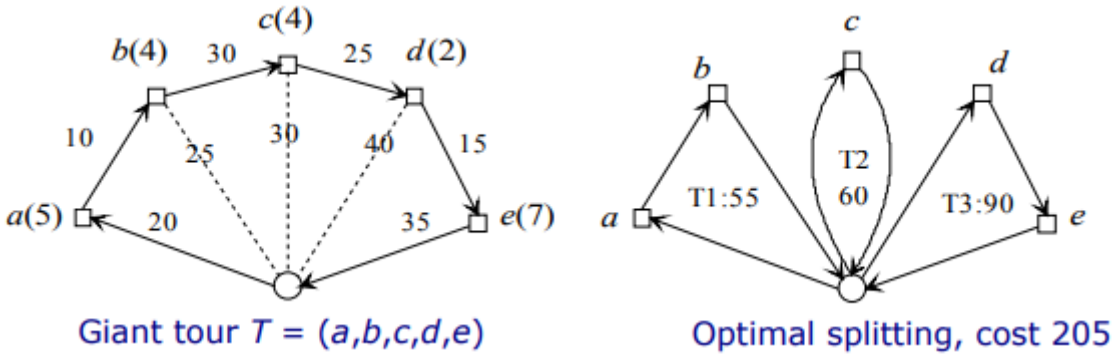
sestavení takzvané Giant Tour (TSP) a následné rozdělení dle proveditelnosti do clusterů. Jedno vozidlo je přiřazeno ke každému clusteru a navštěvuje každého zákazníka tak jak jdou po sobě v trase TSP.

- cluster-first–route-second metoda:

sestavení clusterů pro klienty a následné řešení TSP na každý cluster. Tato heuristika je obvykle více technicky propracována, neboť určování seskupení je založeno na matematicky programovacím přístupu.

- neúplně optimalizační metody:

tyto metody vyžadují vysokou dobu výpočetního času a z tohoto důvodu jsou často i předčasně ukončeny. Většinou se nepoužívají, pokud má problém VRP více než 100 zákazníků. [43]



Obrázek 8: Dělení VRP [39]

1.8.3 VRP s omezením časového okna

Tato modifikace VRP nám ukazuje, že je možné využít také jiné náklady, které se netýkají kapacity nebo obecně ceny zásilky. Každý zákazník má nějaké časové okno, ve kterém je třeba doručit objednanou zásilku. Cílem je tedy najít trasu takovou, aby byly doručeny všechny zásilky a zároveň dodržena časová okna pro doručení. I v této modifikaci je depo a trasa každého vozu v depu začíná a také končí. Tomuto modelu se říká VRPTW nebo také CVRPTW. Tento problém je často zkoumán empirickou analýzou, ale jen velmi málo je zkoumán z analytického hlediska. Je to z toho důvodu, že je snaha popsat teoretické chování heuristiky. Toto chování je následně využito k vytvoření účinných algoritmů pro vyhledávání optimální trasy. [43]

2 PARTICLE SWARM OPTIMIZATION PRO ŘEŠENÍ VRP

Algoritmus optimalizace hejnem částic (PSO), poprvé popsán autory Kennedym a Eberhartem v článku [25]. Tato metoda optimalizace je založena na společenském chování hejn ptáků nebo hejn ryb. Tento algoritmus se nápadně podobá algoritmům genetickým (GA). S genetickými algoritmy má společné hledání optima pomocí předků a následníků – generací. Každá generace začíná na již nejlépe nalezeném řešení. Z tohoto bodu je hledáno následné řešení. Pokud je řešení lepší než řešení předka, řešení potomka je zapsáno jako nejlepší aktuálně nalezené řešení. Z tohoto řešení je výchozí pro další generaci. [42]

Počáteční rozdělení tras v systému je zcela náhodné. Systém si vytvoří náhodné, však kompletní řešení trasy. Trasa však není optimalizována, jedná se pouze o náhodnou trasu, která má křížení tras a propojení bodů (zastávek) je zcela náhodné. Následně probíhá iterativní hledání nejlepšího možného řešení (optima). [42]

V porovnání s GA je PSO založen na zcela jiném algoritmu. Tento algoritmus je výrazně propracovanější než GA. Z tohoto důvodu je PSO vhodný nejen pro výzkum, ale také pro inženýrské aplikace. [42]

2.1 Diskrétní algoritmus PSO pro problém TSP

Předpokládejme, že prohledávaný prostor je *D*-dimenzionální a je tvořen hejnem o velikosti *m*. Každá dimenze *D* je klasifikován vektorem:

$$X_i(i = 1, 2, \dots, m).$$

Tento fakt znamená, že hejna jsou lokalizována v námi prohledávaném prostoru jako:

$$X_i = (x_{i1}, x_{i2}, \dots, x_{iD}), \quad (\mathrm{i} = 1,2, \dots, \mathrm{m}).$$

Poloha každé částice je teoreticky možné řešení. Pokud je hodnota fitness vyšší, korespondující X_i je lepší. Pohybující se částice mají svou velikost. Tyto velikosti jsou předepsány *D*-dimenzionálním vektorem:

$$V_i = (v_{i1}, v_{i2}, \dots v_{iD}), \quad (i = 1,2, \dots, m).$$

Nejlepší pozice celého hejna je popsána jako:

$$P_i = \left(p_{g1}, p_{g2}, \dots, p_{gD}\right).$$

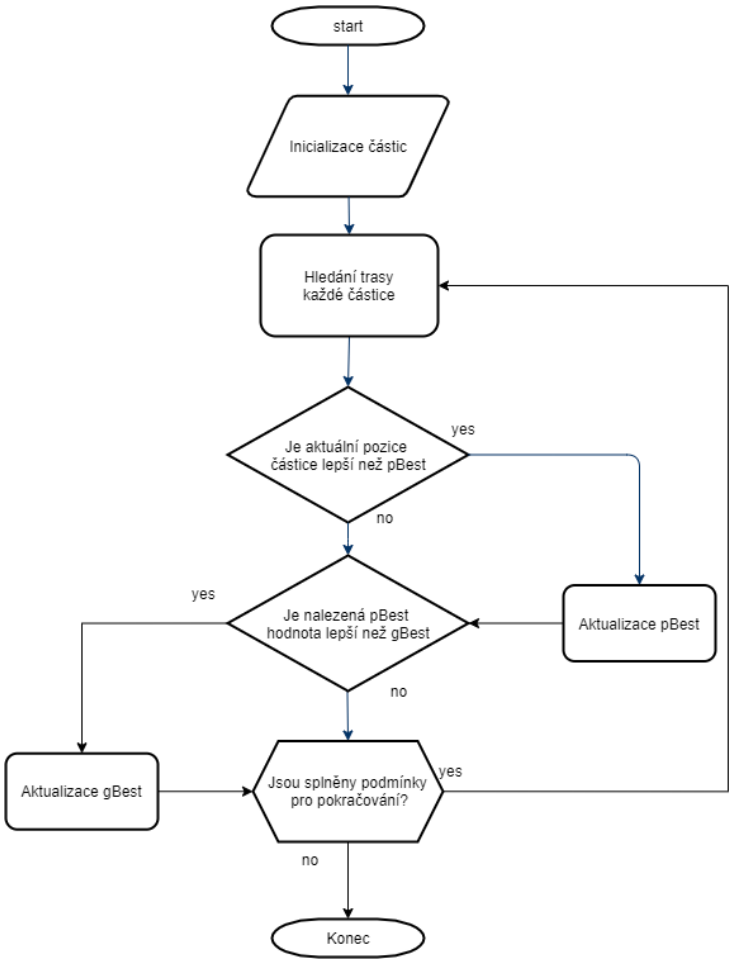
Výpočet trasy pro každou částici je vypočítávána podle následujícího vzorce.

$$V_p^t = \omega V_p^{t-1} + C_p rand_p\left(pBest_p - Y_p^{t-1}\right) + C_g rand_g(gBest - Y_p^{t-1}).$$

Tato vypočítaná trasa je následně připočtena k aktuální pozici. Tímto způsobem je vypočítána pozice nová:

$$Y_p^1 = Y_p^{t-1} + V_p^1.$$

Vývojový diagram algoritmu vypadá následovně:



Obrázek 9: Vývojový diagram algoritmu PSO

2.1.1 Parametry

V algoritmu PSO budeme užívat parametry uvedené v Tabulce 3.

Tabulka 3: Parametry PSO [1]

T	Počet iterací.
P	Počet částic.
D	Velikost dimenze.

	$D = n$, dimenze je rovna počtu adres.
t	Index iterace: $t = \{1, 2, \dots, T\}$.
p	Index částice: $p = \{1, 2, \dots, P\}$.
d	Index dimenze: $d = \{1, 2, \dots, D\}$.
$v_{dp}(t)$	Rychlost částice p v dimenzi d a iteraci t.
$V_p(t)$	Vektor rychlosti částice indexované p v iteraci t: $\left[v_{1p}(t) v_{2p}(t) \dots v_{Dp}(t)\right]$.
$y_{dp}(t)$	Pozice částice indexované p v dimezi D a v iteraci t.
$Y_p(t)$	Vektor pozic částice p v iteraci t: $\left[y_{1p}(t) y_{2p}(t) \dots y_{Dp}(t)\right]$.
$pbest_{dp}$	Lokální nejlepší nalezená hodnota částice p v dimenzi d.
$PBEST_p$	Vektor lokálních nejlepších nalezených hodnot částice s indexem p: $\left[pbest_{1p}\ pbest_{2p} \dots\ pbest_{Dp}\right]$ v dimenzi d.
$gbest_d$	Nejlepší nalezená pozice částic $p = \{1, 2, \dots, P\}$ v dimenzi D.
$GBEST$	Vektor celkové nejlepší nalezené pozice částic: $p = \{1, 2, \dots, P\}$: $[gbest_1\ gbest_2 \dots\ gbest_D]$.
$Sol\left(Y_p(t)\right)$	Nejlepší nalezený vektor rychlosti částice p v iteraci: t: $\left[v_{1p}(t) v_{2p}(t) \dots v_{Dp}(t)\right]$.
$Sol(PBEST_p)$	Nejlepší nalezený vektor lokálních nejlepších nalezených hodnot částice p: $\left[pbest_{1p}\ pbest_{2p} \dots\ pbest_{Dp}\right]$ v dimezi D.
$Sol(GBEST)$	Nejlepší nalezený vektor celkové nejlepší nalezené pozice částic: $p = \{1, 2, \dots, P\}$: $[gbest_1\ gbest_2 \dots\ gbest_D]$.
w	Hmotnost částice p. Jedná se o koeficient setrvačnosti, který je konstantou v intervalu $[0,1]$.
C_g	Parametr vyhledávání ovlivňující $GBEST$ pozici. Jedná se o reálné nezáporné číslo.

C_p	Parametr vyhledávání ovlivňující $PBEST_p$ pozici. Jedná se o reálné nezáporné číslo.
$ymin$	Minimální pozice.
$ymax$	Maximální pozice.
$rnd(RA, RB)$	Náhodné číslo z uniformního rozdělení v rozsahu od RA do RB.
$rand_p$	Náhodné číslo generované v intervalu [0,1].
$rand_g$	Náhodné číslo generované v intervalu [0,1].

2.1.2 Pseudokód

1. Inicializace parametrů $w, C_g, C_p, ymin, ymax$.
2. Inicializace $Sol(PBESTp)$ a $Sol(GBEST)$ na nejlepší dosavadní nalezenou hodnotu.
3. Vytvoření listu LAD (seznam adres).
4. Nastavení parametru $t = 1$.
5. Cyklus pro $d = \{1, \dots, D\}$.
    a. Inicializace vektoru:

    $$V_p(t) = \left[v_{1p}(t)\ v_{2p}(t) \dots\ v_{Dp}(t)\right] = 0, \quad \forall p \in P.$$

6. Cyklus pro $d = \{1, \dots, D\}$:
    a. Inicializace vektoru pozic na náhodné hodnoty v rozsahu $[D, 2 * D]$:

    $$Y_p(t) = \left[y_{1p}(t)\ y_{2p}(t) \dots\ y_{Dp}(t)\right], \quad \forall p\ \in\ P.$$

7. Naplnění seznamu LAD hodnotami vygenerovaných v krocích 5 a 6.
8. Vytvoření listu LPC.
9. Výpočet $Sol\left(Y_p(t)\right), \quad \forall p\ \in\ P.$
10. Cyklus pro $d = \{1, \dots, D\}$:
    a. Aktualizace hodnot:

    $$PBEST_p = Y_p(t), \quad \forall p\ \in\ P.$$

11. Cyklus pro $d = \{1, \dots, D\}$:
    a. Pokud $Sol(GBEST) > PBEST_p$.
        i. $GBEST = PBEST_p, \quad \forall p \in P.$
12. Cyklus pro $t = \{2, \dots, T\}$:
    a. Cyklus pro $p = \{1, \dots, P\}$:
        i. $V_p(t) = w * V_p(t-1) + C_p * rnd_p * PBEST_p + C_g * rnd_g * GBEST.$
        ii. $Y_p(t) = Y_p(t-1) + Y_p(t).$
        iii. Aktualizace listu LPC.
        iv. Výpočet $Sol\left(Y_p(t)\right), \quad \forall p \in P.$
        v. Cyklus pro $d = \{1, \dots, D\}$:
            1. Pokud: $Sol\left(PBEST_p\right) > Sol\left(Y_p(t)\right).$
                a. $PBEST_p = Y_p(t), \quad \forall p \in P.$
        vi. Cyklus pro $d = \{1, \dots, D\}$:
            1. Pokud: $Sol(GBEST) > PBEST_p.$
                a. $GBEST = \left(Y_p(t)\right), \quad \forall p \in P.$

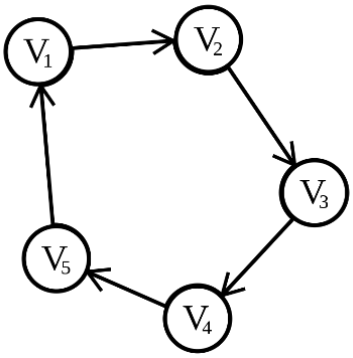
2.2 Strategie při konstrukci řešení

V této sekci budou popsány některé užitečné nástroje a strategie, které v iteračním algoritmu PSO umožňují nalézt počáteční rozdělení zákazníků do sektorů, resp. vylepšení řešení.

2.2.1 Odstranění křížení tras

Při hledání řešení TSP se stává, že trasy jsou kříženy. Obecně lze říct, že pokud máme minimální počet parametrů, trasy prohledávání by se neměly křížit. Pokud se trasy kříží, nebylo nalezeno řešení, které se blíží optimu.

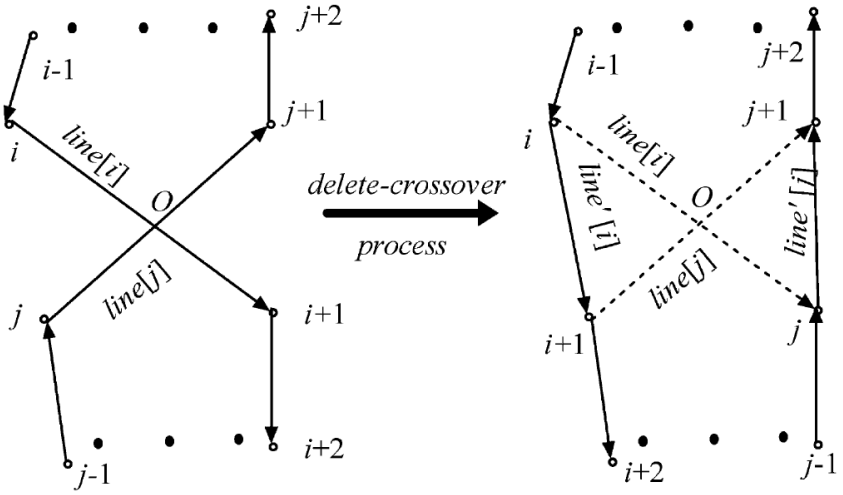
Optimální řešení by měl být tedy kružnice, tj. graf, ve kterém se žádné dvě hrany nekříží. [1]



Obrázek 10: Vzor trasy bez křížení[26]

Řešení takového problému je prohození hran. Pokud je prohození možné, měla by se zmenšit celková délka trasy TSP. Tato metoda úpravy trasy TSP je velice užitečná ke zkrácení vyhledávání optimální trasy. Zároveň je zde snaha odstranění uvíznutí v lokálním extrému. [1]

Tato funkce je použitelná pouze pro hledání trasy bez časových oken. Časová okna lze řešit v následujících funkcích.



