VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

OPTIMALIZACE PRŮCHODU SKLADEM

OPTIMIZATION OF WAREHOUSE PASSING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAN HOLÁŇ**
AUTHOR

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

BRNO 2023

Abstrakt

Práce je zaměřena na nalezení nejkratší cesty při průchodu (konkrétně při skladovém procesu zvaném vychystávání položek) skladem za účelem vychystání daného počtu položek. Posána je nezbytná teorie související s matematickými problémy, vztahující se k tématu této práce. Největší pozornost je věnována obecně nepříliš diskutovanému tématu, kterým je seskupování nerozdělitelných skupin položek, za cílem získání nejkratší výsledné nejkratší trasy mezi položkami. Výsledkem je knihovna v jazyce C++ schopna řešit proces vychystávání za podmínek definovaných konkrétní reálnou firmou.

Abstract

The work focuses on finding the shortest path through the warehouse (specifically in a warehouse process called picking) in order to pick a given number of items. The necessary theory related to mathematical problems related to the topic of this work is presented. Most attention is given to a generally not much discussed topic, which is the grouping of indistinguishable groups of items in order to obtain the resulting shortest path between items. The result is a C++ library capable of dealing with the picking process under conditions defined by a specific real company.

Klíčová slova

sklad,seskupování skupin položek, TSP, Problém obchodního cestujícího, BPP, Bin Packing Problem, nejkratší cesta, optimalizace

Keywords

warehouse, grouping groups items, TSP, Travelling Salesman Problem, BPP, Bin Packing Problem, shortest path, optimization

Citace

HOLÁŇ, Jan. *Optimalizace průchodu skladem*. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Zbyněk Křivka, Ph.D.

Optimalizace průchodu skladem

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Zbyňka Křivky, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jan Holáň
1. května 2023

Poděkování

Děkuji svému vedoucímu, panu Ing. Zbyňku Křivkovi, Ph.D., za milý a vstřícný přístup při konzultacích mé práce a jeho velmi užitečné a věcné rady.

Obsah

1 Úvod **3**

2 Logistika **4**
2.1 Problematika logistiky ve skladech 4
2.2 Matematické problémy v logistice 5
2.3 Problém obchodního cestujícího 6
2.3.1 Formulace problému 6
2.3.2 Heuristiky 7
2.3.3 Held-Karpova spodní hranice 8
2.3.4 Greedy 8
2.4 Bin packing problem 9
2.4.1 Definice BPP pro jeden typ boxu 9
2.4.2 Definice BPP s více typy boxů 9
2.4.3 Heuristiky 9
2.4.4 FFD 9
2.4.5 FFDS 10
2.4.6 Využití v 1D a ve 3D 10
2.5 Metody pro seskupování položek 11
2.5.1 Metoda nejmenších čtverců 11
2.5.2 K-Means clustering 11

3 Problematika řešené úlohy **13**
3.1 Popis systému 13
3.1.1 Dosavadní řešení 13
3.1.2 Předvychystávání 13
3.1.3 Data 14
3.1.4 Pravidla 14
3.1.5 Statistiky 14
3.1.6 Možná rozšíření 14
3.1.7 Popis adres ve skladu 15

4 Návrh a Implementace **16**
4.1 Základní řešení 16
4.1.1 Spuštění výpočtu 16
4.1.2 Fáze výpočtu 16
4.1.3 Seskupení zákazníků do vozíků - funkce runAlgorithm 17
4.1.4 Rozdělení položek do boxů - funkce decompIntoBoxes 18
4.1.5 Seřazení položek ve vozících - funkce sortItemsInCart 18

4.1.6 Adresování a měření vzdáleností . . . 19
4.2 Hlavní řešení . . . 20
4.2.1 Spuštění . . . 20
4.2.2 Načítání dat . . . 20
4.2.3 Adresování a měření vzdáleností . . . 20
4.2.4 Implementace Bin packing - jmenný prostor *bp_solver* . . . 22
4.2.5 Přiřazení zákazníka do vozíku . . . 23
4.2.6 Implementace seskupování zákazníků do vozíku - jmenný prostor *grouping_customers* . . . 24
4.2.7 Implementace řazení položek ve vozíku - jmenný prostor *tsp_solver* 28
4.2.8 Export dat . . . 32
4.3 Grafické rozhraní . . . 32

5 Srovnání, testování a statistiky 33
5.1 Změny v implementaci základního a finálního řešení . . . 33
5.1.1 Načítání specifikací položek . . . 33
5.1.2 Srovnání hlavního a základního řešení . . . 33
5.2 Validace a integrita dat . . . 33
5.3 Srovnání firemního řešení . . . 34
5.3.1 Srovnání patnácti vln . . . 34
5.3.2 Srovnání čtyř vln . . . 35
5.3.3 Srovnání všech devatenácti vln - odlišná konfigurace . . . 36
5.4 Další srovnání . . . 37
5.4.1 Vytváření boxů . . . 37
5.4.2 Srovnání algoritmů pro nejkratší vzdálenost . . . 37
5.4.3 Časová náročnost algoritmů seskupování zákazníků . . . 38
5.4.4 Časová náročnost algoritmů pro nejkratší trasu . . . 40

6 Závěr 41

Literatura 42

A Algoritmy 44

B Skladové prostory 46

C Grafické rozhranní 48

Kapitola 1

Úvod

V rámci této práce je řešena optimalizace vychystávání zboží objednaného zákazníky. Vychystáváním rozumíme celý proces, který nastává poté, co firma zpracuje objednávku od zákazníka - od vyzvednutí všech požadovaných položek ze skladových pozic až po jejich odevzdání ve finálním stanovišti. Jako referenční zde poslouží sklad konkrétní, reálně existující Firmy (firma si výslovně vyžádala zachování anonymity - v celé této práci bude označována jako *Firma* s velkým počátečním písmenem), pro kterou je celá tato práce optimalizována. Nutno podotknout, že detaily jako rozložení skladu, tvar přepravek, druh zboží, které Firma nabízí a mnoho dalších parametrů natolik ovlivňují celkovou implementaci, že je téměř nemožné navrhnout obecnější řešení.

Jako cíl práce bylo stanoveno najít takové uspořádání skladových položek do vozíků, aby se vzdálenosti, které pracovníci s vozíky ve skladu během vychystávání ujedou, blížily nezbytnému minimu.

Celou implementaci a veškerou vykonanou práci je možno rozdělit na tři hlavní části - první je samotné skládání položek do přepravek, kde se naskýtá velký prostor pro optimalizaci vzhledem k tomu, že vychystávané položky mohou být od velmi málo objemných (pro představu uveďme například položku typu USB flash disk) až po ty velmi objemné (například stolní počítače). Další částí je potom seskupování položek do vozíků. Jako důsledek této činnosti vznikají seznamy položek, které se musí objemově vejít do vozíků, se kterými pracovníci jezdí po skladě. Poslední fází, nezbytnou k implementaci, je seřazení položek ve vozících do co nejkratší trasy.

Řešení skládání položek do boxů a problém nejkratší trasy patří mezi velmi diskutované úlohy nejen v logistice, a proto k nim existuje řada materiálů. Přesto je potřeba se těmto oblastem pečlivě věnovat vzhledem ke specifickému skladu Firmy. V kontrastu k těmto úlohám stojí řešení seskupování položek do vozíků - neexistuje mnoho algoritmů řešící takové zadání a to pro specifikum, které Firma zadala. Jedná se o fakt, že položky, které si objednal jeden zákazník, nelze rozdělit mezi vícero vozíků. Vzniká tak nutnost *seskupovat skupiny položek*. Pro tento účel jsou implementovány a srovnány následující přístupy: K-Means clustering, metoda založená na aproximaci pomocí metody nejmenších čtverců a algoritmus využívající průměrnou vzdálenost mezi položkami.

Kapitola 2

Logistika

2.1 Problematika logistiky ve skladech

Skladová logistika je velmi komplexní obor. Její složitost a propracovanost se obvykle odvíjí od mnoha kritérií. Mezi ně patří zkušenosti, které má daná firma, dále velikost skladu a počet produktů, které je třeba uskladňovat až po finanční možnosti, které lze investovat do organizace a fungování skladu. Dříve nebyla velká poptávka po optimalizaci práce na skladě a docházelo často k mnoha nadbytečným a neefektivním úkonům. Vzhledem k rostoucím požadavkům na rychlost a úsporu financí však začala pronikat do světa logistiky tendence vylepšovat skladové procesy a nacházet efektivnější řešení jednotlivých problémů.

Následující odstavec vychází z článku [10] a osobních zkušeností.
Obecně řečeno, každý sklad sestává ze tří hlavních komponent - příjem zboží, uložení zboží a odeslání zboží. Příjem je odpovědný za kontrolu zboží - jedná se o vstupní bod do skladového procesu. Jakmile je zboží přijato, musí být okamžitě uloženo na skladu na dané pozici. Skladové pozice se zpravidla liší podle velikosti položek, které do nich mají být uloženy. Obvykle jsou tvořeny paletami nebo regály. Poslední fáze - odeslání zboží - je podobná jeho příjmu. Na základě objednávek od zákazníků, kterými mohou být celé firmy, menší živnostníci či jednotlivé osoby, jsou položky na skladě posbírány, zkontrolovány a následně po zabalení opouštějí sklad.

Fungování skladu firmy, u které uvažujeme existenci elektronického systému pomáhajícího řídit její interní procesy, můžeme rozdělit do čtyř základních úrovní. Od nich se potom odvíjí míra nutnosti optimalizovat práci v něm vykonávanou:

1. Malý sklad s nízkým počtem položek. Zaměstnanci se umí orientovat po paměti a žádné skladové adresy nejsou potřeba. Zboží je organizováno v některých případech nejvýše do nějakých logických celků. Propojení zboží vedeného v systému firmy a jeho uložení na skladě neexistuje.

2. Sklad větších rozměrů se středním počtem zboží - je v něm přinejmenším obtížné orientovat se po paměti či s nepřesnými označeními lokací jednotlivých položek. Proto taková firma velmi často volí implementaci WMS (Warehouse Management System). Ve skladu se zvolí adresový systém, který představuje způsob organizace a konvence zápisu skladových adres - pozic, na kterých je uloženo zboží. Ten se potom využije takovým způsobem, že každá položka již má v elektronickém systému vedenu i její adresu na skladu, kde se nachází. Zaměstnanci ovšem stále nepoužívají elektronická zařízení. Místo něj je využíván způsob *tužky a papíru*.

3. Velký sklad s vysokým počtem zboží. Zde již nepostačuje pouhé nasazení WMS bez dodatečné techniky poskytnuté pracovníkům. Ti se totiž začínají stávat značně neefektivními společně s nárůstem počtu položek a skladových adres, protože stále využívají během své práce pouze *tužku a papír*. Zaměstnanci skladu bývají činní ve dvou hlavních úkolech:

   - „vezmi položku a zařaď ji na sklad“ (během příjmu zboží do skladu)
   - „najdi a přines položku uloženou ve skladu“ (během exportu zboží ze skladu).

   Přitom je neustále nutné veškeré akce nejprve provést a následně je i registrovat do systému, což bývá náchylné k chybám. Proto je zavedeno používání různé skladové elektroniky, jejímž cílem je zajistit, že pracovníkem prováděné změny, týkající se zboží, budou v reálném čase zaznamenány a reflektovány i v elektronickém systému. Zavádění různých elektronických a mechanických zařízení vede především ke snížení výskytu chyb a zvýšení produktivity zaměstnanců, druhotně potom i ke snížení zátěže pracovníků (například stroji, které umí obalovat palety fólií apod.).

4. Skladové prostory firem s miliardovými obraty. Zde již je práce velmi efektivně řízena, nicméně začíná se utvářet prostor pro použití automatizace ve formě robotů. Jejich zavedení brání ve většině případů nemožnost finanční návratnosti, právě kvůli malému obratu. Robot s sebou přináší výhodu jednorázové investice a ušetření množství lidí. Počet nahrazených pracovníků ovšem na druhou stranu musí být dostatečně velký na to, aby byl robot v pro firmu únosné době splacen. Příkladem takové automatizace může být využití paletových či balících robotů.

2.2 Matematické problémy v logistice

Tato sekce vychází z článku [10].
Logistika je obor, kde se ve velkém množství uplatňují optimalizační procesy, které mají sloužit pro zrychlení skladových prací, úsporu zaměstnanců a zvýšení produktivity. Obvyklý cíl pro všechny firmy bývá snížit počet takzvaných člověko-hodin (jedná se o jednotku práce, kterou vykoná průměrný pracovník za hodinu; význam je převzat ze slovníku [2]) a souběžně s tím zvýšit zisk. Ze všech nákladů na provoz skladu je $55 - 65$ % tvořeno výdaji na sběr objednávek (tím rozumíme proces získávání zboží ze specifikovaných úložných míst na skladě podle požadavků zákazníka; převzato z článku [17]), což poukazuje na důležitost optimalizace všech procesů s tímto souvisejících.

Nejvýznamnější roli hrají algoritmy spadající pod skupinu *Warehouse picking path algorithms*, které se zabývají právě optimalizací sběru položek ze skladu. Tyto algoritmy jsou obecně vzato heuristikami pro problém *TSP - Travelling salesman problem*. Ve vědeckých článcích bylo prokázáno, že sběr, rozložení skladu, úložný prostor, rozdělení skladu do zón, dávkování a směrování jsou klíčovými aspekty pro optimalizaci sběru objednávek (článek [17]).

V rámci algoritmů, které řeší nejkratší průchod skladem při sběru položek, je často nutné použít další algoritmy - obvykle heuristiky, které řeší problémy spadající do třídy NP. Mezi nejzákladnější z nich patří heuristiky pro již zmíněný problém TSP či *Bin packing problem*, který se zabývá skládáním položek do boxů se stanovenými rozměry. V této práci je rovněž rozebrána problematika seskupování položek do vozíků, které již jsou rozděleny do menších, dále nedělitelných celků. Pro řešení tohoto úkolu je využita metoda *nejmenších čtverců* a metoda učení bez dozoru s názvem *K-Means clustering*.

Zmiňme ještě *Problém nejkratší cesty* - ten v této práci není řešen klasickými známými algoritmy (Dijkstrův,...), protože je možno využít jednoduššího přístupu k výpočtu vzdáleností díky euklidovskému (tedy pravoúhlému) prostoru, kterým můžeme charakterizovat skladový prostor Firmy. Díky tomu lze zaměnit složité hledání cesty za nenáročný výpočet euklidovské vzdálenosti mezi dvěma místy - tedy prostý součet rozdílů souřadnic X a Y. Ovšem je třeba řešit výjimky. První z nich tvoří případ, který lze pozorovat na obrázku v příloze B.2 - takové situace jsou řešeny přímo v kódu dle specifických potřeb daného prostoru. Druhá výjimka nastává, když jsou dvě položky uloženy v protějších regálech, které spojuje ulice - tehdy je vzdálenost mezi nimi měřena jako Pythagorova.

Dále je potřeba zmínit dělení algoritmů podle způsobu příjmu vstupních dat na on-line a off-line. On-line algoritmy neznají přesný počet položek, se kterými pracují (tedy například obchodní cestující předem neví přes kolik měst půjde, nebo neznáme přesný počet položek při jejich balení do boxů). Jejich nevýhoda zcela zjevně spočívá ve znalosti pouze aktuálního optimálního řešení, které se v čase může ukázat jako špatně zvolené. Oproti tomu off-line algoritmy znají veškerá vstupní data při spuštění výpočtu a tudíž mohou volit skutečné optimální řešení. V této práci jsou využívány výhradně off-line algoritmy (odstavec vychází z článku [8]).

2.3 Problém obchodního cestujícího

Problém obchodního cestujícího (známý jako TSP - Travelling Salesman Problem) je klasickým kombinatorickým problémem, který je jednoduché popsat, ale naopak velmi složité vyřešit. Máme při něm sadu měst a vzdáleností mezi nimi a startovací bod. Hledáme potom způsob, jakým projít co nejkratší dráhou mezi všemi městy tak, aby každé bylo navštíveno pouze jednou a navrátit se zpět do počátečního bodu.

Jedná se o optimalizační problém, při kterém je prohledáván obrovský prostor, což z něj činí tzv. NP-těžký problém, což znamená, že nemůže být řešen v polynomickém čase: s nárůstem počtu míst roste čas potřebný k vyřešení exponenciálně (vychází z článku [16]). Pokud bychom měli N bodů, bylo by potřeba vyzkoušet $N!$ možných kombinací projití těmito body. Uvažujeme-li fixní počáteční bod, pak by těchto kombinací bylo $(N-1)!$, a tedy by složitost byla $O(n!)$ (srovnání složitostí viz příloha A.1) - pouze pro představu uveďme, že pro pouhých 10 míst by to znamenalo porovnat přes 3,3 milionu možností. K řešení je tedy nutno použít různé heuristiky.

