UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Bin Packing Problem

Karel Steinmetz

Diplomová práce
2013

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2012/2013**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Karel Steinmetz**
Osobní číslo:	**I10416**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Bin Packing problem**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

Dnešní doba je charakteristická obrovskou nabídkou zboží na trhu a proto se dostává logistika do popředí zájmu většiny společností. Cílem práce bude vytvořit aplikaci s řešení významné logistické úlohou, zvanou Vehicle Routing Problem (VRP). Tato úloha má za cíl mimimalizaci nákladů spojených s přepravou zboží z firemního skladu k zákazníkům. S VRP úzce souvisí úloha naplňování zásobníku (Bin Packing Problem - BPP) a problém obchodního cestujícího (Traveling Salesman Problem - TSP). Všechny tyto problémy mají společnou tu vlastnost, že obecně nelze nalézt jejich optimální řešení. Nicméně existují různé heuristické algoritmy. Konkrétně budou naprogramovány metody First-Fit, First-Fit Decreasing, Best-Fit, Best-Fit Decreasing pro úlohu BPP. U úlohy VRP budou do programu implementovány metody Polar Region Partitioning, Rectangular Region Partitioning. Pro řešení TSP může být využita aplikace pro Google maps, viz http://www.tsp.gatech.edu/maps/index.html resp. může být naprogramován např. algoritmus 2-opt. Aplikace bude obsahovat výsledky simulací, které budou demonstrovat, jak použité metody umožní co nejoptimálněji umísťovat náklad o různé velikosti např. do nákladních automobilů a jak vhodně přidělit zákazníky k jednotlivým automobilům při co nejmenší ujeté vzdálenosti.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

Simchi-Levi D., Bramel J., Chen X.: The Logic of Logistics, Springer-Verlag, New York, (2004)
Garey M. R., Graham R. L., Johnson D.S., Yao A.: Resource constrained scheduling as generalized bin packing, J. Combinatorial Theory Ser. A, 21 (1976)
Baker B. S.: A New Proof for the First-Fit Decreasing Bin Packing Algorithm. J. Algorithms 6 (1985)

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

Datum zadání diplomové práce: **31. října 2012**
Termín odevzdání diplomové práce: **17. května 2013**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

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

V Pardubicích dne 15. listopadu 2012

Prohlášení autora

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

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

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích 15. 8. 2013 Karel Steinmetz

Poděkování

Touto cestou bych rád poděkoval vedoucímu diplomové práce Mgr. Jaroslavu Markovi, Ph.D. za poskytnutí cenných rad, literatury a připomínek v průběhu zpracování této diplomové práce.

Anotace

Tato práce se zabývá matematickými problémy z oblasti dopravní logistiky, jejichž řešení vedou na NP-těžké úlohy.

Konkrétně jsou studovány Problém naplnění zásobníků, Problém obchodního cestujícího a Problém rozvozní úlohy.

Nalezení minima, respektive alespoň jeho kvalitní aproximace pomocí heuristických algoritmů umožňuje minimalizovat náklady při distribuci zboží z centrálních skladů ke koncovým zákazníkům.

Klíčová slova

Problém naplnění zásobníků, Problém obchodního cestujícího, Problém rozvozní úlohy

Title

Bin Packing Problem

Annotation

This diploma thesis describes problems of logistic that theirs result passes to NP-hard problems.

Specifically studies problems are Bin Packing Problem, Traveling Salesman problem and Vehicle Routing Problem.

To finding minimum or at least a good approximation that uses heuristic algorithms allows minimalizing costs of distributing goods from central warehouse to the end customers.

Keywords

Bin Packing Problem, Traveling Salesman Problem, Vehicle Routing Problem

Obsah

Seznam zkratek

BPP	Bin Packing Problem - Problém plnění zásobníků
SPP	Problém plnění zásobníku neomezené výšky
1DBPP	Problém plnění zásobníku v jednorozměrném prostoru
2DBPP	Problém plnění zásobníku v dvojrozměrném prostoru
3DBPP	Problém plnění zásobníku v trojrozměrném prostoru
TSP	Traveling Salesman Problem - Problém obchodního cestujícího
STSP	Symetrický TSP
NN	Nearest Neighbor – Metoda nejbližšího souseda
k-opt	K - optimalizace
2-opt	2 - optimalizace
VRP	Vehicle Routing Problem - Problém rozvozní úlohy
PRP	Polar Region Partitioning – Dělení na kruhové výseče
RRP	Rectangular Region Partitioning – Dělení na obdélníkové části
CLR	Common Language Runtime – Společný jazykový modul runtime
CIL	Common Intermediate Language – Společný přechodný jazyk
CSV	Comma-separated values – hodnoty oddělené čárkami