Obrázek 11: Popis prohození tras [42]

2.2.2 Strategie dělení zakázek do sektorů

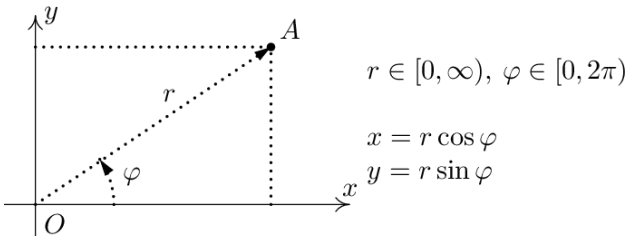
Pro roztřídění a přiřazení zákazníků k jednotlivým vozidlům se ukázalo jako vhodné dělení zákazníků do sektorů tvořených kruhovými výsečemi (Polar Region Partitioning), když dělení do sektorů tvořených obdélníky (Rectangular Region Partitioning) dosahovala horší výsledky, viz [43]. Proto pro roztřídění zákazníků použijeme Wheel algoritmus, který je založen na polárních souřadnicích.

Polární souřadnice

Polární souřadnice jsou transformací souřadnic *x* a *y*, které pocházejí z kartézského souřadnicového systému. Oba systémy jsou charakterizovány dvojicí hodnot v 2D rovině. Rozdíl mezi těmito souřadnicovými systémy je ten, že polární souřadnice vycházejí ortogonální soustavy a kartézské souřadnice vycházejí z kartézské soustavy. Polární souřadnice popisují polohu bodů v rovině v závislosti na poloze počátečního bodu P. Bod P je tzv. pevný bod, od kterého se odvíjí výpočet souřadnic dalších bodů. Bodem P prochází polopřímka, tzv. polární osa.

Polární souřadnice každého bodu jsou definovány dvěma parametry:

- r – je definováno jako vzdálenost bodu P a bodu X, tzv. polární poloměr,
- φ – je definován jako úhel s polární osou, bodem P, bodem X. Úhel je nazýván jako tzv. polární úhel. Hodnota úhlu φ závisí na použitém vzorci pro výpočet.



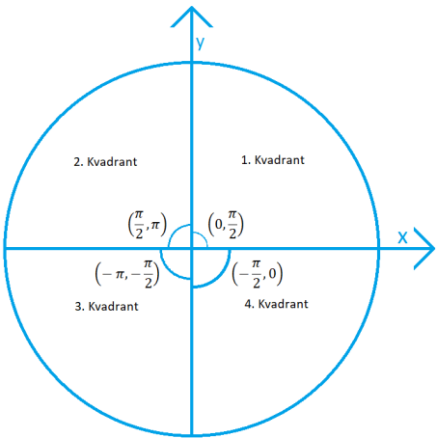
Obrázek 12: Ukázka transformace polárních souřadnic [33]

Převod kartézských souřadnic do polárních je definováno vztahem:

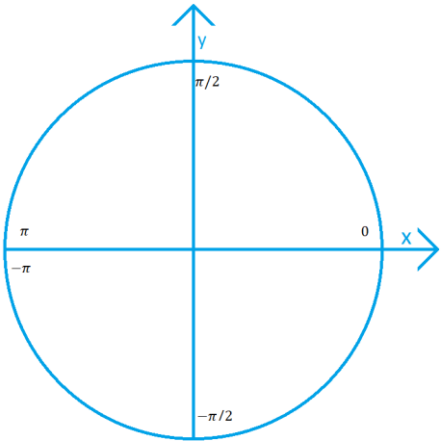
$$\mathrm{r} = \sqrt{x^2 + y^2}\,,$$

$$\varphi = \arctan\left(\frac{y}{x}\right).$$

Tento zápis výpočtu φ je definována pouze pro interval $\varphi \in \langle 0, \frac{\pi}{2} \rangle$. Výpočet v tomto intervalu nám udává souřadnice v rámci prvního kvadrantu v polárním souřadnicovém systému.



Obrázek 13: Rozdělení dle kvadrantů



Obrázek 14: Rozdělení kvadrantů dle π [13]

Výsledek je udáván v radiánech, které jsou posunuty v rámci definičního oboru z původních $\langle 0, 2\pi \rangle$ na definiční obor $\langle -\pi, \pi \rangle$. V případě potřeby přepočtu jsou vzorce následující:

$$stupně = radiány \times 180° / \pi,$$

$$radiány = stupně \times \pi / 180\,°.$$

Převod polárních souřadnic do kartézských je definováno vztahem:

$$x = \mathrm{r} \times \cos(\varphi),$$

$$\mathrm{y} = \mathrm{r} \times \sin(\varphi),$$

[40], [7], [14]

2.3 Popis metody Wheel

Pro každého zákazníka reprezentovaného dvojicí kartézských souřadnic x a y spočteme polární úhel φ a polární vzdálenost r. Zákazníky seřadíme vzestupně (nebo sestupně) podle polárního úhlu do předem připraveného seznamu. Ze seřazeného seznamu vezmeme prvního zákazníka a kontrolujeme, zda námi vybrané vozidlo má dostatečnou kapacitu kapacitou. Pokud vozidlo splňuje požadovanou podmínku, tomuto vozidlu je přiřazena aktuální adresa. Tuto adresu následně odebereme ze seznamu. Pokud již je kapacita vozu naplněna, vybere se další vůz a plnění pokračuje, dokud adresy nedojdou. Výhoda této metody je rychlost. Vhledem k tomu, že procházíme lineární seznam, časová složitost tohoto algoritmu je $O(n)$. Celkově tato metoda výrazně urychluje celé prohledávání, jelikož je zde použita teorie rozděl a panuj.

3 VÝSLEDKY EXPERIMENTÁLNÍ ČÁSTI

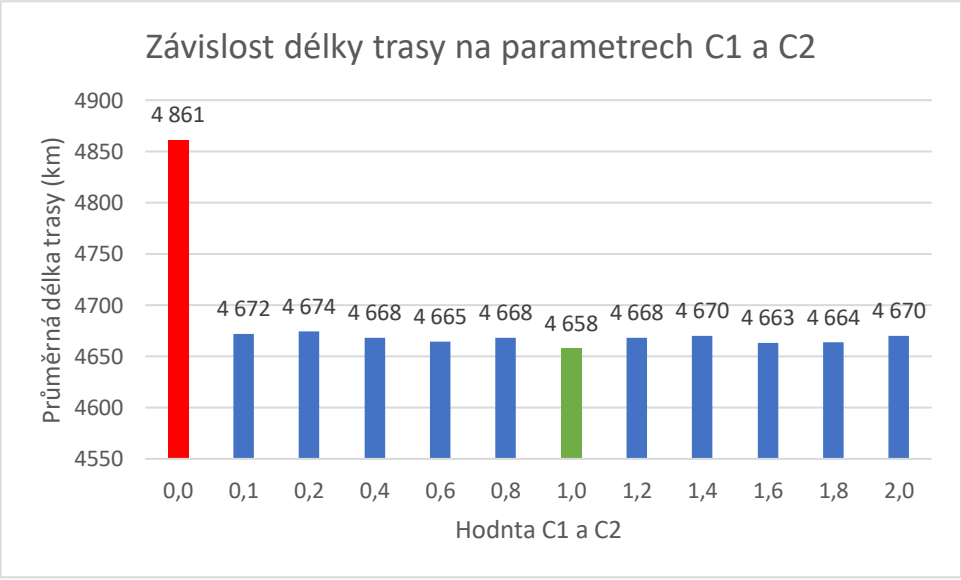
V této části práce je kladen důraz na ovlivnění výsledku při změně parametrů pro vyhledávání trasy. Dále je zde otestování aplikace na testovacích úlohách a porovnání s předem známými výsledky.

3.1 Nastavení parametrů algoritmu PSO

V aplikaci lze v rámci kódu nastavit parametry C1, C2. Tyto parametry jsou definovány jako konstanty, které určují chování k nejlepší lokální a globální hodnotě. Dále lze nastavit počet částic, které prostor prohledávají. Posledním parametrem, které je možné nastavit je počet iterací, které musí algoritmus PSO projít, než vyhledávání nejlepší trasy skončí. V Grafu 2 a Grafu 3 lze vidět změnu parametrů a jejich vliv na vyhledávání nejkratší trasy. Data v Grafu 2 jsou průměrem z 60 nezávislých prohledávání algoritmu PSO.

3.1.1 Nastavení parametrů C1a C2

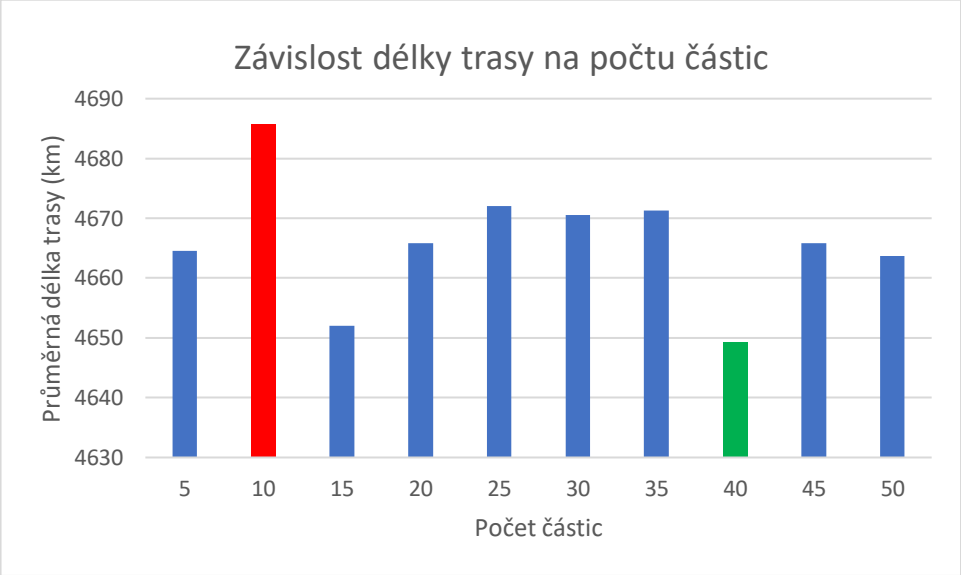
V Grafu 2 můžeme vidět, že nejhorší možnou hodnotou pro parametry C1 a C2 je hodnota 0. Při této hodnotě algoritmus prohledával prostor s velkými odchylkami. Nejlepší se ukázalo být nastavení parametrů C1 a C2 na hodnotu 1. Parametry počet iterací a počet částic je nastaven na hodnotu 20.



Graf 2: Závislost vyhledávání na parametrech C1 a C2

3.1.2 Nastavení parametru počtu částic

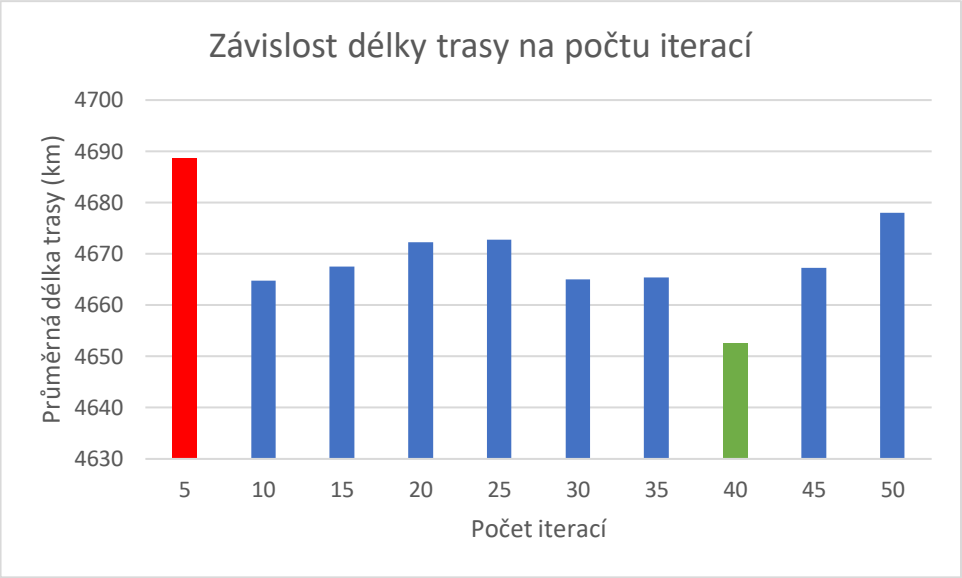
V Grafu 3 můžeme vidět, že nejlépe zvolený parametr je dle grafu hodnota parametru částic 40. Je také však vidět, že i při hodnotě 15 částic bylo dosaženo velice dobrých výsledků. Je však nutné upozornit, že s čím vyšším počtem částic aplikace operuje, tím více se zvyšuje časová náročnost. Další parametry byly nastaveny následovně. C1 a C2 byly nastaveny na hodnotu 1,0 a počet iterací na hodnotu 20.



Graf 3:Závislost vyhledávání na počtu částic

3.1.3 Nastavení parametru iterací

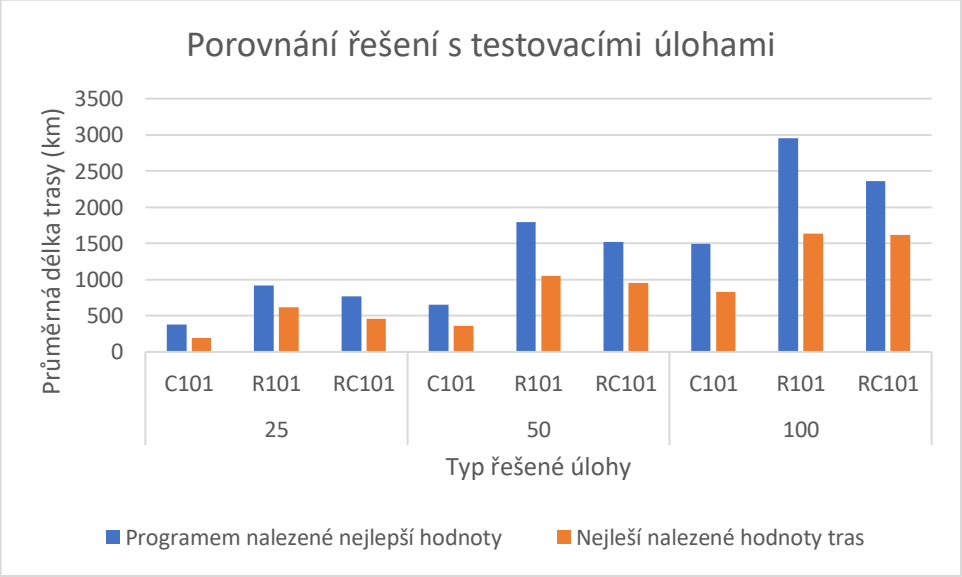
V Grafu 4 můžeme vidět, že nejlepší hodnotou je počet iterací 40. Dále již pravděpodobně nemá smysl zvyšovat počet iterací, jelikož částice, které prostor prohledávali, již našli nějaké optimum (ať lokální nebo globální) a tudíž zvyšující se počet iterací již výsledek nezmění. Další parametry byly nastaveny následovně. C1 a C2 byly nastaveny na hodnotu 1,0 a počet částic na hodnotu 20.



Graf 4:Závislost vyhledávání na počtu iterací

3.2 Porovnání ceny na Solomonových úlohách

Tato část práce je věnována použití testovacích úloh. Tyto úlohy jsou nazývány Solomonovy testovací úlohy. Úlohy byly použity z důvodu, že je již známa jejich nejlepší dosud nalezená hodnota. Z tohoto důvodu jsou tyto úlohy vhodné pro testování. V následném Grafu 5 je vidět hodnota, kterou našel program oproti nejlepší hodnotě Solomonovy úlohy. Data k tomuto grafu se nacházejí v příloze F, viz str. 81.