Pro TSP platí úzká souvislost s problémem Hamiltonova cyklu - to je cyklus obsahující takovou cestu, na které lze všechny uzly navštívit právě jednou a přitom se navrátit zpět do počátečního vrcholu. **Jestliže najdeme takový (tj. Hamiltonovský) cyklus s minimálním součtem vah jeho hran, vyřešili bychom problém obchodního cestujícího** (odstavec vychází z knihy [19]).

2.3.1 Formulace problému

Tato sekce vychází z článku [11].

Mějme města, kde každé dvojici z nich (i, j) přiřadíme celočíselnou hodnotu $x_{ij} \in 0, 1$, která se bude rovnat 1, pokud existuje cesta z města i do města j, jinak se bude rovnat 0. Potom dvojice (i, j) přispívá hodnotou $c_{ij}x_{ij}$ (ta bude rovna hodnotě c_{ij}, pokud $x_{ij} = 1$, a 0, pokud $x_{ij} = 0$) k celkové ceně cesty, přičemž hodnota c_{ij} odpovídá vzdálenosti měst i, j.

Odtud plyne cíl minimalizovat funkci

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}. \tag{2.1}$$

Během cesty navštívíme každé město pouze jednou - do každého města právě jednou vstoupíme a právě jednou z něj odejdeme (zmiňme, že pro počáteční město provedeme tento proces v opačném pořadí - první z něj vystoupíme a pak se do něj na konci cesty vrátíme, nicméně tento fakt nehraje během výpočtu žádnou roli). Tento požadavek vyjádříme následujícími podmínkami:

$$\sum_{1\leq i\leq n,\ i\neq j} x_{ij} = 1 \quad \text{pro každé } j = 1, 2, ..., n \tag{2.2}$$

$$\sum_{1\leq k\leq n,\ k\neq j} x_{jk} = 1 \quad \text{pro každé } j = 1, 2, ..., n \tag{2.3}$$

První rovnice (2.2) říká, že do města j přicházíme právě z jednoho města, zatímco druhá rovnice (2.3) vyjadřuje, že město j opouštíme právě jednou do dalšího města. Tato omezení ovšem nejsou dostačující, jak můžeme pozorovat na obrázku A.2. Z toho důvodu potřebujeme přidat další podmínky, které by úplně popisovaly řešení problému TSP. Tyto podmínky mohou být vytvořeny různými způsoby, my zde uvedeme pouze jeden z nich - přidání velkého množství podmínek, které omezují vznik podcest (metoda uvedena Dantzigem, Fulkersonem a Johnsonem v roce 1954). Mají následující tvar:

$$\sum_{i\in S}\sum_{j\notin S} x_{ij} \geq 1 \quad \text{pro každé S: } 1 \leq |S| \leq n-1 \tag{2.4}$$

Pro každou množinu měst S, která je neprázdná a není rovna množině všech měst, dostáváme sumu hran zahrnující všechny dvojice měst (i, j), kde město i patří do množiny S, zatímco město j do ní nepatří. Požadavkem, aby výsledek sumy byl alespoň 1 chceme docílit toho, aby každá množina měst S byla během průchodu opuštěna alespoň 1. To bude splněno při každém validním průchodu městy, který odpovídá kandidátnímu řešení pro TSP. Zatímco optimální řešení splňující omezení (2.2) a (2.3) podmínku 2.4 porušuje.

Pro jakékoli celočíselné řešení odpovídající podmínkám (2.2) a (2.3) a omezením pro eliminaci podcest (2.4) dostáváme vždy vyhovující kandidátní řešení problému TSP.

2.3.2 Heuristiky

Tento odstavec vychází z webové stránky [6].
Pro vybrání správné heuristiky musíme rozlišovat mezi dvěma typy problému TSP - první je tvořen místy, mezi kterými se dá cestovat v obou směrech za stejnou cenu - tento druh je označován jako STSP (*Symetric TSP*). Opačný případ, tedy pokud cesta z místa A do místa B má jinou váhu, než z místa B do místa A, nazýváme ATSP (*Asymetric TSP*). Protože sklad, pro který je tato práce řešena, vykazuje možnost pohybovat se ve všech místech v obou směrech, budeme se zde zabývat pouze heuristikami pro STSP. Pro předtsavu uveďme pouze, že typickým místem pro řešení ATSP je například cestování po městě. V něm totiž nalezneme jak obousměrné, tak i jednosměrné ulice.

Mezi základní heuristiky pro řešení TSP, které jsou implementovány v této práci, patří *Nearest neighbor* a *Greedy*. Existují řešení mnohem komplexnější, kterým zde není věnován

prostor - zmiňme alespoň *Ant colony optimization (ACO)*. Výhodou takových přístupů je jednoznačně schopnost řešit v reálném čase problém TSP i pro několik tisíc měst a to s přesností velmi blízkou optimálním hodnotám.

2.3.3 Held-Karpova spodní hranice

Tento odstavec vychází z článku [4].
Held-Karpova hranice (HK hranice) je spodní mez ceny optimální cesty při řešení problému TSP, vytvořená Michaelem Heldem a Richardem Karpem. Protože nalezení optimálního řešení není vždy realizovatelné, je tato hranice používána pro ohodnocení výkonnosti zvoleného algoritmu. Held-Karpova hranice určuje míru blízkosti výsledku získaného zvoleným algoritmem k optimálnímu řešení. HK hranice bývá většinou vzdálena o $0,8$ % od optimální délky trasy.

Nearest Neighbor

Odstavec vychází z článků [15] a [10].
Pro tento postup neexistuje žádná garance přiblížení se k optimálnímu řešení - z toho důvodu se příliš nedoporučuje pro optimalizaci tras - zde je využit kvůli své jednoduchosti. Tento algoritmus zpravidla dosahuje výsledků, které jsou v rámci 25% HK spodní hranice. Složitost tohoto algoritmu je $O(n^2)$.

Průběh algoritmu:

1. Nastav všechna města jako nenavštívená.
2. Vyber náhodné město z množiny nenavštívených a nastav jej jako počáteční město n_0.
3. Najdi nejbližší nenavštívené město vzhledem k aktuálnímu vybranému městu a vyber jej.
4. Označ vybrané město jako navštívené.
5. Je množina nenavštívených měst neprázdná? Pokud ano, vrať se k bodu 3.
6. Vrať se do počátečního bodu.

2.3.4 Greedy

Tento odstavec čerpá z článku [15].
Složitost tohoto algoritmu je $O(n^2 log_2(n))$. Tato heuristika vytváří postupně cestu tím, že jsou opakovaně vybírány nejkratší cesty mezi N městy, dokud není vytvořen jediný cyklus s právě N hranami a zároveň nevznikne žádné město o stupni[1] větším než 2. Řešení nalezené tímto algoritmem se běžně pohybuje v rozmezí $15 - 20$% HK spodní hranice.

[1] Stupeň města = stupeň bodu v grafu. Udává počet hran, kterými je takové město spojeno s okolními městy.

Průběh algoritmu:

1. Seřaď všechny hrany a nastav jako nevybrané.
2. Vyber nejkratší hranu z nevybraných a přidej ji k doposud vybraným hranám pokud:
   - Nevznikne žádné město o stupni `>2`.
   - Nevznikne cyklus s počtem hran `<počet_měst`.
3. Vznikl cyklus o počtu hran rovném počtu měst? Pokud ne, jdi na krok 2, jinak skonči.

2.4 Bin packing problem

Bin packing problem (dále BPP) figuruje v této práci na místě, kdy je třeba rozhodnout kolik přepravek bude potřeba pro naložení jednoho zákazníka při cestě po skladu. Opět zde mluvíme o klasickém, kombinatorickém a NP-úplném problému, pro který ovšem existuje řada heuristik. Tuto problematiku rozdělujeme podle typu prostoru, ve kterém se snažíme docílit uspořádání položek do boxů a dále podle toho, zda máme k dispozici pouze jeden či více typů boxů. V implementaci je počítáno s možností použití jak jednoho tak více typů boxů.

2.4.1 Definice BPP pro jeden typ boxu

Tento odstavec vychází z prezentace [12].

Mějme N položek, kde každá z nich je definována číslem w_j $(j = 1, ..., n)$, které zpravidla reprezentuje váhu či objem položky, a dále neomezený počet stejnorozměrných boxů o kapacitě c. Následně balíme všechny položky do co nejmenšího počtu boxů tak, aby u žádného boxu nedošlo k přetečení jeho maximální kapacity. Bez újmy na obecnosti můžeme předpokládat, že $0 < w_j < c$ pro každé j.

2.4.2 Definice BPP s více typy boxů

Tento odstavec vychází z článku [20].

Tato varianta je variantou klasického BPP, kde navíc uvažujeme proměnlivou velikost jednotlivých boxů. Mějme list $L = (a_1, ..., a_n)$ položek, ve kterém každá položka má velikost $s(a_i) \in (0, 1\rangle$, a dále několik typů boxů $B^1, ..., B^l$ o rozměrech $1 = s(B^1) > s(B^2) > \cdots > s(B^l) > 0$, při čemž máme neomezený počet boxů od každého typu. Cílem je potom naskládat všechny položky do boxů tak, aby součet velikosti boxů byl co nejmenší. Pokud budou všechny boxy jednoho typu, budeme řešit klasický BPP.

2.4.3 Heuristiky

2.4.4 FFD

Tento odstavec vychází z článku [18] a prezentace [12].

Pro bližší pochopení následujících odstavců uvedeme nejdříve algoritmus First Fit (FF):

1. Vyber položku ze seznamu položek.
2. Pokus se vložit položku do některého z již otevřených boxů, a to od prvního k poslednímu.

3. Pokud se podařilo umístit položku do boxu, opakuj krok 1, jinak pokračuj krokem 4.

4. Otevři nový box, vlož do něj položku a umísti box na konec řady otevřených boxů.

5. Pokud není seznam položek prázdný opakuj krok 1, jinak skonči.

Mezi obdobné varianty pro algoritmus FF patří algoritmy Best Fit, Any Fit či Next Fit.

FFD nebo-li First Fit Decreasing je lepší - nicméně off-line - variantou algoritmu First Fit (ten spadá mezi on-line algoritmy). U FFD se jedná o algoritmus, který nakládá s položkami seřazenými sestupně podle velikosti tak, že každou z nich dá do prvního boxu (mluvíme o jednom typu boxu), ve kterém je dostatek místa. Toto řešení BPP je v této práci implementováno kvůli jeho poměrně vysoké přesnosti - bylo dokázáno, že výsledek poskytnutý algoritmem FFD nepřekročí hranici $\frac{11}{9}OPT + \frac{2}{3}$, kde OPT označuje optimální počet boxů využitý pro zabalení daného počtu položek. Časová náročnost FFD je rovna $O(n \log n)$. Pro srovnání, algoritmem First Fit lze získat řešení nepřesahující hranici $1,7OPT$, stejně tak algoritmem Best Fit. Pro off-line algoritmus s názvem Best Fit Decreasing platí stejná hranice jako pro FFD.

2.4.5 FFDS

Odstavec vychází z článku [20].

Jedná se o algoritmus *First Fit Decreasing using Smallest possible bin sizes* (dále FFDS). Jeho nasazení přichází v rámci té varianty BPP, kdy máme k dispozici více, než jeden typ boxu. Popišme prvně algoritmus FFS (*First Fit using Smallest possible bin sizes*), který je on-line algoritmem a jehož princip je využit v rámci FFDS:

1. Seřaď boxy od nejmenšího po největší.

2. Přiřaď první položku do nejmenšího možného boxu.

3. Pokud se následující položka vejde do jakéhokoli již otevřeného boxu, aplikuj na ni First Fit (viz výše 2.4.4). Jinak dej položku do nejmenšího možného nového boxu.

4. Existují další nepřiřazené položky? Pokud ano, opakuj krok 3, jinak skonči.

Protože v této práci všechna data již známe, využijeme opět seřazení všech položek od největší po nejmenší. Tento krok se stane prvním krokem výše napsaného algoritmu - tím z FFS získáme FFDS. V kombinaci se seřazením typů dostupných boxů od nejmenšího po největší pak můžeme rychleji nalézt box, jehož rozměry budou nejlépe odpovídat rozměrům dané položky.

2.4.6 Využití v 1D a ve 3D

Algoritmy FFD i FFDS lze použít v 1D i 3D prostoru. Rozdíl pak spočívá v tom, že zatímco v 1D prostoru se řídíme pouze jedním rozměrem a jeho porovnáváním (nejčastěji objemem), tak ve 3D prostoru využíváme všechny rozměry položky a zkoušíme nasimulovat její reálnou polohu v boxu. Časová náročnost je zjevně náročnější pro 3D řešení problému BPP. Pro představu pouze uveďme, že v nejhorším případě může mít algoritmus, popsaný v článku [5], časovou náročnost $O(n^3)$.

2.5 Metody pro seskupování položek

V této sekci je popsán matematický základ pro metody, které byly použity pro nalezení skupin položek s cílem zkrátit celkovou vzdálenost při jejich sběru ze skladu. Implementovány byly tři přístupy, jejichž konkrétní podoba je popsána v sekci 4.

2.5.1 Metoda nejmenších čtverců

Tato sekce vychází z prezentace [7] a webové stránky [9].

V rámci této matematické metody je nalezena přímka, od níž má každý bod z dané množiny bodů co nejmenší vzdálenost.

Mějme danou množinu bodů se souřadnicemi $[x_i, y_i]$, $i \in 0, 1, ..., n$. Hledáme takovou přímku $y = ax+b$, pro kterou bude $\sum_{i=0}^{n}(y_i - a - bx_i)^2$ minimální. Pro nalezení numerického řešení je nutno vyřešit rovnice

$$a.\text{počet\_uzlů} + b.\sum x_i = \sum y_i \tag{2.5}$$

$$a.\sum x_i + b.\sum x_i^2 = \sum y_i.x_i \tag{2.6}$$

Dále je v této práci použit výpočet vzdálenosti d bodu od přímky podle vzorce

$$d = \frac{|w.x_0 + v.y_0 + c|}{\sqrt{w^2 + v^2}} \tag{2.7}$$

V něm je w rovno koeficientu $-1 * a$ a c je rovno koeficientu $-1 * b$, kde koeficienty a, b pocházejí ze, v předešlém kroku získaného, směrnicového tvaru přímky. v je potom rovno 1.

2.5.2 K-Means clustering

Tento odstavec vychází z webových stránek [3] a [1].

K-Means clustering je metoda používaná pro strojové učení bez dozoru, mající za cíl seskupit body podle daného kritéria, přičemž takových skupin bude tolik, kolik bylo na počátku specifikováno číslem K. Určení tohoto čísla hraje velmi významnou roli. Pro jeho získání se využívá různých metod, z nichž jmenujme tu nejznámější - metodu Elbow. Tato práce však vyžaduje vzít v úvahu i další parametry a proto je odhad čísla K proveden jiným způsobem (viz sekce 4).

Centroid je rovněž významným pojmem této metody - jedná se o bod, jehož pozice je na počátku náhodně určena, nebo mu je přiřazena poloha některého z bodů. Od něj se pak odvozuje podobnost jednotlivých bodů vůči sobě, a tedy jejich příslušnost do jedné skupiny. V algoritmu figuruje vždy K centroidů; každý pro jednu skupinu.

Algoritmus

1. Urči číslo K definující počet hledaných skupin.
2. Vyber nebo náhodně rozmísti K bodů - centroidů.
3. Každý bod přiřaď k nejbližšímu centroidu.
4. Vypočítej rozptyl vzdáleností bodů od centroidů a přepočítej podle něj pozice centroidů.

5. Každý bod znovu přiřaď k nejbližšímu centroidu.
6. Změnilo se rozdělení bodů mezi centroidy? Pokud ano, pokračuj bodem 4.
7. Model je připraven.

Kapitola 3

Problematika řešené úlohy

3.1 Popis systému

V systému je provedeno N objednávek od různých zákazníků. Po čase T (odpovídající tomu, že systém vyhodnotí dostatečnou naplněnost) dojde k *„zavlnění“* - v této fázi jsou vytvořeny dávky (dále HD - Hromadný Dodací list - toto označení zastupuje jednoho **zákazníka** a zjednodušuje pojem dávka) a jednotlivým produktům je přidělena adresa. Prodejní doklady na *stejného zákazníka a stejnou dodací adresu* jsou spojeny do jednoho **HD**. Data jsou dále exportována, pracovníky skladu vychystána a odeslána zákazníkům.