Seznam obrázků

Seznam tabulek

Úvod

Dnešní doba je charakteristická velkou nabídkou zboží na trhu a proto se dostává logistika do popředí zájmu většiny společností.

Tato práce se zabývá řešením významné logistické úlohy zvané Vehicle Routing Problem (jedním z možných překladů do češtiny může být Problém rozvozní úlohy). Cílem této úlohy je minimalizace nákladů spojených s přepravou zboží z distribučního skladu ke konečnému zákazníkovi. S touto úlohou úzce souvisí úloha Problém obchodního cestujícího a Problém naplňování zásobníků. Tato práce je zaměřena především na Problém naplňování zásobníků.

Všechny tyto problémy mají tu společnou vlastnost, že obecně nelze najít jejich optimální řešení. Proto tyto úlohy řadíme do NP-těžkých úloh. Ovšem existují heuristické algoritmy, které umožňují nalézt minimum nebo alespoň jeho uspokojivou aproximaci. Minimem jsou v tomto případě myšleny minimální náklady, které úzce souvisí s těmito problémy.

Kapitola *Management v logistice* popisuje samotný management v logistice a obsahuje jednoduchý přehled problémů, na které je možné v logistice narazit.

V následující kapitole *Bin Packing Problém* jsou vybrány a popsány jednotlivé heuristické algoritmy, na které narazíme při balení a skladování zboží. Tato úloha se nazývá Problém plnění zásobníků a je možné ji dělit do třech kategorií dle příslušné prostorové dimenze zásobníků a baleného zboží. Pro tuto oblast byly vybrány heuristiky First-Fit, First-Fit Decreasing, Best-Fit Decreasing.

Kapitola *Traveling Salesman Problém* obsahuje úvod do problematiky Problém obchodního cestujícího. Dále obsahuje vybrané heuristické algoritmy, které si kladou za cíl nalézt minimální ujetou vzdálenost vozidel při obsloužení zákazníků. Pro tuto oblast byl vybrán algoritmus 2-optimalizace.

Kapitola *Vehicle Routing Problém* popisuje komplexní problém, který zahrnuje již zmíněné problémy v logistice. V kapitole je uveden jednoduchý úvod do problematiky a možné rozdělení dle ovlivňujících faktorů, kterými může být kapacita vozidla, termín dodání nebo převoz zboží v obou směrech. Pro tuto oblast byly vybrány heuristické algoritmy Polar Region Partitioning a Rectangular Region Partitioning.

Kapitola *Desktopová Aplikace* obsahuje popis implementace desktopové aplikace, která umožní demonstrovat a hodnotit výstupy heuristických algoritmů pro vybrané problémy.

V kapitole *Porovnání metod* se nachází výsledky a porovnání vybraných metod, které byli použity pro experimentální výpočty reálných problémů.

Kapitola *Závěr* obsahuje shrnutí této diplomové práce.

1 Management v logistice

1.1 Úvod

Dnešní trhy jsou specifické tím, že na nich najdeme produkty s krátkým životním cyklem. Nároční zákazníci také nutí výrobce efektivně investovat do logistiky a vytvářet změny v komunikaci a přepravě, aby co nejlépe uspokojili svého zákazníka.

Zboží je vyráběno ve více výrobnách, poté přepravováno do centrálních nebo lokálních skladů a v konečné fázi je přepravováno k maloobchodníkům nebo koncovým zákazníkům. Na základě toho jsou vytvářeny distribuční logistické sítě. Tyto sítě se skládají z dodavatelů, výrobních míst, skladů, distribučních a zákaznických center.

Pojem logistika můžeme definovat jako proces průběžného a efektivního plánování, výroby, kontroly, skladování zboží, služeb a souvisejících věci za účelem vyhovět požadavkům zákazníka.

