Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

Studijní program: Aplikovaná informatika

Obor: Informatika

Návrh a implementace knihovny pro řešení okružního dopravního problému

BAKALÁŘSKÁ PRÁCE

Student : Michal Drobný

Vedoucí : Ing. Libor Gála

Oponent : Ing. Jan Melechovský, Ph.D.

2014

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze které jsem čerpal.

V Praze dne 16.12.2014

. . . . . .

Jméno a příjmení studenta

Poděkování:

Rád bych tímto poděkoval všem, kteří mi byli podporou při psaní této práce.

Jedná se především o moji nejmilejší Lenku Kubíkovou, která se mnou protrpěla mnoho nudných víkendů a obětovaných horských dobrodružství při psaní práce.

Velký dík patří Ing. Janu Melechovskému PhD., jehož konzultace ohledně problematiky okružního dopravního problému mě vždy navedly tím správným směrem.

Dále bych rád poděkoval svému vedoucímu Ing. Liborovi Gálovi, díky kterému se struktura této práce ubírala cestou obhajitelnou na KIT.

Konečně bych rád poděkoval celé své rodině a přátelům, kteří mě psychicky podpořili v dokončení této práce.

Děkuji.

Abstrakt

S různými typy dopravních problémů se v praxi setkáváme velmi často. Tento problém lze chápat především jako rozvoz zboží od dodavatelů k odběratelům s cílem minimalizace distribučních nákladů. Reálné dopravní problémy se od těch obecných liší především uvažovanými restrikcemi, což mohou být například kapacity vozidel a objednávek, časová okna a různá další speciální distribuční omezení.

Na trhu jsou prezentovány některé specializované komerční nástroje pro optimalizaci distribučních nákladů, které jsou mnohdy součástí větších frameworků, které dále zajišťují další služby jako například sledování vozidel, umístění skladů, výkazy jízd atp. Tyto nástroje poskytují cloudové služby plně customizované dle zákazníka.

Tato práce si klade za cíl navrhnout a zkonstruovat open source softwarovou knihovnu pro řešení okružního dopravního problému, která bude schopna reflektovat požadavky jednoduchosti, rozšiřitelnosti a přenositelnosti. Navrhovaná knihovna může sloužit nejen jako open source řešení okružních dopravních problémů, nástroj pro optimalizaci distribučních nákladů malých a středních firem ale především jako nástroj pro testování nově zkoumaných metod na akademické půdě.

Práce je strukturována do několika hlavních částí. V první části definujeme cíle práce, vymezujeme okružní dopravní problém a popisujeme principy jeho řešení. V další části navrhujeme strukturu softwarového díla, diskutujeme výběr prostředků a dokumentujeme návrh. Poslední část práce se věnuje implementaci, validaci a reálnému využití softwarového díla.

Klíčová slova

Okružní dopravní problém, open source, softwarová knihovna, Java.

Abstract

Various types of transportation issues are a common practice. The issue may be approached mainly as the distribution of products from suppliers to consumers while minimising distribution costs. The difference of real transportation issues predominantly relates to the considered restrictions, such as capacities of vehicles and orders, time windows and other special distribution restrictions.

There are some specialized commercial products for optimizing distribution costs presented on the market. These are often part of larger frameworks, which also provides other services such as vehicle tracking, warehouse location, ride statements etc. These products provide cloud services fully customized according to the customer.

This work aims to design and construct an open source software library for solving vehicle routing problem, which will be able to reflect requirements of simplicity, scalability and portability. The constructed library can serve not only as an open source solution for vehicle routing problems, but also as a tool for testing new methods studied on university premises.

The work is divided into several parts. In the first part we define the goals of this work, we define the vehicle routing problem and we describe the principles of the solution. In the next section, we propose the structure of the software, we discuss the choice of aids and document a design phase. The last part deals with the implementation phase, validation phase and real use of the software.

Keywords

Vehicle Routing Problem, Open Source, Software Library, Java.

Obsah

1 Úvod 1

1.1 Formulace problému 1

1.2 Cíle práce 2

2 Okružní dopravní problém 3

2.1 Definice 3

2.1.1 Úloha obchodního cestujícího 3

2.1.2 Problém čínského listonoše 4

2.1.3 Okružní dopravní problém 5

2.1.4 Podmínky časových oken 5

2.2 Matematický model 6

2.2.1 Zadání 6

2.2.2 Pomocné proměnné 7

2.2.3 Formulace omezení 8

2.2.4 Účelová funkce 9

2.3 Shrnutí 9

3 Algoritmy okružního dopravního problému 11

3.1 Exaktní algoritmy 11

3.2 Heuristiky 12

3.2.1 Metoda Clarke-Wright 12

3.2.2 Metoda nejbližšího souseda 15

3.2.3 Vkládací metody 18

3.3 Metaheuristiky 21

3.3.1 Tabu search 21

3.3.2 Genetické algoritmy 23

3.4 Shrnutí 23

4 Přístup k návrhu a implementaci 24

4.1 Softwarová knihovna jako řešení ODP 24

4.2 Přístupy k řešení softwarových knihoven 25

4.2.1 Typy knihoven 25

4.2.2 Principy 25

4.2.3 Metodika 26

4.2.4 Modelování ... *27*
4.2.5 Programovací jazyk ... *27*
4.2.6 Aplikování principů ... *27*
4.3 SHRNUTÍ ... 28
5 Charakteristika knihovny ... 30
5.1 DOKUMENTACE NÁVRHU ... 30
5.1.1 Route ... *30*
5.1.2 Destination ... *31*
5.1.3 Car ... *32*
5.1.4 Location ... *33*
5.1.5 Evaluator ... *34*
5.1.6 Restriction ... *35*
5.1.7 Algorithm ... *36*
5.1.8 Input/Output ... *37*
5.2 DOKUMENTACE IMPLEMENTACE ... 38
5.2.1 Nástroje ... *38*
5.2.2 Publikování knihovny ... *39*
5.3 VALIDACE KNIHOVNY ... 39
5.3.1 Jednotkové testování ... *39*
5.3.2 Benchmark ... *40*
6 Využití knihovny ... 43
6.1 SCÉNÁŘ POUŽITÍ KNIHOVNY ... 43
6.1.1 Načtení vstupních dat ... *43*
6.1.2 Výpočet ... *44*
6.1.3 Export řešení ... *44*
6.2 UKÁZKA VYUŽITÍ ... 45
6.3 PODMÍNKY APLIKACE ... 46
7 Závěr ... 48
Citovaná literatura ... 50
Terminologický slovník ... 53

1 Úvod

V praxi se velmi často setkáváme s různými typy dopravních problémů. Tyto problémy lze chápat především jako rozvoz zboží od dodavatelů k odběratelům s cílem minimalizace distribučních nákladů. Navíc při řešení reálných dopravních problémů jsou velmi často zavedeny další restrikce (jako kapacita vozidel a objednávek, časová okna, maximální provozní doba vozidel a další speciální distribuční podmínky), které omezují množinu přípustných řešení problémů.

V současné době se stále více zvyšuje poptávka po technologiích, které by dokázaly tyto problémy řešit programově, nejlépe za předpokladu jednoduché interakce s uživatelem. Přirozeně se tedy rýsuje požadavek pro uživatelsky jednoduchý a přitom architektonicky promyšlený framework pro řešení těchto problémů, který by byl lehce rozšiřitelný o nové metody a postupy. Současný trh nabízí některá cloudová řešení, která přizpůsobuje a konkretizuje přesně dle zákazníka, nicméně tato řešení jsou pro malé odběratele nevhodná z důvodu ceny a jsou určena především pro dlouhodobou spolupráci, nikoli pro jednoduché a rychlé využití. Lze zmínit například společnost DIGITECH a jejich produkt PLANTOUR nebo produkt TASHA od společnosti SolverTech.

V této práci se věnujeme návrhu a implementaci knihovny pro řešení okružního dopravního problému.

1.1 Formulace problému

Problematiku klasického dopravního problému formuloval již F. L. Hitchcock v roce 1941, kdy navrhnul i jednoduchou metodu jeho řešení (1), nicméně podrobněji byla tato zkoumána až G. Dantzigem, který ve své publikaci *Application of the Simplex Method to a Transportation Problem* (2) popsal upravenou simplexovou metodu a definoval tak jednu z nejpoužívanějších metod pro řešení klasického dopravního problému.

V současné době je popsáno mnoho stochastických a nedeterministických metod pro řešení klasického dopravního problému, nicméně při zavedení distribučních restrikcí pro

řešení reálných problémů jsou tyto metody obtížně aplikovatelné (3). Také metody klastrování a prořezávání se zdají být nevhodné z důvodu vysoké výpočetní složitosti.

Konstrukce a implementace knihovny pro řešení tohoto problému předpokládá především předem dobře analyzovaný návrh všech struktur a korektnost celé metodiky návrhu. Zkoumané a popsané metody pro řešení okružního dopravního problému nelze obecně generalizovat, neboť každá metoda je strukturně specifická. Je proto nutné uvažovat takové rozhraní, které bude natolik obecné, že bude možné ho nasadit na většinu uvažovaných metod.

1.2 Cíle práce

Na trhu jsou prezentovány některé specializované komerční nástroje pro optimalizaci distribučních nákladů, které jsou mnohdy součástí větších frameworků, které dále zajišťují služby jako například sledování vozidel, umístění skladů, výkazy jízd atp.

Typickým příkladem jsou produkty společnosti DIGITECH, kde produkt PLANTOUR slouží pro řízení rozvozu a optimalizaci tras a další produkty (TRACKMANAGER, STANDORT aj.) zajišťují dodatečné služby jako například výše zmíněné umístění skladů, sledování techniků, vozidel aj. Tyto produkty vždy reflektují několik zvolených přístupů k řešení problému a konkretizují ho dle zákazníka.

Cílem této práce je návrh a implementace obecné, logické a rozšiřitelné softwarové knihovny, která řeší problematiku okružního dopravního problému. Lze uvažovat i nasazení softwarového díla pro vědecké výzkumy nových metod pro řešení tohoto problému nebo jeho variant. Naplnění cíle je dokumentováno v práci následujícím způsobem.

Nejprve je v kapitole 2 rozebrána problematika okružního problému a přístupy k jeho řešení (viz kapitola 3). V kapitole 4 je popsán zvolený přístup k návrhu a implementaci knihovny. Návrh knihovny je dokumentován v kapitole 5.1. Charakteristika implementace knihovny je uvedena v kapitole 5.2 a její validace, reprezentovaná jednotkovým testováním a benchmarkingem je uvedena v kapitole ?. Konečně v kapitole 0 je popsáno typické využití knihovny.

2 Okružní dopravní problém

V této kapitole se zaměříme na definici a formální popis okružního dopravního problému (dále jen ODP) a vymezíme jej oproti podobným NP-úplným problémům.

V práci se zabýváme ODP v podmínkách časových oken, tudíž součástí této kapitoly je charakteristika tohoto problému, včetně definice matematického modelu ODP rozšířeného o podmínky časových oken.

2.1 Definice

Okružní dopravní problém (*Vehicle Routing Problem*) je také často mylně zaměňován s *Úlohou obchodního cestujícího (Travelling Salesman Problem)* nebo s *Problémem čínského listonoše (The Chinese Postman Problem)*, nicméně jedná se o rozdílné problémy. Okružní dopravní problém je pouhým rozšířením Úlohy obchodního cestujícího a Problém čínského listonoše uvažuje pokrytí všech hran grafu, nikoli vrcholů.

2.1.1 Úloha obchodního cestujícího

Jedná se o obtížný diskrétní optimalizační problém, který byl popsán již v první polovině 19. století významnými matematiky W. R. Hamiltonem a T. Kirkmanem. Podrobněji se jím zabýval až Karl Menger na začátku 20. století a brzy nato Hassler Whitney z Princeton University, který ho ve svých publikacích definoval jako Travelling Salesman Problem (4). Později byl důkladně prostudován a jeho NP-úplnost dokázal Karp (5).

Tuto úlohu lze popsat jako hledání cyklu v ohodnoceném grafu tak, aby cyklus obsahoval stanovené vrcholy a zároveň měl co nejnižší celkové ohodnocení. Problém lze tedy chápat jako snahu cestujícího navštívit všechny požadované destinace s cílem najet co nejméně kilometrů a vrátit se do výchozího místa.

Formálněji lze tento problém definovat takto:

Mějme ohodnocený graf $G(V,E)$ s nezáporným ohodnocením $H: E \to \mathbb{R}_0^+$. Úlohou je najít posloupnost k vrcholů $A = \{v_{j_0}, v_{j_1}, \dots, v_{j_{k-1}}\}$ tak, aby hrany $h \in E$ mezi hledanými vrcholy $v_{j_i} \in V, i \in \{0,1,\dots,k-1\}$ tvořily v grafu G cyklus. Dále aby posloupnost A obsahovala každý vrchol z množiny V právě jednou a zároveň aby celkové ohodnocení hran mezi hledanými vrcholy bylo minimální. Jde tedy o úlohu:

$$\forall(w \in V)\ \exists(i \in \{0,\dots,k-1\}): \left(w = v_{j_i}\right), \tag{R 2.1}$$

$$\forall(i \in \{0,1,\dots,k-1\}): \left(\left(v_{j_i}; v_{j_{(i+1) mod\, k}}\right) \in E\right), \tag{R 2.2}$$

$$\sum_{i=0}^{k-1} h\left(\left(v_{j_i}; v_{j_{(i+1) mod\, k}}\right)\right) \underset{min}{\longrightarrow}, \tag{R 2.3}$$

kde $h\left(\left(v_i; v_j\right)\right) \in \mathbb{R}_0^+$ je ohodnocení hrany $\left(v_i; v_j\right) \in E$, pro $v_i, v_j \in V$.

Z hlediska složitosti je úloha obchodního cestujícího NP-úplný problém.

2.1.2 Problém čínského listonoše

Problém čínského listonoše byl popsán již v 60. letech 19. století významným čínským matematikem Kwan Mei-Ko (6).