3.1.1 Dosavadní řešení

Systém firmy ve své nynější podobě pracuje tak, že rozřazuje náhodně položky do vozíků na základě celkové kapacity vozíku - ta nesmí být přesažena. Kapacita vozíku je vypočítána jako součet objemu devíti přepravek s navýšením o $52,6\ \%$ (což je kompenzace otevřeného prostoru nad přepravkou). Dále systém nesmí umístit do jednoho vozíku více, než osm zákazníků a zároveň za všech situací musí splnit kritérium nedělitelnosti zákazníka (viz dále). Výstupem jsou potom seznamy položek, kde každý seznam představuje jeden „vozík“.

Jednotlivé položky, které jsou umístěny na vozíku, jsou potom řazeny do trasy podle adres. Každá adresa má fixně přiřazeno pořadí odpovídající její vzdálenosti od místa, kde pracovník musí vzít vozík. Průchod všemi adresami na skladě podle jejich vzestupného pořadí by potom vytvořil *„hadovitou“* trasu skladem.

Zaměstnanci skladu se řídí podle seznamů položek, vytvořených pro jednotlivé vozíky. Průběh *vychystávání* potom spočívá ve sběru položek ze skladu a jejich shromažďování na balicím pracovišti.

3.1.2 Předvychystávání

Nutno zmínit, že během celého dne probíhá proces předvychystávání. Jedná se o interní proces Firmy, v rámci nějž dochází k přemisťování položek ze vzdálenějších prostor skladu do sektorů blíže samotnému výchozímu a cílovému stanovišti pro vychystávání. V této práci není skutečnost fáze předvychystávání nijak zohledňována. Implementovaný algoritmus by měl být schopen pracovat s položkami rozmístěnými kdekoliv po skladu. Nicméně řešení, které přinese tato práce, může ukázat, že dvojitý[1] pohyb položky je redundantní a vše lze zvládnout bez předvychystávání.

[1] Prvním pohybem míníme předvychystání, druhým pohybem pak chápeme samotné vychystání položky.

3.1.3 Data

Vstupem algoritmu jsou položky, patřící jednotlivým zákazníkům, které se mají vychystat. Položky jsou ve definovány v tabulce, jejíž sloupce obsahují hodnoty (i s příklady):

- HD (Alza Praha 3, CZC Opava, COMFOR Poruba,...)
- Položka (Lenovo IdeaPad AX, Apple MacBook Pro 10, ThinkPad T150,...)
- Počet kusů produktu (1, 5, 20,...)
- Adresa (G000159545, PO10494564,...) určující místo na skladě

3.1.4 Pravidla

Pravidla níže napsaná je nutno dodržet, aby nedošlo ve skladu k nekonzistentním situacím vzhledem k zásadám firmy.

- Jeden zákazník (objednávky od jednoho zákazníka spojené do HD) je nedělitelný mezi vozíky.
- Vozík je schopen pojmout až devět přepravek (rozměry přepravky jsou 28x19x45 cm).
- Zboží mající větší rozměry, než je možno umístit do přepravky, musí být stejně vychystáno (řešeno násilným přiřazením do vozíku a ponecháním na pracovníkovi, aby případný problém vyřešil).
- Jedna přepravka může obsahovat zboží pouze od jednoho zákazníka (ten samozřejmě může být ve více přepravkách).
- Jedna přepravka může obsahovat zboží od více zákazníků pouze v tom případě, pokud každé zboží je položka o jednom kusu.

3.1.5 Statistiky

- Je řešen prostor o cca třiceti tisíci adresách a dvaceti tisíci produktech.
- Množství zpracovávaných dat se pohybuje od jedné do tří tisíc položek.
- Každému vozíku je přiřazeno maximálně osm HD (osm zákazníků).
- Nejvíce bývá aktivních čtyřicet pracovníků (tedy naráz bylo možno řešit čtyřicet vozíků) v sezóně, mimo sezónu patnáct.

3.1.6 Možná rozšíření

Protože se logistika velmi rychle vyvíjí, je nezbytné uvažovat při řešení této práce možnosti, které firma aktuálně pouze zamýšlí - patří zde zvětšení rozměrů vozíku, čímž by zároveň došlo k navýšení počtu přepravek, které by se do něj vešly. Další ideou je poskytnutí více typů přepravek, tudíž by algoritmus měl být do budoucna schopen řadit položky efektivně do patřičných boxů.

3.1.7 Popis adres ve skladu

Pro pohyb na skladě je nezbytné specifikovat a lokalizovat jednotlivá skladová místa. Toto se setkává s komplikací spočívající v tom, že skladový prostor se obvykle skládá z více sekcí, které často vypadají různě z hlediska šířky regálů, pater a ulic (průchozích koridorů).

Protože sklady jsou rozličné a stejně tak adresování míst v nich, je způsob reprezentace pohybu a vzdáleností vztažen na konkrétní jeden sklad.

Jeho prostor lze klasifikovat jako euklidovský (tedy pravoúhlý) - to zajišťuje vzájemné rozestavení regálů. Ve skladu jsou pro sběr položek relevantní tři sektory, kdy každý z nich má odlišnou velikost. Jejich adresy nicméně mají stejný formát, který v sobě zahrnuje čísla odpovídající souřadnicím ve 3D prostoru (použitelné pouze v prostoru sektoru) a označení daného sektoru. Reprezentace os v adrese je následná:

- Osa X reprezentuje počet buněk se zbožím v jedné řadě - jejich počet se může lišit v rámci jednotlivých řad (přestože ty mají stejnou délku) v jednom sektoru. Například sektor G obsahuje v jedné řadě 246 nebo 31 buněk - samozřejmě o rozdílné šířce. Stejně tak i další sektory mívají rozdílné počty buněk ve svých řadách.
- Osa Y představuje počet řad v jednom sektoru.
- Osa Z udává výšku skrz číslo police - ty jsou v regálech číslovány od země vzestupně.

Ulice a obecně prostory, kde není zboží, nijak adresovány nejsou.

Kapitola 4

Návrh a Implementace

4.1 Základní řešení

Základní řešení si kladlo za cíl implementovat jednoduché řešení optimalizace logistického procesu vychystávání, který je popsaný výše. Toto řešení pak slouží jako výchozí bod pro vytvoření finální implementace.

4.1.1 Spuštění výpočtu

Ovládání celého výpočtu je umístěno do objektu `CartsLoader`, který zodpovídá za uložení všech nezbytných dat během výpočtu a zároveň disponuje všemi nezbytnými funkcemi pro výpočet. Důvodem pro jeho existenci je především možnost uchovávat v průběhu programu, který bude knihovnu využívat, specifikace jednotlivých položek. Jeho konstruktor přijímá jako parametr instanci třídy `Warehouse` - v ní je řešena implementace výpočtu vzdáleností ve skladu.

Na začátku programu je nutné nejdříve načíst data, která jsou předpokládána ve formátu JSON. K tomu nám slouží funkce třídy CartsLoader `setItems()` a `setItemsSpecifications()`, obě přijímají jako parametr cestu k souboru. První ze zmíněných funkcí zajistí načtení položek, které se mají vychystat, druhá pak načtení rozměrových specifikací těchto položek. V rámci specifikací rozlišujeme *kartón (carton)* a *jednotku (unit)*. Kartón popisuje rozměry balení více kusů jednoho produktu. Jednotka udává rozměry jednoho kusu zboží. Následně spustíme výpočet funkcí `loadingCartsRun()`.

Zmiňme, že pro optimalizaci rychlosti výpočtů je v kódu vytvářeno mnoho proměnných typu *mapa* nebo *multimapa* za účelem rychlejšího přístupu k datům pod vhodným klíčem (mapa slouží pro uložení dvojic klíč-hodnota, multimapa umožňuje uložení více hodnot k jednomu klíči).

4.1.2 Fáze výpočtu

Po načtení dat zmíněnými funkcemi dojde ke spojení položek s jejich rozměrovými specifikacemi . Jednotlivé položky jsou potom sdruženy podle zákazníků, pod které spadají, do tříd typu `Customer`.

Dále je vypočítána celková velikost všech zákazníků a probíhá rozdělení jejich položek do boxů pomocí funkce `decompIntoBoxes()` třídy `Customer`. Dojde k vytvoření nové *mapy*, ve které je klíčem celkový objem všech položek zákazníka (sestupné řazení). Pak je volána funkce `runAlgorithm()` pro seskupení zákazníků do vozíků podle požadavku nejkratší vý-

sledné cesty, která vrátí jako výsledek seznam vozíků se seřazenými položkami - samotné řazení položek je řešeno ve funkci `sortItemsInCart()`.

4.1.3 Seskupení zákazníků do vozíků - funkce runAlgorithm

V této funkci probíhá rozřazení zákazníků do vozíků. Algoritmus, který toto zaštiťuje pracuje následujícím způsobem:

Procházejí se zákazníci od těch nejvíce objemných. Vždy je vzat jeden zákazník a k němu jsou následně procházeni všichni zbylý zákazníci mající objem, který vyhovuje zbylému volnému místu na vozíku po načtení prvního zákazníka. Pro každého jednoho z nich jsou vypočítány vzdálenosti všech jeho položek vůči položkám od prvního zákazníka a je určena průměrná vzdálenost jako podíl sumy všech vzdáleností mezi jednotlivými položkami a počtem těchto vzdáleností. Podle této průměrné vzdálenosti jsou zákazníci opět seřazeni sestupně do pole. Toto pole je procházeno a každý zákazník je otestován, zda se vejde do vozíku - pokud ano, je do něj přiřazen. Tento postup je opakován dokud nejsou všichni zákazníci rozřazeni (tedy dokud není vstupní pole s nimi prázdné).

Algoritmus zapsaný do pseudokódu:

```
sort the customers in cart in descending order by volume
cartList ← empty;
for biggestCustomer in customerMap do
    //nalož největšího zákazníka do vozíku
    cart ← newCart
    cart.loadCustomer(biggestCustomer);
    customerMap.remove(biggestCustomer);
    bestFitCustomersMap ← empty;
    //projdi zbylé zákazníky, kteří vyhovují zbývající kapacitě vozíku
    for nextCustomer in customerMap.lowerBound(cart.getCartFreeSpace()) do
        //vypočítej průměrnou vzdálenost všech položek aktuálně procházeného zákazníka
        //od zákazníka načteného ve vozíku
        distance ← 0;
        countOfDistances ← 0;
        for bigCitem in biggestCustomer.items do
            for item in nextCustomer.items do
                distance ← distance+
                    getDistanceBetweenAddresses(bigCitem.address, item.address);
                countOfDistances ← countOfDistances + 1;
            end for
        end for
        bestFitCustomerMap ← bestFitCustomerMap+
                {distance/countOfDistances, nextCustomer};
    end for
    //projdi všechny nejbližší zákazníky, pravděpodobně vyhovující objemem
    //a pokus se je načíst do vozíku
    for bestFitCustomer in bestFitCustomerMap do
        customerLoaded ← cart.loadCustomer(bestFitCustomer);
        if customerLoaded then
            customerMap.remove(bestFitCustomer);
```

```
        end if
    end for
    cartList.pushback(cart);
end for
```

4.1.4 Rozdělení položek do boxů - funkce decompIntoBoxes

Zde probíhá rozdělení položek zákazníka do přepravek - boxů. Jejich kapacita je předem známa a pevně stanovena. Je třeba zmínit, že položky na skladě mohou mít rozměry, které neodpovídají celkové kapacitě boxů, ale přesto musí být načteny - v takovém případě dojde k primitivnímu ošetření způsobem přiřazení této položky do prázdné přepravky (to je prováděno s ohledem na požadavek nutnosti vychystání **všech** položek).

Algoritmus rozdělování do boxů je v základním řešení realizován jako *First fit decreasing (FFD)*.

Algoritmus zapsaný do pseudokódu:

```
sort the items in cart in descending order by volume
box ← newBox;
cust ← Customer;
for item in cust.items do
    if box.freeVolume ≥ item.volume then
        //položku lze načíst do aktuálně připraveného boxu
        box.load(item);
    else
        is_loaded ← false;
        //projdi předcházející boxy a zkus do nich načíst položku
        for prevBox in cust.boxes do
            if prevBox.freeVolume ≥ item.volume then
                prevBox.load(item);
                is_loaded ← true;
            end if
        end for
        if is_loaded then
            //položka je načtena, pokračuj další položkou
            break;
        end if
        //vytvoř nový box pro aktuální položku
        cust.boxes.add(box);
        box ← newBox;
        box.load(item);
    end if
end for
```

4.1.5 Seřazení položek ve vozících - funkce sortItemsInCart

Jejím úkolem je seřadit položky jednoho vozíku tak, aby dráha ujitá při průchodu byla co nejkratší. Je zde implementován algoritmus *Nearest Neighbor*, který je nejjednodušší heuristikou problému TSP (viz strana 6). V rámci něj je procházeno všemi položkami,

pro každou je spočítána vzdálenost od výchozího bodu a je vybrána ta, která je nejblíže. Výchozí položkou je vždy ta, která byla naposledy vybrána.

Jako výchozí bod je uvedena adresa *G000000136*, která odpovídá přibližně středovému bodu mezi dvěma vjezdy do sektoru **G** (viz nákres v příloze B.1).

```
items ← cart.Items;
sortedVector ← empty;
//načti počáteční adresu - od ní se odvíjí výběr dalších blízkých položek
prevAddress ← "G000000136";
while items not empty do
    //projdi zbývající položky a najdi mezi nimi nejbližší vzhledem k uložené adrese
    curBestDistance ← MAX_INT;
    curBestItem ← empty;
    for item in items do
        tmp ← getDistanceBetweenAddresses(prevAddress, item.address);
        if tmp < curBestDistance then
            curBestDistance ← tmp;
            curBestItem ← item;
        end if
    end for
    //nejbližší položku přiřaď do výsledného seznamu
    //a ulož si její adresu jako adresu poslední přidané položky
    items.remove(curBestItem);
    sortedVector.append(curBestItem);
    prevAddress ← curBestItem.address;
end while
```

Vozík je reprezentován třídou `Cart` - ten v sobě ukládá všechny naložené zákazníky, jejich rozdělení do přepravek - reprezentovaných třídou `Box` - a údaje o své zaplněnosti. Dále disponuje dvěma důležitými funkcemi - funkcí `loadCustomer()` s parametrem `Customer`, která vrací *pravdivostní hodnotu* v závislosti na tom, zda vozík v sobě našel dost místa pro načtení zákazníka, nebo ne. Druhou funkcí je `getItemVect()`, která vrací vektor všech položek umístěných ve vozíku.

4.1.6 Adresování a měření vzdáleností

Zodpovídají za ně třídy `WAddress` a `Warehouse`.

Třída `WAddress` obsahuje funkce pro načtení jedné adresy a také pro extrahování souřadnic, které daná adresa reprezentuje.

V třídě `Warehouse` je zajištěno vypočítání vzdáleností. Přestože je sektorů více - tři - je v této základní implementaci tato skutečnost pominuta za účelem zjednodušení (přechody mezi sektory vyžadují nalezení jistého ideálního přístupu jelikož, jak je zmíněno na straně č.15, koridory nejsou adresovány). V úvahu je vzat pouze sektor s označením **G** (viz nákres v příloze B.1, protože v něm se odehrává většina veškeré vychystávací činnosti. Počítání vzdáleností v tomto sektoru zajišťuje funkce `getDistanceInSectorG()`, která je volaná skrz funkci `getDistanceBetweenAddresses()` - ta má jako operandy dvě adresy, mezi kterými chceme zjistit vzdálenost.

Před samotným výpočtem vzdálenosti je třeba si definovat dva vektory - v prvním jsou uloženy řady, které obsahují rozdílné počty buněk oproti ostatním řadám a ve druhém jsou uloženy čísla vertikálních koridorů (souběžných s abstraktní osou Y). Čísla těchto vektorů

jsou uměle vytvořena jako desetinná čísla odpovídající buňkám, mezi kterými se ulice nachází (pro příklad uveďme ulici se souřadnicí $7,5$, která odpovídá pozici mezi buňkami s čísly 7 a 8). Dále musíme přepočítat buňky (tedy osu X) v těch řadách, které mají 246 buněk vzhledem k řadám o 31 buňkách. Tím dosáhneme možnosti porovnávat mezi sebou souřadnice, které se vztahují k různým řadám.

Při výpočtu vzdálenosti mezi dvěma adresami se postupuje takto:

1. Načtení souřadnice X první adresy a získání nejbližšího koridoru skrz porovnání s definovanými koridory pro sektor G.
2. Získání vzdálenosti mezi nejbližším vertikálním koridorem a souřadnicí X první adresy.
3. Získání vzdálenosti mezi souřadnicemi Y od první a druhé adresy (prostý odečet).
4. Načtení souřadnice X druhé adresy a získání vzdálenosti od vybraného koridoru v kroku 1.
5. Návrat součtu absolutních hodnot všech získaných vzdáleností.