Graf 5: Porovnání výsledků vyhledávání se Solomonovými úlohami

4 VYUŽITÉ VÝVOJOVÉ TECHNOLOGIE

Program byl psán ve vývojovém prostředí IntelliJ IDEA 2021.2 (Ultimate Edition). Pro vývoj byl využit jazyk Java 8 a s tím související JavaFX pro grafické zobrazení. JavaFX je součástí JDK 8. Dále byl pro vývoj využit nástroj pro správu, řízení a automatizaci buildů – Maven. Celý program byl zálohován pomocí distribuovaného systému správy verzí – GitHub. Dokumentace je vygenerována pomocí generátoru technické dokumentace – Doxygen 1.9.1.

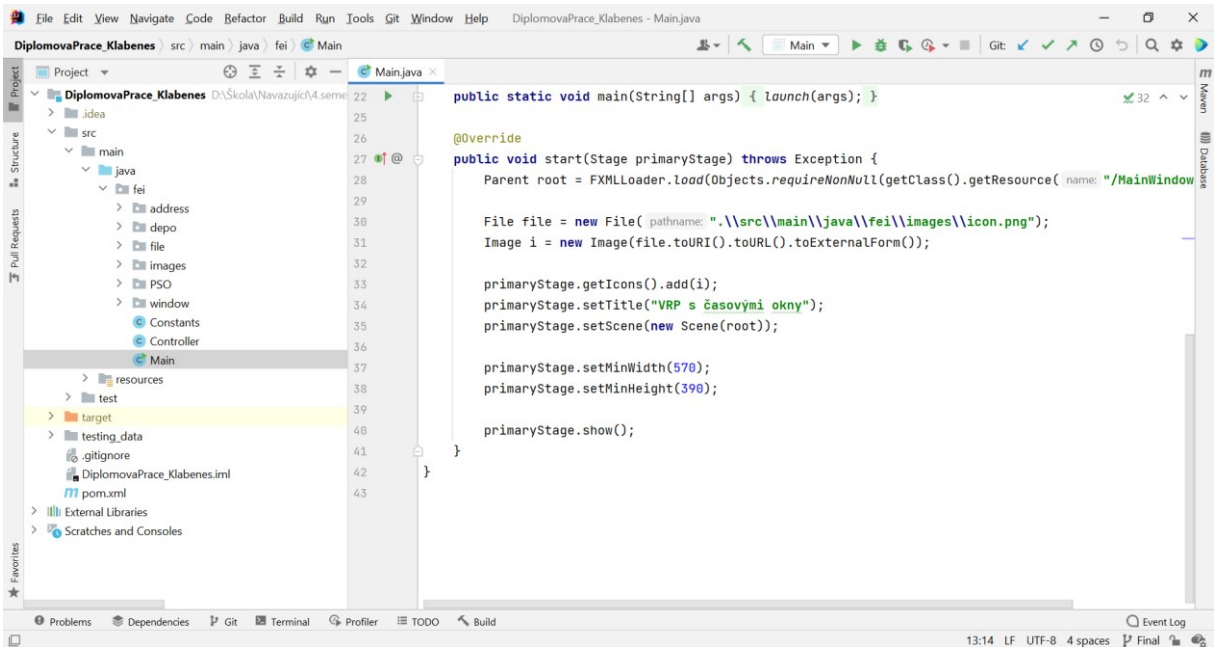
4.1 Vývojové prostředí

Praktická část diplomové práce je psána ve vývojovém prostředí IntelliJ IDEA od společnosti JetBrains. IntelliJ IDEA je vývojové prostředí uzpůsobené jazykům běžících na JVM. Samotné vývojové prostředí je multiplatformní, to znamená, že toto vývojovém prostředí lze spustit nejen na operačním systému Windows, ale také na systémech macOS a Linux. [17]

IntelliJ IDEA je dostupná ve dvou variantách:

- Community Edition – jedná se o bezplatnou verzi vývojového prostředí. Kód této verze je Open-source, tj. zcela dostupný a upravitelný. Kód je dostupný pod licencí Apache 2.0. Ve verzi Community Edition je možné využívat veškeré základní funkcionality pro vývoj aplikací JVM a android.
- IntelliJ IDEA Ultimate – jedná se o komerční distribuci. Je zde možnost využití 30 dní zkušební lhůty. V této zkušební lhůtě je vývojové prostředí zcela bez omezení. Oproti Community Edition verzi jsou zde navíc dostupné nástroje pro webové aplikace a podnikový vývoj. [16]

Vývojové prostředí má přehledně strukturovaný balíčkovací systém a přístup k verzovacím systémům. Balíčkovací systém lze vidět na Obrázku 15 po levé straně.



Obrázek 15: Ukázka vývojového prostředí

4.1.1 Systémové požadavky

K využívání vývojového prostředí není třeba instalovat Javu. JetBrains Runtime, který je součásti vývojového prostředí, je založen na JRE 11. Je tedy jen požadována instalace JDK. Další požadavky na hardware a operační systém jsou sepsány níže. Je zde i vyjádření k 32bitové verzi vývojového prostředí. Tato zmínka je zde spíše doplňující, neboť JetBrains oficiálně nikdy 32bitové verze IntelliJ IDEA nepodporoval. [24]

Tabulka 4: Systémové požadavky IntelliJ IDEA [16]

Požadavek	**Minimální**	**Doporučeno**
RAM	2 GB.	8 GB.
Procesor	CPU 64bitové architektury (neoficiálně i 32bitové, ale vývoj těchto vývojových prostředí je již ukončen).	Vícejádrový procesor. IntelliJ IDEA podporuje multithreading.

Místo na disku	2,5 GB a další 1 GB mezipaměti.	Disk typu SSD a alespoň 5 GB volného místa.
Rozlišení monitoru	102 x 768.	1920 × 1080.
Operační systém	Oficiálně veškeré 64bitové systémy, které byly vydány po vydání vývojového prostředí, tj: ❖ Microsoft Windows 8, ❖ macOS 10.13, ❖ Linuxu s podporou Gnome, KDE nebo Unity DE.	Nejnovější 64bitová verze Windows, macOS nebo Linux.

4.1.2 Programovací jazyky ve vývojovém prostředí

Ve vývojovém prostředí IntelliJ IDEA lze vyvíjet a překládat aplikace do bytekódu pro JVM v následujících jazycích:

- Java,
- Kotlin,
- Scala,
- Groovy. [17]

Další jazyky lze podporovat po přidání pluginu, který je součástí IntelliJ IDEA. Tyto pluginy jsou zabaleny, aby zbytečně nenavyšovali velikost a náročnost vývojového prostředí. V případě nutnosti není problém plugin rozbalit a nainstalovat. Jazyky jsou následně podporovány, ale pro plný vývoj v těchto jazycích je doporučeno nainstalovat vývojová prostředí k tomu přizpůsobených. Jazyky, které jsou po doinstalování pluginu podporovány jsou následující:

- Python (vývojové prostředí PyCharm),
- Ruby (vývojové prostředí RubyMine),
- PHP (vývojové prostředí PhpStorm),
- SQL (vývojové prostředí DataGrip),
- Go (vývojové prostředí GoLand),
- JavaScript (vývojové prostředí WebStorm),
- TypeScript (vývojové prostředí WebStorm),

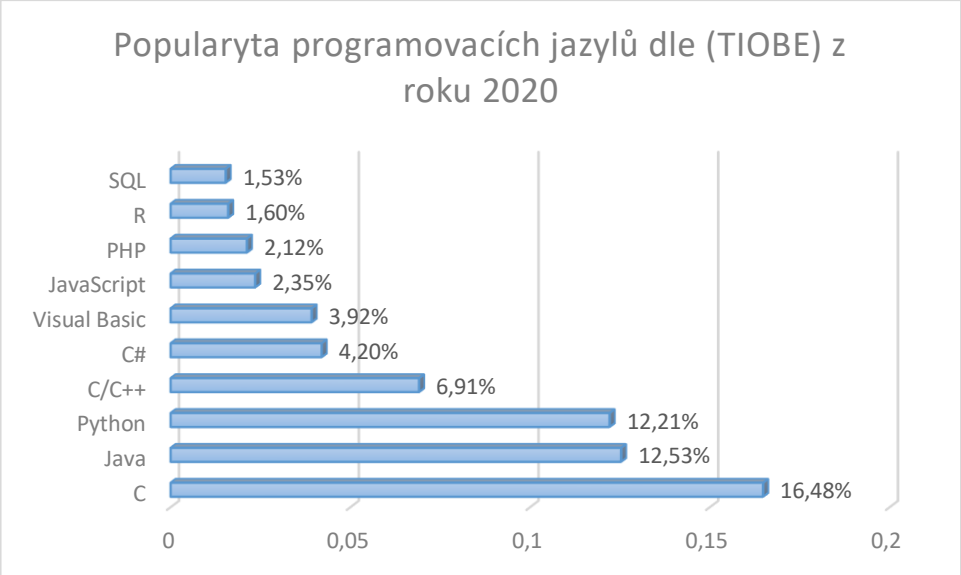
- JSON,
- HTML and XHTML,
- Stylesheets (CSS, Less, Sass),
- a dalších 7. [17]

4.2 Programovací jazyk Java

Program byl napsán v programovacím jazyce Java. Jazyk Java vyvinula společnost Sun Microsystems. Představen byl již v roce 1995. Java je jeden z nejoblíbenějších programovacích jazyků na celém světě. Aktuálně je platforma Java vlastněna společností Oracle, která se sloučila původní společnost Sun Microsystems.

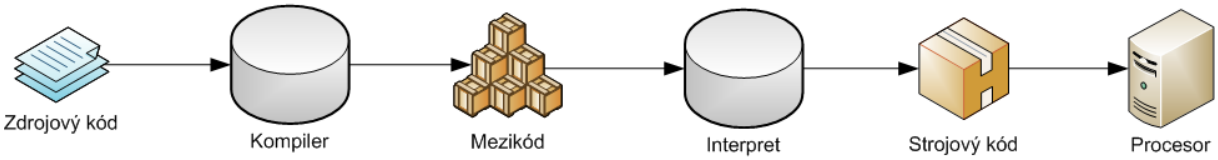
Jazyk Java je objektově orientovaný programovací jazyk, stejně jako:

- C++,
- C#,
- PHP,
- Python,
- Ruby,
- Go,
- atd.



Graf 6: Popularita jazyků [28]

Jazyk Java patří mezi programovací jazyky, které ke kompilaci využívají virtuální stroj. Do kategorie jazyků využívajících virtuální stroj spadá také jazyk C#. Zdrojový kód napsaný v jazyce Java je nejprve přeložen do mezikódu. Tomuto přeloženému mezikódu se říká bytekód. Jedná se o binární kód s jednodušší instrukční sadou a přímou podporou objektového programování. Tento mezikód je následně zpracován pomocí virtuálního stroje (např. JVM). Výstupní strojový kód je připravený ke zpracování samotným procesorem. [3]



Obrázek 16: Zpracování kódu Java [3]

Výhody tohoto přístupu jsou:

1. Nalezení chyb – překompilací do bytekódu lze relativně jednoduše nalézt chyby v kódu zdrojovém.
2. Stabilita – díky interpretu lze jsme upozorněni na chyby či nebezpečné operace.
3. Jednoduchý vývoj – jsou zde možnosti využít již vytvořené datové struktury, knihovní funkce. Není třeba ani hlídat uvolňování pamětí, neboť za správu paměti zodpovídá garbage collector.
4. Rychlost – první zpuštění bývá delší než u nižších programovacích jazyků. Je to z důvodu, že při první kompilaci je třeba nalinkovat knihovny. Výhodou však je, že virtuální stroj dokáže cachovat a následně sám optimalizovat. Tudíž další zpuštění jsou mnohem rychlejší a je teoreticky možnost dosáhnout rychlosti kompilace pomocí kompilátoru.
5. Zranitelnost – aplikace se šíří v podobě bytekódu, a tedy nejsou zcela dobře lidsky čitelné jako v kód zdrojovém.
6. Přenositelnost – program je možné spustit na jakémkoliv stroji, který má přístup k virtuálnímu stroji, např. JVM. [3]

4.2.1 Platformové edice Java

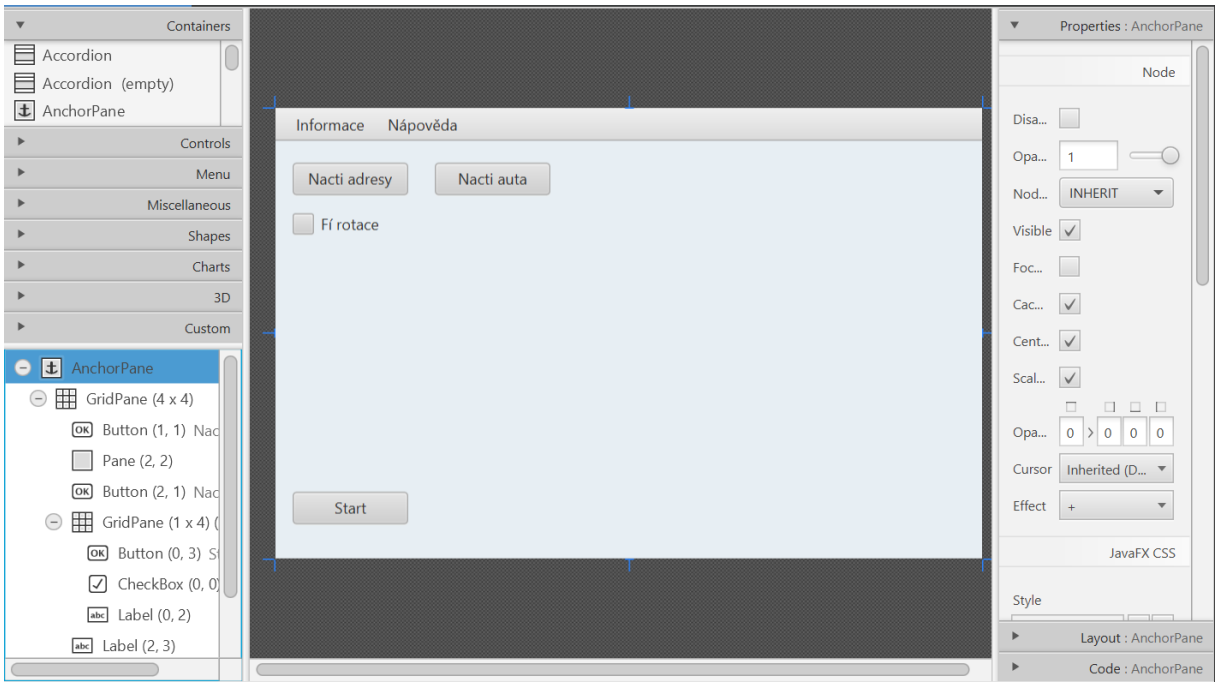
Jazyk Java má více edicí. Každá edice je vhodná k jinému využití. Edice platforem Java jsou následující:

- Java Card – jedná se o platformu, která využívá bezpečnostních zařízení jako jsou čipové karty, bezpečnostní čipy apod. Lze těmito tyto zařízení hostovat aplikace vytvořené v jazyce Java. Díky této platformě lze na jednom zařízení bezpečně spustit více aplikací. Aktuální verze Java Card je 3.1. [18]
- Java ME – Java Micro Edition je prostředí pro aplikace, které běží na vestavěných a mobilních zařízeních. Příkladem těchto zařízení jsou např. mikrořadiče, senzory, mobilní telefony, televizní přijímače top boxy, tiskárny a další. Java ME obsahuje uživatelská rozhraní. Z bezpečnostní stránky je Java ME dobře zabezpečena. Dokáže komunikovat přes síťové protokoly a podporuje síťové i offline aplikace. Aplikace založené na Java ME jsou dobře přenositelné. Aktuální verze je 8.3. [20]
- Java SE – Java Standard Edition je prostředí pro aplikace, které jsou vyvíjeny a nasazovány na stolních počítačích či serverech. Java SE má taktéž uživatelské rozhraní, vysoké zabezpečení a rychlost. Samozřejmě neopomenutelnou součástí je přenositelnost mezi operačními systémy. Aktuální verze je 16.0.2. [21]
- Java EE – Java Enterprise Edition je prostředí pro aplikace, která jsou vyvíjena a provozována jako podnikové aplikace a informační systémy. Jedná se o nadstavbu Java SE díky sadě knihoven určených k vytváření webových aplikací. Aktuální verze je 8. [3], [19]