Rozhodování v logistice můžeme rozdělit do třech základních skupin dle jejich cíle:

- *Strategický* – tento cíl má dlouhodobý vliv na podnik. Zabývá se počtem, umístěním, kapacitami skladů a výrobních závodů nebo o dodávku distribuované materiálu přes logistickou síť,
- *Taktický* – tento cíl je typicky plánován na kvartály nebo na jeden celý rok. Zahrnuje nákup a rozhodování o produkci, zásobovací politiku a dopravní strategii, včetně frekvence dodání,
- *Provozní* – zabývá se každodenním rozhodováním pro plánování, rozvoz a nakládání vozidel.

1.2 Základní problémy v logistice

1.2.1 Rozmístění v distribuční síti

Je potřeba zvážit, kde se geograficky nachází výroba, která produkuje zboží pro své odběratele. Může se stát, že aktuální síť skladů není dlouho době vyhovující a proto vedení společnosti má za cíl přeorganizovat distribuční síť. Tuto situaci může způsobit například poptávka na zboží nebo ukončení nájemních smluv objektů, ve kterých se aktuálně nachází sklady. Cílem je potom v této situaci vybudovat síť skladů a kapacit tak, aby celkové náklady v distribuční síti byly co nejmenší, ať už jde o transport ze skladu do skladu nebo ze skladu k maloobchodníkům. Náklady se zjednodušeně skládají z výrobních nákladů, nákladů při skladování a transportu k zákazníkům.

1.2.2 Plánování produkce

Produkce zboží musí být schopná pokrýt poptávku zákazníků v určitém časovém horizontu za minimální možnou cenu. Časový horizont vzniká při vytvoření objednávky nebo při uzavření dlouhodobého kontraktu. Cena produkce se skládá z fixní a

proměnlivé ceny. Fixní cena odpovídá nákladům na stroje a jejich údržbu. Proměnlivá cena koresponduje s časem, který byl stráven na výrobě jednoho produktu. Tento problém se stává složitějším s narůstajícím počtem vyráběných produktů.

1.2.3 Řízení zásob

U řízení zásob je především nutné zvážit, které zboží udržovat ve skladě. Je nutné rozhodovat o tom, na jaké místo, kdy doobjednat další várku zboží, také v jakém množství. Typicky se cena skládá z fixní a proměnlivé ceny, kde fixní cena odráží náklady na přepravu, která se odvíjí od velikosti objednávky. Proměnlivá cena odráží množství a cenu objednaného zboží. Cílem je mít zásobovací politiku, která minimalizuje náklady spojené s objednáváním a skladováním zboží. Tento problém se stává ještě více obtížný, protože množství nabízených produktů se stále zvyšuje a cena objednávky je závislá na objednaném zboží.

1.2.4 Integrace zásob a dopravy

Mějme sklad zásobující část zákazníků, kteří poptávají různé produkty. Pro redukci nákladů je nutné, aby management určil vyvážení mezi skladovacími a transportními náklady. Časté dodávání ze skladu k maloobchodníkům znamená, že zásilky jsou malého objemu. Z toho plyne, že náklady na skladování jsou malé a náklady na přepravu velké a naopak. Cílem je vytvářet takovou skladovací a přepravní strategii, aby co nejvíce minimalizovala náklady.

1.2.5 Správa vozového parku

V logistice sklady obecně dodávají zboží zákazníkům prostřednictvím vozidel s limitovanou kapacitou, popř. jinými dopravními prostředky. Dispečerovým úkolem je optimalizovat zatížení vozidel a vytváření jednotlivých tras pro tyto vozidla. V první řadě dispečer vytváří skupiny zákazníků tak, aby je bylo možné reálně obsloužit z hlediska geografického umístění. V druhé řadě dispečer optimalizuje a hledá nejkratší trasu pro obsloužení zákazníků. Při hledání je cílem minimalizovat náklady spojené s dopravním prostředkem. Tímto problémem se zabývá problematika nazvaná VRP. [6]