4.2 Hlavní řešení

4.2.1 Spuštění

Pro spuštění zpracování dat je nezbytné vybrat algoritmy, které budou následně použity pro výpočet. Tento výběr algoritmů se ukládá ve třídě `SolverConfiguration`. Dále je možné přidat různé typy boxů - pro tento účel je využita třída `BoxConfiguration`. Pokud žádný box do této třídy není přidán, je implicitně použit pouze jeden typ boxu, jehož rozměry byly definovány Firmou. Nakonec je nutné přidat definici (rozměry a počet polic) vozíku skrz třídu `CartConfiguration`.

4.2.2 Načítání dat

Zatímco v základním řešení se využívalo načítání dat dvoufázově (viz strana 16), zde se jedná pouze o načtení jediného souboru dat pomocí funkce `loadItemsAndSpec()`. Ta přijímá jako parametr opět buď cestu k souboru typu JSON, nebo přímo data ve formátu JSON, předaná jako textový řetězec. Vstupní soubor musí obsahovat jak jednotlivé položky s jejich umístěním na skladě, tak jejich rozměrové specifikace. Tím došlo ke zvýšení rychlosti zpracování vstupních dat.

Dále bylo přidáno rozšíření, které umožňuje načíst Firmou vytvořené rozřazení položek do vozíků společně s jejich pořadím na trase. Z těchto dat je následně získána odhadovaná vzdálenost pro každý vozík, kterou pracovník nachodí při sběru položek přiřazených k danému vozíku. Toto je implementováno ve funkci `testUsedData()` třídy `CartsLoader`. Rozšíření bylo využito v rámci testování a srovnávání výsledků.

4.2.3 Adresování a měření vzdáleností

Oproti základnímu řešení se již počítá se všemi sektory - tedy G, V, P a GP (sektor GP je součástí G sektoru). Při implementaci převodu skladových adres, které patří ke zmíněným sektorům, je třeba si uvědomit, že jejich formát je platný v jednom konkrétním sektoru. Sektory jsou specifické svými buňkami, které mají rozdílné rozměry. Stejně tak i jednotlivé koridory (ulice), kterými se prochází v daném sektoru, mají své vlastní šířky a délky a jsou

na různých pozicích. K ulicím je nutné navíc zmínit, že jejich přítomnost není v řešeném skladě nijak značena.

Hlavní otázka se týkala řešení skladového prostoru jako celku - nabízela se možnost řešit jednotlivé sektory zvlášť s jejich vlastním adresováním - obdobně, jako tomu je v základním řešení této práce. To se ovšem setkává s mnoha problémy, které značně zhoršují přesnost výpočtu. Mluvíme především o nutnosti lokalizovat jednotlivé ulice v rámci daných sektorů. Jednotlivým koridorům sice lze přiřadit nějaká čísla, odpovídající prostoru mezi buňkami (viz základní řešení), nicméně toto řešení ústí do komplikace při východu ze sektoru a následně při výpočtu vzdálenosti mezi sektory. Tento mezi-sektorový prostor (obdobně s ulicemi v sektorech) by mohl mít vlastní způsob značení, ovšem to by globálně bylo pravděpodobně velmi nejednoznačné a zmatečné, a tak se v novém řešení od tohoto způsobu využívaní adres upouští.

Výhodou práce se skladovou reprezentací adres je jednoduchá extrahovatelnost souřadnic a práce s nimi. Ovšem proti tomu opět převažuje nevýhoda zdánlivě stejných jednotek, ve kterých jsou jednotlivé složky - X,Y a Z - v adrese zaznamenány. Jejich nepatrné odchylky znamenaly značné rozdíly ve výpočtu vzdáleností napříč skladem.

V novém řešení je stále využita výhoda euklidovského prostoru. Hlavní předností nového přístupu je implementace mapování skladových adres do souřadnic ve 2D prostoru - tímto prostorem je chápán celý prostor skladu. To vyžadovalo určení počátečního bodu - ten je umístěn do levého horního rohu skladu.

Každý sektor má zadefinovanou specifikaci, která nese informace o vnitřním rozložení daného sektoru (šířka buněk, ulic,...). Pro lepší variabilitu a flexibilitu je navíc využito odsazení v osách X a Y. Toto odsazení odpovídá vzdálenosti sektoru od jeho sousedů. Jeho zavedením je umožněno jednoduše reflektovat případné změny v rámci rozložení skladu.

Přechody

Jednotlivé přechody mezi sektory jsou implementovány tak, že pro každý sektor jsou definovány jeho východy (*exit*), které slouží jako body na krajích daného sektoru, vůči kterým je vypočtena vzdálenost od první adresy. Potom následuje výpočet délky trasy mezi východy jednotlivých oblastí a navazuje výpočet vzdálenosti mezi výstupem cílového sektoru a druhé adresy, která se v něm nachází.

Nutno poznamenat, že všechny ulice jsou příliš úzké na to, aby v nich bylo možné využít možnosti jít *šikmo* - po přeponě - a tudíž veškeré výpočty lze značně zjednodušit čistým výpočtem euklidovské vzdálenosti mezi dvěma body. Tuto skutečnost navíc podporuje fakt existence vozíku, se kterým zaměstnanec nedokáže zvlášť účinně manipulovat. Jediným místem pro optimalizaci je případ, kdy dané dvě adresy jsou ohraničené stejnými ulicemi, a tedy musí bý provedeno porovnání mezi vzniklými dvěma možnostmi trasy, která tato místa může spojovat (viz příloha B.2).

Uveďme ještě, že sektory G a V se liší od P a GP tím, že jejich regály jsou v horizontálním směru. První dvojice zmíněných sektorů má navíc velmi příhodně rozmístěny spojující koridory tak, že na sebe přímo navazují. Výpočet vzdálenosti mezi dvěma adresami, kde každá z nich patří jednomu ze sektorů G nebo V, se téměř neliší od výpočtu vzdálenosti dvou adres, nacházejících se pouze v jednom z nich. Oproti tomu sektory P a GP mají regály ve vertikálním směru a tudíž výpočty vzdáleností mezi adresami, ve kterých tyto sektory figurují, jsou komplikovanější.

4.2.4 Implementace Bin packing - jmenný prostor *bp_solver*

Fáze skládání položek do boxů přichází na úplném počátku výpočtu, obdobně jako u základního řešení. Všechny algoritmy vycházejí z heuristiky First Fit pro BPP. Rozdíl mezi implementacemi spočívá především v tom, zda pracují v 1D nebo 3D a zda umí pracovat s více typy boxů. Celkem je v kódu realizována šestice funkcí, přičemž jejichž činnost se u některých z nich překrývá - ty funkce, které umí pracovat s více typy přepravek umí pracovat i s jedním. Implementace funkcí, které pracují s jedním typem přepravky, jsou nicméně mírně jednodušší, a také nepatrně rychlejší - proto zůstaly zachovány.

Prvním krokem při řešení tohoto problému je vždy rozdělení dané objednávkové položky na jednotky a balení (kartóny) - pokud položka disponuje specifikací svého balení. Jednotku i balení potom zastupuje instance třídy `BoxItem`, se kterou se pracuje dále. Tato fáze není v rámci pseudokódu uvedena.

Implementace musí dokázat vyřešit případ, ve kterém se položka do žádného boxu nevejde - v této práci je tato výjimka řešena tak, že je přidán nový box, kterému jsou přiřazeny rozměry dané položky a který je automaticky označen jako *naplněný*.

Algoritmus FFDS v 1D

Implementován ve funkci `mainSolution1DFFDS_multipleBoxTypes()` s využitím pomocné funkce `mainSolution1DFFDS_helper()`. Teoretický popis tohoto algoritmu se nachází na stránce 10. Níže je uveden pseudokód implementace FFDS v této práci:

Pomocná funkce: `bool mainSolution1DFFDS_helper()`

```
//projdi předcházející boxy a zkus do nich načíst položku
for box in opened_boxes do
    if box.getCapacity() ≥ item.getVolume() then
        //vyhovující box nalezen - položka načtena
        box.addItem(item);
        return true;
    end if
end for
//pokud se nepodařilo položku nikam přidat,
//vytvoř pro ni nový box s nejvíce vyhovujícím objemem
for box_type in box_types do
    if box_type.getCapacity() ≥ item.getVolume() then
        new_box ← new Box(box_type);
        new_box.addItem(item);
        opened_boxes.add(new_box);
        return true;
    end if
end for
```

Hlavní funkce: `mainSolution1DFFDS_multipleBoxTypes()`

```
sort items decreasingly according to the volume
sort box types increasingly according to the volume
opened_boxes[] ← NONE;
```

```
for item in items do
    if mainSolution1DFFDS_helper(item) then
        continue;
    else
        //pokud se položku nepodařilo nikam přidat
        if položka je jednokusová then
            box ← new Box(item dimensions);
            new_box.addItem(item);
            opened_boxes.add(new_box);
        else
            //pokud položku nelze nikam přidat, rozděl ji po jednom kusu
            //pokud to lze tak udělat
            pieces[] ← break the item down into units;
            for piece in pieces do
                if mainSolution1DFFDS_helper(item) then
                    continue;
                else
                    //položka má nestandardní rozměry - vytvoř fiktivní box
                    //o těchto rozměrech
                    box ← new Box(item dimensions);
                    new_box.addItem(item);
                    opened_boxes.add(new_box);
                end if
            end for
        end if
    end if
end for
```

Implementace BP ve 3D s posílením FFDS v 1D

Pro řešení ve 3D prostoru je využita knihovna napsaná v jazyce python[1]. Aby bylo možné ji použít, je vždy nutno specifikovat počet boxů každého typu. Řešení s touto knihovnou bylo vylepšeno využitím algoritmu FFDS v 1D a to takovým způsobem, že výsledná konfigurace boxů v 1D, získána algoritmem FFDS, je následně předána algoritmu převzaté knihovny. S položkami, které algoritmus nezvládl nikam umístit, je opětovně proveden celý proces.

4.2.5 Přiřazení zákazníka do vozíku

Běžný způsob přiřazení zákazníka do vozíku probíhá jako proces Bin Packing v jednorozměrném prostoru. Vždy jsou vzaty všechny boxy daného zákazníka a s každým z nich je proveden pokus vložit jej do dané police ve vozíku. Jako veličina, která určuje, zda lze box umístit do police či nikoli, je vzata šířka boxu a šířka police. Velikost Firmou používaných boxů je optimalizovaná tak, že za boxem nezůstává v polici žádné místo. Nicméně v případě použití přepravek s menší hloubkou dochází vlivem popsaného přístupu ke ztrátě prostoru za nimi.

Pokud je zákazník jednokusový a zároveň vozík obsahuje přepravku s takovými zákazníky, je proveden pokus dát všechny kusy společně do stejné přepravky - využit je pro to

[1] https://github.com/enzoruiz/3dbinpacking

1D nebo 3D algoritmus v závislosti na dříve použitém algoritmu pro rozdělování položek zákazníků do boxů. Pokud toto nevyjde, je zákazník přidán běžným způsobem.

Řešení implementované v této práci je schopné pokrýt i případ, kdy dojde k nutnosti zpracovat zákazníka, který je větší než kapacita vozíku. V takovém případě je potom tento zákazník přiřazen triviálně do $N + 1$ vozíků, kde N vozíků je naplněno a poslední vozík je dále k dispozici pro potenciální další zákazníky. Krajní případ, kdy je všech $N + 1$ vozíků maximálně naplněno, zde nevyžaduje speciální pozornost.

4.2.6 Implementace seskupování zákazníků do vozíku - jmenný prostor *grouping_customers*

Tato část je implementována ve třech různých algoritmech, které se všechny snaží maximalizovat zaplnění vozíku a minimalizovat dráhu. První z nich je převzat ze základního řešení - ten se řídí průměrnými vzdálenostmi všech položek mezi dvěma zákazníky. Druhý algoritmus vychází z metody nejmenších čtverců, třetí potom z metody K-Means clustering.

Využití metody nejmenších čtverců

Obdobně jako algoritmus využívající průměrnou vzdálenost i tento je založen na výběru prvního - označme jej jako hlavního - zákazníka, který je největší z dosud nepřiřazených zákazníků do žádného vozíku. K němu je následně vybírán vhodný - označme jej jako potenciálního - zákazník. Pro odhad vzdálenosti potenciálního zákazníka od hlavního je využita přímka, získaná metodou nejmenších čtverců - množinou bodů se stávají všechny položky (přesněji jejich pozice) od hlavního a potenciálního zákazníka. Následně je vypočítána průměrná vzdálenost všech těchto položek od získané přímky. Všichni potenciální zákazníci jsou seřazeni vzestupně podle takto získaných vzdáleností a poté probíhá jejich nakládání do vozíku - pokud se zákazník do vozíku nevejde, je přistoupeno k dalšímu...

Algoritmus:

```
cartList ← empty;
for biggestCustomer in customerMap do
    cart ← newCart;
    //vybrán první zákazník - načti jej do vozíku
    while not cart.loadCustomer(biggestCustomer, partly_load_allowed = true) do
        //pokud se zákazník nevejde do jednoho vozíku, přidej další vozík
        //a opakuj průběh se zbylými položkami zákazníka
        //dokud není zákazník zcela načten
        cartList.pushback(cart);
        cart ← newCart;
    end while
    customerMap.remove(biggestCustomer);
    bestFitCustomersMap ← empty;
    //projdi všechny nenačtené zákazníky, kteří mají vyhovující kapacitu
    for nextCustomer in customerMap do
        //vypočítej koeficienty a a b přímky vzniklé použitím metody nejmenších čtverců
        //ve které jako body figurují položky jak zákazníka ve vozíku (biggestCustomer),
        //tak zákazníka aktuálně procházeného (nextCustomer)
        bothCustomerItems[] ← biggestCustomer.items + nextCustomer.items;
```

```
        sumx ← 0; sumy ← 0; sumxx ← 0; sumxy ← 0;
        n ← bothCustomerItems.size;
        for item in bothCustomerItems do
            sumx ← sumx + item.x;
            sumy ← sumy + item.y;
            sumxx ← sumxx + item.x * item.x;
            sumxy ← sumxy + item.x * item.y;
        end for
        a ← (n * sumxy − sumx * sumy)/(n * sumxx − sumx * sumx);
        b ← (sumy − a * sumx)/n;
        //vypočítej průměrnou vzdálenost každé položky (od obou zákazníků)
        //od získané přímky
        distance ← 0;
        for item in bothCustomerItems do
            distance ← distance + | − 1 * a * item.x + 1 * item.y + −1 * b)/√(a * a + 1 * 1)|;
        end for
        bestFitCustomerMap.pushback({distance/n, nextCustomer});
    end for
    //projdi všechny nejbližší zákazníky (vypočítané v předchozím kroku),
    //pravděpodobně vyhovující objemem, a pokus se je načíst do vozíku
    for bestFitCustomer in bestFitCustomerMap do
        if cart.loadCustomer(bestFitCustomer) then
            customerMap.remove(bestFitCustomer);
        end if
    end for
    cartList.pushback(cart)
end for
```

Využití metody K-Means clustering

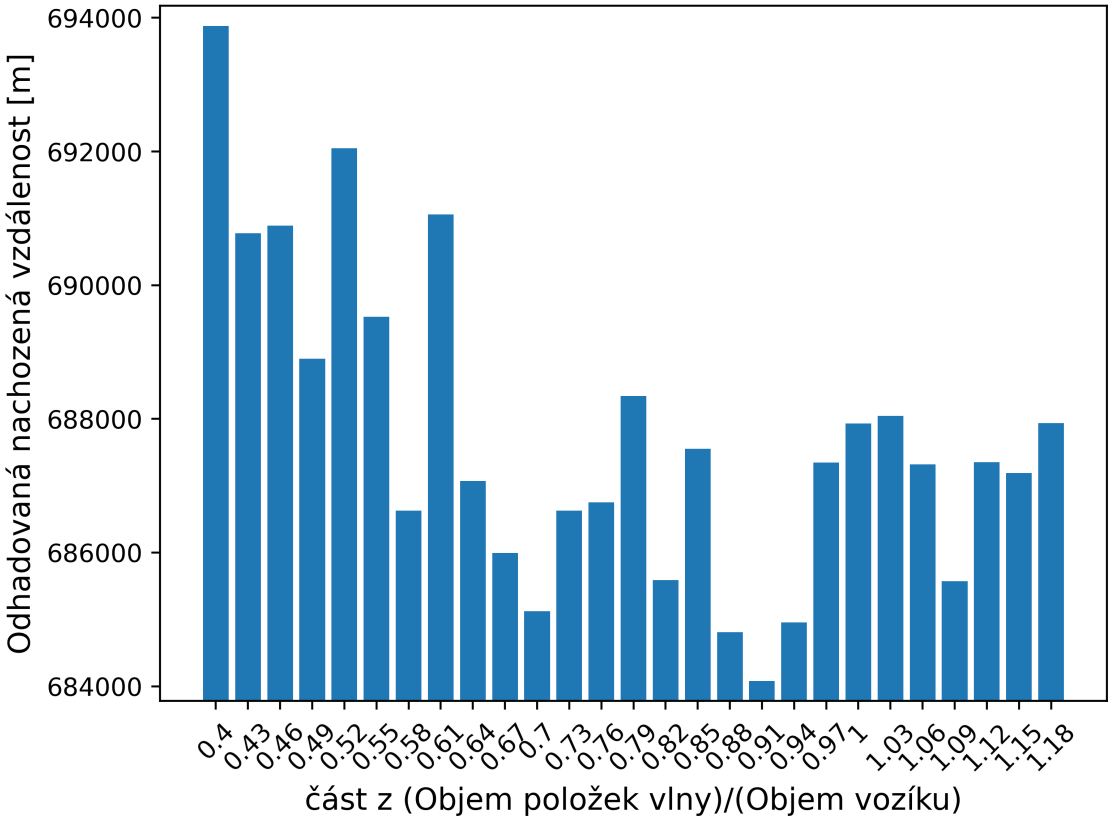
Tento přístup se od předchozích dvou liší především v tom, že se zde nezačíná od celkově největšího zákazníka. V první fázi je vypočten střed pro každého zákazníka, a to jako průměr souřadnic všech jeho položek. Poté je vytvořeno K skupin, kde jako centroid každé skupiny je zvolena pozice středu největšího, dosud nevybraného zákazníka. Následně probíhá N iterací (kde číslo N je zadáno před startem algoritmu uživatelem), ve kterých je prováděno přiřazování jednotlivých zákazníků do skupin podle vzdálenosti jejich středu od centroidu skupiny. Na konci každé iterace se přepočítává pozice centroidu jako průměr souřadnic k němu v dané iteraci přiřazených zákazníků a iterace se opakuje.