4.2.2 JavaFX

JavaFX je open source platforma pro vytváření grafického výstupu pro PC, mobilní telefony a systémy vytvořené v jazyku Java. Jedná se moderní a efektivní sadu nástrojů pro vývoj klientských aplikací. Vytvoření grafického rozhraní lze vytvořit dvěma způsoby: [23]

1. Scene Builder – je to nástroj pro vizualizaci a rozvržení grafického výstupu. Vytváření vizualizace přetahováním komponent do pracovní oblasti. Dále je zde možnost upravovat vlastnosti a styly komponent. Přidáním komponent vzniká v přidruženém souboru FXML kód, která je využit jako zdrojový při vizualizaci. Tento soubor FXML se následně propojuje s projektem Java na samotnou logiku aplikace. Scene builder po otevření vypadá takto: [22], [41]



Obrázek 17: Ukázka Scene builderu

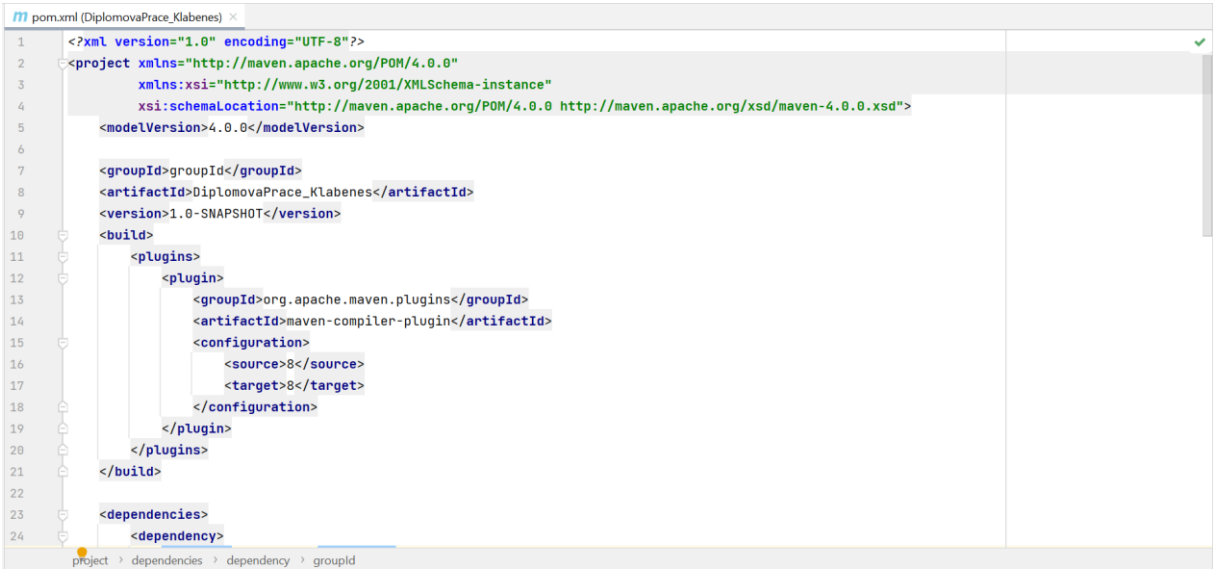
2. Soubor FXML – FXML samotný je jazyk pro vytváření formulářů. Tento jazyk je odvozený od jazyku XML. Java, stejně jako C#, pracuje s principy přenosu standardů HTML a CSS do desktopových aplikací. FXML souboru vypadá takto: [4]

Obrázek 18: Ukázka FXML souboru

4.2.3 Apache Maven

Další použitou technologií pro vývoj aplikace byl použit nástroj Apache Maven. Na základě objektového projektového modelu (POM) spravovat, řídit a automatizovat sestavování buildů aplikace. Tento nástroj je podporovaný více jazyky. Především je však podporován jazykem

Java. Maven je v podstatě správa knihoven a automatické stažení těchto knihoven při potřebě jejich použití – nalinkování. Struktura souboru POM je psána v jazyku XML. Jedná se o standardní soubor XML, který využívá komponenty jazyka JavaFX. Struktura souboru pom.xml vypadá následovně: [48]



Obrázek 19: Ukázka POM souboru

4.3 GitHub

Další použitý nástroj pro tvorbu aplikace je GitHub. Jedná se o webovou službu verzování open – source projektů. Dříve šlo ukládat projekty zdarma jen pokud byli veřejně přístupné. Od roku 2019 lze ukládat repositáře i soukromě. Vlastníkem této webové služby je Microsoft. První zpuštění proběhlo již v roce 2008.

Přístup ke službě GitHub je možný třemi způsoby:

- přes webové rozhraní,
- přes desktopovou aplikaci,
- přes textové uživatelské rozhraní – příkazový řádek. [49]

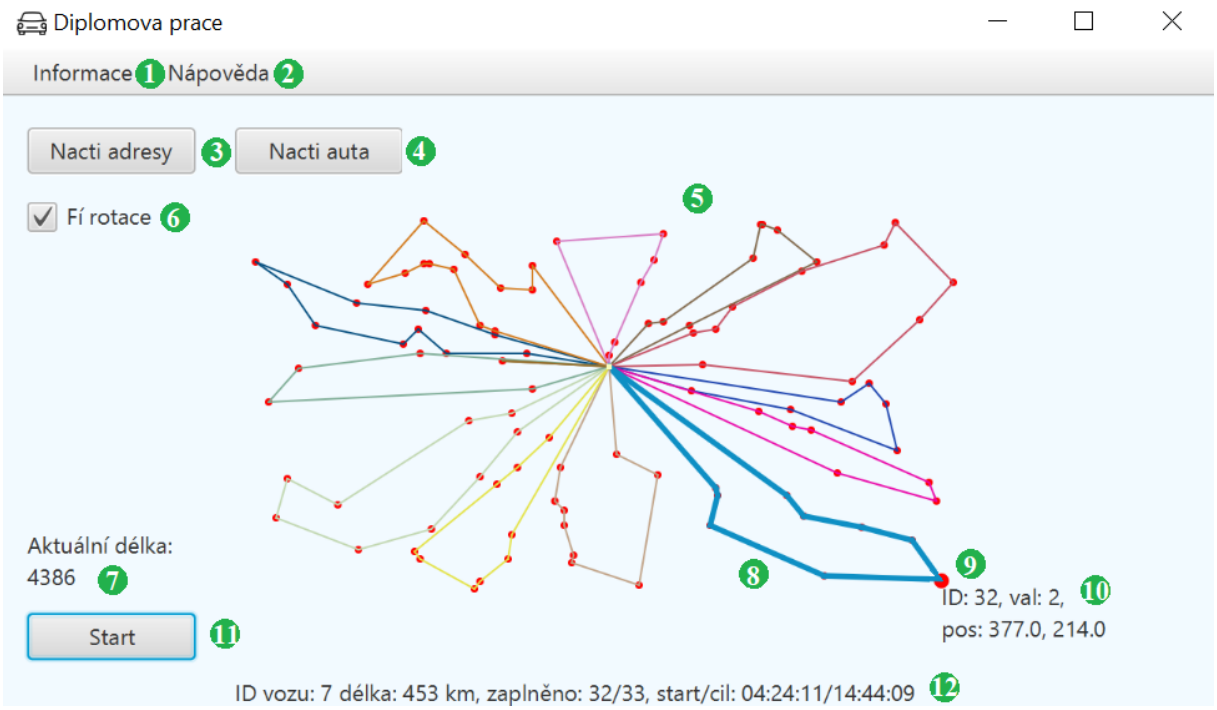
5 POPIS PRAKTICKÉ ČÁSTI

Praktická část je zaměřena na řešení problému VRP s časovými okny a omezením kapacitou vozu (CVRP s časovými okny). Data do programu jsou načítána skrze soubory formátu *.csv. Data jsou načítána ve dvou samostatných souborech. Každý z těchto souborů obsahuje specifické informace:

1. Informace o adresách včetně informace o velikosti zásilky a časovém okně,
2. Informace o vozu.

5.1 Vzhled aplikace

Grafická část aplikace je napsána pomocí JavaFX, tudíž jsou zde komponenty, které jsou součástí tohoto standardu. Vzhled aplikace vypadá následovně:



Obrázek 20: Vzhled aplikace

Obrázek programu je značen body, které lépe popisují funkcionality aplikace:

1. Informace – jedná se o možnosti, které program nabízí. Jsou zde možnosti zobrazení vozů včetně jejich tras, pokud nějaké jsou. Dále jsou zde možnosti výstupu do souboru ve formátu *.txt nebo *.pdf.
2. Nápověda – po výběru možnosti Nápověda se otevře okno s informacemi podobnými těmto.

3. Načti adresy – po stisknutí tohoto tlačítka se otevře okno s výběrem souboru pro načtení adres. Formát souboru bude popsán v kapitole 5.2.1.
4. Načti auto – po stisknutí tohoto tlačítka se otevře okno s výběrem souboru pro načtení vozů. Formát souboru bude popsán v kapitole 5.2.2.
5. Graf tras – grafické vykreslení informací o trase. Je zde hezky znázorněno, jak vypadají projeté trasy. Adresy na trase jsou zapsány v kartézských souřadnicí. Stejně tak je zapsán graf.
6. Fí rotace – výběr, zda je třeba rotace výchozího bodu Fí. Rotace Fí je popsána v kapitole 5.4.2.
7. Aktuální délka – výpočet kompletní vzdálenosti ujeté všemi vozy. Dá se říct, že je to výsledné hodnota řešeného problému.
8. Zvýraznění trasy – zvýraznění aktuální trasy po najetí na trasu kurzorem myši. Zobrazí se také informace o trase a vozu, viz bod 12.
9. Zvýraznění adresy – zvýraznění aktuální adresu po najetí na adresu kurzorem myši. Zobrazí se také informace o adrese, viz bod 10.
10. Informace o adrese – zobrazí se informace o adrese, včetně id adresy, pozice a časového okna.
11. Start – tlačítko, které spustí prohledávání tras. Start lze zpustit až poté, co jsou načtena veškerá data potřebná k výpočtu trasy.
12. Informace o trase – informace o trase jako jsou ID vozu, délka trasy, zaplnění vozu, doba startu vozidla, doba dojetí zpět na depo.

5.2 Vstupní soubory

Pro běh programu jsou třeba dva vstupní soubory. První vstupní soubor obsahuje informace o adresách, druhá obsahuje informace o vozech. Veškerá data jsou oddělena středníkem a každá informace (adresa nebo vůz) jsou na samostatném řádku.

5.2.1 Vstupní soubor adres

Vstupní soubor adres je definován jako soubor *.csv. Struktura souboru vypadá následovně.

Tabulka 5: Vzor vstupních adres

Id adresy	Pozice x	Pozice y	Typ zásilky	Velikost zásilky	Začátek časového okna	Konec časového okna

- Id adresy – unikátní identifikátor adresy. Pokud se jedná o depo, jsou díky identifikátoru této adrese přiřazeny vozy. Identifikátor je celočíselného typu.
- Pozice *x* – hodnota souřadnice *x* v rámci kartézského souřadnicového systému.
- Pozice *y* – hodnota souřadnice *y* v rámci kartézského souřadnicového systému.
- Typ zásilky – typ zásilky zde z důvodu rozšiřitelnosti programu. Typ zásilky nabývá hodnot S (suché), M (mražené), CH (chlazené). Pokud adresa nemá přiřazený typ zásilky, jedná se o depo.
- Začátek časového okna – jedná se o časový údaj v hodinách. Hodnota může být celočíselného či desetinného typu. Pokud zde není žádná hodnota, adresa nemá přiřazené časové okno.
- Konec časového okna – jedná se o časový údaj v hodinách. Hodnota může být celočíselného či desetinného typu. Pokud zde není žádná hodnota, adresa nemá přiřazené časové okno.

5.2.2 Vstupní soubor vozů

Vstupní soubor vozů je definován jako soubor *.csv, jehož struktura vypadá následovně.

Tabulka 6: Vzor vstupních vozů

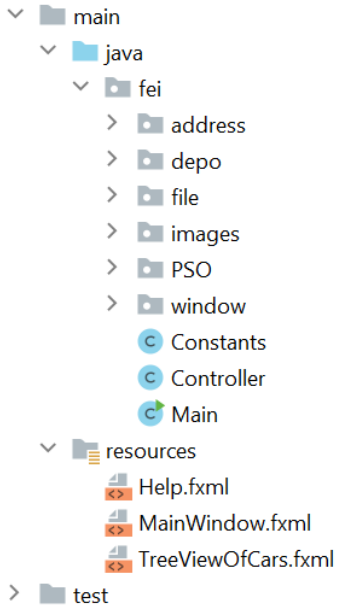
Id adresy	Typ vozu	Počet vozů	Kapacita vozu

- Id adresy – id adresy (depa) ke kterému jsou vozy přiřazeny. Jedná se o celočíselnou hodnotu.
- Typ vozu – jedná se o možnost budoucího rozšíření aplikace o typ vozu. Typ vozu může nabývat hodnot evaporatorSemitrailer, twoEvaporatorsSemitrailer, LorryA, LorryA2, LorryB, LorryC. Po rozšíření aplikace by bylo možné aplikovat rozdělení zásilek do teplotních režimů.
- Počet vozů – jedná se o celočíselnou hodnotu udávající počet vozu stejného typu přidělené konkrétnímu depu.

- ❖ Kapacita vozu – jedná se o celočíselnou hodnotu udávající kapacitu vozů v řádku, které jsou přidělena konkrétnímu depu.

5.3 Struktura programu

Program je strukturován přehledně do balíčků. Každý balíček obsahuje třídy, které přímo souvisí s řešenou problematikou balíčku. Struktura programu vypadá následovně:



Obrázek 21: Struktura balíčků programu

Kompletní struktura je detailněji popsána pomocí diagramu tříd. Diagram tříd je obsažen v přílohách, viz příloha B str. 71 a příloha C str. 72.

5.3.1 Balíček address

Balíček address obsahuje popis tříd Adress.java, PointPolar.java, TimeWindow.java,TypeOfPackage.java.

1. Třída Address.java je předlohou pro objekty typu Address. Tyto objekty nesou informace o názvu adresy (stejný jako id), hodnota adresy, typ balíčku, časové okno, pozice doba návštěvy a polární souřadnice od depa, kterému tato adresa náleží.
2. Třída PointPolar.java je předlohou pro objekty typu PointPolar. Tyto objekty nesou informaci o polárních souřadnicích, tj. primitivní typy r a phi.
3. Třída TimeWindow.java je předlohou pro objekty typu TimeWindow. Tyto objekty nesou informace o časovém okně, tj. kdy časové okno začíná a kdy končí.

4. Výčet TypeOfPackage.java předpovídá typ zásilky.

5.3.2 Balíček depo

Balíček depo obsahuje třídy Depo.java a SaveValueDepos.java. Dále balíček depo obsahuje další balíček car. Balíček car obsahuje třídy Car.java a TypeOfCar.java.

1. Třída Depo.java je předlohou pro objekty typu Depo. Depo obsahuje seznam vozů. Dále obsahuje adresu depa a seznam adres, které depu náleží. V poslední řadě jsou zde ještě informace o celkové ujeté vzdálenosti v rámci depa. Vzdálenost je dopočítávána až na konci výpočtu trasy.
2. Třída SaveValueDepos.java je předlohou pro objekty typu SaveValueDepos. Objekt typu SaveValueDepos je objekt sloužící jako dočasné úložiště nejlepší nalezené trasy a celkovou nejlepší nalezenou vzdálenost. Pro uložení dat je zvolena třída z důvodu, že je třeba držet jen nelepší pozici. V případě potřeby uložení více informací by bylo vhodné data uložit do databáze.
3. Třída Car.java je předlohou pro objekty typu Car. Objekt typu Car nese informace o maximální kapacitě vozu, typu vozu, jedinečné id vozu, aktuální ujetá vzdálenost, aktuálně zaplněná kapacita vozu a aktuální trasa, která je reprezentována instancí objektu Route, viz kapitola 5.3.5.
4. Výčet TypeOfCar.java předpovídá informaci typu vozu.

5.3.3 Balíček file

Balíček file obsahuje třídy FileReader.java a FileWriter.java.