1.2.6 Plánování tras pro vozidla

Je nutné zvážit kolik vozidel, popř. jaké kapacity, opustí centrální sklad. Každému vozidlu je přidělen segment zákazníku, který mají být obslouženi. U každého vozidla je pro nás důležitá délka trasy, časová náročnost trasy. Snahou je plánovat tak, aby se bylo vozidlo schopné vrátit zpět do skladu. Tímto problémem se zabývá TSP. [5]

1.2.7 Balení zboží

V logistice se setkáme s problematikou balení zboží popř. skladováním zboží. Zboží je nutné balit nejprve balit do kartónů, zásobníků nebo vozidel a vše je omezeno svou kapacitou. Pokaždé je cílem zabalit zboží do co nejmenšího počtu těchto přepravních nebo skladovacích prostředků. Tento problém nazýváme jako BPP. [4]

1.2.8 Termín dodání

V mnoha případech je nutné zákazníkům dodat zboží ve specifikovaném termínu. Například některý zákazník má kapacity na příjem zboží pouze ve specifikovaný čas, např. od 9 do 11 hodin. V takovém to případě je nutné naplánovat trasu, aby splňovala tyto požadavky. S rostoucím počtem zákazníků tohoto typu může být efektivní plánování tras velmi obtížné.

1.2.9 Předání zboží

V některých distribučních sítích může zákazník specifikovat místo předání resp. doručení. Po té potřebuje dispečer naplánovat takovou trasu pro jednotlivá vozidla tak, aby obsloužila všechny své zákazníky a uspokojila jejich potřeby. Dispečer musí také brát ohled na požadavky zákazníka, termín dodání a omezenou kapacitu vozidel.

2 Heuristické metody v logistice

Heuristická metoda slouží k získání dostatečně přesného řešení, které však nelze obecně dokázat, neboli jedná se NP-těžký problém[3]. Jedná se o nalezení uspokojujícího řešení daného problému v rozumném čase. Výsledné řešení úzce souvisí s vybranou heuristikou, také na konkrétním zadání problému a na vstupních datech. V neposlední řadě výsledný čas doby řešení také ovlivní prostředí, na kterém je heuristika vykonávaná. Z toho je také zřejmé, že je obtížné vyhodnotit a určit efektivitu těchto heuristik vzhledem k své komplexnosti. Vyhodnocení efektivity lze rozdělit do těchto třech kategorií:

- **Empirické porovnání** – spočívá ve vybrání reprezentativních vzorků problému a v porovnání jejich výsledků z několika heuristik. Porovnání může být založeno na přesnosti výsledku, délce zpracování nebo obou vlastností zároveň.
- **Worst-Case Analýza** – spočívá v určení maximální odchylky od optimálního řešení, pokud je počítáno s relativní chybou. Například heuristika pro řešení BPP může zaručit, že jakékoliv řešení používá nanejvýš o 50 % více zásobníků než optimální řešení. Heuristika nám tímto zaručuje, že výsledek se bude nacházet v určitém rozpětí od optimálního řešení.
- **Average-Case Analýza** – spočívá v určení průměrné odchylky od optimálního řešení. Uvádí se jako průměrná relativní chyba výsledku heuristického řešení za určitých předpokladů. Předpoklady mohou být např. umístění skladu, poptávka, velikost, čas dodání, kapacita dopravního prostředku atd. Tato analýza může být provedena pouze za předpokladu velké množiny vstupních dat. Příkladem může být použití BPP, kde vkládáme položky o velikosti větší než 0 a menší nebo rovné než velikost zásobníků. Pokud bychom položky na vstupu seřadili dle velikosti a následně vkládali do jednotlivých zásobníků, tak se s rostoucím počtem vstupních položek budeme přibližovat k optimálnímu řešení, tedy k nulovému rozdílu mezi průměrným a optimálním řešením.

3 NP-těžká úloha

S NP-těžkými úlohami se můžeme setkat v mnoha odvětvích, jako je např. doprava, výpočetní technika, teorie grafů nebo strojírenství.

Jedná se o úlohy, které je možné intuitivně řešit pomocí prozkoumání všech možných variant daného problému, ovšem v případě rozsáhlých úloh, resp. jejich vstupních dat, počet variant výsledného řešení velmi narůstá a není možné dosáhnout přesných výsledků v konečném čase (viz [1]).