Po poslední iteraci se potom prochází všemi skupinami a v nich každým zákazníkem od nejobjemnějšího. Nejdříve je proveden pokus načíst každého z nich do nějakého z již vytvořených vozíků. Následně je první zákazník skupiny (zákazníci jsou stále seřazeni podle objemu sestupně) naložen do nového vozíku a k němu jsou postupně přiřazováni všichni ostatní zákazníci z dané skupiny, dokud není vozík plný. Vozík je poté uložen mezi vytvořené vozíky a zákazníci, které nebylo možné naložit, se ukládají stranou.

Celý proces od vytváření skupin (a určování centroidů) se opakuje tak dlouho, dokud nejsou všichni zákazníci naloženi.

Číslo K: Číslo K bylo v této práci určeno jako podíl celkového objemu všech zákazníků a objemu jednoho vozíku, zaokrouhlený nahoru. To se ovšem ukázalo na několika vlnách jako

ne zcela efektivní. Proto byl proveden pokus: vypočítané číslo K bylo postupně násobeno koeficienty $0,4$ ... $1,2$ a následně byl měřen celkový počet nachozených metrů u devatenácti vln. Nejlépe se potom ukázalo násobit popsaný podíl koeficientem $0,91$, viz graf 4.1. Nutno poznamenat, že výsledek v závislosti na tomto koeficientu kolísá od přibližně 684 km do 694 km, takže úspora je v nejlepším případě přibližně $1,44$ % celkově nachozené odhadované vzdálenosti během všech devatenácti vln.



Obrázek 4.1: Součet délek dvaceti vln při jednotlivých koeficientech

Počet iterací[2]**:** Jedná se opět o proměnnou, která může znamenat mnohem lepší výsledek co do celkové délky trasy. Počet iterací je možné změnit v rámci proměnné třídy `SolverConfiguration`. Pro zjištění co nejideálnějšího počtu iterací, který by dosahoval co nejlepšího výsledku pro nejvíce vln, byl proveden pokus na sedmi vlnách (změřena celková vzdálenost pro každou z těchto vln) pro pět až padesát sedm iterací. Z výsledků vyplývalo, že k ustálení pozic centroidů dojde po třicáté deváté iteraci - tedy toto číslo (zaokrouhleno na čtyřicet) bylo nastaveno jako základní počet pro opakování iterací. Kromě toho byla samozřejmě implementována ukončující podmínka pro K-Means clustering - k zastavení iterací dojde v okamžiku, kdy žádný z centroidů nezmění po přepočítání souřadnic svou pozici.

[2]Jako iteraci zde chápeme jeden běh metody K-Means clustering, během které jsou rozděleny body mezi centroidy a přepočítány pozice centroidů.

Algoritmus:

```
sort the customers in descending order by volume
cartList ← empty;
k ← user-defined number of iterations;
while customers not empty do
    //inicializuj skupiny (clustery) pomocí N prvních zákazníků (podle objemu)
    num_of_clusters ← customers.volume/cart.volume;
    Clusters[] ← empty;
    for i < num_of_clusters do
        new_cluster ← new Cluster;
        new_cluster.centroid ← customers[i].centerPoint;
    end for
    while k-- ≠ 0 do
        //přiřaď zákazníky k jejich nejbližším centroidům
        for customer in customers do
            cluster ← nearest cluster from Clusters according to the distance;
                        between its centroid and customer's center point
            cluster.addCustomer(customer);
        end for
        //přepočítej souřadnice centroidů
        for cluster in Clusters do
            sumx ← 0;  sumy ← 0;
            n ← cluster.assignedCustomers.size;
            for customer in cluster.assignedCustomers do
                sumx ← sumx + customer.centerPoint.x;
                sumy ← sumy + customer.centerPoint.y;
            end for
            cluster.centroid ← (sumx/n, sumy/n);
        end for
    end while
    //vymaž všechny zákazníky z hlavního pole - do něj budou přiřazeni
    //z každé skupiny jen ti, kteří se nikam nevejdou
    customer ← none;
    for cluster in Clusters do
        for customer in cluster.assignedCustomers do
            for cart in cartList do
                if cart.loadCustomer(customer) then
                    cluster.assignedCustomers.erase(customer);
                    continue;
                end if
            end for
        end for
        cart ← newCart;
        //vytvoř tolik vozíků, kolik je potřeba na načtení hlavního zákazníka
        while not cart.loadCustomer(biggestCustomer, partly_load_allowed = true)
do
            cartList.pushback(cart);
```

```
            cart ← new Cart;
        end while
        customerMap.remove(biggestCustomer);
        for customer in cluster.assignedCustomers do
            if cart.loadCustomer(customer) then
                cluster.assignedCustomers.erase(customer);
                continue;
            else
                //přidej zákazníka, který nemohl být načten, znovu pro další běh
                //algoritmu K-Means
                customers.pushback(customer);
            end if
        end for
        cartList.pushback(cart);
    end for
end while
```

4.2.7 Implementace řazení položek ve vozíku - jmenný prostor *tsp_solver*

Tato sekce obsahuje dva hlavní algoritmy - první z nich, algoritmus Nearest Neighbor, je obdobně implementován jako v základním řešení. Jeho efektivita není příliš vysoká a proto byl k němu přidán druhý algoritmus - Greedy. Protože časové požadavky firmy byly během vývoje splňovány, bylo rozhodnuto přidat navíc řešení metodou známou jako brute force - tedy řešení hrubou silou.

Veškeré implementace umožňují v rámci měření délky trasy nastavit výchozí a cílový bod. Díky tomu je možné i vytvoření cyklické trasy nastavením totožného výchozího a počátečního bodu.

Řešení hrubou silou

Je implementováno takovým způsobem, že první jsou nahrány do vektoru celých čísel všechny možné kombinace řazení čísel od 1 až N, kde N je celkový počet položek ve vozíku. Použití této metody je omezeno na vozíky o maximálním počtu deseti položek. Důvodem je velmi dramatickým způsobem narůstající časová složitost ($10! = 3628800$) pro vyšší počty položek. Každé číslo ve vektoru značí danou položku a nejlepší řazení je určeno podle nejmenší ujité vzdálenosti (ze všech permutací).

Pro efektivní vytvoření všech možných permutací N čísel je použit algoritmus z webové stránky [14].

```
n ← items.size;
order[] ← {0..n};
bestSolutionDistance ← MAX_INT;
bestSolutionItemOrder[] ← empty;
for i = 0; i < factorial(n); i++ do
    //získej další permutaci (z čísel 0..N)
    nextOrder[] ← getNextPermutation(order);
    tmpDistance ← 0;
    tmpItemOrder[] ← empty;
    //začni od počátečního bodu a pokračuj podle pozic položek v poli,
```

```
    //kde pozice jsou postupně udány podle získané permutace
    lastPoint ← start_point;
    for d = 0; d < n; d++ do
        tmpDistance ← tmpDistance +
                    getDistanceBetweenAddresses(lastPoint, items[nextOrder[d]]);
        lastPoint ← items[nextOrder[d]];
        tmpItemOrder.pushback(lastPoint);
    end for
    //přidej vzdálenost do koncového bodu od poslední položky
    tmpDistance ← tmpDistance + getDistanceBetweenAddresses(lastPoint, end_point);
    if tmpDistance < bestSolutionDistance then
        bestSolutionDistance ← tmpDistance;
        bestSolutionItemOrder ← tmpItemOrder;
    end if
end for
```

Implementace algoritmu Greedy

Algoritmus Greedy funguje podle principu popsaného v podkapitole 2.3.4. Zde je vypsána jeho činnost pro tuto práci, skládající se z hlavní funkce `greedyAlgorithm()` a pomocné funkce `checkForCycle()`:

Pseudokód funkce `checkForCycle()`, má návratovou hodnotu *true*, pokud najde cyklus, jinak $false$.

```
//vstupními parametry jsou: vektor hran (edges) a počet všech bodů (n)
mark ← edges[0].begin;
last_point ← edges[0].end;
edges.erase(edges[0]);
num_of_consecutive_edges ← 1;
consecutive_edge_found ← false;
while 1 do
    for edge in edges do
        if last_point in edge then
            if last_point == edge.begin then
                last_point ← edge.begin;
            else
                last_point ← edge.end;
            end if
            num_of_consecutive_edges++;
            consecutive_edge_found ← true;
            edges.erase(edge);
            break;
        end if
    end for
    //pokud byla nalezena následná hrana, zkontroluj, zda byla nalezen jiný
    //než malý cyklus
    if consecutive_edge_found then
        if last_point == mark && n ≠ num_of_consecutive_edges then
            return true;
```

```
        else
            continue;
        end if
    end if
    if !consecutive_edge_found && edges.size > 0 then
        mark ← edges[0].begin;
        last_point ← edges[0].end;
        num_of_consecutive_edges ← 1;
        edges.erase(edges[0]);
    else
        //byl nalezen cyklus o n hranách - ten je v pořádku
        return false;
    end if
end while
return false;
```

Pseudokód hlavní funkce `greedyAlgorithm()`:

```
//vstupními parametry jsou: vektor položek (items) a výchozí a
//koncový bod (start_point a end_point)
//získej všechna unikátní místa, na kterých se položky vyskytují
unique_points[] ← {start_point, end_point};
for item in items do
    if (item.x, item.y) in unique_points then
        unique_points.pushback(Point(item.x, item.y));
    end if
end for
//získej všechny hrany
edges[] ← {}
for i = 0; i < unique_points.size; i++ do
    for d = 0; d < unique_points.size; d++ do
        edges.pushback(Edge(unique_points[i], unique_points[d],
                    getDistanceBetweenAddresses(unique_points[i], unique_points[d])));
    end for
end for
//seřaď všechny hrany vzestupně podle délky
sort(edges);
//spoj všechny body greedy algoritmem
final_edges[] ← {Edge(start_point, end_point)};
used_points ← {start_point, end_point};
for edge in edges do
    first_exists ← 0;  second_exists ← 0;
    for point in used_points do
        if point in edge.begin then
            first_exists+ = 1;
        end if
        if point in edge.end then
```

```
            second_exists+ = 1;
        end if
    end for
    if first_exists == 2 || second_exists == 2 then
        //pokud z jednoho z bodů vedou 2 hrany, pokračuj další hranou
        continue;
    end if
    used_points.pushback(edge.begin);
    used_points.pushback(edge.end);
    final_edges.pushback(edge);
    if final_edges.size! = 1 && checkForCycle(final_edges, unique_points.size)
then
        //jestliže vznikl cyklus o počtu hran menším než počet všech bodů,
        //hranu odeber
        used_points.popback(edge.begin);
        used_points.popback(edge.end);
        final_edges.popback(edge);
    end if
end for
//ze souboru hran tvořící finální cyklus podle Greedy seřaď všechny položky
//přičemž začni od zadaného počátečního bodu
final_distance ← 0;
last_point ← start_point;
final_edges.erase(Edge(start_point, end_point));
final_item_order[] ← {};
while final_edges.size > 0 do
    for edge in final_edges do
        if last_point in edge then
            if last_point == edge.begin then
                last_point ← edge.begin;
            else
                last_point ← edge.end;
            end if
            for item in items do
                if (item.x, item.y) == last_point then
                    final_item_order.pushback(item);
                    //přidej do finálního uspořádání položek všechny položky
                    //které se nacházejí na daném bodu
                end if
            end for
            final_distance ← final_distance + edge.length;
            final_edges.erase(edge);
        end if
    end for
end while
return final_distance;
```

4.2.8 Export dat

Pro finální řešení je implementován export dat podle požadavků firmy. Data jsou exportována ve formátu JSON do pole, jehož jednotlivé položky - vozíky - tvoří jednak pole položek seřazených do co nejkratší vzdálenosti podle zvoleného algoritmu, jednak pole polic, kde každá z nich opět obsahuje pole položek, které do dané police patří. Každá položka má specifikace vyžadované skladovým systémem.

4.3 Grafické rozhraní

Pro názornější práci s implementovaným algoritmem a možnost vizuálního otestování toho, jak byly položky seskupeny do vozíků, bylo vytvořeno jednoduché grafické rozhraní. V něm je možné načíst vstupní soubor, a to jak s originálními daty, tak s daty reprezentujícími finální seskupení položek. Rozhraní dále přímo využívá zde vyvinutou knihovnu pro možnost přímého spuštění vytváření vozíků z dat načtených na vstupu.

Aplikace se skládá ze dvou hlavních oken - v prvním lze načíst vstupní i finální soubor dat (oba ve formátu JSON, tlačítka `Load original JSON` a `Load final JSON`). Po korektním načtení je možné data vizualizovat po kliknutí na tlačítka `Show original` a `Show final`. Finální data není třeba mít připravená - lze je získat spuštěním knihovní funkce `loadingCartsRun()` třídy `CartsLoader` po načtení vstupních dat kliknutím na tlačítko `Run`. Algoritmy, které budou použité pro výpočet, je možné ručně nastavit vybráním z rozbalitelných seznamů nebo lze ponechat základní nastavení algoritmů tak, jak jsou vybrány po startu aplikace. Pro metodu K-Means clustering je navíc možno specifikovat počet iterací pro vytváření clusterů a jejich centroidů. Dále je nezbytné zvolit rozměry jedné či více přepravek, které budou během výpočtu využity. Po startu aplikace se v polích objeví hodnoty odpovídající těm, které jsou v této práci využívány jako referenční pro Firmu. Nedostatek zde spočívá v tom, že rozměry vozíku určit nelze - je fixně nastaven na celkový počet tří polic o šířce 90.

V rámci vizualizace vstupních dat je možné vidět rozmístění položek po skladě ve 2D prostoru, kde každému zákazníkovi je přiřazena jedna barva. Pro lepší orientaci lze vybrat konkrétního zákazníka, který je následně označen výraznou žlutou barvou. U finálních dat jsou opět položky rozlišeny, tentokrát podle jejich náležitosti k danému vozíku - každému patří jedna barva. Zaškrtávacími poli je potom možné nechat si zobrazit jenom jeden vozík, rozlišit barevně zákazníky ve vozíku či graficky pospojovat všechny položky tak, jak jsou za sebou řazeny (tzn. výsledek algoritmů Greedy nebo Nearest Neighbor). Je třeba upozornit, že vizualizace následnosti položek je pouze orientační co do reálného tvaru výsledné cesty - je to dáno tím, že dvojice položek jsou jednoduše spojeny *„vzdušnými přímkami“*.

Tlačítkem `RESTORE` je možno navrátit původní podobu vizualizovaných dat.

Kapitola 5

Srovnání, testování a statistiky

5.1 Změny v implementaci základního a finálního řešení

5.1.1 Načítání specifikací položek

Při spuštění načítání dat základního a finálního řešení došlo k pozoruhodnému zrychlení - 919 položek se u starého způsobu načítalo 13 vteřin, ale u nového pouhých 0,5 vteřiny. Důsledkem byla zcela zjevně skutečnost, že spojování položek s jejich rozměrovými specifikacemi, kterých bylo pro celý sklad přes 20000, zabralo téměř veškerý čas výpočtu. I samotné načtení těchto dat znamenalo ztrátu 1,6 vteřiny.