1. Třída FileReader.java je předlohou pro objekty typu FileReader. Tento objekt slouží jako prostředník mezi souborem typu *.csv a samotným programem. Skrze této objekt jsou načteny jak data o adresách, tak data o vozech.
2. Třída FileWriter.java je předlohou pro objekty typu FileWriter. Tento objekt slouží jako prostředník mezi programem a výstupním souborem.

5.3.4 Balíček images

Balíček images obsahuje obrázky help.png, icon.png a model.png. Tyto obrázky jsou použity v aplikaci.

1. help.png – obrázek, který je použit jako nápověda. Obsahuje popisky k aplikaci.
2. icon.png – obrázek, který je použit jako ikona aplikace. Tato ikona byla získána z veřejné open kolekce.
3. model.png – tento obrázek je použit jako vzor pro budoucí zobrazení grafu tras a zastávek.

5.3.5 Balíček PSO

Balíček PSO obsahuje třídy Particle.java a PSOptimalization.java. Dále balíček PSO obsahuje další balíček PSOstructures. Balíček PSOstructures obsahuje třídy Locations.java, Route.java a Velocity.java

1. Třída Particle.java je předlohou pro objekty typu Particle. Objekt Particle obsahuje informace potřebné k prohledávání algoritmem PSO a je základnám kamenem celého prohledávání. Objekt Particle obsahuje informace o počtu adres, instance trasy, kterou momentálně částice našla, instance na objekt obsahující informace o rychlostech částice, instance na objekt s informacemi o lokacích, instance na objekt, který obsahuje nejlepší dosud nalezenou trasu a lokace nejlépe dosud nalezené trasy.
2. Třída PSOptimalization.java je předlohou pro objekty typu PSOptimalization. Tento objekt je hlavním prohledávacím nástrojem. Tato třída obsahuje informace o počtu adres, seznamu s objekty typu Particle, seznam adres, globální nejlepší nalezené řešení všech částic, globální seznam lokací nejlepšího nalezeného řešení.
3. Třída Locations.java je předlohou pro objekty typu Locations. Tato třída drží informace o lokacích potřebných k fungování algoritmu PSO.
4. Třída Route.java je předlohou pro objekty typu Route. Objekty třídy Route drží kompletní informace o trase, včetně samotných adres. Dále jsou objekty typu Route využity v objektech třídy Car, viz kapitola 5.3.2.
5. Třída Velocity.java je předlohou pro objekty typu Velocity. Objekty třídy Velocity drží seznam informací o rychlostech částic.

5.3.6 Balíček window

Balíček window obsahuje třídy ErrDialog.java, FileInputDialog.java, Help.java, TreeViewOfCars. Java.

1. Třída ErrDialog.java je předlohou pro objekty typu ErrDialog. Objekt třídy ErrDialog je v aplikaci použit jako grafické okno s oznámením chybové hlášky, pokud je to třeba.
2. Třída FileInputDialog.java je předlohou pro objekty typu FileInputDialog. Objekt třídy FileInputDialog využit při načítání dat ze vstupního souboru. Z tohoto objektu je následně získána cesta k souboru s daty.
3. Třída Help.java je předlohou pro objekty typu Help. Objekt třídy Help je v aplikaci použit jako grafické okno zobrazující obrázek s informacemi o aplikaci, viz kapitola 5.3.4.
4. Třída TreeViewOfCars.java je předlohou pro objekty typu TreeViewOfCars. Objekt třídy TreeViewOfCars je v aplikaci použit jako grafické okno, které zobrazí seznam vozů.

5.3.7 Třídy Constants, Contoller a Main

Třída Constants obsahuje informace, jež jsou třeba aby byli dostupný globálně, ale zároveň aby nebyli měněny.

- Konstanty pro algoritmus PSO. Tyto konstanty slouží k definici počtu částic hledající trasu, počet iterací prohledávání a také oba parametry prohledávání C_1 (C_p) a C_2 (C_g).
- Pomocná konstanta – LIVELOCK_DETECT. Tato konstanta je použita pro zabránění křížení tras, které může nastat. Je však také možné, že křížení na trase není možné odstranit a v tu chvíli by došlo k zacyklení a program by nemohl dokončit prohledávání. Pokud toto nastane, tato konstanta nám pomůže ukončit prohledávání a začít hledat trasu novou.
- Konstanty vozů – mezi konstanty vozů jsou zařazeny hodnoty rychlosti, doba, než se naloží zásilka, délka nakládání každého bloku zásilky a maximální doba čekání vozu na časové okno.

Třída Controller – tato třída je využita díky navázání na fxml soubor. Obsahuje veškeré informace o všech grafických komponentech a také je v této třídě možnost měnit jejich parametry.

Třída Main – spouštěcí třída obsahující metodu main(). Tato metoda se spouští vždy když je třeba spustit program. Díky třídě Main se vytváří počáteční okno dle nakonfigurovaného fxml souboru.

5.4 Stěžejní metody aplikace

Celá aplikace stojí na několika stěžejních funkcích, které zajišťují chod celého programu. Jedná se o funkce, které buď ve velké míře ovlivňují prohledávání, nebo by bez nich řešení nebylo možné nalézt. V této podkapitole budou jen popsány části kódu. Teoretické fungování je popsáno v předchozích kapitolách.

5.4.1 PSO vyhledávání

Tato část funkce findShortestRoute() slouží k vyhledávání nejkratší trasy. Tato trasa je vytvořena ze seznamu adres, které jsou součástí každé částice, která trasu hledá. Nejdůležitější část je výpočet newVelocity. Tato hodnota je vypočítána vzorcem PSO. Dále ze studie PSO je použit posun, což je v kódu vidět jako newPosition. Tato část kódu je teoreticky popsána v kapitole 2.

```
for (int i = 1; i < Constants.MAXIMUM_ITERATIONS; i++) {
    for (Particle particle : particlesList) {
        for (int j = 0; j < Particle.numberOfAddresses - 1; j++) {
            double r1 = RANDOM.nextDouble();
            double r2 = RANDOM.nextDouble();
            double aktVelocity = particle.velocity.getVelocity().get(j);
            double aktPostition = particle.location.getLocationArrayList().get(j);
            double pBestLoc = particle.locationPBest.getLocationArrayList().get(j);
            double gBestLoc = gBestLocation.getLocationArrayList().get(j);

            // vzorec pro PSO
            double newVelocity = (w * aktVelocity) + Constants.C_P * r1 * (pBestLoc - aktPostition) +
                    Constants.C_G * r2 * (gBestLoc - aktPostition);
            double newPosition = aktPostition + newVelocity;

            // ulozeni nove velocity
            particle.velocity.getVelocity().set(j, newVelocity);
            particle.location.getLocationArrayList().set(j, newPosition);
            particle.swapLocations((int) Math.abs(aktPostition - newPosition));

        }
        particle.hasIntersection();
    }
    sort();
```

Obrázek 22: Ukázka kódu – PSO vyhledávání

5.4.2 Phi rotace

Lze říct, že rotace není úplně z programátorského hlediska přesné. Jde spíše o procházení seznamu než o rotaci takovou jakou ji známe. Tato část kódu slouží jako možnost nalezení lepší trasy za použití posunu v setříděném seznamu dle úhlu phi od nejnižší hodnoty k nejvyšší. Zároveň slouží k redukci složitosti hledání trasy. Rotace phi je popsána více v kapitole 2.3.

Program má možnost zvolit variantu s rotací phi a bez rotace phi. Funkce je popsána následovně. V první iteraci prohledávání (v prohledávání bez rotace) je hledána nejkratší trasa od nejmenší hodnoty. V druhé iteraci probíhá vyhledávání od druhé nejmenší hodnoty a hodnota nejmenší – ta která byla přeskočena – se přesouvá na konec seznamu. Takto to pokračuje dál, dokud každá adresa nebude na seznamu použita jako první. Tudíž počet iterací se rovná počtu adres v seznamu.

```
if (usePhiChanging) {
    for (Depo depo : depoArray) {
        for (int i = 0; i < depo.getAddressesForDepo().size(); i++) {
            for (int j = 0; j < i; j++) {
                depo.getAddressesForDepo().add(depo.getAddressesForDepo().remove( index: 0));
            }

            SaveValueDepos tmp = searchRoutesForBestValue();
            if (best == null || tmp.getValue() < best.getValue()) {
                best = tmp;
            }
            getAddressesForDepo(depoArray, addressArray);
            sortByPolarLocationsForDepos(depoArray);
        }
    }
} else {
    best = searchRoutesForBestValue();
}
```

Obrázek 23: Ukázka kódu – phi rotace

5.4.3 Kontrola oken

Čas výjezdu vozu je měněn dle potřeb časových oken viz Obrázek 24. Kontrolu oken jsem vložil do 4 fází:

1. Pokud okna jdou za sebou bez jakéhokoliv čekání, není třeba měnit čas výjezdu. Tato varianta je vidět v bloku switche case 0. Jediné, co je třeba udělat je kontrola, jestli velikost prvního časového okna je menší než hodnota druhého časového okna. Je to z důvodu tato hodnota je uložena pro možnost posunutí startu vozu.
2. Pokud okna za sebou jdou, ale budu u okna dříve a musel bych čekat. Z tohoto důvodu je zde právě možnost posunu v rámci časových oken. Proto se v předchozím bodu kontrolovala hodnota časových oken a jejich trvání. V rámci této hodnoty se mění startovací čas vozu. Pokud je tato hodnota v nižší než doba čekání vozu na okno, hodnota se sníží a příjezd se naplánuje na začátek okna. Tato část je popsána kódem case 1.
3. Pokud okna jdou za sebou a došla hodnota k posunu oken, je zde ještě jedna možnost, jak okno stihnout. Tato možnost musí být definována při začátku programu. Jedná se o volbu čekání, respektive dobu, jak dlouho bude vůz na dané časové okno čekat. Tato část je v kódu popsána jako case 2.

4. Tato varianta nastává, pokud již není možné okno projet. Buď na nej nelze čekat, protože je v daleké budoucnosti (daleké v rámci jízdy vozu). Nebo je již v minulosti. Tato část je popsána v kódu jako case -1.

```
switch (tWsValue) {
    case 1:
        long second2 = SECONDS.between(beforeAddressWithTW.getTimeWindow().getFrom().plus((long) timeTaveling, SECONDS)
                .plus(manipulationTimeSeconds, SECONDS), actual.getTimeWindow().getFrom());
        startTime = startTime.plus(second2, SECONDS);

        if (second2 >= 0)
            timeForChanging -= second2;
    case 0:
        long second = SECONDS.between(actual.getTimeWindow().getFrom(), actual.getTimeWindow().getTo());
        if (second <= timeForChanging)
            timeForChanging -= second;
        return;
    case 2:
        long second3 = SECONDS.between(beforeAddressWithTW.getTimeWindow().getFrom()
                .plus( amountToAdd: (long) timeTaveling + timeForChanging, SECONDS)
                .plus(manipulationTimeSeconds, SECONDS), actual.getTimeWindow().getFrom());
        startTime = startTime.plus(timeForChanging, SECONDS);
        plusTime += second3;
        actual.setLocalPlusTime((int) second3);

        timeForChanging = 0;
        return;
    case -1:
        addresses.remove(actual);
```

Obrázek 24: Ukázka kódu – kontrola časových oken

6 ZÁVĚR

Tato práce byla věnována heuristickému algoritmu Particle Swarm Optimization aplikovanému k řešení víceokruhové dopravní úlohy. Vzhledem k exponenciální složitosti problému je aplikace heuristik založena na redukci dimenze prohledávaného prostoru možných řešení. Strategie zjednodušení v této práci je založena na algoritmu Wheel, který rozděluje zákazníky do bloků, konkrétně do kruhových výsečí. Původní idejí bylo rozdělení do bloků pomocí získání Giant Tour jako nejlepšího řešení úlohy TSP nad všemi zákazníky doplněnými o depo. Tato strategie však vykazoval mírně horší výsledky a z časového hlediska byl algoritmu velmi pomalý. Pro nalezení vhodného dělení do výsečí je použita metoda rotace phi, která hledá nejlepší počáteční úhel pro první sektor. Souběžně s rozdělením do bloků byla kontrolována také velikost zásilky a řešen tedy problém CVRP. Trasa v rámci jednoho bloku je hledána pomocí algoritmu PSO aplikovaného na úlohu TSP s okny a omezeními na kapacitu. Po nalezení nejkratší trasy proběhla kontrola časových oken. Dle časových oken byly specifikovány výjezdy vozů. Z důvodu zaokrouhlení jsou časová okna dodržena až na chybu několika sekund. Po testování aplikace na Solomonových úlohách lze konstatovat, že aplikace dosahovala lepších výsledků v testovacích úlohách s rovnoměrným rozmístěním zákazníků ve čtvercové oblasti. Cena nalezených řešení byla poměrně vzdálena od ceny dosud nejlepších řešení uváděných v literatuře. Tato řešení byla pochopitelně získána při větším počtu iterací, vyšším výpočetním výkonu a s využitím různých algoritmů. Je zjevné, že existuje značný prostor pro vylepšení aplikace. Účinnost heuristik na různých úlohách bývá v literatuře sledována v závislosti např. na počtu vrcholů konvexního obalu všech zákazníků nebo na poměru největší vzdálenosti a mediánem od depa atd. Podle takových charakteristik je pak možné na konkrétní úloze nastavit vhodné parametry algoritmu. Vzhledem k rozsahu práce takové vyhodnocení a autonomní nastavení parametrů v aplikaci nebylo provedeno.

Aplikace byla napsána v jazyku Java s použitím frameworku JavaFX. V rámci aplikace lze hledat trasy pro úlohy typu CVRP s časovými okny. Finální řešení je znázorněno graficky. Aplikace umožňuje načtení úlohy z *.csv souboru a export tras naplánovaných linek. Ovládání aplikace je popsáno v kapitole 5. V menu aplikace je dostupná položka Nápověda, která obsahuje uživatelskou příručku.

POUŽITÁ LITERATURA

[1] BELMECHERI, Farah, Christian PRINS, Farouk YALAOUI a Lionel AMODEO. Particle swarm optimization algorithm for a vehicle routing problem with heterogeneous fleet, mixed backhauls, and time windows. *Journal of Intelligent Manufacturing* [online]. 2013, **24**(4), 775-789 [cit. 2021-03-10]. ISSN 0956-5515. Dostupné z: http://dx.doi.org/10.1007/s10845-012-0627-8

[2] BELMECHERI, Farah, Christian PRINS, Farouk YALAOUI a Lionel AMODEO, 2010. *Particle Swarm Optimization to solve the Vehicle Routing Problem with Heterogeneous fleet, Mixed Backhauls, and time windows* [online]. IEEE, 2010, 1-6 [cit. 2021-8-13]. ISBN 978-1-4244-6533-0. Dostupné z: http://dx.doi.org/10.1109/IPDPSW.2010.5470702

[3] ČÁPKA, David, 2021. Lekce 1 - Úvod do jazyka Java. *ITnetwork.cz* [online]. Praha: ITnetwork.cz [cit. 2021-8-13]. Dostupné z: https://www.itnetwork.cz/java/zaklady/java-tutorial-uvod-do-jazyka-java

[4] ČÁPKA, David, 2021. Lekce 2 - FXML a první formulářová aplikace v JavaFX. *ITnetwork.cz* [online]. Praha: ITnetwork.cz [cit. 2021-8-13]. Dostupné z: https://www.itnetwork.cz/java/javafx/java-tutorial-uvod-do-javafx

[5] CARIĆ, Tonči a Hrvoje GOLD, ed., 2008. *Vehicle Routing Problem.* 1. Croatia: In-teh. ISBN 978-953-7619-09-1. Dostupné z: http://dx.doi.org/10.5772/64

[6] CORDEAU, Jean-François a Gilbert LAPORTE, 2007. The dial-a-ride problem: models and algorithms. *Annals of Operations Research* [online]. **153**(1), 29-46 [cit. 2021-8-13]. ISSN 0254-5330. Dostupné z: http://dx.doi.org/10.1007/s10479-007-0170-8

[7] ČUŘÍK, František, 1944. *Matematika*. Praha. Kniha. Česká matice technická.