Pokud se budeme držet výše definovaného značení, pak problémem čínského listonoše chápeme nalezení cyklické cesty v grafu G, která pokrývá všechny hrany grafu G a je z hlediska ohodnocení minimální.

Úlohou je najít posloupnost l hran $B = \{e_{j_1}, e_{j_2}, \dots, e_{j_l}\}$, $e_{j_i} \in E$, $i \in \{1,2,\dots,l\}$ tak, aby hrany B tvořily v grafu G cyklus, dále aby posloupnost B obsahovala každou hranu z E alespoň jednou a zároveň aby celkové ohodnocení hran B bylo minimální.

$$\forall(u \in E)\ \exists(i \in \{0,1,\dots,l-1\}): \left(u = e_{j_i}\right), \tag{R 2.4}$$

$$\forall(i \in \{0,1,\dots,l-1\}): \left(\left(e_{j_i} \cap e_{j_{(i+1) mod\, l}}\right) \neq \emptyset\right), \tag{R 2.5}$$

$$\sum_{i=0}^{l-1} h(e_{j_i}) \underset{min}{\longrightarrow}, \qquad \text{(R 2.6)}$$

NP-úplnost problému čínského listonoše je dokázána např. v (7).

2.1.3 Okružní dopravní problém

Okružní dopravní problém je rozšířením úlohy obchodního cestujícího popsané v kapitole 2.1.1. Pro úlohu obchodního cestujícího zavedeme kapacitní omezení okruhů a povolíme vícenásobné cykly, tzn. ve výsledné posloupnosti A může vzniknout více nezávislých cyklů.

Tato omezení lze chápat v kontextu reálných dopravních problémů jako problém distribuce zboží prostřednictvím vozidel pro jednotlivé objednávky. U vozidel i objednávek evidujeme kapacitní omezení a vozidla mohou absolvovat více okruhů z výchozího bodu.

Formální matematický model je uveden dále v kapitole 2.2.

2.1.4 Podmínky časových oken

S okružním dopravním problémem se setkáváme v celé řadě reálných dopravních situací, především lze zmínit logistické problémy distribuce zboží mezi dodavateli a odběrateli. Z tohoto důvodu se vedle obecného formulování problému nabízí mnoho variant rozšíření, které jsou těmito problémy inspirovány. Nejčastěji uvažovaným rozšířením jsou podmínky časových oken.

Pokud pro každou destinaci, kterou chceme navštívit, zavedeme časový interval, kdy má být obsloužena, hovoříme o ODP s podmínkou časových oken (dále jen ODPČO). Pro korektní konstrukci těchto intervalů je nutné uvažovat synchronizovaný čas mezi všemi prvky, které vstupují do řešení ODPČO.

Jestliže budeme uvažovat ODPČO v reálném rozvozním dopravním problému, pak lze zavést časová okna pro jednotlivé objednávky, vozidla, vozové parky i sklady. Tyto podmínky lze chápat jako provozní dobu skladů a vozových parků, pracovní dobu vozidel (resp. jejich řidičů) a požadovanou dobu obsluhy objednávek.

V poslední době je studiu lokálních heuristik pro řešení ODPČO věnována velká pozornost (8).

2.2 Matematický model

Matematickým modelem rozumíme abstraktní model používající matematický zápis pro vyjádření reálného problému. Tyto modely převádí složité lingvistické vyjádření podmínek a omezení na strojově zpracovatelný matematický jazyk, který lze dále efektivně integrovat do obecných algoritmů.

Pro okružní dopravní problém popsaný v kapitole 2.1.3 a 2.1.4 lze sestavit jednoduchý matematický model. Model lze formulovat tak, že standardní úlohu obchodního cestujícího popsanou v kapitole 2.1.1 rozšíříme dále o vícenásobné cykly, časová okna a kapacitní podmínky.

Nejprve zavedeme vstupní hodnoty zadání, což bude především vyjádření nákladového ohodnocení jednotlivých tras mezi destinacemi.

V modelu budeme využívat pomocné proměnné. Tyto pomocné proměnné nám slouží pouze k tomu, abychom mohli jednoduše matematicky popsat tento model, pro reálný problém jsou tyto hodnoty dopočítány nějakou z vybraných metod pro řešení okružního dopravního problému. V prvé řadě se jedná o proměnné, které budou charakterizovat pořadí, v jakém jsou navštíveny uvažované destinace v nalezeném řešení ODP. Dále využíváme předdefinovaný počet cyklů.

Konečně sestavíme matematické rovnice tak, aby všechny podmínky v rozšířeném okružním dopravním problému byly splněny.

Výchozí destinaci (chceme-li základnu) budeme indexovat 0 a nebudeme ji zahrnovat do destinací, které chceme navštívit.

2.2.1 Zadání

Hodnoty zadání lze definovat takto:

$N \in \mathbb{N}$	...	Počet všech destinací.
$T_0 \in \mathbb{R}_0^+$	...	Počáteční (výchozí) čas pro start úlohy.
$\pi \in \mathbb{R}^+$	...	Čas obsloužení objednávky. Pro jednoduchost uvažujme konstantu.
$O_i = \langle \mathbb{R}_0^+; \mathbb{R}_0^+ \rangle$ $i \in \{0,1,\dots,N\}$	...	Časové okno pro i-tou destinaci.
$C_{ij} \in \mathbb{R}_0^+$ $i,j \in \{0,1,\dots,N\}, C_{ij} = 0\ pro\ i = j$	...	Nákladové ohodnocení trasy mezi i-tou a j-tou destinací.
$T_{ij} \in \mathbb{N}$ $i,j \in \{0,1,\dots,N\}, T_{ij} = 0\ pro\ i = j$	...	Časové ohodnocení trasy mezi i-tou a j-tou destinací.

2.2.2 Pomocné proměnné

Dále definujme některé pomocné proměnné pro zjednodušení matematického zápisu.

$K \in \mathbb{N}$	...	Počet cyklů.
$X_{ij} \in \{0,1\}$ $i,j \in \{0,1,\dots,N\}, X_{ij} = 0\ pro\ i = j$	...	Pomocná proměnná, která vyjadřuje, zda mezi i-tou a j-tou destinací povede trasa v daném řešení úlohy. Povede-li, pole $X_{ij} = 1$, v opačném případě $X_{ij} = 0$.

$\delta_{iz} \in \{1,2,\dots,N\}$, $z \in \{1,2,\dots K\}, i \in \{1,2,\dots,N\}$, $\delta_{iz} \neq \delta_{jz}\ pro\ i \neq j$	...	Pomocná proměnná vyjadřující pořadí, v jakém je navštívena i-tá destinace v rámci z-tého okruhu.

2.2.3 Formulace omezení

Dále sestavíme rovnice, které zajistí splnění podmínek úlohy obchodního cestujícího. Je zřejmé, že každá destinace musí být navštívena právě jednou a proto i z každé destinace vede pouze jedna trasa. Pokud povolíme více cyklů, pak tuto podmínku nesplňuje pouze výchozí destinace (základna), pro kterou zavedeme podmínky zvlášť.

$$\sum_{i=0}^{N} X_{ij} = 1, \tag{R 2.7}$$

$$\sum_{j=0}^{N} X_{ij} = 1, \tag{R 2.8}$$

kde $i \in \{1,2,\dots,N\}$ a $j \in \{1,2,\dots,N\}$.

Pro výchozí destinaci (základnu) musí platit, že počet cest ze základny i do základny musí být roven počtu cyklů, tj. K.

$$\sum_{j=1}^{N} X_{0j} = K, \tag{R 2.9}$$

$$\sum_{i=1}^{N} X_{i0} = K. \tag{R 2.10}$$

Aby vznikly cykly, (kde základna bude vždy startovní i cílová destinace) je nutné zahrnout podmínku cyklu:

$$\delta_{iz} - \delta_{jz} + NX_{ij} \leq N - 1, \quad i,j \in \{1,2,\dots,N\}, i \neq j, z \in K. \tag{R 2.11}$$

Je zřejmé, že podmínka nabývá na účinnosti pouze pokud $X_{ij} > 0$. Jinak je podmínka splněna vždy. Pokud $X_{ij} > 0$, pak nutně i $\delta_{jz} > \delta_{iz}$. V jednom cyklu je vždy $\delta_{iz}, \delta_{jz} \in \{1, 2, \dots, P\}$. Odvození této podmínky ukázal Miller (9).

V neposlední řadě je třeba splnit podmínky časových oken.

Předpokládejme, že pro každou destinaci i sklad je vytvořeno korektní časové okno, tzn. jestliže označíme časové okno i-té destinace jako $O_i = \langle O_{i1}; O_{i2} \rangle$, kde $O_{i1}, O_{i2} \in \mathbb{R}_0^+$, pak platí $O_{i1} \leq O_{i2}$ a $T_0 \leq O_{i1}$ pro všechny $i \in \{1,2,\dots,N\}$. Dále označíme čas příjezdu vozidla do i-té destinace jako π_i, pak musí platit podmínka horní meze intervalu $\pi_i + \pi \leq O_{i2}$ pro všechny $i \in \{1,2,\dots,N\}$. Podmínka dolní meze intervalu $\pi_i \geq O_{i1}$ se mnohdy neuvažuje a poté je dovoleno čekání vozidla po čas $\Delta_i = O_{i1} - \pi_i$. Toto čekání může být penalizováno a zahrnuto v účelové funkci.

2.2.4 Účelová funkce

V původních výzkumech Solomona byla účelová funkce konstruována jako minimalizace počtu vozidel, což bylo odůvodněno faktem, že při optimalizaci celkové vzdálenosti je třeba využít vyšší počet vozidel (10) (11). My se budeme držet konstrukce na základě parametrického vyjádření obou kritérií, kterou navrhli Labadi et al (12). Tato konstrukce optimalizuje součet váženého počtu vozidel a vážené celkové vzdálenosti

$$\Theta \cdot K + \Phi \cdot \sum_{i=0}^{N} \sum_{j=0}^{N} C_{ij} X_{ij} \overset{min}{\longrightarrow}, \qquad \text{(R 2.12)}$$

kde $\Theta, \Phi \in \mathbb{R}_0^+$ jsou konstanty vážící daná kritéria.

2.3 Shrnutí

V předchozích podkapitolách byly vymezeny problémy podobné okružnímu dopravnímu problému. Dále byl zkonstruován matematický model pro exaktní definici okružního dopravního problému rozšířeného o podmínky časových oken.

Je zřejmé, že základní okružní dopravní problém popsaný v kapitole 2.1.3 lze rozšířit mnoha způsoby. Nejčastější variantou je rozšíření o kapacitní omezení a podmínky časových oken, tuto variantu jsme ukázali na modelu v rámci kapitoly 2.2. Kapacitní omezení jsou nejčastěji definována pro destinace a vozidla, nicméně lze je definovat i pro startovní destinaci aj. Stejně tak podmínky časových oken lze aplikovat jak pro destinace, tak pro vozidla.

V další kapitole věnované popisu vybraných algoritmů budeme zabývat pouze základním okružním dopravním problémem, nicméně při konstrukci knihovny je nutné architekturu všech objektů ODP přizpůsobit akceptaci dodatečných omezujících podmínek jako jsou například výše zmiňované kapacitní omezení a podmínky časových oken.

3 Algoritmy okružního dopravního problému

V následující kapitole je poskytnut přehled nejznámějších principů exaktních metod, heuristik a metaheuristik pro řešení okružního dopravního problému a okružního dopravního problému v podmínkách časových oken (dále jen ODPČO). Přehled heuristik a metaheuristik byl z velké části převzat z článků Bräysy a Gendreau (13) (14). Některé exaktní metody byly studovány z článků (15) (16).

3.1 Exaktní algoritmy

Exaktní algoritmy poskytují optimální řešení, nicméně jejich časová náročnost je v případě „velkých problémů“ neúnosná. Dle Fishera et al (16) lze exaktní metody dělit dle přístupu na dynamické programování, generování sloupců, metody založené na Langrangeově rozkladu a K-stromy.

Výzkumu v oblasti řešení ODP a ODPČO na základě dynamického programování se věnoval především Kolen et al (17), nicméně tento princip byl známý již v 50. Letech minulého století. Optimální trasy pro tyto problémy lze konstruovat až do počtu cca 15 destinací, neboť algoritmy dosahují vysoké časové složitosti.

Na konci 90. Let minulého století byly navrženy algoritmy založené na kombinaci lineárního programování a generování sloupců. Na základě výzkumu Desrochers et al v roce 1992 (18) byl popsán algoritmus, který řešil tyto problémy až pro 100 destinací. Tato metoda se v současnosti těší největší pozornosti ve vědeckém výzkumu.

Princip Langrangeova rozkladu je využíván v mnoha oborech matematiky a i pro tento výzkum bylo popsáno mnoho metod založených na Langrangeově rozkladu. Mezi známější patří metody od autorů Jörnsten (19) a Madsen (20). Vybrané metody založené na Langrangeově rozkladu dokáží řešit tyto problémy až pro 100 destinací.

V nedávné době se zaměřil výzkum i na metody založené na K-stromech. Poprvé je aplikoval Fisher v roce 1994 (21), nicméně další výzkum v této oblasti dosud není znám.

3.2 Heuristiky

Heuristický přístup k řešení okružního dopravního problému sice neposkytuje optimální řešení, nicméně díky jeho rychlosti a snadné adaptaci je mu stále věnována největší pozornost nejen v reálné praxi, ale především v teoretických výzkumech.

V následujících podkapitolách se zaměříme na obecné heuristiky, jejichž principy jsou využity při konstrukci konkrétních heuristik pro řešení ODP a ODPČO. Konkrétní heuristiky byly shrnuty v článku (13).

3.2.1 Metoda Clarke-Wright

Jedná se o jednu z nejznámějších heuristických metod pro řešení úlohy obchodního cestujícího, kterou v roce 1964 popsali Clarke a Wright (22).

Metoda byla vybrána pro svou implementační a časovou efektivitu. V rámci porovnání známých metod pro řešení okružního dopravního problému, které provedl RNDr. Petr Kučera ve své práci Metodologie řešení okružního dopravního problému, lze předpokládat, že tato metoda se zdá být vhodná pro vyhledání efektivních řešení „velkých"problemů (23).