Protože 919 položek znamenalo celkově jen asi 2/5 celkové reálné možné velikosti jedné vlny (soubor objednávek za daný čas), ukázalo se toto zpoždění jako neúnosné, protože narůstalo lineárně s počtem položek (pro každou položku je třeba najít její specifikaci mezi celkovým počtem všech specifikací).

5.1.2 Srovnání hlavního a základního řešení

Hlavní řešení přineslo zcela nový koncept zpracovávání dat, který spočíval v implementování mnoha dalších algoritmů týkajících se skládání položek do boxů, seskupování zákazníků do vozíků a řazení položek ve vozících. Kromě toho byl změněn přístup k „překladu“ adres do souřadnic. Tedy základní řešení již není relevantní pro porovnání s řešením hlavním. Srovnání dvou řešení proběhlo pouze ve formě srovnání algoritmů ze základního řešení použitých (a mírně upravených) v řešení hlavním.

5.2 Validace a integrita dat

Integrita dat byla ověřena pomocí skriptu, který kontroloval, že nedochází k žádným ztrátám dat mezi vstupem a výstupem algoritmu.

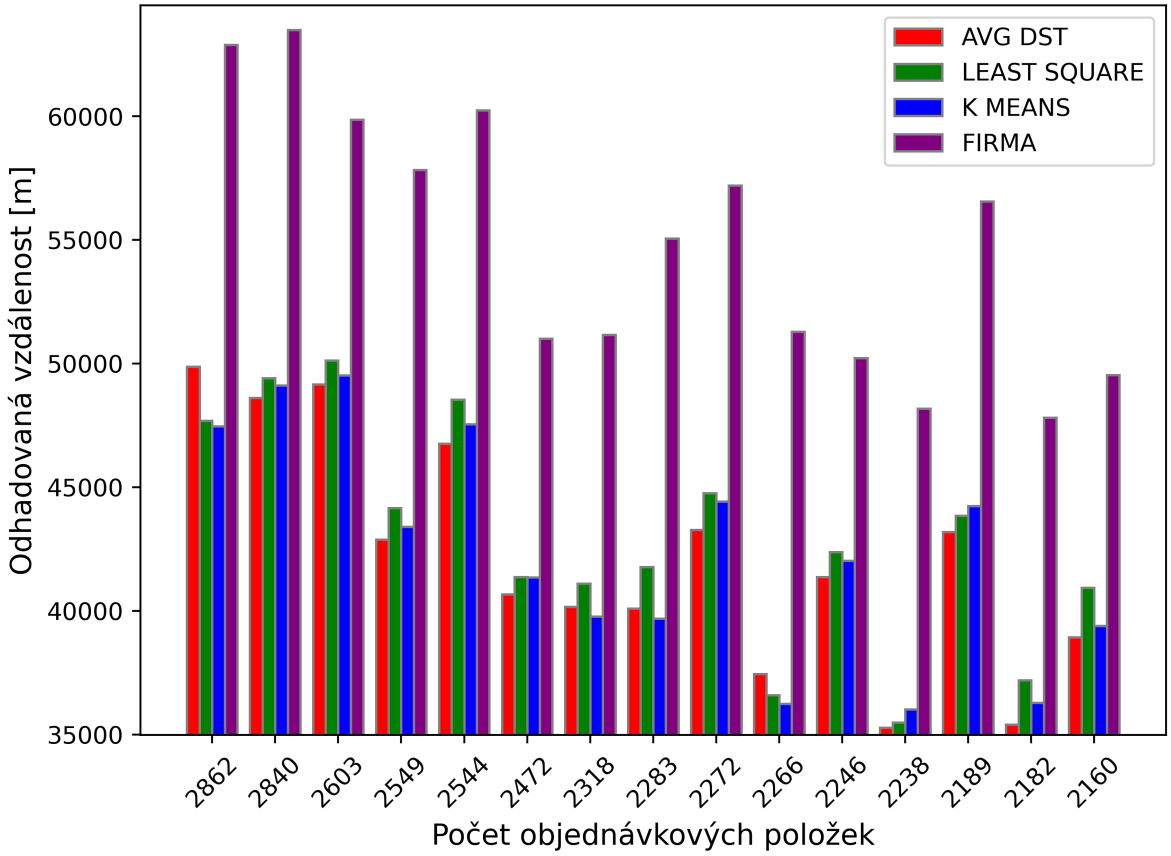
Výsledné rozřazení do vozíků nemohlo být ověřeno v praxi, nicméně naměřené vzdálenosti (a tedy podstatný prvek výpočtu) byly označeny za validní Firemním ředitelem logistiky. I když během provádění algoritmu dochází pravděpodobně k měřícím chybám, odhadované nachozené vzdálenosti u vozíků, které byly vytvořeny a jejichž položky byly seřazeny Firmou, odpovídaly očekávaným hodnotám.

5.3 Srovnání firemního řešení

Firma dala k dispozici patnáct vln, mající adresu před předvychystávacím procesem (tedy adresy jsou po celém skladu) a další čtyři vlny, které mají již adresy po předvychystávacím procesu. Data jednotlivých vln obsahují také informace o Firemním rozřazení položek do vozíků i s jejich pořadím v nich - díky tomu je možné srovnat dosavadní výsledky Firmy s výsledky této práce. Všechny grafy jsou zobrazeny v závislosti na počtu objednávkových položek (osa x). [1]

5.3.1 Srovnání patnácti vln

Figurují zde vlny, jejichž položky mají adresy neoptimalizované předvychystávacím procesem, a tedy jsou po celém skladě. Zde dominoval algoritmus ze základního řešení, založený na průměrné vzdálenosti položek mezi sebou. Algoritmus K-Means byl potom lepší v případech, kdy nestačil právě zmíněný algoritmus, což bylo právě ve čtyřech případech vln. Řešení seskupování zákazníků metodou nejmenších čtverců zde mělo ve většině případů nejhorší výsledky. Vše lze názorně pozorovat v grafu 5.1.



Obrázek 5.1: Odhadovaná vzdálenost pro patnáct vln mající adresy po celém skladě.

[1] Celá vlna se skládá z objednávkových položek od zákazníků. Každá položka je potom unikátní pouze pro jednoho zákazníka (tj. mezi více zákazníky může existovat více stejných objednávkových položek). Objednávková položka je určena zákazníkem, produktem a počtem objednaných kusů.

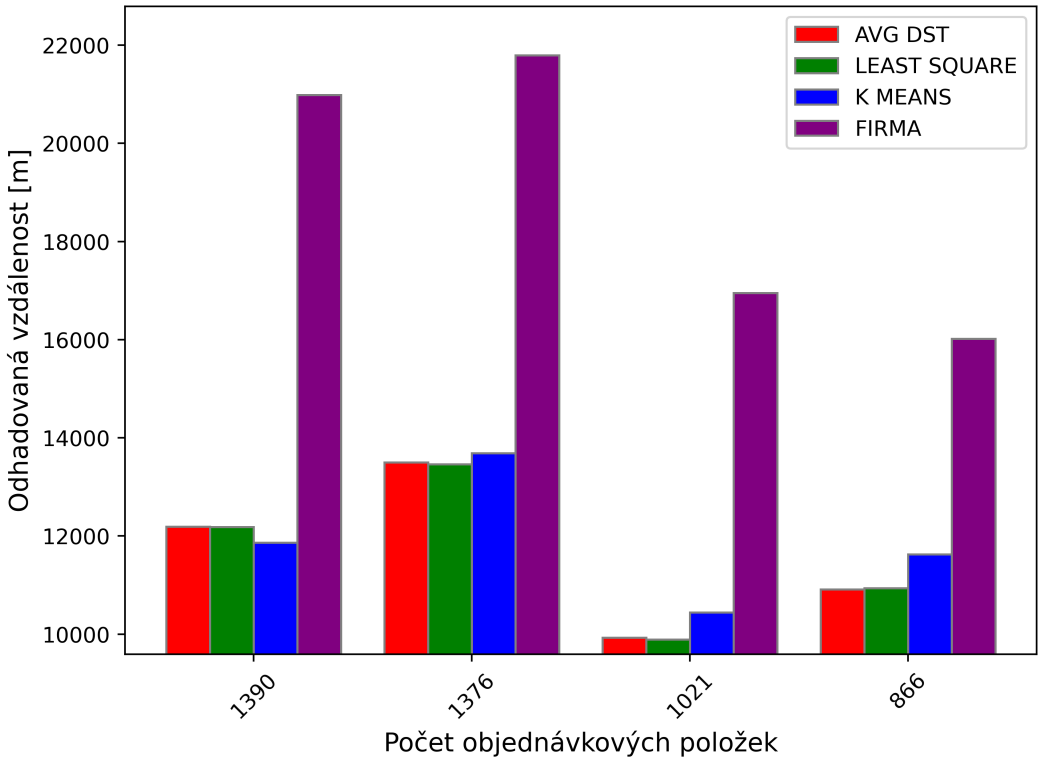
Celkově došlo ke zlepšení odhadované vzdálenosti oproti firemnímu řešení o průměrně 23, 55 %, nejmenší úspora potom byla 17, 62 % a největší 29, 33 %.

Lze pozorovat mírný trend klesání vzdálenosti s počtem objednávkových položek. Místo Firemního řazení adres bylo potom ještě vyzkoušeno řazení Greedy algoritmem (pro vozíky vytvořené Firmou). Tím došlo ke srovnání efektivity umisťování zákazníků do vozíků mezi firemním zpracováním a algoritmy implementovanými v této práci - výsledkem bylo průměrné zkrácení trasy pro všech patnáct vln o 16, 12 %. Odtud lze tedy zároveň odvodit, že úspora trasy pouhým řazením položek ve vozíku je průměrně 7, 43 %.

5.3.2 Srovnání čtyř vln

V tomto srovnání figurují ty vlny s položkami, jejichž adresy jsou rozmístěny v rámci dvou sektorů (G a V) díky procesu předvychystání. Protože se jedná o malý vzorek, nelze výsledky příliš zobecňovat. Pro vyzkoušené vlny vyplývá, že efektivita metody nejmenších čtverců se výrazně přiblížila algoritmu, který využívá průměrnou vzdálenost a u tří vln poskytla lepší řešení (v průměru o 26 metrů). Metoda K-Means clustering zde vykazuje ve třech vlnách horší výsledek v porovnání s ostatními dvěma metodami, zatímco v první vlně přináší mírné zlepšení. Výsledek lze pozorovat v grafu 5.2.

Výsledky zde nabádají k teorii, že metoda nejmenších čtverců je lepší na malém prostoru s větší koncentrací položek. Rovněž se naskýtá doporučení využívat automatického výběru správné metody (který je implementován triviálním způsobem - vyzkoušení všech metod a následně výběr té, která poskytla celkovou nejkratší vzdálenost) a tím dosáhnout vždy největší možné úspory.



Obrázek 5.2: Odhadovaná vzdálenost pro čtyři vlny mající adresy v sektorech G a V

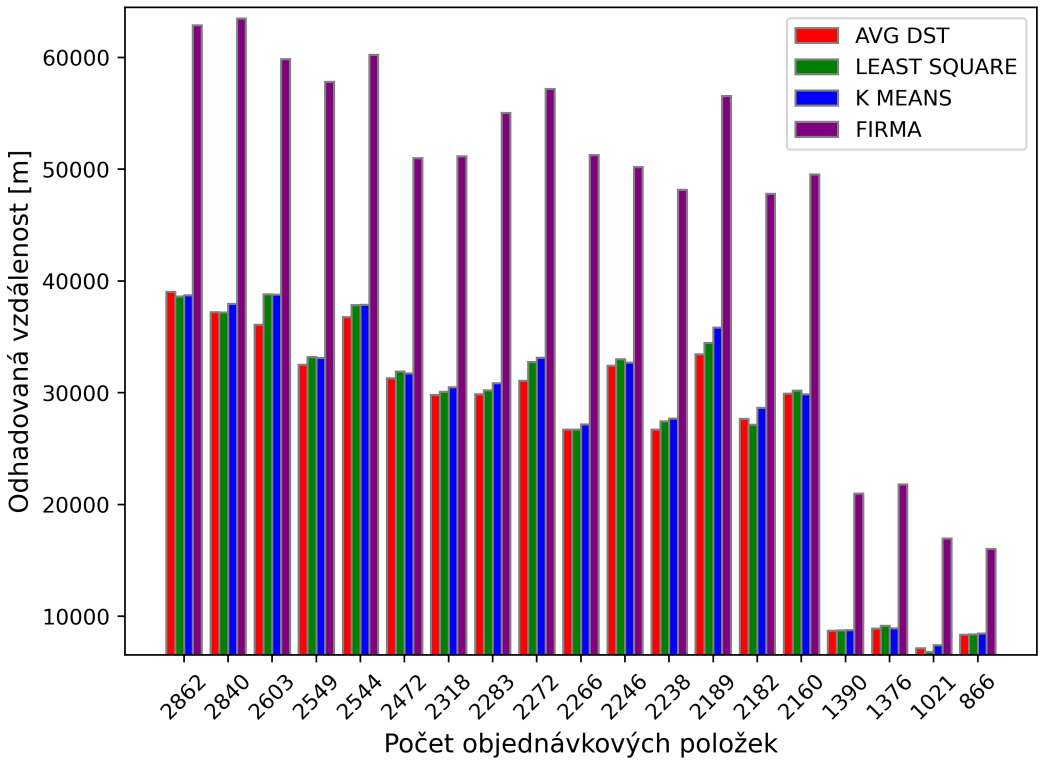
Celkově zde došlo k průměrnému zlepšení délky trasy o 39, 1 %; nejlepší zlepšení pak zkrátilo trasu o 43, 45 % zatímco to nejnižší zlepšení bylo o 31, 88 %.

5.3.3 Srovnání všech devatenácti vln - odlišná konfigurace

Protože Firma označila za potenciálně plánovanou možnost měnit konfigurace vozíku i boxů (jiné rozměry, různé typy), byly implementovány třídy, ve kterých lze různé konfigurace použité při běhu algoritmů měnit. Firma podala konkrétní rozměry vozíku a boxů, které zamýšlí do budoucna zavést:

vozík má mít čtyři police, přičemž každá z nich by měla být schopna pojmout čtyři nynější boxy - odtud plyne nastavení vozíku na šířku ($4 * \ box = 4 * 28.0 = 112cm$), výška a hloubka není řešena (jejich využití není implementováno v nynější verzi knihovny - ovšem toto omezení nijak neznehodnocuje následující test) - nicméně výšku vozíku odráží nastavení počtu polic na čtyři. Boxy potom mají být o dvou druzích - první se stejnými rozměry jako doposud (tzn.: $28 * 29 * 45$), druhý o poloviční šířce (tzn.: $14 * 29 * 45$) - to umožňuje dát na polici až osm těchto „menších“ přepravek.

Výsledky jsou opět srovnány s dosavadním řešením Firmy v grafu 5.3.



Obrázek 5.3: Odhadovaná vzdálenost pro všech devatenáct vln při větších rozměrech vozíku a dvou typech boxů

Lze pozorovat klesající trend v grafu, který odpovídá předpokladu, že menší počet objednávkových položek bude zvládnutelný v kratší dráze (závislost os x a y).

Pro všechny vlny potom vychází průměrné zlepšení odhadované nachozené vzdálenosti o 42, 96 %, maximální zlepšení o 59, 77 % a nejméně zkrácená trasa pak dosahovala délky o 35, 4 % kratší oproti Firemnímu řešení.

5.4 Další srovnání

5.4.1 Vytváření boxů

Protože ve firmě je zkušenost s využíváním pouze jednorozměrného kritéria při skládání položek do vozíku, bylo pro srovnávání rozhodnuto využít rovněž algoritmus pro 1D (tedy ve všech předchozích srovnáních je použit tento přístup). Níže je srovnání algoritmů pro balení položek do boxů s údaji o množství vygenerovaných boxů a míře průměrné zaplněnosti všech boxů pro každou vlnu.