[8] DANTZIG, George a John RAMSER, 1959. The Truck Dispatching Problem. *Management Science* [online]. **6**(1), 80-91 [cit. 2021-8-12]. ISSN 0025-1909. Dostupné z: http://dx.doi.org/10.1287/mnsc.6.1.80

[9] DORRONSORO, Bernabe. CVRP Instances. *Bernabé DORRONSORO, PhD* [online]. Cadiz: University of Cádiz [cit. 2021-8-14]. Dostupné z: http://www.bernabe.dorronsoro.es/vrp/index.html?/results/resultsSolom.htm

[10] FELD, Sebastian, Christoph ROCH, Thomas GABOR, Christian SEIDEL, Florian NEUKART, Isabella GALTER, Wolfgang MAUERER a Claudia LINNHOFF-POPIEN, 2019. A Hybrid Solution Method for the Capacitated Vehicle Routing Problem Using a Quantum Annealer. Frontiers in ICT [online]. 6 [cit. 2021-8-13]. ISSN 2297-198X. Dostupné z: http://dx.doi.org/10.3389/fict.2019.00013

[11] GUNES, Canan, Willem-Jan HOEVE a Sridhar TAYUR, 2010. *Vehicle Routing for Food Rescue Programs: A Comparison of Different Approaches*. Heidelberg: Springer-Verlag Berlin. ISBN 978-3-642-13519-4.

[12] HAO, Jin-Kao a Martin MIDDENDORF, 2012. *Evolutionary Computation in Combinatorial Optimization: 12th European Conference, EvoCOP 2012, Málaga, Spain, April 11-13, 2012, Proceedings*. 1. Málaga: Springer-Verlag Berlin Heidelberg, 264 s. ISBN 978-3-642-29124-1. Dostupné z: http://dx.doi.org/10.1007/978-3-642-29124-1

[13] Help Centre: atan2, 1994. *MathWorks* [online]. Switzerland: MathWorks [cit. 2021-8-12]. Dostupné z: https://ch.mathworks.com/help/matlab/ref/atan2.html

[14] HEŘMÁNEK, Libor, 2008. *Sbírka příkladů z matematiky I ve strukturovaném studiu*. Vyd. 2., přeprac. Praha: Vydavatelství VŠCHT. ISBN 978-80-7080-688-3.

[15] HOSNY, Manar a Christine MUMFORD, 2010. *Our Research on Pickup and Delivery Problems*. Wales. Cardiff University.

[16] Install IntelliJ IDEA, 2021. *IntelliJ IDEA* [online]. Praha: JetBrains [cit. 2021-8-13]. Dostupné z: https://www.jetbrains.com/help/idea/installation-guide.html

[17] IntelliJ IDEA overview, 2021. *IntelliJ IDEA* [online]. Praha: JetBrains [cit. 2021-8-13]. Dostupné z: https://www.jetbrains.com/help/idea/discover-intellij-idea.html

[18] Java Card Technology, 2021. *Oracle* [online]. Santa Clara: Oracle [cit. 2021-8-13]. Dostupné z: https://www.oracle.com/java/technologies/java-card-tech.html

[19] Java EE at a Glance, 2021. *Oracle* [online]. Santa Clara: Oracle [cit. 2021-8-13]. Dostupné z: https://www.oracle.com/java/technologies/java-ee-glance.html

[20] Java Platform, Micro Edition (Java ME), 2021. *Oracle* [online]. Santa Clara: Oracle [cit. 2021-8-13]. Dostupné z: https://www.oracle.com/java/technologies/javameoverview.html

[21] Java SE at a Glance, 2021. *Oracle* [online]. Santa Clara: Oracle [cit. 2021-8-13]. Dostupné z: https://www.oracle.com/java/technologies/java-se-glance.html

[22] JavaFX Scene Builder, 2021. *Oracle* [online]. Santa Clara: Oracle [cit. 2021-8-13]. Dostupné z: https://www.oracle.com/java/technologies/javase/javafxscenebuilder-info.html

[23] JavaFX, 2001. *JavaFX* [online]. Mountain View: Creative Commons [cit. 2021-8-13]. Dostupné z: https://openjfx.io/

[24] KALYUZHNAYA, Zlata, 2021. End of Support for 32-bit Operating Systems in IntelliJ-based IDEs. *IntelliJ IDEA Blog* [online]. Praha: JetBrains [cit. 2021-8-13]. Dostupné z: https://blog.jetbrains.com/idea/2021/04/end-of-support-for-32-bit-operating-systems-in-intellij-based-ides/

[25] KENNEDY, J. a R. EBERHART, 1995. Particle swarm optimization. In: *Proceedings of ICNN'95 - International Conference on Neural Networks* [online]. IEEE, s. 1942-1948 [cit. 2021-8-13]. ISBN 0-7803-2768-3. Dostupné z: http://dx.doi.org/10.1109/ICNN.1995.488968

[26] Kružnice (graf), 2001-. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2021-8-13]. Dostupné z: https://cs.wikipedia.org/wiki/Kru%C5%BEnice_(graf)

[27] KUMAR, Suresh Nanda a Ramasamy PANNEERSELVAM, 2012. A Survey on the Vehicle Routing Problem and Its Variants. *Intelligent Information Management* [online]. **04**(03), 66-74 [cit. 2021-8-12]. ISSN 2160-5912. Dostupné z: http://dx.doi.org/10.4236/iim.2012.43010

[28] LALAIANTS, Daria, 2021. The most popular programming languages in 2021. *Darly Solutions* [online]. Kharkiv: Darly Solutions [cit. 2021-8-13]. Dostupné z: https://darly.solutions/the-most-popular-programming-languages-in-2021/

[29] LAPORTE, Gilbert, 1992. The vehicle routing problem: An overview of exact and approximate algorithms. European Journal of Operational Research [online]. 59(3), 345-358 [cit. 2021-8-13]. ISSN 03772217. Dostupné z: http://dx.doi.org/10.1016/0377-2217(92)90192-C

[30] LAWLER, E., Jan LENSTRA, A. KAN a D. SHMOYS, 1985. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. 1. East Kilbride: Royal Irish Academy. ISBN 978-0-471-90413-7.

[31] LI, Lin, Jiao MENG a Ying CHEN, 2019. Brief Algorithm Review on Vehicle Routing Problems with Different Backhaul Constraints. *2019 Chinese Control And Decision Conference (CCDC)* [online]. IEEE, 2019, 4152-4157 [cit. 2021-8-13]. ISBN 978-1-7281-0106-4. Dostupné z: http://dx.doi.org/10.1109/CCDC.2019.8832941

[32] LIU, Wan-Yu, Chun-Cheng LIN, Ching-Ren CHIU, You-Song TSAO a Qunwei WANG, 2014. Minimizing the Carbon Footprint for the Time-Dependent Heterogeneous-Fleet Vehicle Routing Problem with Alternative Paths. *Sustainability* [online]. **6**(7), 4658-4684 [cit. 2021-8-13]. ISSN 2071-1050. Dostupné z: http://dx.doi.org/10.3390/su6074658

[33] MAŘÍK, Robert, 2007. Polární souřadnice. *Inženýrská matematika* [online]. Brno: Mendelova univerzita v Brně [cit. 2021-8-13]. Dostupné z: https://user.mendelu.cz/marik/in-mat-web/in-mat-webse21.html

[34] MIN, Jianing a Cheng JIN, 2016. A two-phase greedy strategy in one to many PDVRP. *2016 International Conference on Logistics, Informatics and Service Sciences (LISS)* [online]. IEEE, 2016, 1-5 [cit. 2021-8-13]. ISBN 978-1-5090-1102-5. Dostupné z: http://dx.doi.org/10.1109/LISS.2016.7854420

[35] Modeling guide for routing problems, 2019. *LocalSolver* [online]. Innovation 24 [cit. 2021-8-13]. Dostupné z: https://www.localsolver.com/docs/8_5/advancedfeatures/routing.html

[36] NAGY, Gábor, Niaz A. WASSAN, M. Grazia SPERANZA a Claudia ARCHETTI, 2015. The Vehicle Routing Problem with Divisible Deliveries and Pickups. *Transportation Science* [online]. **49**(2), 271-294 [cit. 2021-8-13]. ISSN 0041-1655. Dostupné z: http://dx.doi.org/10.1287/trsc.2013.0501

[37] PARRAGH, Sophie, Karl DOERNER a Richard HARTL, 2008. *A survey on pickup and delivery problems Part I: Transportation between customers and depot* [online]. Wien [cit. 2021-8-12]. Dostupné z: https://prolog.univie.ac.at/research/surveyPDP/PDPsurvey-Part1_web.pdf. Faculty of Business, Economics and Statistics.

[38] PARRAGH, Sophie, Karl DOERNER a Richard HARTL, 2008. *A survey on pickup and delivery problems Part II: Transportation between pickup and delivery locations* [online]. Wien [cit. 2021-8-12]. Dostupné z: https://prolog.univie.ac.at/research/surveyPDP/PDP-surveyPart2_web.pdf. Faculty of Business, Economics and Statistics.

[39] PRINS, Christian, 2008. *The route-first cluster-second principle in vehicle routing*. France. University of Technology of Troyes.

[40] Radians to Degrees conversion calculator. *RapidTables* [online]. [cit. 2021-8-13]. Dostupné z: https://www.rapidtables.com/convert/number/radians-to-degrees.html

[41] Scene Builder, 2015. *Gluon* [online]. Belgium: Gluon [cit. 2021-8-13]. Dostupné z: https://gluonhq.com/products/scene-builder/

[42] SHI, X.H., Y.C. LIANG, H.P. LEE, C. LU a Q.X. WANG, 2007. Particle swarm optimization-based algorithms for TSP and generalized TSP. *Information Processing Letters* [online]. **103**(5), 169-176 [cit. 2021-8-12]. ISSN 00200190. Dostupné z: http://dx.doi.org/10.1016/j.ipl.2007.03.010

[43] SIMCHI-LEVI, David, Xin CHEN a Julien BRAMEL. *The Logic of Logistics: Theory, Algorithms, and Applications for Logistics Management*. Third Edition. New York: Springer Science+Business Media, 2014, 447 s. ISBN 978-1-4614-9148-4.

[44] TAVAKKOLI-MOGHADDAM, R., M. RABBANI, A. SAREMI a N. SAFAEI, 2005. *SOLVING THE BACKHAUL VEHICLE ROUTING PROBLEM BY GENETIC ALGORITHMS*. Tehran. University of Tehran.

[45] TURZÍK, Daniel, 1999. Matematika III: základy optimalizace. Vyd. 3. Praha: Vydavatelství VŠCHT. ISBN 80-7080-363-0.

[46] Vehicle Routing Problem (VRP) and Traveling Salesman Problem Types. *LOG-VRP* [online]. Türkiye: NetAkıl [cit. 2021-8-13]. Dostupné z: https://logvrp.com/log-vrpsite/en/vrp.types.algorithms.html

[47] Vehicle Routing Problem, 2013. Networking and Emerging Optimization [online]. Malaga: University of Malaga [cit. 2021-8-12]. Dostupné z: https://neo.lcc.uma.es/vrp/vehicle-routing-problem/

[48] Welcome to Apache Maven, 2021. *Apache Maven Project* [online]. Wakefield: The Apache Software Foundation [cit. 2021-8-13]. Dostupné z: https://maven.apache.org/

[49] Where the world builds software, 2021. *GitHub.com* [online]. San Francisco: GitHub [cit. 2021-8-13]. Dostupné z: https://github.com/about

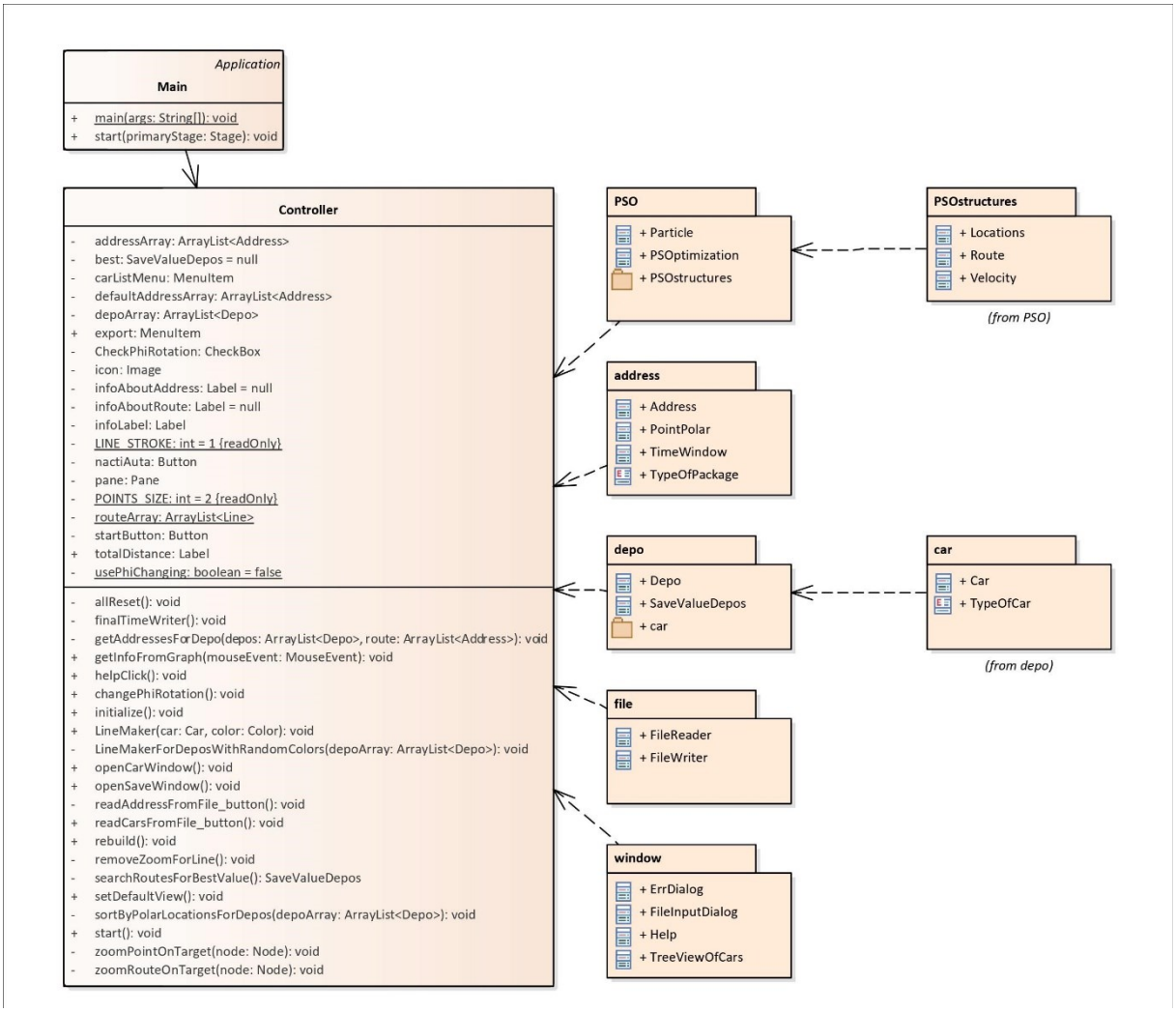
[50] WIKIPEDIA, contributors, 2001. Travelling salesman problem. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2021-8-12]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Travelling_salesman_problem&oldid=1035459634