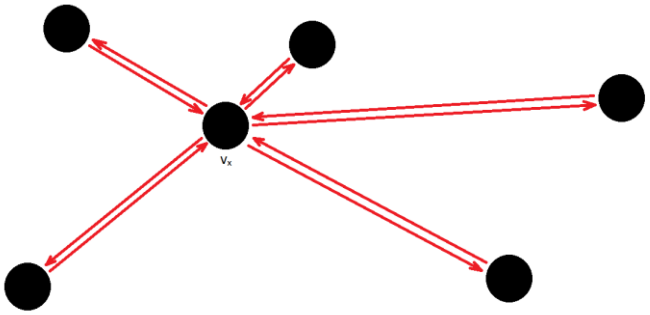
Do „výhodnostního koeficientu" (definován níže) lze také zakomponovat některá další omezení v rámci objednávky, příp. vozidla, a jednoduše tak metodu přizpůsobit konkrétnímu problému (např. zmiňovaným podmínkám časových oken).

Uvažujme úlohu obchodního cestujícího jako graf $G = (V; E)$, kde $V = \{v_1, v_2, \ldots, v_N\}$ jsou destinace a E hrany mezi nimi, $N \in \mathbb{N}$ je počet destinací. Ohodnocení hran $H: E \to \mathbb{R}_0^+$ budeme značit jako $h\left(\left(v_i; v_j\right)\right)$ pro hranu $\left(v_i; v_j\right) \in E$. Při popisu budeme značit trasu z v_i do v_j jako $v_i \to v_j$. Zavedeme termín „cena okruhu", což lze chápat jako součet vzdáleností všech tras daného okruhu.

V prvním kroku algoritmu je náhodně zvolena jedna destinace $v_x \in V$, kterou budeme dále nazývat výchozí. Touto destinací můžeme chápat sklad nebo továrnu, ze které rozvážíme zboží do cílových destinací. Tato destinace je tedy startovním i cílovým místem okruhu. Zavedeme také pomocné struktury $\mathcal{A}, \mathcal{B} \subseteq V$, které budou charakterizovat vrcholy, do nichž

vede trasa z výchozí destinace (v případě $\mathcal{A}$), resp. z nichž vede trasa do výchozí destinace (v případě $\mathcal{B}$). Na počátku volíme $\mathcal{A} = \mathcal{B} = \emptyset$.

Nyní zkonstruujeme elementární řešení problému, tzn. pro každé $v_i \in V, v_i \neq v_x$ přidáme do okruhu T trasu $v_x \rightarrow v_i$ a $v_i \rightarrow v_x$ (viz Obr. 3.1) a přidáme v_i do množiny vstupních krajních vrcholů $\mathcal{A}$ i do množiny výstupních krajních vrcholů $\mathcal{B}$. Nyní je tedy $\mathcal{A} = \mathcal{B} = V \setminus \{v_x\}$.



Obr. 3.1 Elementární řešení metody Clarke-Wrighta

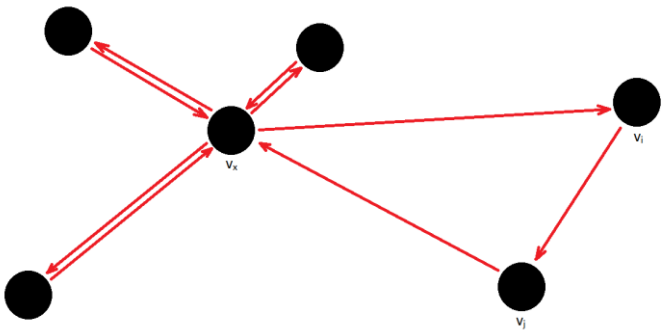
V dalším kroku metody určíme dvojici destinací (v_i, v_j), kde $v_i \neq v_x$, $v_j \neq v_x$ a $v_i \neq v_j$. Samotný výběr je založen na myšlence prioritního řazení takových dvojic destinací, kde vzdálenost mezi těmito destinacemi je výrazně nižší než vzdálenost každé z nich do destinace výchozí. Tento poměr vzdáleností charakterizujeme pomocí tzv. „výhodnostního koeficientu“ z_{ij} pro i-tou a j-tou destinaci:

$$z_{ij} = h\big((v_x; v_i)\big) + h\Big(\big(v_j; v_x\big)\Big) - h\Big(\big(v_i; v_j\big)\Big), \quad i, j \in \{1,2, \dots, N\}; i, j \neq x; i \neq j. \tag{R 3.1}$$

Spočítáme výhodnostní koeficienty pro všechny dvojice destinací (v_i, v_j), $i, j \in \{1, \dots, N\}; i, j \neq x; i \neq j$. Seřadíme dvojice destinací dle výhodnostních koeficientů sestupně. Nyní vybereme dvojici (v_i, v_j) s nejvyšším kladným výhodnostním koeficientem, kde $v_j \in \mathcal{A}, v_i \in \mathcal{B}$. Zároveň musí být splněna podmínka, že odebráním tras $v_i \rightarrow v_x$, $v_x \rightarrow v_j$ z okruhu T a přidáním trasy $v_i \rightarrow v_j$ do okruhu T nevznikne nový okruh, který by neobsahoval výchozí

vrchol $v_x \in V$. Pokud taková dvojice neexistuje, pak skončíme a výsledek prohlásíme za výsledné řešení.

Odebereme trasy $v_i \rightarrow v_x$, $v_x \rightarrow v_j$ a přidáme trasu $v_i \rightarrow v_j$ (viz Obr. 3.2). Odebereme v_i z množiny výstupních krajních vrcholů $\mathcal{B}$ a v_j z množiny vstupních krajních vrcholů $\mathcal{A}$.



Obr. 3.2 Přepojení tras pro dvojici destinací vi a vj

Nyní opět vybíráme další dvojici (v_i, v_j) s nejvyšším výhodnostním koeficientem, kde $v_j \in \mathcal{A}$ a $v_i \in \mathcal{B}$ tak, aby odebráním tras $v_i \rightarrow v_x$, $v_x \rightarrow v_j$ z okruhu T a přidáním trasy $v_i \rightarrow v_j$ do okruhu T nevznikl nový okruh, který by neobsahoval výchozí vrchol $v_x \in V$. Pokud taková dvojice existuje, opakujeme postup s přepojováním tras zmíněný v odstavci výše, jinak skončíme a okruh prohlásíme za výsledné řešení.

Výsledné řešení zkonstruované metodou Clarke-Wright může být složeno z více okruhů, z nichž každý začíná i končí ve výchozím vrcholu $v_x \in V$. Je to způsobeno tím, že výhodnostní koeficient pro určitou dvojici vrcholů může být i záporný. Přepojení tras dle této dvojice vrcholů je nežádoucí, neboť by zvýšilo cenu okruhu.

3.2.1.1 Popis algoritmu

Při popisu algoritmu se budeme držet značení definovaného výše.

I. Náhodně zvolíme výchozí uzel $v_x \in V$.Vytvoříme okruh $T = \{v_x \rightarrow v_i, v_i \rightarrow v_x\}$, $v_i \in V, i \in \{1,2,\dots,N\}, i \neq x. \mathcal{A} = \mathcal{B} = V \setminus \{v_x\}$.

II. Spočítáme výhodnostní koeficienty z_{ij} pro všechny dvojice destinací (v_i, v_j), kde $i, j \in \{1,2, \dots, N\}; i, j \neq x; i \neq j$:

$$z_{ij} = h\big((v_x; v_i)\big) + h\Big(\big(v_j; v_x\big)\Big) - h\Big(\big(v_i; v_j\big)\Big). \qquad \text{(R 3.2)}$$

III. Vybereme dvojici (v_i, v_j) s nejvyšším kladným výhodnostním koeficientem, kde $v_j \in \mathcal{A}$ a $v_i \in \mathcal{B}$ a zároveň odebráním tras $v_i \to v_x$, $v_x \to v_j$ z okruhu T a přidáním trasy $v_i \to v_j$ do okruhu T nevznikl nový okruh, který by neobsahoval výchozí vrchol $v_x \in V$. Pokud taková dvojice neexistuje, pak skončíme, a výsledek prohlásíme za výsledné řešení.

IV. $T := T \setminus \{v_i \to v_x, v_x \to v_j\} \cup \{v_i \to v_j\}$. $\mathcal{A} := \mathcal{A} \setminus \{v_j\}$, $\mathcal{B} := \mathcal{B} \setminus \{v_i\}$. Opakujeme krok III.

3.2.2 Metoda nejbližšího souseda

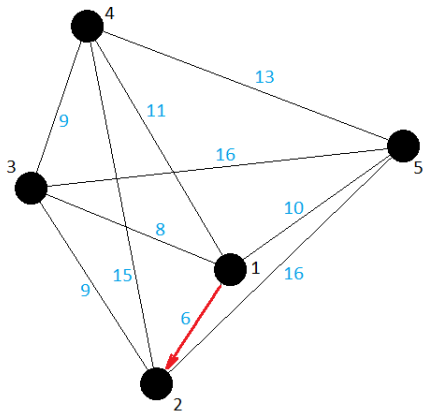
Tato metoda je považována za jednu z nejjednodušších metod pro řešení úlohy obchodního cestujícího, okružního dopravního problému a jeho variant. V zahraniční literatuře ji lze najít pod názvem „Nearest Neighbor method“ (24). Lze ji použít i pro víceokruhové a nesymetrické varianty úlohy obchodního cestujícího, nicméně na rozdíl od jiných metod zde není možné určit odhad odchylky daného řešení od optimálního.

Uvažujme problém jako graf $G = (V; E)$, kde $V = \{v_1, v_2, \dots, v_N\}$ jsou destinace, $N \in \mathbb{N}$ je počet destinací a E hrany mezi nimi. Ohodnocení hran $H: E \to \mathbb{R}_0^+$ budeme značit jako $h\Big(\big(v_i; v_j\big)\Big)$ pro hranu $(v_i; v_j) \in E$. Při popisu budeme značit trasu z v_i do v_j jako $v_i \to v_j$ a zavedeme $\mathcal{A}$ jako pomocný seznam použitých vrcholů. Okruh bude značit posloupnost tras: $T = \{v_m \to v_n, \dots\}$. Na počátku volíme $\mathcal{A} = \emptyset$.

Nejprve zvolíme náhodně výchozí destinaci $v_x \in V$, přidáme ji do seznamu použitých vrcholů $\mathcal{A}$ a poté určíme destinaci $v_i \in V, v_i \notin \mathcal{A}$ s nejnižším ohodnocením tak, aby platil vztah

$$\bigvee_{v_j \in V, v_j \notin \mathcal{A}} \Big(h\big((v_x; v_i)\big) \leq h\Big(\big(v_x; v_j\big)\Big)\Big) \qquad \text{(R 3.3)}$$

Přidáme v_i do seznamu použitých vrcholů $\mathcal{A}$ a přidáme do okruhu T trasu $v_x \rightarrow v_i$ (viz Obr. 3.3).



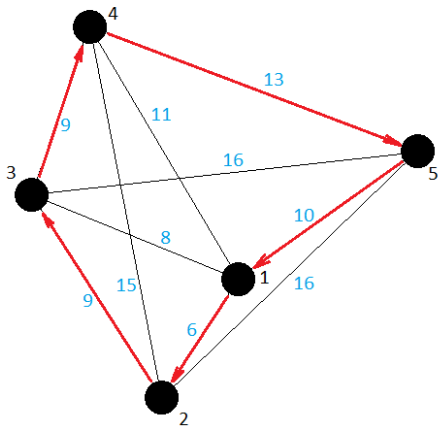
Obr. 3.3 První iterace metody nejbližšího souseda (pro výchozí vrchol 1)

Je zřejmé, že destinace v_i je v rámci ohodnocení nejbližší nenavštívenou destinací k destinaci výchozí. V každém dalším kroku algoritmu hledáme nejbližší nenavštívenou destinaci k destinaci předchozí a přidáme ji do okruhu. Označíme-li předchozí destinaci jako $v_j \in V, v_j \in \mathcal{A}$, pak následující destinace musí splňovat

$$\bigvee_{v_k \in V, v_k \notin \mathcal{A}} \left(h\left((v_i; v_j)\right) \leq h\left((v_i; v_k)\right) \right) \qquad \text{(R 3.4)}$$

Iterujeme, dokud nejsou použity všechny vrcholy, tj. $\mathcal{A} = V$.

Předpokládejme, že poslední vrchol, který přidáme do okruhu, je $v_u \in V$. Na závěr přidáme do okruhu T trasu $v_u \rightarrow v_x$, abychom dokončili cyklus (viz Obr. 3.4).



Obr. 3.4 Výsledné řešení dle metody nejbližšího souseda (pro výchozí vrchol 1)

3.2.2.1 Popis algoritmu

Při popisu se budeme držet značení definovaného výše.

I. Náhodně zvolíme výchozí vrchol $v_x \in V$, $\mathcal{A} = \{v_x\}$.

II. $(v_i \in V, v_i \notin \mathcal{A})\colon \bigvee_{v_j \in V, v_j \notin \mathcal{A}} \left(h\big((v_x; v_i)\big) \leq h\left((v_x; v_j)\right)\right)$.

III. $T \coloneqq T \cup \{v_x \to v_i\}$, $\mathcal{A} \coloneqq \mathcal{A} \cup \{v_i\}$.

IV. Pokud $\mathcal{A} = V$, pak přejdeme na krok 0, jinak označme poslední přidanou destinaci do okruhu jako $v_i \in V$ a poté $\big(v_j \in V, v_j \notin \mathcal{A}\big)\colon \bigvee_{v_k \in V, v_k \notin \mathcal{A}} \left(h\left((v_i; v_j)\right) \leq h\big((v_i; v_k)\big)\right)$.

V. $T \coloneqq T \cup \{v_i \to v_j\}$, $\mathcal{A} \coloneqq \mathcal{A} \cup \{v_j\}$. Pokud $\mathcal{A} \neq V$ přejdeme na krok IV, jinak přejdeme na krok 0.