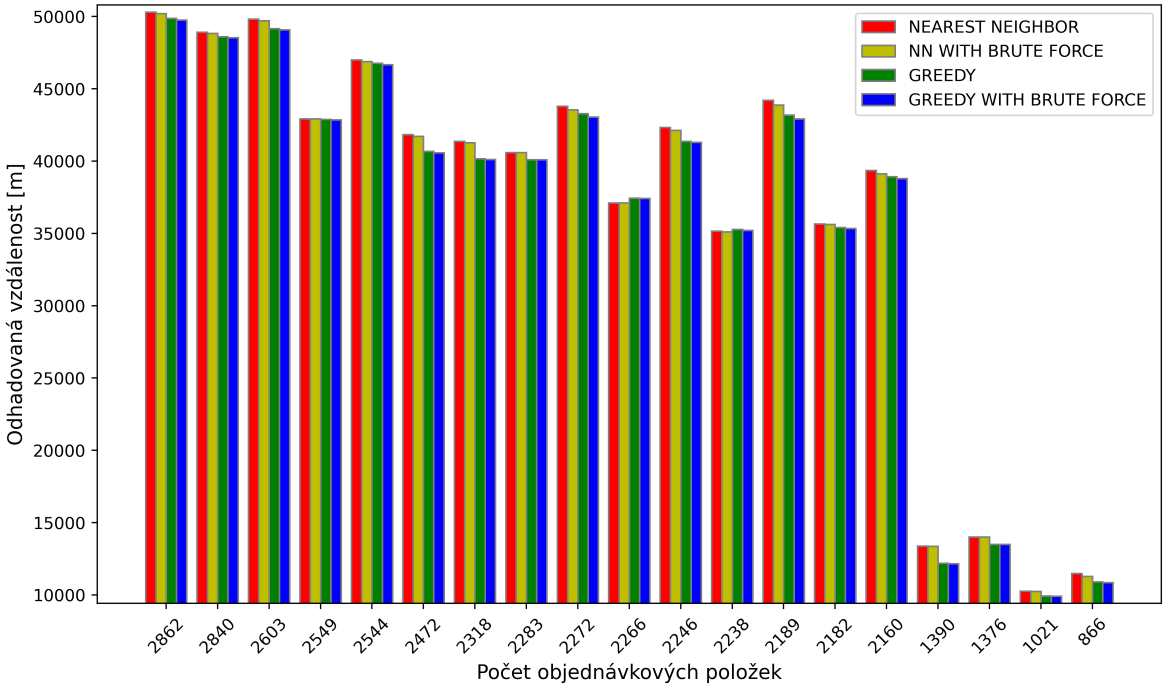
počet objednávkových položek	základní řešení FFD		FFD 1D		3D	
	počet	zaplněnost [%]	počet	zaplněnost [%]	počet	zaplněnost [%]
866	362	62.35	468	48.23	619	36.46
1021	470	35.81	520	32.36	655	25.69
1376	547	24.52	564	23.78	706	19.00
1390	455	38.24	487	35.72	645	26.97
2160	469	69.27	571	56.89	876	37.08
2182	469	59.14	561	49.44	859	32.29
2189	532	66.77	664	53.50	1049	33.86
2238	437	62.82	536	51.22	905	30.33
2246	494	79.04	680	57.42	1181	33.06
2266	532	50.17	635	42.03	942	28.33
2272	579	62.07	773	46.49	1146	31.36
2283	533	50.74	590	45.84	909	29.75
2318	464	61.93	555	51.77	908	31.64
2472	417	72.56	524	57.75	853	35.47
2544	567	69.79	742	53.33	1218	32.48
2549	584	57.47	701	47.88	1087	30.87
2603	603	77.32	839	55.57	1416	32.93
2840	551	66.15	690	52.82	1120	32.54
2862	585	61.42	679	52.92	1006	35.71
průměrná zaplněnost boxu	59,35 %		48,16 %		31,36 %	

Důležité je upozornit, že první z algoritmů (označen jako „základní řešení FFD“) dokáže boxy zaplnit nejvíce proto, že se nesnaží dělit jednotlivé položky objednávek na kusy, ale ukládá do boxu celou položku objednávky (a v případě, že se nevejde, je stejně do něj přiřazena). Oproti tomu zbývající dva testované algoritmy neporuší objem boxu a v případě, že je položka příliš velká, vytvoří box o rozměrech této položky.

5.4.2 Srovnání algoritmů pro nejkratší vzdálenost

V předchozích testech byl vždy využit algoritmus Greedy, jakožto průměrně lepší vzhledem k algoritmu Nearest Neighbor. Oba dva tyto algoritmy navíc mají implementovánu možnost využít řešení „hrubou silou“, pokud je na vozíku méně, než 10 položek a pokud je tato optimalizace nastavena ve třídě `SolverConfiguration`. V grafu 5.4 lze pozorovat

výsledky všech čtyř možných implementací získání nejkratší trasy, které jsou předvedeny na devatenácti vlnách. Pro seskupování položek do vozíku byl ve všech případech využit algoritmus využívající průměrnou vzdálenost.

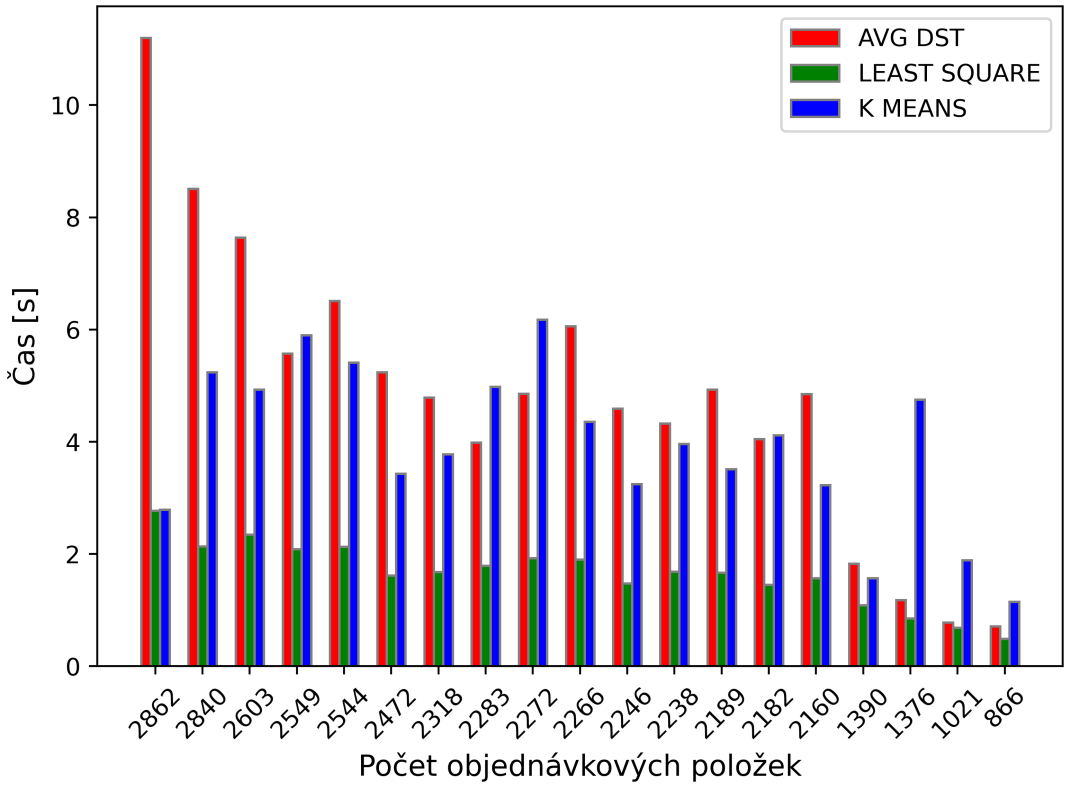


Obrázek 5.4: Odhadovaná vzdálenost pro devatenáct vln ve srovnání algoritmů pro zisk nejkratší vzdálenosti

Použití hrubé síly průměrně zlepšilo výsledek algoritmu Nearest Neighbor o $0,31$ %, u algoritmu Greedy se potom jednalo o $0,21$ %. Algoritmus Nearest Neighbor dosáhl lepšího výsledku u dvou vln, ve zbylých případech byl potom dle očekávání výkonnější algoritmus Greedy a to s kratší trasou v průměru o $2,03$ % (oproti Nearest Neighbor).

5.4.3 Časová náročnost algoritmů seskupování zákazníků

V grafu 5.5 pozorujeme časovou náročnost všech algoritmů použitých pro seskupování zákazníků u devatenácti vln, seřazených sestupně dle počtu objednávkových položek.

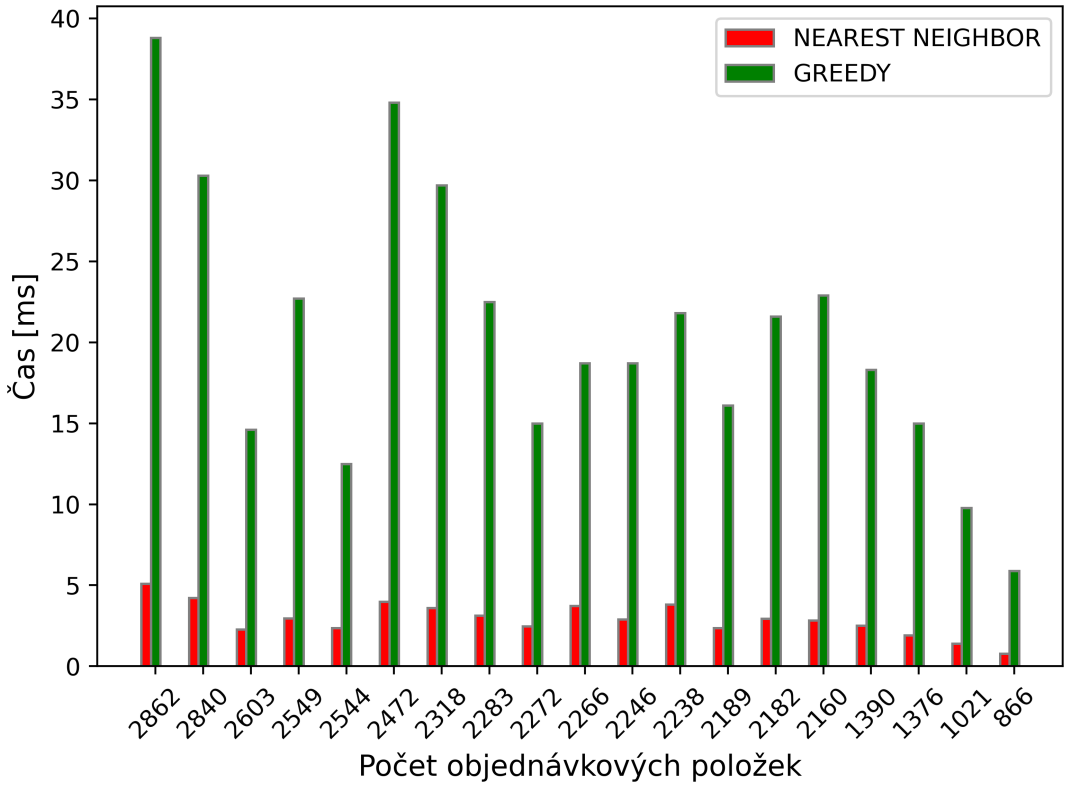


Obrázek 5.5: Časová náročnost algoritmů pro nejkratší vzdálenost.

Algoritmus využívající průměrnou vzdálenost (označen v grafu jako *AVG DST*) pracoval v průměru $4,82\ s$, algoritmus využívající metodu nejmenších čtverců (označen v grafu jako *LEAST SQUARE*) $1,65\ s$ a algoritmus pracující jako K-Means clustering (v grafu označen jako *K MEANS*) vykonával sdružování zákazníků průměrně $3,92\ s$.

5.4.4 Časová náročnost algoritmů pro nejkratší trasu

V následujícím grafu 5.6 lze vidět časovou náročnost algoritmů, které řadí položky ve vozících do nejkratší trasy.



Obrázek 5.6: Časová náročnost algoritmů pro nejkratší vzdálenost.

Průměrná doba práce algoritmu Nearest Neighbor byla $2,9\ ms$, u algoritmu Greedy potom průměrná doba výpočtu činila $20,5\ ms$.

Kapitola 6

Závěr

V této práci byla rozebrána problematika optimalizace skladového procesu zvaného vychystávání, a popsány tři hlavní oblasti, které bylo nutno optimalizovat - skládání položek do přepravek, seskupování nedělitelných skupin položek do vozíků a řazení položek ve vozících do nejkratší trasy. V rámci bližšího pochopení skladové problematiky byl obecně popsán chod skladu a uvedeny typy skladů. Rovněž byly formalizovány matematické problémy, související s touto prací.

Během řešení této práce byly vytvořeny dvě implementace - první - zjednodušené základní řešení - sloužilo pro vytvoření kostry a podkladu pro následnou tvorbu druhé implementace - hlavního řešení. To je připraveno pro nasazení v reálném skladu Firmy. Veškeré významné algoritmy, které byly přizpůsobeny pro řešení této práce, byly v textu uvedeny formou pseudokódu.

Statistiky na devatenácti vlnách prokázaly v teoretické rovině úspěšnost implementovaného řešení co do zkrácené trasy v porovnání s dosavadním řešením Firmy.

Řešená problematika jeví potenciál v přidání různých parametrů - například počtu pracovníků s takovou optimalizací, aby docházelo k jejich efektivnímu využití. Dále se také naskýtá možnost implementovat pokročilejší algoritmy pro řazení položek ve vozíku do nejkratší trasy. Prostor pro zkvalitnění má také měření vzdáleností - přesné změření vzdáleností ve skladu by mohlo přinést relevantnější výsledky, nicméně nelze tvrdit, že by vyšší míra přesnosti v tomto ohledu přinesla výraznější zlepšení co do reálného zkrácení trasy.

Literatura

[1] *K-Means Clustering Algorithm* [online]. [cit. 2023-04-03]. Dostupné z: https://www.javatpoint.com/k-means-clustering-algorithm-in-machine-learning.

[2] *Man-hour definition and meaning: Collins english dictionary.* HarperCollins Publishers Ltd [cit. 2023-03-29]. Dostupné z: https://www.collinsdictionary.com/dictionary/english/man-hour.

[3] *Understanding K-means Clustering in Machine Learning* [online]. Education Ecosystem (LEDU), 2018 [cit. 2023-04-03]. Dostupné z: https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1.

[4] Abdulkarim, H. a Alshammari, I. F. Comparison of Algorithms for Solving Traveling Salesman Problem. *International Journal of Engineering and Advanced Technology.* 2015. ISSN 2249 – 8958.

[5] Dube, E. a Kanavathy, L. OPTIMIZING THREE-DIMENSIONAL BIN PACKING THROUGH SIMULATION. 2006. Dostupné z: https://raw.githubusercontent.com/enzoruiz/3dbinpacking/master/erick_dube_507-034.pdf.

[6] Gao, Y. *Heuristic Algorithms for the Traveling Salesman Problem* [online]. Open Analytics, 2020 [cit. 2023-03-22]. Dostupné z: https://medium.com/opex-analytics/heuristic-algorithms-for-the-traveling-salesman-problem-6a53d8143584.

[7] Hliněná, D. *Metóda najmenších štvorcov.* [cit. 2023-04-02]. Dostupné z: https://www.umat.fekt.vut.cz/~hlinena/IMA1/Prednasky/mnc.pdf.

[8] Karp, R. M. *On-Line Algorithms Versus Off-Line Algorithms: How Much is it Worth to Know the Future?* 1992. Dostupné z: https://www1.icsi.berkeley.edu/pubs/techreports/TR-92-044.pdf.

[9] Kenton, W. *Least Squares Method: What It Means, How to Use It, With Examples* [online]. [cit. 2023-04-01]. Dostupné z: https://www.investopedia.com/terms/l/least-squares-method.asp.

[10] Key, R. a Dasgupta, A. Warehouse Pick Path Optimization Algorithm Analysis. *The 2015 International Conference on Foundations of Computer Science.* 2015.

[11] Lavrov, M. Lecture 35: The Traveling Salesman Problem. 2020. Dostupné z: https://faculty.math.illinois.edu/~mlavrov/docs/482-spring-2020/lecture35.pdf.

[12] Martello, S. *Bin packing problems* [online]. 2019 [cit. 2022-12-04]. Dostupné z: https://mathopt.be/Slides_LaRoche_Martello.pdf.

[13] Meghanathan, N. *Module 1: Asymptotic Time Complexity and Intro to Abstract Data Types* [online]. [cit. 2023-04-13]. Dostupné z: https://www.jsums.edu/nmeghanathan/files/2018/01/CSC228-Sp2018-Module-1-Algorithm-Efficiency.pdf.

[14] myankdv. *C++ / Brute Force / Optimal / Time O(N) / Auxiliary Space O(1)* [online]. 2021 [cit. 2023-03-20]. Dostupné z: https://leetcode.com/problems/next-permutation/solutions/1394208/c-brute-force-optimal-time-on-auxiliary-space-o1/.

[15] Nilsson, C. *Heuristics for the Traveling Salesman Problem.* 2003. Dostupné z: https://scholar.google.com/scholar_lookup?title=Heuristics%20for%20the%20traveling%20salesman%20problem&author=C.%20Nilsson&publication_year=2003.

[16] Sangwan, S. a Dahiya, C. Literature Review on Travelling Salesman Problem. *International Journal of Research.* 2018, sv. 5, s. 1152.