PŘÍLOHY

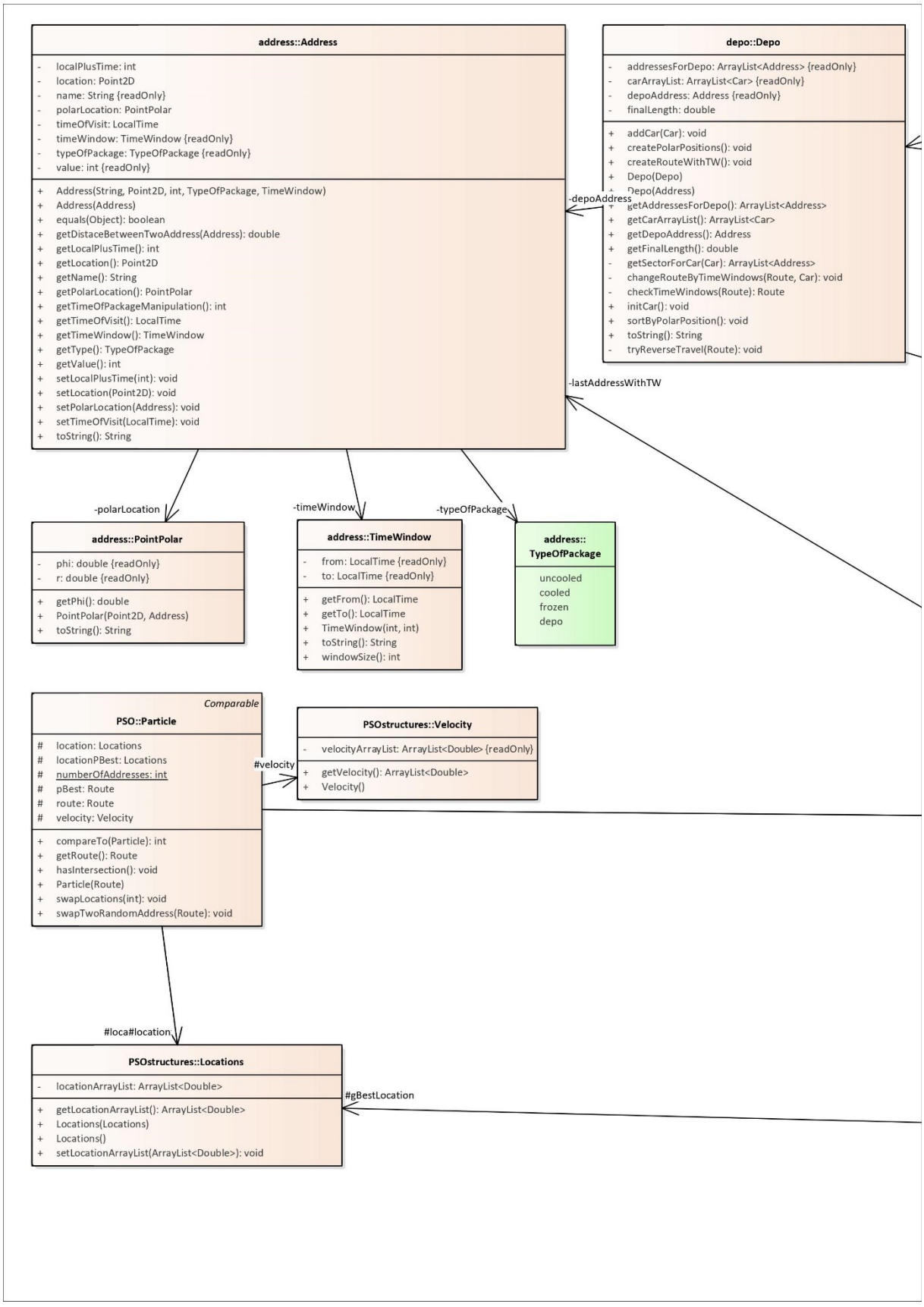
PŘÍLOHA A – ZDROJOVÝ KÓD APLIKACE

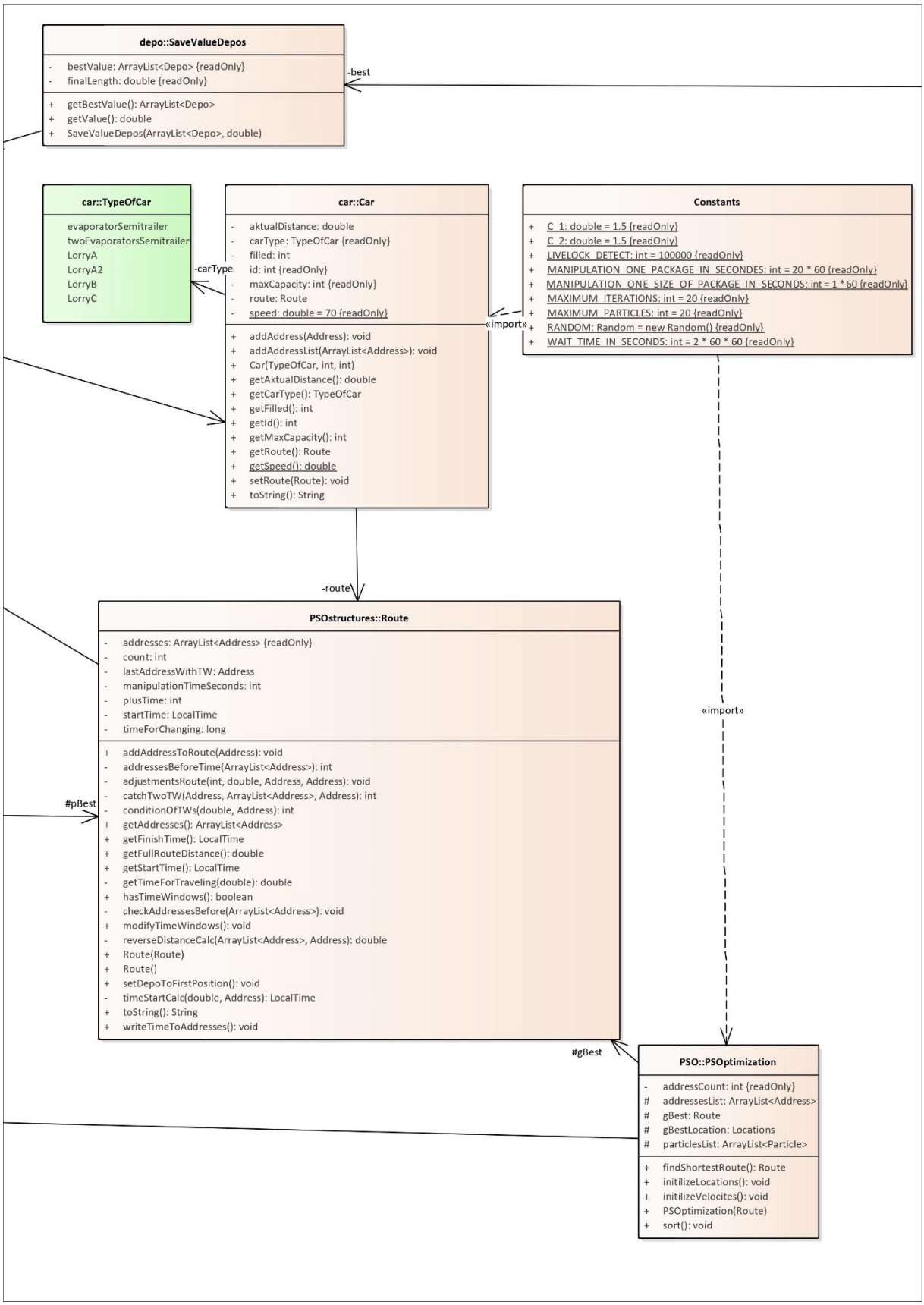
K práci jsou přiloženy zdrojové kódy aplikace včetně spustitelného jar souboru.

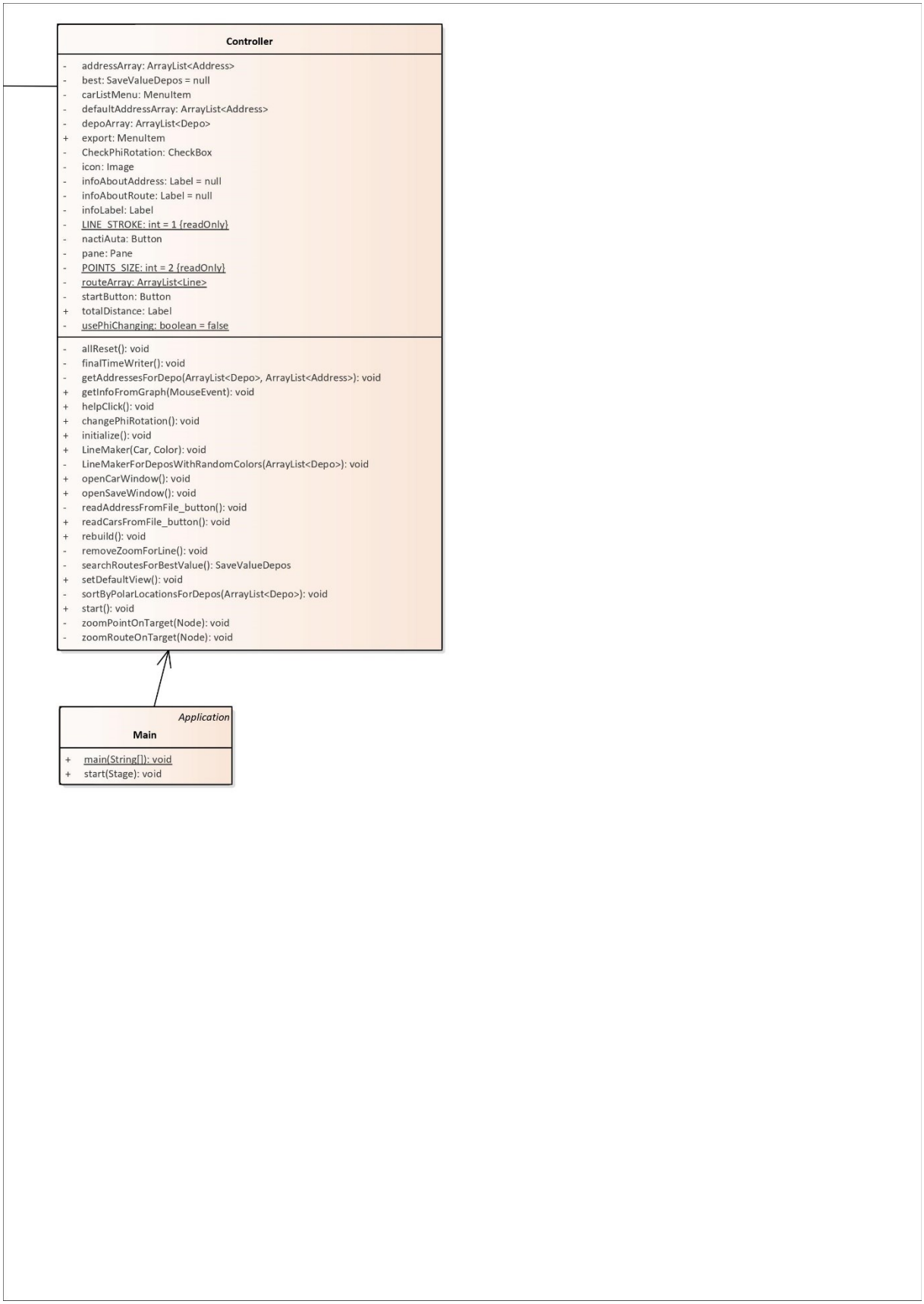
PŘÍLOHA B – DIAGRAM TŘÍD S BALÍČKY



PŘÍLOHA C – DIAGRAM TŘÍD BEZ BALÍČKŮ




depo::SaveValueDepos
- bestValue: ArrayList<Depo> {readOnly}
- finalLength: double {readOnly}
+ getBestValue(): ArrayList<Depo>
+ getValue(): double
+ SaveValueDepos(ArrayList<Depo>, double)
-best
car::TypeOfCar
evaporatorSemitrailer
twoEvaporatorsSemitrailer
LorryA
LorryA2
LorryB
LorryC
-carType
car::Car
- aktualDistance: double
- carType: TypeOfCar {readOnly}
- filled: int
- id: int {readOnly}
- maxCapacity: int {readOnly}
- route: Route
- speed: double = 70 {readOnly}
+ addAddress(Address): void
+ addAddressList(ArrayList<Address>): void
+ Car(TypeOfCar, int, int)
+ getAktualDistance(): double
+ getCarType(): TypeOfCar
+ getFilled(): int
+ getId(): int
+ getMaxCapacity(): int
+ getRoute(): Route
+ getSpeed(): double
+ setRoute(Route): void
+ toString(): String
«import»
Constants
+ C_1: double = 1.5 {readOnly}
+ C_2: double = 1.5 {readOnly}
+ LIVELOCK_DETECT: int = 100000 {readOnly}
+ MANIPULATION_ONE_PACKAGE_IN_SECONDES: int = 20 * 60 {readOnly}
+ MANIPULATION_ONE_SIZE_OF_PACKAGE_IN_SECONDS: int = 1 * 60 {readOnly}
+ MAXIMUM_ITERATIONS: int = 20 {readOnly}
+ MAXIMUM_PARTICLES: int = 20 {readOnly}
+ RANDOM: Random = new Random() {readOnly}
+ WAIT_TIME_IN_SECONDS: int = 2 * 60 * 60 {readOnly}
-route
PSOstructures::Route
- addresses: ArrayList<Address> {readOnly}
- count: int
- lastAddressWithTW: Address
- manipulationTimeSeconds: int
- plusTime: int
- startTime: LocalTime
- timeForChanging: long
+ addAddressToRoute(Address): void
- addressesBeforeTime(ArrayList<Address>): int
- adjustmentsRoute(int, double, Address, Address): void
- catchTwoTW(Address, ArrayList<Address>, Address): int
- conditionOfTWs(double, Address): int
+ getAddresses(): ArrayList<Address>
+ getFinishTime(): LocalTime
+ getFullRouteDistance(): double
+ getStartTime(): LocalTime
- getTimeForTraveling(double): double
+ hasTimeWindows(): boolean
- checkAddressesBefore(ArrayList<Address>): void
+ modifyTimeWindows(): void
- reverseDistanceCalc(ArrayList<Address>, Address): double
+ Route(Route)
+ Route()
+ setDepoToFirstPosition(): void
- timeStartCalc(double, Address): LocalTime
+ toString(): String
+ writeTimeToAddresses(): void
#pBest
«import»
#gBest
PSO::PSOptimization
- addressCount: int {readOnly}
addressesList: ArrayList<Address>
gBest: Route
gBestLocation: Locations
particlesList: ArrayList<Particle>
+ findShortestRoute(): Route
+ initilizeLocations(): void
+ initilizeVelocites(): void
+ PSOptimization(Route)
+ sort(): void


Controller
- addressArray: ArrayList<Address>
- best: SaveValueDepos = null
- carListMenu: MenuItem
- defaultAddressArray: ArrayList<Address>
- depoArray: ArrayList<Depo>
+ export: MenuItem
- CheckPhiRotation: CheckBox
- icon: Image
- infoAboutAddress: Label = null
- infoAboutRoute: Label = null
- infoLabel: Label
- LINE_STROKE: int = 1 {readOnly}
- nactiAuta: Button
- pane: Pane
- POINTS_SIZE: int = 2 {readOnly}
- routeArray: ArrayList<Line>
- startButton: Button
+ totalDistance: Label
- usePhiChanging: boolean = false
- allReset(): void
- finalTimeWriter(): void
- getAddressesForDepo(ArrayList<Depo>, ArrayList<Address>): void
+ getInfoFromGraph(MouseEvent): void
+ helpClick(): void
+ changePhiRotation(): void
+ initialize(): void
+ LineMaker(Car, Color): void
- LineMakerForDeposWithRandomColors(ArrayList<Depo>): void
+ openCarWindow(): void
+ openSaveWindow(): void
- readAddressFromFile_button(): void
+ readCarsFromFile_button(): void
+ rebuild(): void
- removeZoomForLine(): void
- searchRoutesForBestValue(): SaveValueDepos
+ setDefaultView(): void
- sortByPolarLocationsForDepos(ArrayList<Depo>): void
+ start(): void
- zoomPointOnTarget(Node): void
- zoomRouteOnTarget(Node): void
Application
Main
+ main(String[]): void
+ start(Stage): void

PŘÍLOHA D – POROVNÁNÍ PC A TEORETICKÉHO VÝPOČTU – DATA

Vypočítané časové hodnoty pro porovnání teoretické doby výpočtu oproti reálné době výpočtu. Informace o aplikaci a výpočetním stroji více popsány v kapitole 1.2.

Počet adres	Teoretická doba výpočtu (s)	Reálná doba výpočtu (s)	Počet permutací
2	0,000000000625	0,0000037	1
3	0,00000000125	0,000004301	2
4	0,00000000375	0,000004801	6
5	0,000000015	0,000008701	24
6	0,000000075	0,000069401	120
7	0,00000045	0,0002956	720
8	0,00000315	0,001401301	5040
9	0,0000252	0,007917099	40320
10	0,0002268	0,0171787	362880
11	0,002268	0,085739801	3628800

PŘÍLOHA E – TESTOVÁNÍ PARAMETRŮ – DATA

Vypočtené ceny dopravy pro různé hodnoty parametrů C1 a C2. První řádek tabulky udává hodnotu, která byla parametrem udána.