Předpokládejme, že poslední vrchol, který jsme přidali do okruhu, je $v_u \in V$, $T \coloneqq T \cup \{v_u \to v_x\}$.

3.2.3 Vkládací metody

Bylo popsáno mnoho heuristik, které naleznou elementární okruh (většinou s jedinou a sice výchozí destinací) a poté do tohoto okruhu iterativně vkládají dosud nenavštívené destinace. S každou vloženou destinací tak rozšiřují okruh o další destinace, dokud okruh neobsahuje všechny destinace.

Tyto metody jsou obecně nazývány vkládacími a liší se způsobem vyhledání a přidání další destinace do okruhu. My se omezíme pouze na některé z těchto metod a to metodu nejbližšího přídavku, metodu nejbližšího vložení a metodu nejlevnějšího vložení (23).

Uvažujme problém jako graf $G = (V; E)$, kde $V = \{v_1, v_2, \dots, v_N\}$ jsou destinace, $N \in \mathbb{N}$ je počet destinací a E hrany mezi nimi. Ohodnocení hran $H: E \to \mathbb{R}_0^+$ budeme značit jako $h\left(\left(v_i; v_j\right)\right)$ pro hranu $\left(v_i; v_j\right) \in E$. Při popisu budeme značit trasu z v_i do v_j jako $v_i \to v_j$ a zavedeme $\mathcal{A}$ jako pomocný seznam použitých vrcholů v okruhu. Na počátku je $\mathcal{A} = \emptyset$.

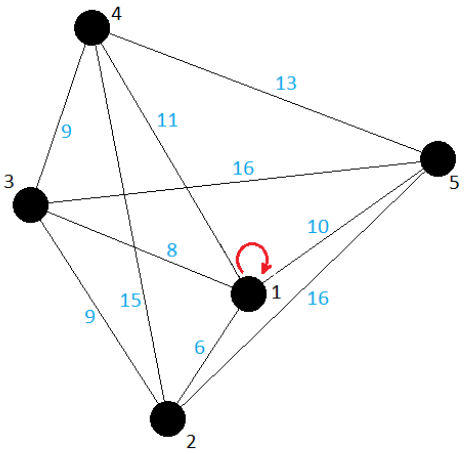
Pro okruh $\acute{T}$, jež obsahuje vrcholy $\acute{V} \subseteq V$, zavedeme speciální značení vzdálenosti vrcholu v_m od okruhu $\acute{T}$:

$$\left(v_m; \acute{T}\right) \coloneqq h\big((v_m; v_i)\big),$$

$$\forall\big(v_j \in \acute{V}\big): \left(h\big((v_m; v_i)\big) \leq h\left(\left(v_m; v_j\right)\right)\right), \qquad \text{(R 3.5)}$$

$$v_i \in \acute{V}, v_m \notin \acute{V}, v_m \in V.$$

Zvolíme náhodně výchozí vrchol $v_x \in V$ a vytvoříme elementární řešení (viz Obr. 3.5): Přidáme okruh pouze pro výchozí vrchol, tzn. $T = \{v_x \to v_x\}$. Přidáme v_x do seznamu použitých vrcholů $\mathcal{A}$.



Obr. 3.5 Elementární řešení vkládacích metod

Další postup je již specifický pro každou ze zmiňovaných metod.

3.2.3.1 Metoda nejbližšího přídavku

Metoda nejbližšího přídavku vyhledá takový vrchol $v_i \in V$, $v_i \notin \mathcal{A}$, že $(v_i; T)$ je minimální. Předpokládejme, že se jedná o vrchol $v_j \in \mathcal{A}$, který je nejblíže vrcholu $v_i \in V$, tzn. $(v_i; T) = h\left((v_i; v_j)\right)$. Dále přepokládejme, že z vrcholu v_j vystupuje trasa $v_j \rightarrow v_k$, kde $v_k \in \mathcal{A}$.

Z okruhu T odebereme trasu $v_j \rightarrow v_k$ a přidáme trasy $v_j \rightarrow v_i$ a $v_i \rightarrow v_k$. Nakonec přidáme vrchol v_i do seznamu použitých vrcholů $\mathcal{A}$.

3.2.3.2 Metoda nejbližšího vložení

Tato metoda je drobnou modifikací metody nejbližšího přídavku. Výběr přidávaného vrcholu probíhá stejně, modifikace se projeví až při napojení vrcholu.

Mějme vrchol $v_i \in V$, $v_i \notin \mathcal{A}$, takový že $(v_i; T)$ je minimální. Předpokládejme, že je to vrchol $v_j \in \mathcal{A}$, který je nejblíže vrcholu $v_i \in V$, tzn. $(v_i; T) = h\left((v_i; v_j)\right)$. Nyní hledáme v okruhu T trasu $v_k \rightarrow v_l$ takovou, aby pro všechny trasy $v_m \rightarrow v_n$ v okruhu T byla splněna podmínka

$$h\big((v_k; v_i)\big) + h\big((v_i; v_l)\big) - h\big((v_k; v_l)\big) \leq h\big((v_m; v_i)\big) + h\big((v_i; v_n)\big) - h\big((v_m; v_n)\big). \quad \text{(R 3.6)}$$

Vrchol v_k zařadíme mezi vrcholy v_k a v_l, tzn. odebereme z okruhu T trasu $v_k \to v_l$ a přidáme trasy $v_k \to v_i$ a $v_i \to v_l$. Nakonec přidáme vrchol v_i do seznamu použitých vrcholů $\mathcal{A}$.

3.2.3.3 Metoda nejlevnějšího vložení

Tato metoda je oproti předchozím metodám z kapitoly 3.2.3 nejefektivnější, ale implementačně nejnáročnější.

V okruhu T je třeba najít trasu $v_i \to v_j$ a vrchol $v_k \in V$, $v_k \notin \mathcal{A}$ takový, aby pro všechny ostatní trasy $v_m \to v_n$ v okruhu T byla splněna podmínka

$$h\big((v_i; v_k)\big) + h\left(\big(v_k; v_j\big)\right) - h\left(\big(v_i; v_j\big)\right) \leq h\big((v_m; v_k)\big) + h\big((v_k; v_n)\big) - h\big((v_m; v_n)\big) \quad \text{(R 3.7)}$$

Vrchol v_k zařadíme mezi vrcholy v_i a v_j, tzn. odebereme trasu $v_i \to v_j$ z okruhu T a přidáme trasy $v_i \to v_k$ a $v_k \to v_j$. Nakonec přidáme vrchol v_k do seznamu použitých vrcholů $\mathcal{A}$.

3.2.3.4 Obecný algoritmus vkládacích metod

Při popisu algoritmu se budeme držet značení použitého výše.

I. Zvolíme výchozí vrchol $v_x \in V$ a vytvoříme elementární řešení, $T = \{v_x \to v_x\}$, $\mathcal{A} = \{v_x\}$ (viz Obr. 3.5).

II. Dle vybrané metody pro přidání vrcholu určíme vrchol $v_k \in V$, $v_k \notin \mathcal{A}$ a hranu $v_i \to v_j \in T$.

III. $T \coloneqq T \setminus \{v_i \to v_j\} \cup \{v_i \to v_k, v_k \to v_j\}$.

$\mathcal{A} \coloneqq \mathcal{A} \cup \{v_k\}$. Pokud $\mathcal{A} \neq V$, přejdeme na bod II, jinak skončíme a prohlásíme okruh T za výsledné řešení.

3.3 Metaheuristiky

Metaheuristikou obecně chápeme proceduru, která může interně využívat principy heuristik k dosažení výchozích řešení ve svém paměťovém prostoru, které dále modifikuje příp. mutuje k dosažení nového řešení.

Jak již bylo zmíněno, výchozí paměťový prostor je většinou naplněn řešeními získanými na základě různých heuristik. Některé z nich jsou popsány v rámci kapitoly 3.2.

Metaheuristiky lze dělit dle principu modifikace vybraných řešení (získávání nové řešení), příp. dle metody dosažení výchozích řešení. My se zde omezíme pouze na principy Tabu search a genetických algoritmů, přehled dalších algoritmů a konkrétních metod lze nalézt v (14).

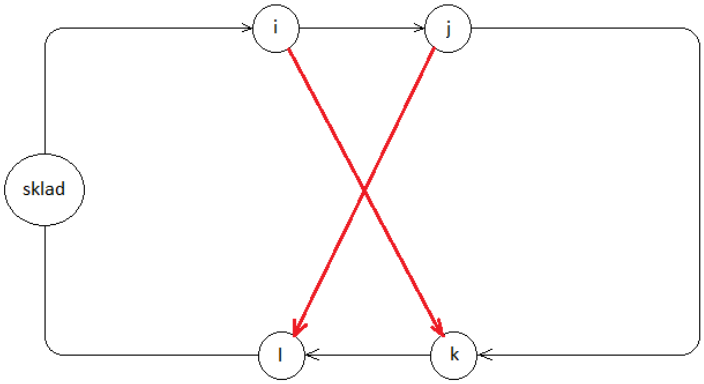
3.3.1 Tabu search

Tabu search je lokální vyhledávající metaheuristika, kterou popsal a studoval Glover již v roce 1986 (25). Jedná se o proceduru, která prozkoumává uložená řešení (tzv. paměťový prostor) a vybrané řešení s nahradí nejlepším řešením z jeho lokálního okolí. Lokálním okolím chápeme množinu řešení, které vzniknou z řešení s jednoduchým přepojením tras. Detaily a konkrétní implementace této metody lze nalézt např. v (26) (27).

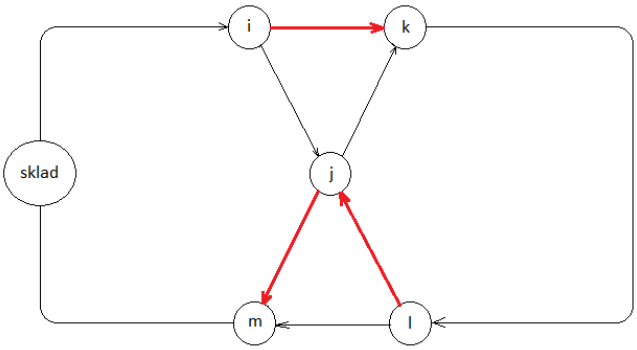
3.3.1.1 Lokální okolí

V následující podkapitole popíšeme některé varianty lokální okolí vybraného řešení s.

Nejjednodušším lokálním okolím vybraného řešení s je k-výměna (viz Obr. 3.6) nebo relokace (viz Obr. 3.7). K-výměnu můžeme definovat jako množinu řešení získanou přepojením k cest ve vybraném řešení s, zatímco relokace je množina řešení získaná přepojením pouze jedné destinace (lze tedy říci, že 1-výměna je relokace).



Obr. 3.6 Ilustrační příklad 2-výměny



Obr. 3.7 Ilustrační příklad relokace

Další příklady lokálního okolí a jejich ilustrace lze nalézt např. v (28).

3.3.1.2 Lokální prohledávání

Tabu search prohledává množinu řešení (tzv. paměť nebo prostor řešení) a vybírá řešení s, které nahradí zvoleným řešení z lokálního okolí s. Vybírání řešení s většinou probíhá čistě náhodně tak, aby se eliminovala možnost vzniku lokálního optima. Také ne vždy je vybíráno nejlepší řešení z lokálního okolí, neboť tak by procedura mohla snadno iterovat do lokálního optima.

Abychom se vyhnuli zacyklení procedury, lze dočasně zakázat modifikaci nebo vybírání některých řešení nebo atributů (tzv. „tabu“). Status „tabu“ může být vymazán například při dosažení nejlepšího řešení z uvažovaného prostoru.

Bylo popsáno mnoho modifikací a variant tohoto algoritmu, přehledně jsou shrnuty v publikaci (14).

3.3.2 Genetické algoritmy

Genetickým algoritmem v tomto případě chápeme adaptivní heuristickou metodu založenou na populační genetice. Základní koncept této techniky popsal již v roce 1975 Holland (29), nicméně praktické využití této techniky testoval až De Jong (30) a Goldberg (31).

Techniku lze jednoduše popsat jako křížení (mutace) vybraných dvou řešení m a n z paměťového prostoru procedury, na jehož základě vznikne nové řešení p. Toto řešení je buď zahozeno, nebo zařazeno do paměťového prostoru procedury. Iterativně vytváříme nová řešení a obměňujeme paměťový prostor procedury.

Thangian et al byl první, kdo aplikoval techniku genetických algoritmů na okružní dopravní problém v podmínkách časových oken (32). Podrobnější popis výzkumu v oblasti genetických algoritmů pro ODPČO lze nalézt v (14).

3.4 Shrnutí

V literatuře bylo popsáno mnoho algoritmů a metod pro řešení okružního dopravního problému. Každý z těchto algoritmů reflektuje některou variantu okružního dopravního problému a pro tu funguje korektně a efektivně. Největšího vědeckého zájmu se v současné době těší ODP varianta s kapacitním omezením a podmínkami časových oken (13) (14).

V dalších kapitolách se věnujeme návrhu obecné rozšiřitelné knihovny pro řešení ODP, která bude schopna řešit různé varianty ODP. Při návrhu knihovny uvažujeme základní variantu ODP popsanou v podkapitole 2.1.3, ale také variantu v podmínkách časových oken, jejíž model je popsán v podkapitole 2.2.

4 Přístup k návrhu a implementaci

Tato kapitola se věnuje přístupu k návrhu a implementaci softwarové knihovny.

Nejprve v podkapitole 4.1 jsou diskutovány důvody, proč problém ODP řešíme jako softwarovou knihovnu a nikoli jako cloudovou službu. V podkapitole 4.2 se věnujeme přístupům k řešení softwarových knihoven včetně diskuze vhodného programovacího jazyka a platformy. Konečně v podkapitole 4.3 dokumentujeme rozvržení fází vývoje softwarové knihovny.