[17] Sebo, J. a Busa, J. Comparison of Advanced Methods for Picking Path Optimization: Case Study of Dual-Zone Warehouse. *International Journal of Simulation Modelling.* 2020.

[18] Sgall, J. Online bin packing: Old algorithms and new results. *Proc. of the 10th Conference on Computability in Europe (CiE).* Springer. 2014. Dostupné z: https://iuuk.mff.cuni.cz/~sgall/ps/cie.pdf.

[19] Wróblewski, P. *Algoritmy.* 4. vyd. Computer Press, 2015. ISBN 978-80-251-4126-7.

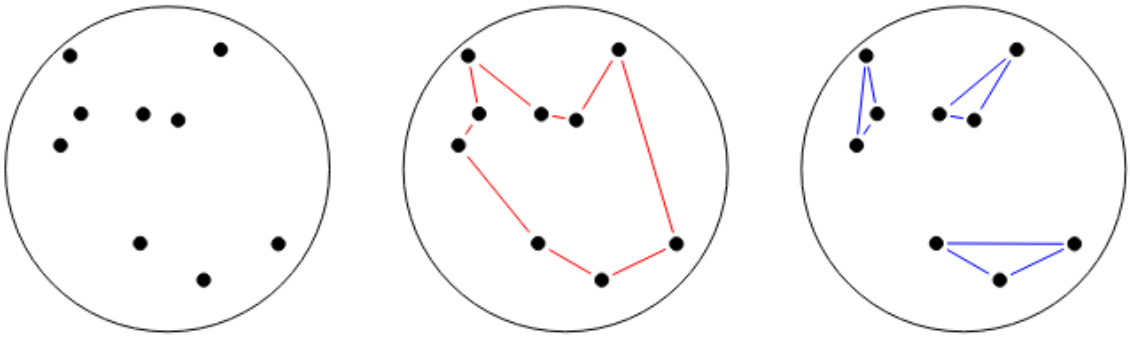
[20] Zhang, G. On Variable-Sized Bin Packing. *Proc. of 3rd International Workshop on ARANCE, Roma, Italy.* 2002.

Příloha A

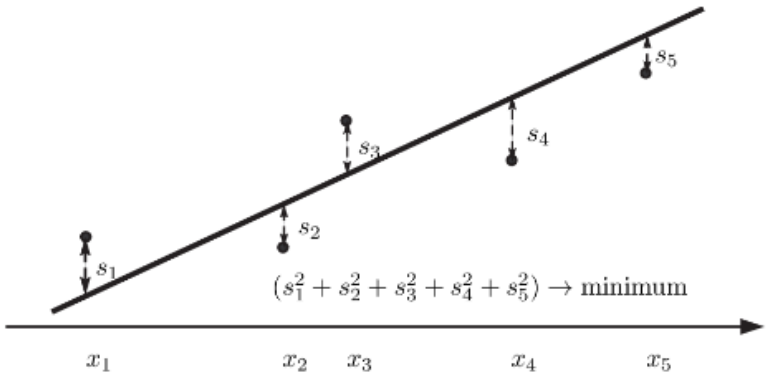
Algoritmy

n	$log_2 n$	n	$n\ log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 * 10^1$	10^2	10^3	10^3	$3.6 * 10^6$
10^2	6.6	10^2	$6.6 * 10^2$	10^4	10^6	$1.3 * 10^{30}$	$9.3 * 10^{157}$
10^3	10	10^3	$1.0 * 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 * 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 * 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 * 10^7$	10^{12}	10^{18}		

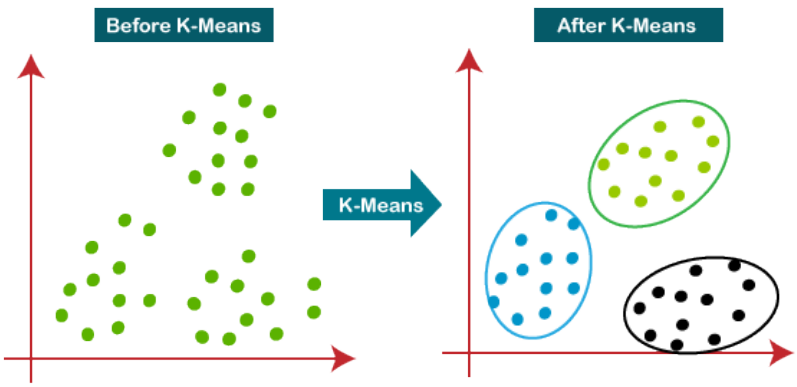
Obrázek A.1: Počet provedených operací algoritmů různých stupňů složitosti v závislosti na velikosti vstupu (převzato z prezentace [13]).



Obrázek A.2: Zleva počáteční množina bodů, mezi kterými je cena cesty stanovena jako vzdálenost mezi každými dvěma z nich. Dále optimální řešení nalezené *brute-force* metodou a vpravo řešení, odpovídající omezením 2.2 a 2.3, které ovšem vůbec není řešením problému TSP (jak zde chceme ukázat).



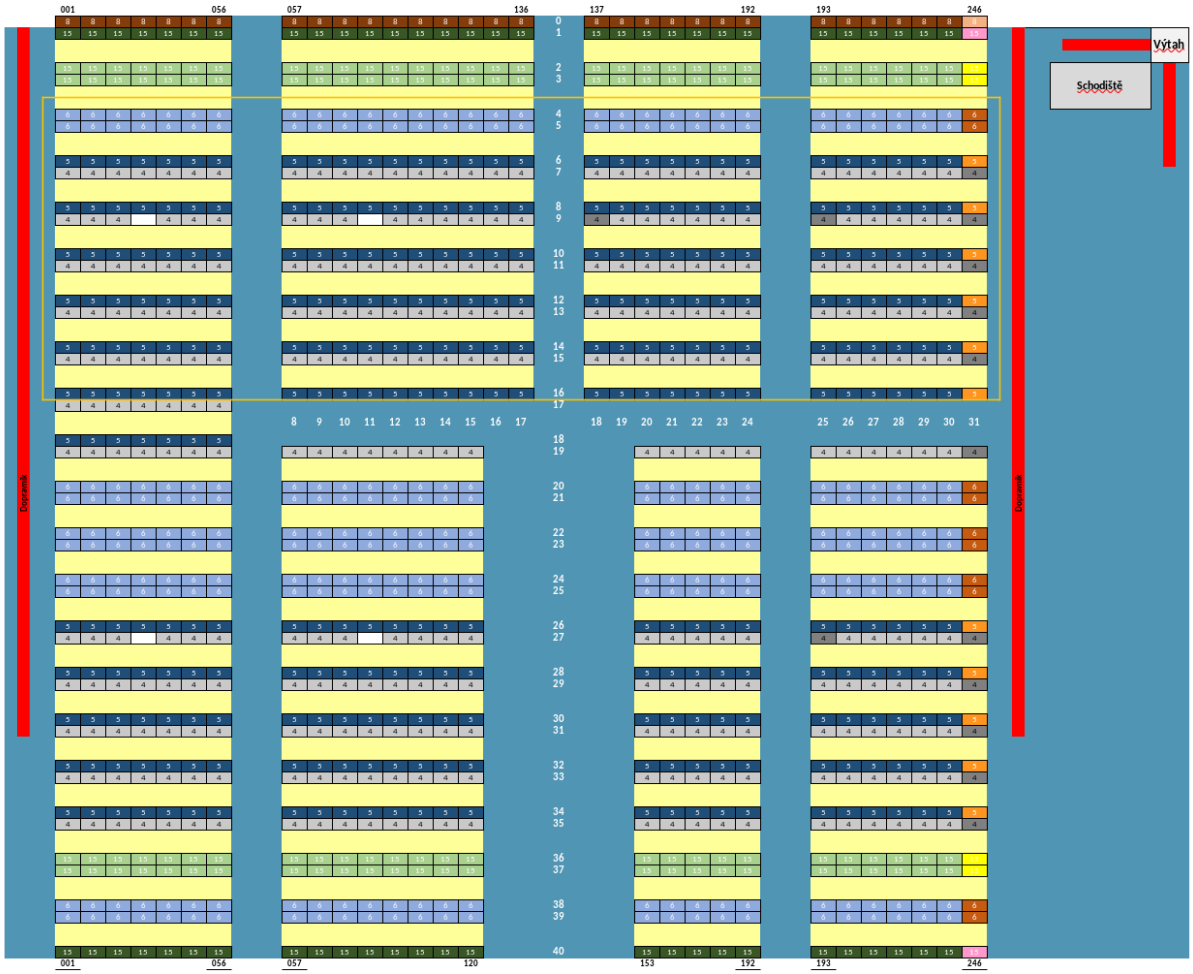
Obrázek A.3: Princip metody nejmenších čtverců. Převzato z [7].



Obrázek A.4: Princip metody K-Means clustering. Převzato z [1].

Příloha B

Skladové prostory

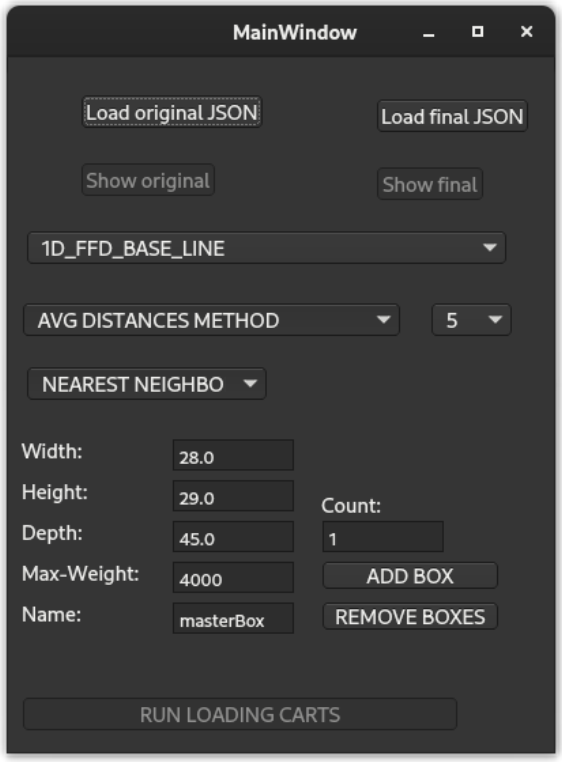


Obrázek B.1: Mapa sektoru G ve skladu konkrétní firmy.

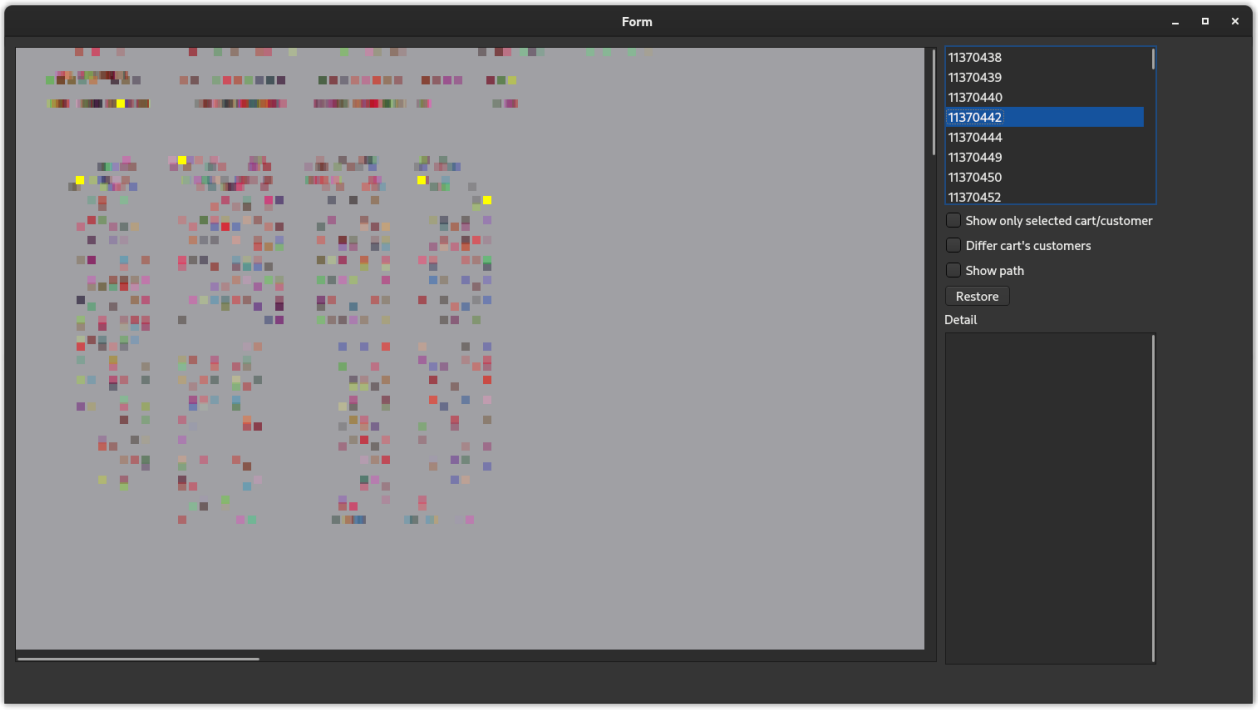
Obrázek B.2: Ukázka jediné nutnosti zohlednit dvě možné trasy mezi dvěma adresami. K obdobné situaci může dojít i v horizontálním směru.

Příloha C

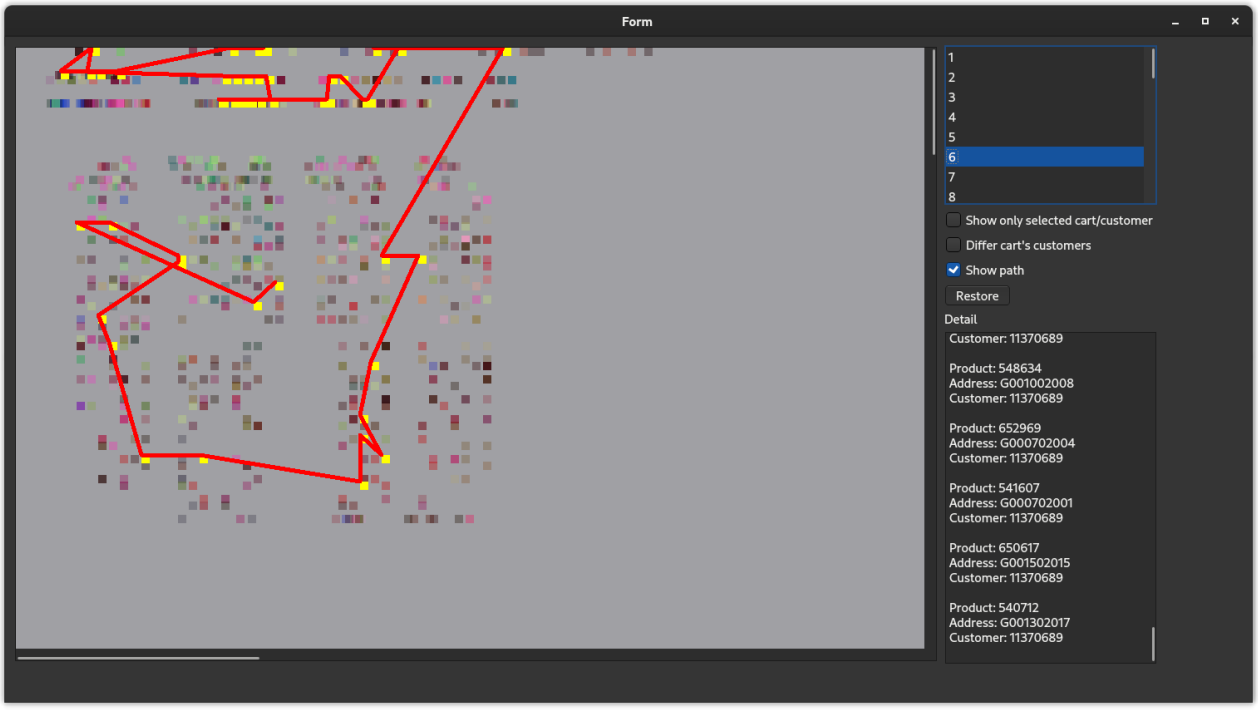
Grafické rozhranní



Obrázek C.1: Hlavní okno GUI umožňující načítat a zobrazovat soubory JSON a spouštět řazení vstupních dat do vozíku.



Obrázek C.2: Vizualizace vstupních dat, žluté body ukazují položky vybraného zákazníka, ostatní body podle barev patří dalším zákazníkům.



Obrázek C.3: Vizualizace finálního rozřazení zákazníků a jejich položek do vozíků, žlutě jsou označeny položky vybraného vozíku, při čemž ty jsou navíc pospojovány tak, jak jsou řazeny za sebou do nejkratší trasy (zde algoritmem Nearest Neighbor).