0,0	**0,1**	**0,2**	**0,4**	**0,6**	**0,8**	**1,0**	**1,2**	**1,4**	**1,6**	**1,8**	**2,0**
5128	4582	4714	4597	4714	4597	4600	4582	4597	4582	4714	4591
5202	4714	4597	4714	4588	4714	4591	4582	4714	4597	4714	4714
5100	4714	4714	4714	4597	4714	4714	4714	4597	4582	4582	4714
4741	4714	4714	4582	4714	4582	4597	4714	4585	4714	4714	4714
4686	4714	4588	4714	4714	4714	4714	4714	4714	4582	4714	4597
4636	4582	4714	4714	4714	4714	4714	4597	4714	4714	4714	4714
4723	4714	4714	4714	4582	4714	4714	4714	4582	4714	4714	4597
4683	4714	4714	4582	4714	4714	4714	4714	4714	4714	4714	4714
4765	4714	4714	4597	4714	4582	4714	4597	4714	4582	4714	4714
5076	4714	4588	4582	4714	4582	4714	4714	4714	4597	4714	4582
5047	4714	4714	4714	4582	4714	4714	4585	4714	4714	4582	4714
5124	4597	4714	4582	4597	4714	4582	4714	4714	4714	4582	4714
4673	4714	4714	4714	4714	4597	4597	4714	4714	4714	4714	4714
4639	4597	4714	4714	4714	4714	4583	4714	4714	4582	4714	4714
5005	4714	4714	4582	4582	4714	4714	4714	4714	4714	4582	4714
4886	4714	4714	4714	4714	4714	4582	4597	4588	4582	4714	4597
4644	4714	4714	4582	4714	4714	4597	4714	4714	4714	4582	4714
5083	4714	4714	4714	4714	4597	4588	4714	4597	4714	4582	4714
4612	4714	4582	4585	4582	4714	4597	4714	4582	4714	4714	4714
4784	4585	4597	4714	4714	4582	4582	4714	4714	4714	4714	4582
5158	4714	4714	4582	4582	4714	4597	4582	4714	4597	4714	4582
4721	4582	4714	4597	4714	4714	4714	4714	4597	4582	4597	4597
4632	4714	4588	4714	4582	4714	4714	4714	4714	4714	4597	4714
4632	4597	4714	4582	4714	4714	4714	4714	4714	4714	4582	4714
4729	4714	4714	4714	4714	4598	4714	4714	4714	4714	4714	4590
4606	4714	4714	4714	4714	4582	4582	4597	4582	4714	4714	4590
5194	4597	4714	4714	4582	4714	4714	4597	4714	4582	4714	4714
4735	4714	4714	4714	4714	4597	4714	4714	4714	4714	4582	4714
4918	4714	4714	4714	4582	4714	4597	4597	4714	4714	4714	4597
5132	4582	4714	4714	4714	4714	4714	4714	4597	4714	4714	4714
5112	4714	4590	4714	4714	4582	4714	4714	4597	4582	4714	4714
4747	4582	4597	4714	4714	4714	4582	4714	4714	4714	4597	4714
5094	4714	4582	4714	4714	4714	4714	4582	4714	4717	4598	4582
4767	4714	4582	4582	4582	4714	4714	4714	4714	4597	4582	4582
5116	4714	4714	4714	4714	4588	4582	4714	4582	4714	4597	4714
5158	4714	4582	4597	4714	4714	4714	4582	4597	4583	4597	4714
4754	4714	4714	4597	4597	4714	4582	4714	4714	4597	4597	4714
4805	4714	4588	4714	4714	4582	4597	4714	4714	4582	4588	4591
4732	4582	4714	4714	4714	4597	4714	4714	4582	4714	4714	4714
4671	4714	4714	4714	4588	4597	4582	4582	4714	4582	4714	4714
4973	4714	4585	4714	4714	4582	4601	4582	4597	4714	4582	4714

4666	4714	4714	4714	4714	4582	4714	4597	4714	4714	4714	4714
5146	4714	4714	4600	4714	4714	4714	4714	4714	4714	4582	4582
4998	4582	4597	4714	4582	4598	4714	4714	4714	4714	4714	4714
4727	4582	4714	4714	4714	4714	4582	4714	4714	4714	4597	4714
4647	4582	4714	4714	4582	4588	4714	4714	4714	4714	4714	4714
4620	4714	4597	4714	4582	4714	4597	4582	4597	4582	4714	4714
5076	4714	4714	4714	4582	4582	4714	4714	4714	4582	4714	4714
4813	4597	4597	4582	4582	4597	4585	4600	4714	4597	4714	4714
5123	4714	4714	4582	4582	4714	4714	4714	4714	4714	4582	4714
4527	4714	4714	4598	4714	4714	4582	4583	4588	4582	4714	4714
5086	4582	4714	4714	4714	4714	4714	4714	4714	4598	4714	4588
4884	4714	4714	4714	4714	4582	4714	4714	4582	4714	4582	4582
5019	4714	4714	4714	4600	4714	4582	4597	4714	4582	4582	4588
4631	4582	4714	4597	4714	4714	4582	4714	4714	4714	4714	4714
4617	4597	4582	4598	4582	4714	4717	4714	4582	4714	4714	4588
4784	4714	4714	4597	4714	4714	4714	4714	4714	4714	4714	4597
4732	4714	4598	4714	4714	4714	4714	4597	4582	4597	4714	4714
4964	4582	4582	4714	4714	4714	4714	4582	4582	4714	4597	4714
4973	4588	4714	4714	4585	4714	4582	4582	4714	4714	4714	4597

Výběrový průměr:

4861	4672	4674	4668	4665	4668	4658	4668	4670	4663	4664	4670

Výběrová směrodatná odchylka:

200,6	59,7	57,9	60,1	62,4	60,2	62,4	60,1	59,4	62,1	61,7	59,3

Vypočtené ceny dopravy pro různou hodnotu počtu částic. První řádek tabulky udává hodnotu, která byla parametrem udána.

5	**10**	**15**	**20**	**25**	**30**	**35**	**40**	**45**	**50**
4718	4714	4585	4714	4714	4582	4714	4597	4582	4597
4714	4714	4583	4714	4714	4597	4714	4597	4582	4582
4604	5116	4714	4714	4714	4714	4588	4714	4714	4597
4727	4714	4714	4714	4597	4597	4714	4582	4597	4714
4596	4714	4582	4597	4582	4714	4714	4714	4597	4714
4714	4585	4714	4714	4714	4714	4714	4597	4714	4714
4586	4714	4598	4582	4582	4714	4714	4597	4714	4714
4502	4714	4714	4714	4582	4714	4714	4582	4714	4582
4599	4714	4714	4582	4714	4582	4582	4714	4597	4714
4715	4714	4585	4714	4597	4597	4582	4582	4714	4714
4605	4593	4597	4714	4582	4714	4714	4714	4597	4582
4714	4591	4582	4714	4714	4714	4714	4714	4714	4714
4725	5086	4582	4714	4600	4714	4714	4714	4582	4582
4718	4583	4714	4717	4714	4597	4714	4597	4714	4714
4715	4586	4714	4597	4714	4714	4714	4582	4714	4582
4587	4714	4714	4582	4714	4714	4714	4582	4714	4714
4718	4583	4582	4714	4714	4582	4597	4597	4714	4714

4714	4717	4714	4714	4714	4714	4714	4714	4714	4714
4715	4714	4717	4714	4714	4714	4582	4582	4714	4597
4715	4583	4488	4714	4597	4597	4582	4597	4714	4714
5116	4585	4597	4714	4714	4714	4582	4597	4582	4714
4587	4714	4591	4714	4582	4714	4597	4582	4714	4714
4606	4714	4714	4714	4597	4714	4714	4714	4714	4582
4610	4591	4714	4714	4582	4714	4582	4714	4582	4597
4715	4591	4714	4714	4582	4583	4714	4582	4714	4714
4604	4714	4714	4597	4714	4714	4714	4714	4582	4714
4608	4597	4597	4714	4714	4714	4582	4714	4714	4714
4715	4714	4714	4714	4714	4714	4714	4714	4597	4714
4598	4598	4714	4714	4714	4582	4714	4714	4714	4597
4714	4714	4597	4597	4714	4582	4714	4714	4714	4597
4714	4714	4582	4583	4714	4714	4582	4597	4597	4714
4714	4714	4714	4714	4714	4714	4714	4582	4582	4714
4723	4714	4597	4714	4582	4714	4714	4714	4714	4597
4603	4714	4582	4582	4582	4714	4582	4714	4714	4582
4714	4714	4582	4714	4582	4597	4714	4597	4597	4597
4496	4597	4714	4717	4714	4582	4597	4582	4597	4714
4605	4723	4601	4598	4714	4714	4714	4714	4714	4582
4588	4714	4714	4582	4714	4714	4714	4582	4714	4582
4717	4717	4715	4714	4714	4597	4714	4714	4597	4582
4601	4714	4583	4582	4714	4597	4597	4714	4714	4714
4718	4600	4714	4714	4582	4714	4714	4597	4714	4582
4715	4714	4597	4582	4714	4714	4714	4714	4597	4714
4714	4714	4582	4714	4714	4714	4714	4714	4714	4714
4720	4714	4582	4714	4714	4714	4582	4582	4714	4582
4714	4714	4714	4714	4714	4597	4597	4582	4714	4582
4714	4599	4714	4597	4714	4714	4582	4714	4714	4714
4714	4582	4597	4598	4714	4582	4714	4714	4582	4597
4592	4714	4714	4714	4714	4714	4714	4714	4714	4714
4585	4586	4714	4582	4714	4714	4714	4582	4714	4714
4607	4598	4597	4582	4714	4597	4714	4597	4582	4714
4598	4598	4714	4714	4582	4714	4714	4582	4714	4714
4598	4714	4714	4582	4714	4714	4714	4714	4714	4714
4714	4714	4714	4582	4597	4597	4582	4582	4714	4597
4715	4714	4593	4597	4597	4714	4714	4597	4714	4714
4714	4582	4714	4714	4714	4714	4714	4714	4714	4714
4592	4597	4597	4714	4714	4582	4582	4714	4582	4582
4599	4952	4714	4714	4597	4714	4714	4714	4582	4714
4591	4714	4714	4598	4714	4714	4714	4597	4582	4714
4488	4582	4582	4582	4714	4582	4582	4582	4582	4714
4724	4714	4582	4582	4714	4714	4714	4714	4714	4714

Výběrový průměr:

4664	4686	4652	4666	4672	4671	4671	4649	4666	4664

Výběrová směrodatná odchylka:

89,82	103,76	65,51	61,25	59,19	59,03	60,12	62,46	60,84	61,44

Vypočtené ceny dopravy pro různou hodnotu počtu iterací. První řádek tabulky udává hodnotu, která byla parametrem udána.

5	**10**	**15**	**20**	**25**	**30**	**35**	**40**	**45**	**50**
4605	4714	4714	4714	4714	4714	4714	4714	4597	4714
4593	4723	4582	4597	4714	4714	4582	4714	4714	4714
4714	4714	4582	4714	4582	4714	4714	4714	4714	4714
4715	4588	4600	4714	4714	4714	4714	4582	4714	4582
4608	4717	4582	4714	4714	4714	4582	4582	4714	4714
4717	4714	4714	4714	4714	4714	4714	4714	4714	4582
4716	4591	4714	4714	4714	4714	4714	4714	4714	4714
4717	4714	4582	4582	4597	4714	4582	4714	4714	4582
4592	4714	4714	4714	4714	4714	4597	4582	4714	4714
4717	4714	4714	4597	4714	4714	4714	4582	4714	4714
4604	4714	4588	4597	4582	4582	4582	4582	4714	4714
4728	4591	4714	4714	4598	4714	4582	4597	4714	4714
4592	4597	4714	4714	4714	4714	4714	4597	4714	4582
4608	4714	4714	4714	4714	4582	4714	4597	4714	4597
4586	4715	4588	4597	4582	4582	4597	4714	4714	4582
4715	4717	4714	4597	4597	4582	4714	4597	4714	4714
5121	4717	4714	4714	4714	4714	4714	4714	4714	4582
4724	4714	4714	4714	4714	4582	4714	4714	4714	4714
4726	4599	4714	4714	4597	4582	4714	4582	4582	4714
4600	4714	4714	4714	4714	4714	4582	4597	4714	4714
4717	4714	4583	4714	4714	4714	4714	4714	4582	4714
4731	4598	4714	4714	4714	4714	4714	4714	4714	4714
4488	4585	4714	4714	4714	4588	4714	4714	4582	4714
4718	4597	4714	4714	4582	4582	4582	4714	4714	4714
4607	4714	4594	4582	4714	4714	4714	4714	4714	4597
4714	4601	4582	4582	4714	4714	4597	4582	4714	4714
4718	4590	4714	4717	4714	4714	4714	4582	4714	4714
4591	4714	4714	4597	4714	4597	4582	4582	4582	4582
4714	4601	4597	4714	4714	4597	4714	4714	4714	4714
4593	4714	4582	4714	4582	4714	4597	4714	4714	4714
4595	4714	4714	4714	4714	4714	4714	4582	4582	4714
4714	4583	4714	4714	4597	4582	4714	4582	4582	4714
4595	4715	4582	4714	4714	4582	4714	4714	4597	4597
5121	4714	4714	4714	4714	4597	4714	4597	4582	4714
4715	4714	4714	4714	4597	4582	4714	4714	4597	4714
4715	4583	4597	4583	4714	4597	4582	4714	4582	4714
4591	4714	4714	4714	4714	4714	4714	4582	4597	4597
4715	4597	4714	4714	4597	4714	4582	4714	4582	4714
4714	4583	4714	4597	4714	4582	4714	4714	4714	4714

4706	4715	4714	4714	4714	4714	4714	4582	4714	4582
4592	4714	4582	4582	4714	4714	4714	4714	4582	4714
4604	4714	4582	4598	4597	4714	4597	4597	4597	4714
5086	4582	4585	4582	4714	4582	4714	4582	4714	4714
4715	4714	4597	4582	4582	4714	4714	4714	4714	4582
4583	4714	4714	4582	4714	4714	4714	4714	4714	4714
4720	4714	4714	4714	4714	4597	4714	4597	4714	4714
4714	4598	4714	4597	4714	4582	4597	4714	4714	4714
4717	4714	4714	4714	4714	4714	4714	4714	4714	4714
4718	4582	4582	4582	4714	4714	4714	4582	4714	4714
4714	4715	4714	4582	4714	4714	4597	4714	4582	4714
4601	4598	4714	4714	4600	4714	4714	4714	4714	4714
4586	4582	4714	4714	4582	4714	4714	4714	4582	4714
4718	4714	4585	4714	4714	4714	4714	4714	4714	4714
4716	4718	4714	4714	4597	4714	4582	4582	4582	4582
4597	4714	4717	4714	4582	4582	4582	4582	4597	4714
4733	4714	4582	4714	4714	4597	4582	4582	4714	4597
4714	4582	4714	4582	4714	4714	4582	4597	4582	4714
4593	4585	4714	4714	4714	4582	4597	4714	4597	4597
4717	4582	4714	4714	4582	4714	4714	4582	4714	4582
4715	4583	4600	4714	4597	4582	4582	4582	4582	4714

Výběrový průměr:

4689	4665	4667	4672	4673	4665	4665	4652	4667	4678

Výběrová směrodatná odchylka:

114,85	61,23	61,12	58,91	58,11	61,95	61,46	63,47	61,12	56,91

PŘÍLOHA F – TESTOVÁNÍ SOLOMONOVÝCH ÚLOH – DATA

Zde se nacházejí data, která byla použita při tvorbě grafu v části testování programu Solomonovými úlohami.

		Nejlepší nalezené hodnoty z aplikace	Nejlepší nalezená dle literatury
25	C101	375,74	191,30
	R101	917,84	617,10
	RC101	763,85	461,10
50	C101	649,48	362,40
	R101	1793,80	1047,00
	RC101	1517,14	957,90
100	C101	1491,62	827,30
	R101	2956,46	1637,70
	RC101	2357,28	1619,80