4.1 Softwarová knihovna jako řešení ODP

Okružní dopravní problém je komplexní problém, který je popsán v mnoha variantách (viz kapitola 2). Z tohoto důvodu současný trh nabízí cloudová řešení, která customizují své řešení dle přání a požadavků zákazníka. Typickým příkladem jsou produkty společnosti DIGITECH, konkrétně produkt PLANTOUR.

Pokud chceme reflektovat požadavky open source rozšiřitelného a jednoduchého řešení, pak je nutné vzdát se možnosti cloudového řešení, neboť cloudové řešení přepokládá vzdálenou obsluhu a customizaci řešení dle konkrétního uživatele.

Jestliže nebudeme uvažovat cloudové řešení problému, pak je nutné definovat společné rozhraní okružního dopravního problému včetně jeho nejznámějších variant a navrhnout strukturu, která bude rozšiřitelná tak, aby mohla být jednoduše nastavitelná uživatelem výsledného systému pro akceptaci konkrétních variant ODP.

Je zřejmé, že zde se dostáváme do rozporu s požadavkem jednoduchosti, neboť při každém využití výše zmíněné struktury je nutné parametrizovat ji dle specifických požadavků varianty ODP.

Cílem práce může být mimo jiné i nasazení softwarového díla pro vědecké výzkumy nových metod pro řešení ODP, proto parametrizace struktury je nutnou podmínkou pro zachování obecnosti tohoto řešení. Při zachování využitelnosti knihovny ji lze jednoduše rozšířit o další algoritmy a ověřit tak jejich korektnost.

Na základě výše zmíněných úvah požadavků byla zvolena softwarová knihovna jako ideální struktura pro softwarové řešení ODP.

4.2 Přístupy k řešení softwarových knihoven

Pojmem softwarová knihovna v informatice nejčastěji chápeme označení pro soubor funkcí a procedur (v objektovém programování též objektů, datových typů a zdrojů), který může být sdílen více počítačovými programy. Knihovna usnadňuje programátorovi tvorbu zdrojového kódu tím, že umožňuje použít již vytvořený kód i v jiných programech. Knihovna navenek poskytuje své služby prostřednictvím navrženého API (aplikační rozhraní), což jsou názvy funkcí (včetně popisu jejich činnosti), předávané parametry a návratové hodnoty. Knihovny lze rozdělit podle vazby na program, který je používá, na statické a dynamické. Z hlediska práce s kódem knihovny v operační paměti je dělíme na sdílené a nesdílené.

4.2.1 Typy knihoven

Z technického hlediska je možné rozdělit knihovny podle způsobu propojení s programem, který je bude využívat:

- statická knihovna (anglicky static linking library)
- dynamická knihovna (anglicky dynamic linking library)

Druhé možné rozdělení je z hlediska možnosti sdílení kódu knihovny mezi více běžícími programy (tj. procesy):

- sdílená knihovna (anglicky shared library)
- nesdílená knihovna

4.2.2 Principy

Součástí návrhu většího softwarového projektu je obvykle i oddělení části funkčnosti do jedné nebo několika samostatných komponent nebo dalších knihoven. Při návrhu knihovny se berou v úvahu především některé známé vlastnosti z objektově orientovaného programování:

- Zapouzdřenost – komplexní funkčnost knihovny by měla být dostupná přes jednoduché a snadno použitelné rozhraní. Na hotovou knihovnu se pak dá dívat jako na černou skříňku.
- Znovupoužitelnost – dobře navržená knihovna může dobře posloužit při vývoji budoucích projektů, kdy bude potřeba stejná nebo podobná funkčnost.

4.2.3 Metodika

V současné době rozvoje softwarového inženýrství je popsáno velké množství různých metodik pro tvorbu softwarových systémů (33), nicméně tyto postupy jsou určeny především k tvorbě rozsáhlých podnikových systémů, kde jsme schopni modelovat podnikové procesy a postupy. Návrh systémů je poté tvořen na základě analýzy a modelování jednotlivých procesů a artefaktů (34). Artefaktem zde chápeme softwarový prvek, který jsme schopni identifikovat a který poskytuje určité rozhraní a služby.

Artefaktem lze chápat i softwarovou knihovnu, která slouží pro řešení určitého problému. Tyto artefakty jsou tvořeny na základě přístupů objektově orientovaného programování, díky nimž zajistíme zapouzdřenost a znovupoužitelnost celého artefaktu.

Objektově orientované programování předpokládá následující programátorské principy (34):

- Abstrakce
- Polymorfismus
- Dědičnost
- Zapouzdření

Tyto principy jsou základní stavební kameny pro znovupoužitelné artefakty.

4.2.4 Modelování

Návrh systémů je tvořen na základě analýzy a modelování jednotlivých procesů a artefaktů. Je definováno velké množství nástrojů pro modelování, nicméně jedním z nejpoužívanějších formátů pro modelování softwarových systémů je formát UML (35).

UML, Unified Modeling Language je v softwarovém inženýrství grafický jazyk pro vizualizaci, specifikaci, navrhování a dokumentaci programových systémů. UML nabízí standardní způsob zápisu jak návrhů systému včetně konceptuálních prvků jako jsou business procesy a systémové funkce, tak konkrétních prvků jako jsou příkazy programovacího jazyka, databázová schémata a znovupoužitelné programové komponenty. UML také podporuje objektově orientovaný přístup k analýze, návrhu a popisu programových systémů.

4.2.5 Programovací jazyk

Návrh softwarové knihovny je veden jako objektově orientovaný model, tudíž pro efektivní implementaci celého návrhu jsou uvažovány objektově orientované programovací jazyky, které poskytují rozumnou časovou složitost i snadnou rozšiřitelnost celé knihovny.

Z důvodu platformní nezávislosti byl vybrán jazyk Java, neboť lze předpokládat především externí využití knihovny a tudíž platformní dependence by v tomto případě byly nežádoucí.

Knihovnu budeme publikovat ve formátu Java Archive (.jar), který lze spustit v rámci virtuálního stroje JVM prakticky na všech známých platformách.

Pro kompilaci zdrojového kódu využíváme verzi Java SE 1.8.

4.2.6 Aplikování principů

Při návrhu knihovny byl problém popsaný v rámci kapitoly 2 rozložen na dílčí podproblémy, které chápeme jako nezávislé problémy.

Tyto podproblémy byly navrhnuty na základě studia jednotlivých variant ODP a vymezení jejich společné struktury. Budeme se držet anglických termínů pro označení těchto

podproblémů, neboť v dalších částech práce budeme těmito názvy označovat i jednotlivé package v rámci implementace knihovny a tudíž by české názvosloví bylo nežádoucí.

Všechny tyto podproblémy byly identifikovány v rámci studia podkapitoly 2.2.

Tab. 4.1 Tabulka charakteristiky podproblémů

Název podproblému	Teoretická charakteristika
Route	Problém charakteristiky okruhu, posloupnosti destinací a operací s nimi.
Destination	Základní problém charakteristiky destinace a uchování dodatečných podmínek (kapacitní omezení, časová okna).
Car	Problém charakteristiky vozidla, dodatečných podmínek, přidávání okruhů.
Location	Charakteristika lokace. Uvažujeme především u destinace.
Evaluator	Problém ohodnocení vzdálenosti – nákladové i časové.
Restriction	Stěžejní problém definování a akceptování dodatečných podmínek pro okruh, destinace, vozidla aj.
Algorithm	Problém charakteristiky obecné struktury algoritmu.
Input/Output	Problém čtení a zapisování vstupních a výstupních hodnot.

Jednotlivé podproblémy jsou detailně studovány v rámci fáze návrhu v podkapitole 5.1.

4.3 Shrnutí

Na základě studia výše zmíněných přístupů k řešení softwarových knihoven byly navrženy hlavní tři fáze vývoje. Tyto fáze vycházejí z obecné metodiky pro vývoj artefaktů softwarových systémů (33).

Pro korektní průběh návrhu a implementace definujeme 3 fáze:

- Návrh
- Implementace

- Validace

Návrhová fáze slouží k modelování struktur a vazeb, které jsme matematicky popsali v rámci kapitoly 2. Tato fáze je čistě analyzační a návrhová. V rámci této fáze navrhneme Class Diagramy jednotlivých oddílů celé knihovny v jazyce UML a popíšeme jednotlivé vazby. Na tuto fázi navazuje fáze implementační, kdy implementujeme struktury dle navrženého modelu. Tato fáze je stěžejní fází celé knihovny, neboť struktura rozhraní knihovny musí odpovídat požadavkům a cílům, které jsme definovali v úvodní kapitole 1.

Fází implementační rozumíme implementaci modelů, které jsme navrhli v rámci fáze návrhové. Tyto modely definují rozhraní a komunikaci mezi jednotlivými strukturami. Pro fázi implementační je stěžejním bodem implementace zvolených algoritmů a heuristik. Tyto metody jsou nasazeny na obecné rozhraní algoritmu pro řešení okružního dopravního problému definované v rámci fáze návrhové.

Ve fázi validační je v prvé řadě implementace validována na základě JUnit testů, které testují každý z podproblémů navržených v rámci kapitoly 4.2.6. Tyto testy ověřují jednotkové chování struktur. Vývojové a komplexní testování pak zajišťuje korektnost implementace knihovny na sadě dat pro ODPČO, kterou publikoval Solomon. Tato sada je v současné době používána jako benchmark pro výzkum nových heuristik a algoritmů pro řešení ODPČO. Tato fáze zaručí, že celá knihovna včetně implementovaných algoritmů a heuristik funguje korektně.

Detailní generovaná vývojová dokumentace softwarové knihovny je obsažena na přiloženém CD.

5 Charakteristika knihovny

Následující kapitola se věnuje dokumentaci jednotlivých fází vývoje knihovny.

Nejprve dokumentujeme samotný návrh knihovny včetně modelování struktur, class diagramů a rozhraní v kapitole 5.1. Další část kapitoly je věnována fázi validační (viz kapitola 5.2). Nakonec v kapitole dokumentujeme validační fázi knihovny.

5.1 Dokumentace návrhu

Dle kapitoly 4.2.6 jsme problém rozdělili na několik podproblémů, které budeme řešit separovaně. Ve výsledné knihovny budou tyto podproblémy odděleny do jednotlivých package.

Struktura podproblémů:

- Route
- Destination
- Car
- Location
- Evaluator
- Restriction
- Algorithm
- Input/Output

Jednotlivé podproblémy jsme modelovali v jazyce UML a jejich návrh popíšeme podrobněji v dalších částech kapitoly.

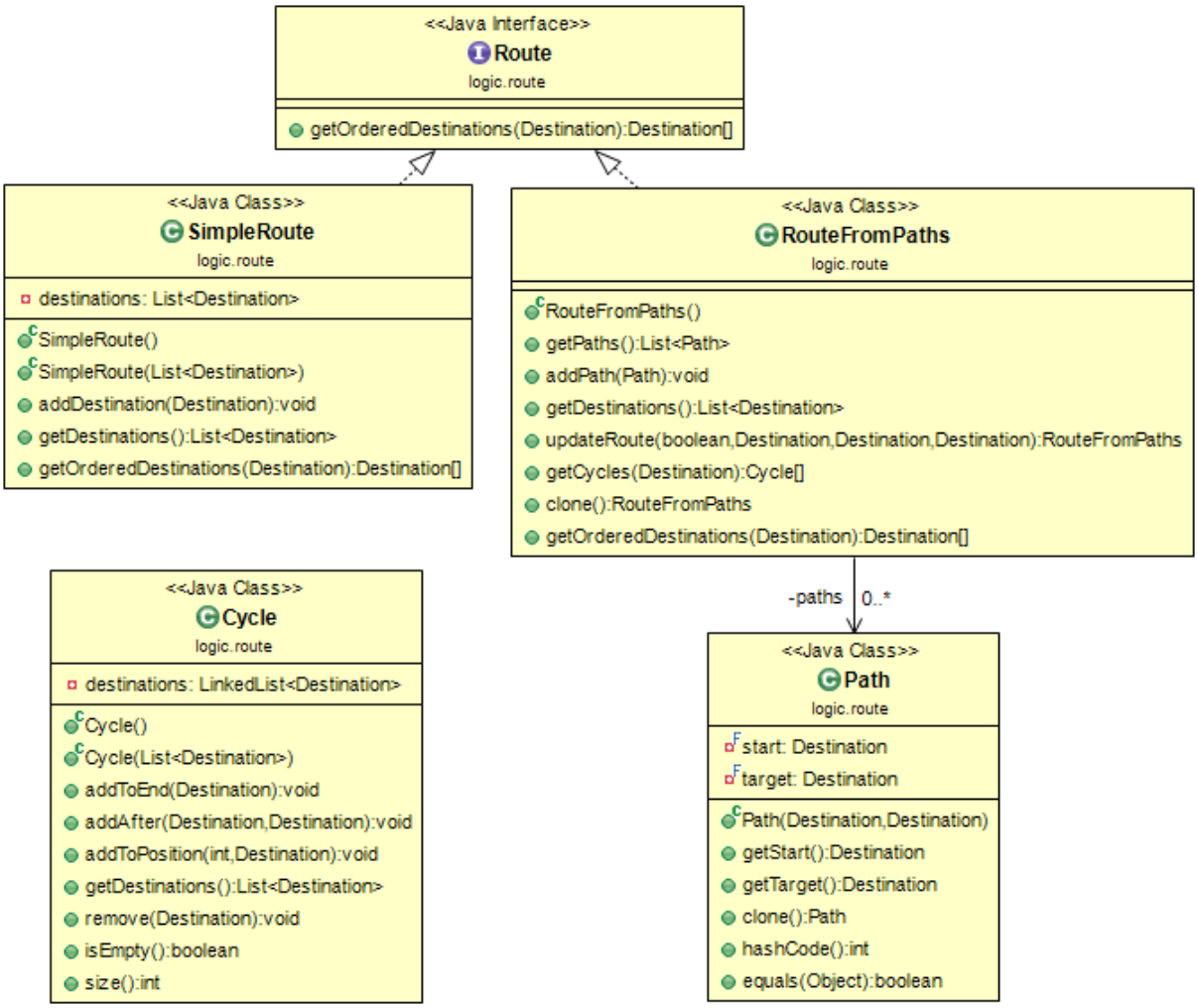
5.1.1 Route

Tento podproblém se týká návrhu struktur okruhu.

Okruhem chápeme posloupnost destinací, kde nás zajímá především pořadí jednotlivých destinací v okruhu, případně startovní (výchozí) destinace.

Okruh může být určen několika způsoby. V této práci uvažujeme řazenou posloupnost destinací, případně seznam hran mezi destinacemi společně s určením startovní (výchozí) destinace.

Na následujícím Obr. 5.1 je navržena struktura okruhu.



Obr. 5.1 Class Diagram Route

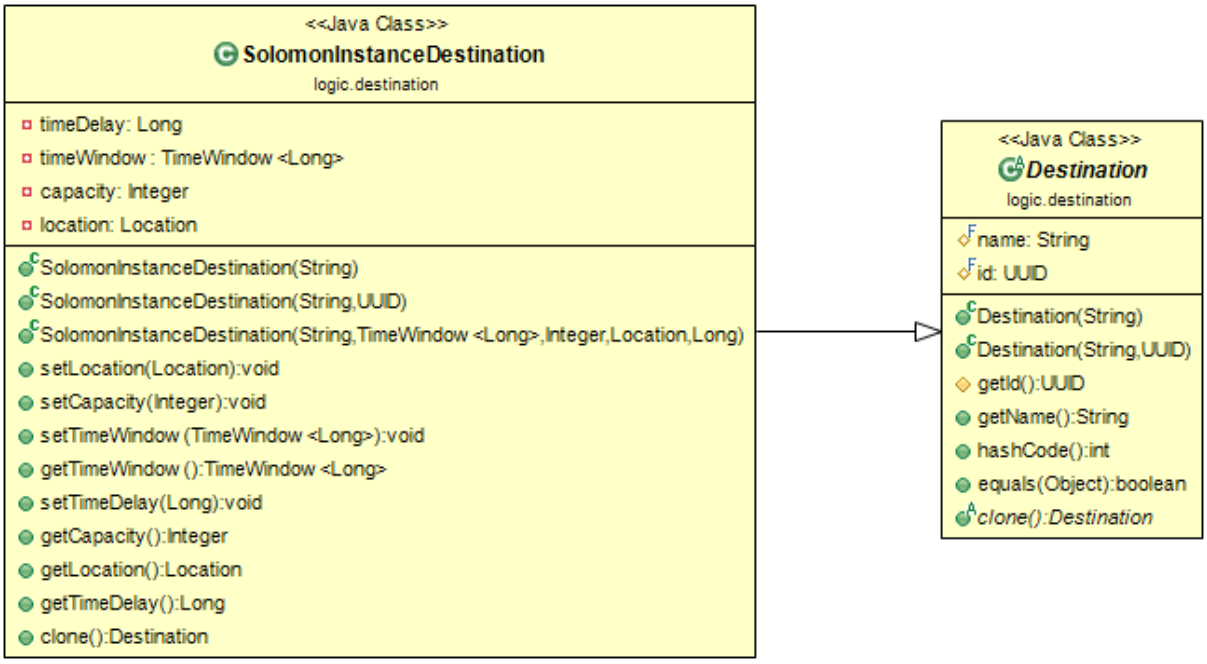
Struktura okruhu je jedním ze stěžejních struktur celé knihovny, neboť konstrukce výsledných okruhů a řazení jednotlivých destinací určuje sumu nákladů pro daný okružní dopravní problém.

5.1.2 Destination

Problémem Destination chápeme návrh struktur, které se týkají destinace.

Destinaci lze chápat jako jednotku okruhu. V rámci destinace mohou být definovány některé restrikce pro okruh, jako například kapacitní omezení, časové okno aj. Destinace musí být navržena tak, aby rozhraní umožňovalo jednoznačně identifikovat destinaci v rámci okruhu a určit tak náklady pro jednotlivé přesuny mezi destinacemi.

Na následujícím Obr. 5.2 je navržena struktura destinace.



Obr. 5.2 Class Diagram Destination

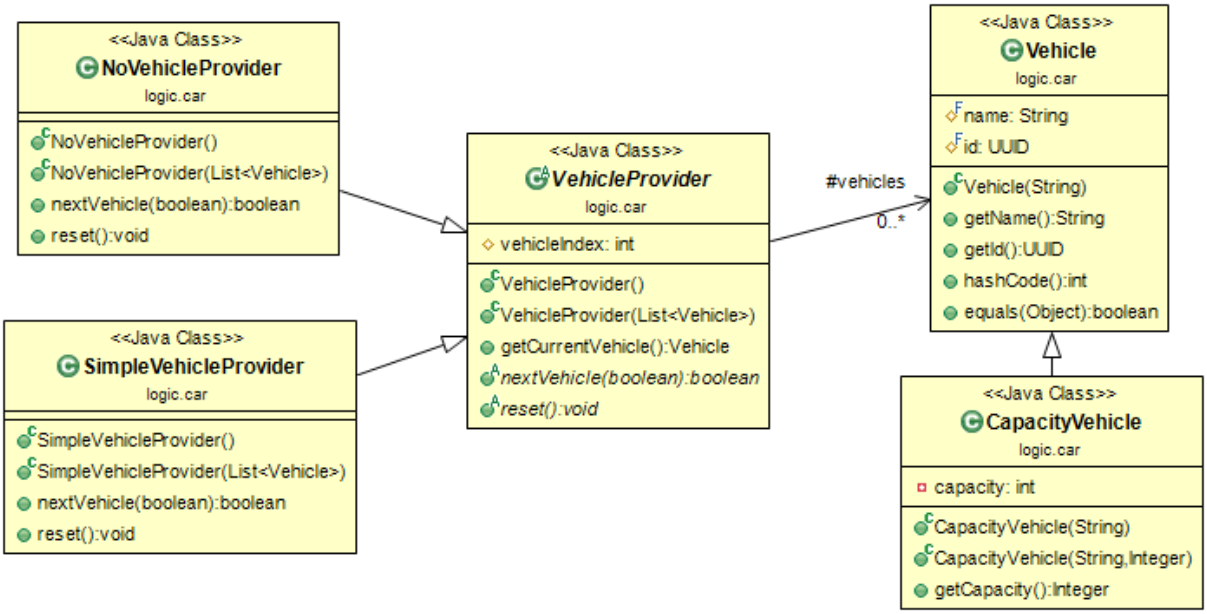
Struktura destinace je navržena tak, že destinace je identifikována názvem a unikátním Java UUID identifikátorem. Porovnání dvou destinací pak proběhne tak, že jsou porovnány identifikátory.

5.1.3 Car

Problém Car se týká navržení struktur pro definování vozidla.

K navrženému okruhu lze přiřadit vozidlo. Stejně jako u destinace, u vozidla mohou být definována některá omezení jako například kapacitní omezení, časové okno aj. Vozidla jsou přiřazena tak, aby byly splněny restrikce definované v rámci okruhu, vozidla i všech destinací.

Na následujícím Obr. 5.3 je navržena struktura vozidla.



Obr. 5.3 Class Diagram Car

Stejně jako destinace, každé vozidlo je identifikováno názvem a unikátním Java UUID identifikátorem. Porovnání dvou vozidel pak proběhne tak, že jsou porovnány identifikátory.

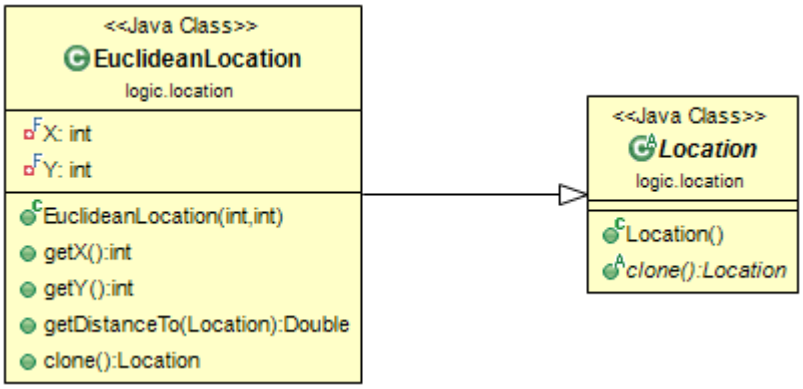
5.1.4 Location

V rámci podproblému Location navrhujeme strukturu pro definování lokace.

Lokace může být určena mnoha způsoby, z obecného hlediska nás u každé lokace zajímá pouze porovnání a ohodnocení vzdálenosti dvou lokací.

V rámci této práce navrhujeme euklidovskou lokaci, což je definice umístění ve dvourozměrném prostoru (v rovině). V tomto případě je každá lokace popsána pomocí souřadnic X a Y.

Na následujícím Obr. 5.4 je navržena struktura lokace.



Obr. 5.4 Class Diagram Location

V rámci rozšíření problému lze lokaci definovat do trojrozměrného prostoru, případně jako GPS souřadnice.

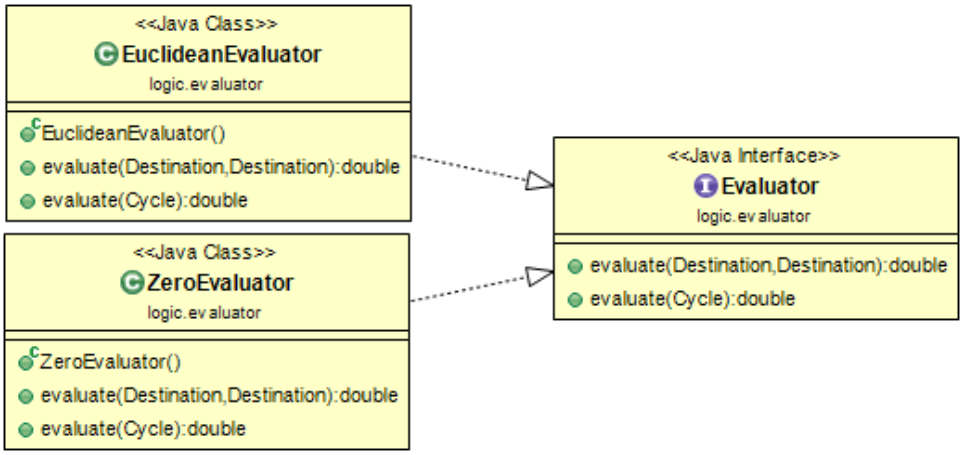
5.1.5 Evaluator

Pod pojmem Evaluator chápeme strukturu pro ohodnocení vzdálenosti mezi dvěma destinacemi. Toto ohodnocení nemusí být pouze finančně nákladové, lze uvažovat i časové náklady, ušlý zisk firmy atp.

V nejjednodušší formě uvažujeme euklidovské ohodnocení vzdálenosti mezi dvěma destinacemi (za předpokladu, že lokace u každé destinace je navržena také jako euklidovská, tj. že obsahuje souřadnice ve dvojrozměrném prostoru). V tomto ohodnocení nelze uvažovat zakázané trasy, parametrizovanou účelovou funkci aj.

Další velmi používaný typ je statické případně dynamické tabulkové ohodnocení.

Na následujícím Obr. 5.5 je navržena struktura Evaluatoru.



Obr. 5.5 Class Diagram Evaluator

Každý typ ohodnocení musí podporovat rozhraní, které dokáže ohodnotit vzdálenost (lépe hranu) mezi dvěma destinacemi.

5.1.6 Restriction

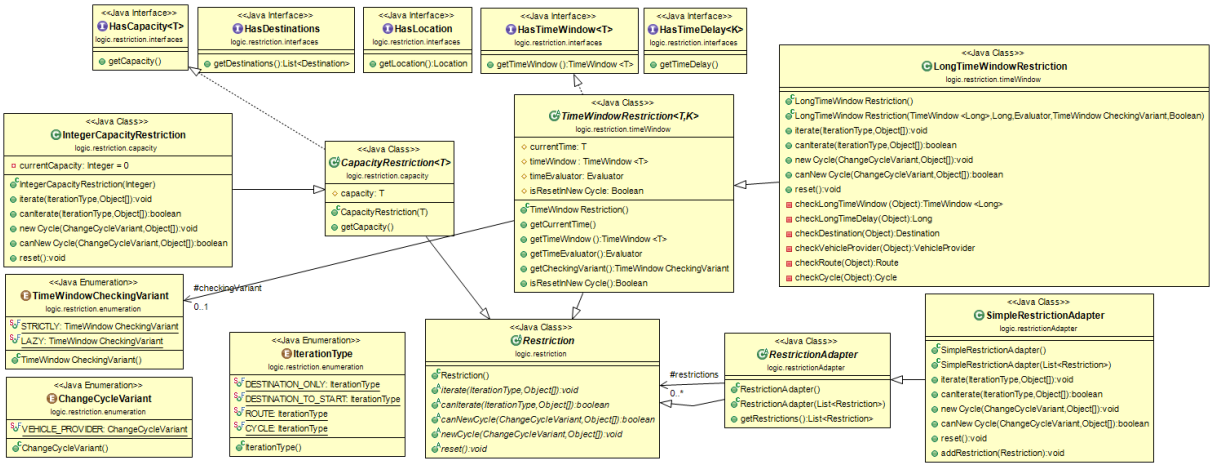
Tento podproblém se týká obecné definice omezení, které bude použitelné pro destinace, vozidla, okruhy aj.

Návrh této struktury je obzvlášť složitý, neboť je žádoucí, aby logika algoritmu pro řešení okružního dopravního problému nezasahovala do logiky struktury omezení a naopak.

Lze uvažovat různé druhy omezení. V tomto případě se omezíme na základní druhy, což jsou časová okna a kapacitní omezení.

Také se pro jednoduchost omezíme na přípustné řešení okružního dopravního problému bez relaxace některých podmínek. To znamená, že pro navrženou trasu (okruh) musí být vždy splněny všechny podmínky, které byly specifikovány. Tento fakt je zajištěn třídou *SimpleRestrictionAdapter*.

Na následujícím Obr. 5.6 je navržena struktura omezení.



Obr. 5.6 Class Diagram Restriction

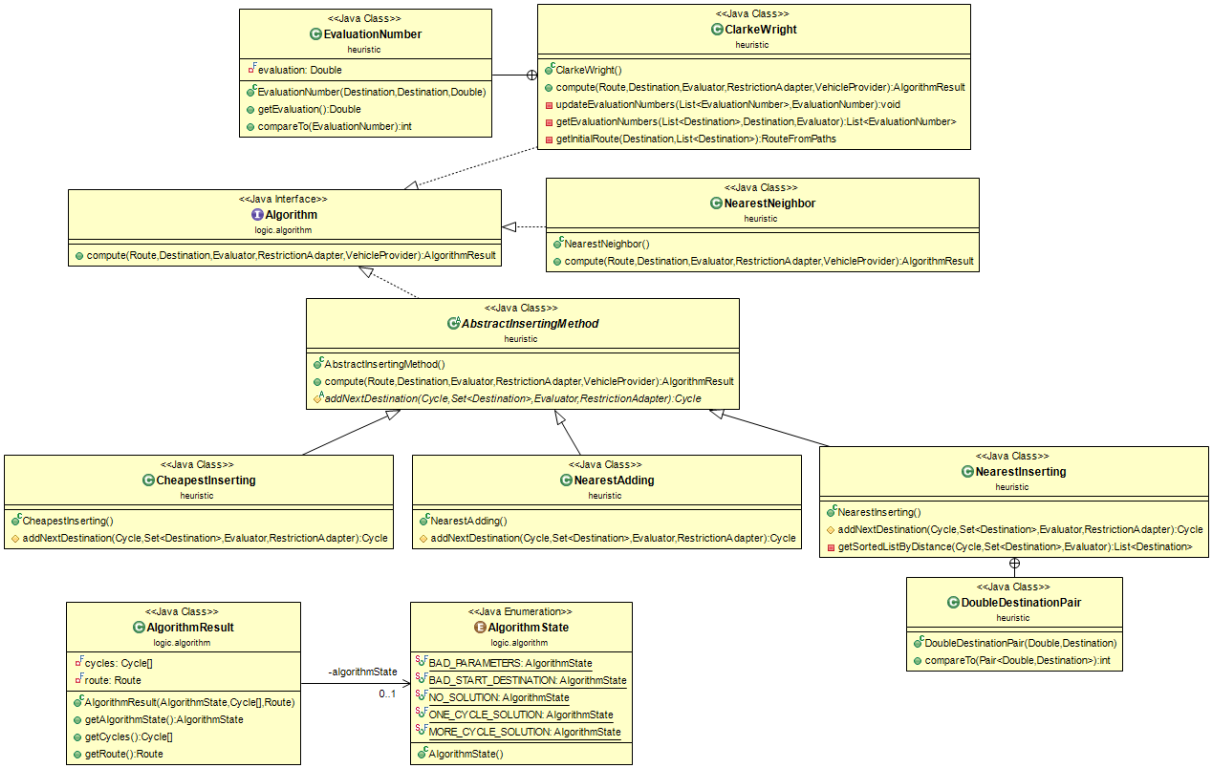
Lze snadno prohlédnout, že návrh struktury omezení vychází od abstraktního návrhu, který nevyžaduje žádné restrikce pro vstupní parametry a poté ho konkretizujeme dle použitého typu restrikce.

5.1.7 Algorithm

V rámci tohoto problému byla popsána obecná struktura algoritmu pro řešení okružního dopravního problému a pro tuto strukturu navrženy a implementovány jednotlivé heuristiky pro řešení okružního dopravního problému.

Každá z těchto heuristik aplikuje jinou metodu pro získání výsledného řešení problému, nicméně vstupní parametry jsou určeny strukturou algoritmu a jsou stejné pro všechny heuristiky (a obecně i pro jakékoli metody řešení okružního dopravního problému). Tento požadavek zajistí korektní práci s výsledky algoritmu, validaci a znovupoužitelnost.

Na následujícím Obr. 5.7 je navržena struktura algoritmu.



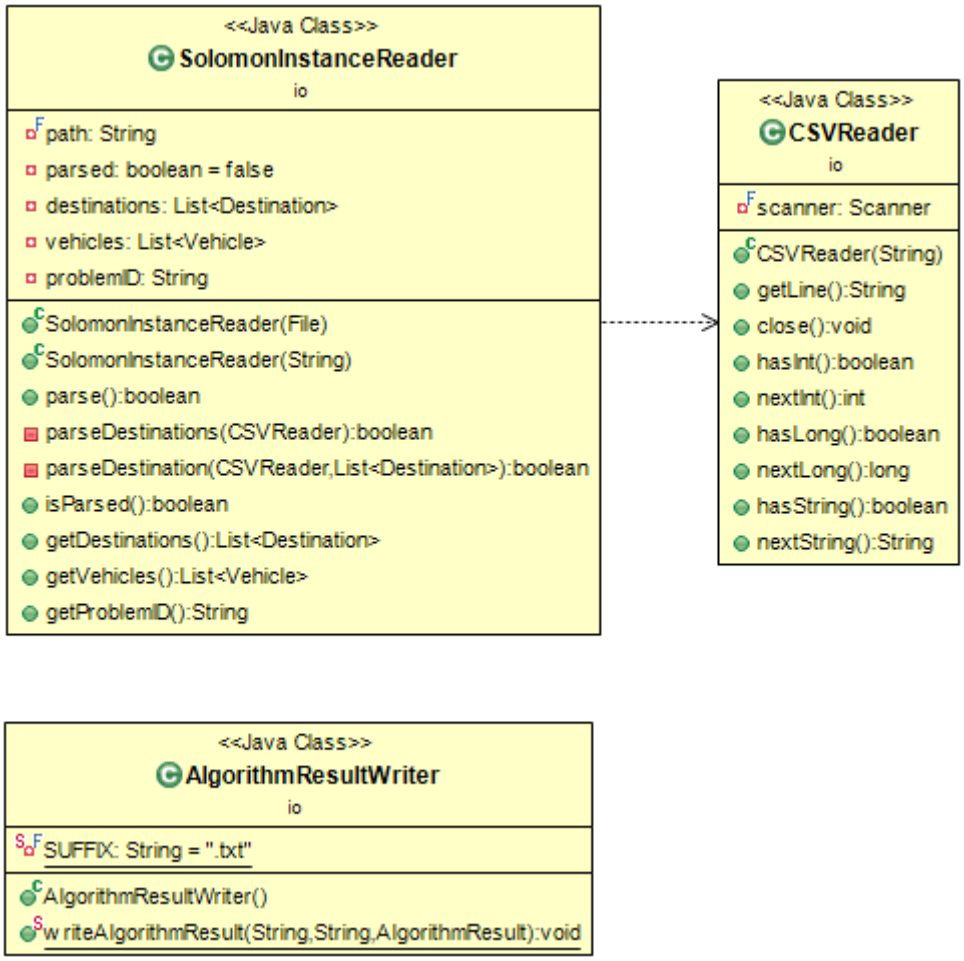
Obr. 5.7 Class Diagram Algorithm

Obecná struktura algoritmu uvažuje základní metodu pro řešení okružního problému, která zahrnuje vstupní parametry okruh, startovní destinaci, objekt pro ohodnocení trasy a objekt s definovanými omezeními.

5.1.8 Input/Output

V rámci této části navrhujeme některé pomocné vstupně výstupní struktury, které jsou užitečné pro načítání instancí ODP a ukládání jejích výsledků.

Na následujícím Obr. 5.8 je navržena struktura vstupně výstupního podproblému.



Obr. 5.8 Class Diagram IO

5.2 Dokumentace implementace

V této podkapitole popíšeme detaily implementace navržené softwarové knihovny. Nejprve představíme další nástroje a frameworky, které byly při implementaci využity. Nakonec zveřejníme adresu úložiště pro implementovanou softwarovou knihovnu.

5.2.1 Nástroje

Pro implementaci knihovny v jazyce Java bylo využito vývojového prostředí Eclipse, konkrétně verzi 4.4.1 Luna Service Release.

V rámci návrhové fáze byl použit Eclipse plugin ObjectAid UML ve verzi 1.1.6, který zajistil korektní export a generování Class Diagramů ze zdrojového kódu.

Pro zálohování a verzování byl využit nástroj Subversion, který společně s veřejným repozitářem Google Repository poskytnul efektivní způsob správy, sdílení a publikování zdrojového kódu.

V neposlední řadě je nutné zmínit Framework JUnit verze 4, který byl využit pro jednotkové testování zdrojového kódu a celkovou validaci knihovny.

5.2.2 Publikování knihovny

V rámci vývoje knihovny využíváme veřejné úložiště Google Repository.

Pro verzování používáme nástroj Subversion a k projektu lze přistupovat s následujícími přístupovými údaji:

- `svn checkout http://vrptw-drobas.googlecode.com/svn/trunk/ vrptw-drobas-read-only`

5.3 Validace knihovny

Tato podkapitola pojednává o průběhu validace knihovny. Jedná se o testování korektnosti všech struktur a implementovaných algoritmů.

Validaci jsme rozdělili na dvě fáze:

- fáze jednotkového testování struktur
- fáze benchmarkového srovnání

Obě fáze popíšeme důkladněji v dalších částech podkapitoly.

5.3.1 Jednotkové testování

Pro každou stavovou i bezestavovou strukturu, která v rámci svého rozhraní čte, příp. modifikuje jiné struktury, byl vytvořen JUnit Test Case, který testuje korektnost chování struktury. Tyto testy ověřují základní funkcionalitu a rozhraní každé struktury.

Jednotlivé Test Case jsou přiloženy k publikovaným zdrojovým kódům a umístěny do složky src\test. JUnit testy jsou strukturovány do složek dle rozvržení podproblémů v rámci podkapitoly 5.1.

5.3.2 Benchmark

Solomon publikoval sadu dat pro testování efektivity algoritmů a metod pro řešení ODPČO. Tato sada se postupem času stala uznávaným zdrojem dat pro srovnání a testování nových metod navržených pro řešení tohoto problému.

Jednotlivé problémy jsou děleny do dvou kategorií dle časového horizontu řešení na kratší (označeny 100) a delší (označeny 200). Problémy s delším časovým horizontem jsou považovány za složitější k řešení. Každá kategorie se potom dělí do tří skupin (c, r a rc), kde c znamená clustered (vrcholy jsou umístěny do clusterů), r je random (náhodné umístění vrcholů) a rc(mix clustered a random). Každá skupina potom obsahuje jednotlivé testovací soubory. Základní jsou problémy se 100 vrcholy (označené _100 v názvu). Dále sada obsahuje i zadání se 25 a 50 vrcholy. Kompletní sada dat je obsažena na přiloženém CD.

Pro problémy z této sady dat volíme jednoduchou euklidovskou účelovou funkci, která destinace ohodnotí pouze na základě euklidovské vzdálenosti lokací jednotlivých destinací. Lokace pro každou destinaci jsou zadány pomocí souřadnic X a Y.

Pro každý implementovaný algoritmus uvedeme procentuální poměr jeho průměrných výsledků s průměrnými optimálními výsledky. Optimálním výsledem chápeme nejlepší možné řešení jakého lze dosáhnout při řešení daného problému. Tyto výsledky rozdělíme do kategorií dle časového horizontu a počtu vrcholů.

Detailní neprůměrované výsledky jsou obsaženy na přiloženém CD.

Následující tabulky prezentují srovnání jednotlivých implementovaných metod.

Tab. 5.1 Výsledky srovnání pro metodu nejbližšího souseda

Počet dest. \ Časový hor.	*100*	*200*
25	256%	274%
50	285%	316%
100	305%	379%

Tab. 5.2 Výsledky srovnání pro metodu Clarke-Wrighta

Počet dest. \ Časový hor.	*100*	*200*
25	375%	402%
50	458%	530%
100	505%	678%

Tab. 5.3 Výsledky srovnání pro metodu nejbližšího přídavku

Počet dest. \ Časový hor.	*100*	*200*
25	230%	243%
50	267%	294%
100	289%	360%

Tab. 5.4 Výsledky srovnání pro metodu nejbližšího vložení

Počet dest. \ Časový hor.	*100*	*200*
25	231%	243%
50	266%	294%
100	288%	360%

Tab. 5.5 Výsledky srovnání pro metodu nejlevnějšího vložení

Počet dest. \ Časový hor.	*100*	*200*
25	230%	243%
50	266%	293%
100	289%	360%

Výsledky uvádíme pouze proto, aby byla zajištěna validace funkce softwarové knihovny na testovacích problémech. Korektní průběh výpočtů nad celou sadou testovacích dat zajistil záruku komplexní funkcionality.

6 Využití knihovny

Tato kapitola se věnuje reálnému využití knihovny včetně typických scénářů nasazení.

Nejprve je v podkapitole popsán typický scénář použití, dále je demonstrována jednoduchá ukázka a na závěr jsou formulovány všechny podmínky aplikace tohoto řešení.

6.1 Scénář použití knihovny

Existuje hned několik typických scénářů využití knihovny.

- edukační ukázka funkce implementovaných algoritmů
- testování nově navrženého algoritmu nebo metody
- pomůcka při optimalizaci distribučních nákladů pro malé a střední firmy
- studium problematiky okružního dopravního problému
- a další

Pro všechny tyto scénáře lze sestavit jednoduchou obecnou osnovu:

I. Načtení vstupních dat
II. Výpočet
III. Export řešení

Jednotlivými body se budeme zabývat blíže.

6.1.1 Načtení vstupních dat

V rámci implementace podproblému Input/Output navrhovaného v podkapitole 5.1.8 byl implementován jednoduchý Parser CSV dat (CSVReader), který je vhodný pro načítání instancí problémů ze souborů formátu CSV.

V současné podobě je tento Parser (SolomonInstanceReader) vhodný pro načítání Solomonových benchmarků, tudíž pro načítání dat ve specifické struktuře je nutné Parser upravit.

6.1.2 Výpočet

Samotnou fázi výpočtu zajistí rozhraní *Algorithm*, které implementuje každý algoritmus a metoda implementovaná v rámci této knihovny.

Rozhraní obsahuje metodu, která zajistí výpočet nad danou instancí ODP:

```
AlgorithmResult compute(Route, Destination, Evaluator, RestrictionAdapter,
                  VehicleProvider);
```

Vstupní hodnoty této metody:

- Route – reprezentuje okruh. Více o problematice okruhu lze nalézt v 5.1.1.
- Destination – reprezentuje startovní destinaci pro řešení této instance problému. Více o problematice destinace lze nalézt v 5.1.2.
- Evaluator – reprezentuje ohodnocení nákladů pro přesun mezi destinacemi. Více o problematice Evaluator lze nalézt v 5.1.5.
- RestrictionAdapter – reprezentuje objekt pro správu a validaci podmínek pro řešení ODP. Více o problematice RestrictionAdapter lze nalézt v 5.1.6.
- VehicleProvider – reprezentuje objekt pro správu vozidel. Více o problematice VehicleProvider lze nalézt v 5.1.3.

Volání metody `compute` s korektními parametry zajistí výpočet daného algoritmu nad daty instance problému. Návratovou hodnotou je objekt `AlgorithmResult`, který reprezentuje výsledek řešení problému.

6.1.3 Export řešení

V rámci implementace podproblému Input/Output navrhovaného v podkapitole 5.1.8 byl implementován jednoduchý Writer dat (AlgorithmResultWriter), který umožní uložení výsledku algoritmu (AlgorithmResult) do souboru formátu CSV.

Tento Writer je v současné době navržen tak, aby výsledky dosažené v rámci jedné instance ODP zapisoval do jednoho souboru. Pro specifické ukládání výsledku je nutné tento Writer upravit.

6.2 Ukázka využití

V rámci druhé fáze testování softwarové knihovny (viz 5.3.2) byla knihovna testována na Solomonově sadě dat instancí ODP. Pro korektní spuštění byla vytvořena Main třída, kterou lze nalézt ve složce src/main/execution/Main.java.

Tato třída může být použita jako jednoduchý příklad využití knihovny.

Úryvek z metody `main`:

```
File initDirectory = new File(BENCHMARKS_DIRECTORY_PATH);
if (!initDirectory.isDirectory() || !initDirectory.canRead())
	return;

for (File groupDirectory : initDirectory.listFiles()) {
	if (!groupDirectory.isDirectory() || !groupDirectory.canRead())
			continue;

	for (File file : groupDirectory.listFiles())
		if (file.canRead() && file.isFile())
			solveSolomonInstance(file);
}
```

Tato sekce čte každý soubor `file` ve složce ***BENCHMARKS_DIRECTORY_PATH*** a volá metodu *solveSolomonInstance*(file).

Úryvky z metody *solveSolomonInstance*(file):

```
if (file == null || !file.isFile())
	return;

SolomonInstanceReader reader = new SolomonInstanceReader(file);
reader.parse();

if (!reader.isParsed() || reader.getDestinations().isEmpty() ||
reader.getVehicles().isEmpty())
	return;
```

Tato sekce parsuje vstupní soubor jako instanci ODP.

```
List<Destination> destinations = reader.getDestinations();
List<Vehicle> vehicles = reader.getVehicles();
String problemID = reader.getProblemID();

SimpleRestrictionAdapter restrictionAdapter = new SimpleRestrictionAdapter();

int vehicleCapacity = ((HasCapacity<Integer>)vehicles.get(0)).getCapacity();
restrictionAdapter.addRestriction(new
IntegerCapacityRestriction(vehicleCapacity));

TimeWindow<Long> timeWindow =
((HasTimeWindow<Long>)destinations.get(0)).getTimeWindow();
LongTimeWindowRestriction timeRestriction = new LongTimeWindowRestriction(
                          timeWindow,
                          0L,
                          new EuclideanEvaluator(),
                          TimeWindowCheckingVariant.LAZY,
                          true);
restrictionAdapter.addRestriction(timeRestriction);
```

Dále z parsovaných parametrů (destinace, vozidla) sestavuje restrikce pro řešení problému.

```
AlgorithmResult result = algorithm.compute(
                          new SimpleRoute(destinations),
                          destinations.get(0),
                          new EuclideanEvaluator(),
                          restrictionAdapter,
                          new SimpleVehicleProvider(vehicles));

AlgorithmResultWriter.writeAlgorithmResult(
                    RESULT_DIRECTORY_PATH,
                    problemID + "_" + String.valueOf(destinations.size()-1),
                    result);
```

Následně spouští výpočet. Výsledek výpočtu je exportován do souboru formátu CSV.

6.3 Podmínky aplikace

Jedinou podmínkou pro korektní funkci výše uvedeného příkladu je přidání těchto závislostí:

- Zdrojové kódy knihovny. Nejjednoduším způsobem je přidání dependence přímo na zkompilovaný Java Archive knihovny. Lze také importovat celý projekt včetně nezkompilovaných zdrojových kódů do jiného projektu a kompilovat kód při

spouštění rodičovského projektu. Druhý způsob je výhodnější pokud chceme provádět funkcionální úpravy na této knihovně.

- Java Runtime Environment. Pro kompilaci této knihovny byla použita verze 1.8, tudíž pro korektní kompilaci kódu doporučujeme verzi 1.8 nebo vyšší.

7 Závěr

Metody pro řešení okružního dopravního problému jsou svým postupem vždy specifické, neboť pro vybrané případy lze uvažovat různé distribuční restrikce, specifika destinací, vozidel a další podmínky, které musíme brát v úvahu. Většina metod a algoritmů pro řešení okružního dopravního problému je popsána pro nejjednodušší variantu okružního dopravního problému a neuvažují dodatečné restrikce. Z tohoto důvodu je velmi těžké zkonstruovat obecný návrh knihovny, která by pokrývala všechny výše zmíněné potřeby a zároveň rozhraní pro všechny uvažované metody.

Problém jsem dekomponoval na menší podproblémy a ty pak řešil separovaně. Tyto podproblémy dále při návrhu softwarové knihovny vyplynuly jako nezávislé komponenty, které tvoří strukturu výsledné knihovny. Komunikace mezi komponentami probíhá na základě předem definovaného obecného rozhraní a každá komponenta je nahraditelná. Lze to chápat tak, že pro každý specifický druh okružního dopravního problému lze vybrané komponenty modifikovat (případně přidat jiné) tak, aby struktura knihovny odpovídala těmto potřebám.

Práci lze strukturně rozdělit do 3 částí. První částí je část návrhová. Tato část je stěžejní, proto jsem jí věnoval zdaleka nejvíce prostoru. V rámci této části jsem důkladně popsal problematiku okružního dopravního problému v kapitole 2 a 3. Následně jsem zdokumentoval principy návrhu výsledné knihovny v kapitole 4 a v podkapitole 5.1. Následovala část implementační, které jsem se věnoval v podkapitole 5.2. Poslední částí jsem vymezil fázi validační, kterou dokumentuji v podkapitole 5.3. Následuje kapitola 6, která prezentuje typické využití knihovny a popisuje ukázkový příklad.

I přes některé drobné problémy při obecném návrhu struktury knihovny (lze zmínit stěžejní návrh podproblému Restriction dokumentovaný v podkapitole 5.1.6) byly naplněny všechny cíle práce vymezené v úvodní kapitole 1. Výsledná knihovna je prezentována a publikována jako open source projekt (viz 5.2.2). Rád bych pokračoval v dalším vývoji knihovny a implementoval další známé algoritmy - při rozšiřování knihovny bych se zaměřil především na okružní dopravní problém v podmínkách časových oken, kterému je v současnosti věnována největší pozornost (13).

Implementace knihovny včetně zdrojových kódů je obsažena na přiloženém CD. Dále CD obsahuje Java Archive výsledné knihovny, návrhové diagramy, testovací soubory (JUnit Test Case) a programátorskou dokumentaci knihovny (Javadoc). Přiloženy jsou také Solomonovy benchmarky a výsledky validace knihovny.

Citovaná literatura

1. **Hitchcock, F. L.** The distribution of a product from several sources to numerous localities. *Journal Article.* 1941.
2. **Dantzig, G.** Application of the simplex method to a transportation problem. *Activity analysis of production and allocation.* 1951.
3. **Dupačová, J., Gröwe-Kuska, N. a Römisch, W.** Scenario reduction in stochastic programming. *Mathematic Programming.* 2003, stránky 493-511.
4. **Alexander, S.** On the history of combinatorial optimization (till 1960). *Handb. Oper. Res. Manag. Sci. Discrete Optim.* 2005, Sv. 12, 1.
5. **Karp, R. M.** *Reducibility, among Combinatorial Problems.* New York : Plennum Press, 1972. pp. 85-104.
6. **Kwan, M. K.** Graphic programming using odd or even points. *Chin. Math.* 1, 1962, 110.
7. **Lenstra, J. K. a Kan, A.** Complexity of vehicle routing and scheduling problems. *Networks.* 11, 1981, 2.
8. **Golden, B. L., Raghavan, S. a Wasil, E. A.** *The Vehicle Routing Problem. Latest Advances and New Challenges.* New York : Springer, 2008.
9. **Miller, C. E., Tucker, A. W. and Zemlin, R. A.** Integer Programming Formulation of Travelling Salesman Problems. *Journal of the ACM.* 1960, Vol. 7, 4, pp. 326-329.
10. **Solomon, M. M.** Algorithms for the vehicle routing and scheduling problems with time. *Oper. Res.* 1987, Sv. 35, 2.
11. **Solomon, M. M.** On the worst-case performance of some heuristics for the vehicle. *Networks.* 1986, Sv. 16, 2.
12. **Labadi, N., Prins, C. a Reghioui, M.** A memetic algorithm for the vehicle routing. *RAIRO-Oper. Res.* 2008, Sv. 42, 3.
13. **Bräysy, O. a Gendreau, M.** Vehicle routing problem with time windows, Part I: Route construction and local search algorithms. *Transp. Sci.* 2005, Sv. 39, 1.
14. **Bräysy, O. a Gendreau, M**. Vehicle routing problem with time windows, Part II: Metaheuristics. *Transp. Sci.* 2005, Sv. 39, 1.

15. **Feillet, D., a další.** An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks.* 2004, Sv. 44, 3.
16. **Fisher, M. L., Jörnsten, K. O. a Madsen, O. B.** Vehicle routing with time windows: Two optimization algorithms. *Oper. Res.* 1997, Sv. 45, 3.
17. **Kolen, A. W., Kan, A. R. a Trienekens, H.** Vehicle routing with time windows. *Oper. Res.* 1987, Sv. 35, 2.
18. **Desrochers, M., Desrosiers, J. a Solomon, M.** A new optimization algorithm for the vehicle routing problem with time windows. *Oper. Res.* 1992, Sv. 40, 2.
19. **Jörnsten, K. O., Madsen, O. B. a Sorensen, B.** Exact solution of the vehicle routing and scheduling problem with time windows by variable splitting. 1986.
20. **Madsen, O. B.** Lagrangean relaxation and vehicle routing. *IMSOR Tech. Univ. Den.* 1990.
21. **Fisher, M. L.** Optimal solution of vehicle routing problems using minimum k-trees. *Oper. Res.* 1994, Sv. 42, 4.
22. **Clarke, G. a Wright, J.W.** Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operational research.* 1964, stránky 568-581.
23. **Kučera, P.** *Metodologie řešení okružního dopravního problému.* Praha : Kučera, 2009.
24. **Rosenkrantz, D. J., Stearns, R. E. a Lewis, P. M.** An analysis of several heuristics for the traveling salesman problem. *SIAM J. Comput.* 1977, Sv. 6, 3.
25. **Glover, F.** Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.* 13, 1986, 5.
26. **Glover, F**. Tabu search-part I. *ORSA J. Comput.* 1989, Sv. 1, 3.
27. **Glover, F.** Tabu search-part II. *ORSA J. Comput.* 1990, Sv. 2, 1.
28. **Kindervater, G. A. a Savelsbergh, M. W.** Vehicle routing: handling edge exchanges. *Local Search Comb. Optim.* 1997.
29. **Holland, J. H.** *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.
30. **De Jong, K. A.** *Analysis of the behavior of a class of genetic adaptive systems.* 1975.
31. **Goldberg, D. E:.** *Genetic algorithms in search, optimization, and machine learning.* místo neznámé : Addison-wesley Reading Menlo Park, 1989.

32. **Thangiah, S. R., Nygard, K. E. a Juell, P. L.** Gideon: A genetic algorithm system for vehicle routing with time windows. *Artificial Intelligence Applications.* 1991, Sv. 1.

33. **Bruckner, Tomáš, a další.** *Tvorba informačních systémů.* Praha : Tomáš Bruckner, 2012. ISBN 978-80-904661-6-6.

34. **Buchalcevová, Alena.** *Metodiky budování informačních systémů.* Praha : Oeconomica, 2009. ISBN 978-80-245-1540-3.

35. **Buchalcevová, Alena a Stanovská, Iva.** *Příklady modelů analýzy a návrhu aplikace v UML.* Praha : Oeconomica, 2013. ISBN: 978-80-245-1922-7.

36. **Arms, W. Y.** *Digital Libraries.* Cambridge : MIT Press, 2000.

Terminologický slovník

Termín	Zkratka	Význam
Algorithm		Podproblém definovaný dle 4.2.6.
Car		Podproblém definovaný dle 4.2.6.
Destination		Podproblém definovaný dle 4.2.6.
Evaluator		Podproblém definovaný dle 4.2.6.
Input/Output		Podproblém definovaný dle 4.2.6.
Location		Podproblém definovaný dle 4.2.6.
Okružní dopravní problém	ODP	Definice významu dle 2.1.3.
Okružní dopravní problém v podmínkách časových oken	ODPČO	Rozšíření ODP dle 2.1.4.
Restriction		Podproblém definovaný dle 4.2.6.
Route		Podproblém definovaný dle 4.2.6.
Softwarová knihovna		Definice významu dle 4.2.