Proto pro nalezení efektivního řešení používáme heuristické algoritmy a metody, které nezaručují optimální řešení, ale za cenu odchylky od přesného řešení, lze výsledného řešení dosáhnout v přiměřeném a především konečném čase.

4 Bin Packing Problém (BPP)

Bin Packing problem je možné přeložit jako Problém plnění zásobníků. Je to jeden z několika problémů, na které můžeme v logistice narazit. Zabývá se problematikou, jak co nejlépe naplnit *n* zásobníku tak, aby jejich počet byl ve výsledku co nejmenší.

Tento problém se vyskytuje nejen v logistice. Tyto heuristiky je možné použít, všude tam, kde je naším cílem efektivně zpracovat nebo minimalizovat délku zpracovávaného materiálu, z kterého jsou tvořeny části konečných produktů. Tento problém se často objevuje jako podproblém komplexnějšího problému v daném oboru nebo průmyslu.

Metody můžeme dělit do dvou základních skupin dle strategie zpracování položek:

- *on-line metody* – položky z fronty postupně odebíráme, dokud není fronta prázdná a není proto tedy nutné znát předem počet položek,
- *off-line metody* – před vkládáním známe přesný počet položek, což nám umožňuje s těmito položkami pracovat před samotným vkládáním pro dosažení lepších výsledků.

BPP je velice komplexním problémem a je ho možné dělit na typy 3D-BPP[4.3] a dále na jeho postupně zjednodušené formy 2D-BPP[4.2] a 1D-BPP[4.1]. Pokud bychom uvažovali nejsložitější trojrozměrný problém, tak mějme trojrozměrné předměty v trojrozměrném prostoru, kde je nutné řešit zároveň dva problémy. Řešíme nejen, do jakého zásobníku bude prvek vložen, ale také na jakou pozici prvek v zásobníku umístíme v ose *x*, *y* a *z*. Pokud problém zjednodušíme na typ dvojrozměrného prostoru, je nutné vyřešit, do jakého zásobníku položky vložíme a zároveň také na jakou pozici v zásobníku položky umístit na ose *x*. Pokud bychom BPP zjednodušili na jednorozměrný prostor, zabýváme se pouze do jakého zásobníku vložit položku za předpokladu, že zásobník disponuje potřebnou kapacitou.

4.1 BPP v jednorozměrném prostoru (1D-BPP)

Typ 1D je nejjednodušším případem BPP, kde řešíme pouze problém do jakého zásobníku předmět umístit. Pokud bychom uvažovali tuto zjednodušenou metodu, měli bychom dodržet jistá pravidla. Předměty, které vkládáme do zásobníku, musí mít stejnou šířku *i* délku jako použité zásobníky. Předměty se se liší pouze ve své výšce.

Obecným předpokladem při použití těchto metod je, že výška prvků musí být menší nebo rovna výšce zásobníků. V opačném případě by nemohl být prvek zařazen do žádného prázdného zásobníku.

1D – BPP je možné definovat takto, mějme n položek obdélníkového tvaru, kde každá má šířku w_j, výšku h_j a hloubku d_j $(j \in J = \{1, \dots n\})$, pak platí, že $w_j = W$, $h_j \leq H$ a $d_j = D$ $(j \in J)$ (viz [4]).

Reálným příkladem tohoto problému může být naplnění nákladních vozidel. Pokud bychom měli vozidla o kapacitě x palet. Dále bychom měli y objednávek zboží, které je převáženo na paletách. Pak bychom těmito heuristikami dostali řešení, které by nám udávalo počet nákladních vozidel a využití jejich kapacit pro přepravu do cílového místa. U tohoto příkladu je také nutné zmínit, že daný typ přepravovaného zboží je nutné přepravit společně v jednom vozidle.

4.1.1 Next-Fit

Metoda Next-fit je nejjednoduší heuristický algoritmus pro řešení BPP a patří do skupiny on-line metod.

Algoritmus je navržen tak, aby první položku vždy vložil do prvního zásobníku. Následující každá položka je vkládána do aktuálních zásobníků za předpokladu, že výška položky nepřesahuje zbývající kapacitu aktuálního zásobníku. V opačném případě je vytvořen nový zásobník, který je zařazen jako následující aktuálního zásobníku a je do něho vložena položka. Tento zásobník se zároveň stává aktuálním.

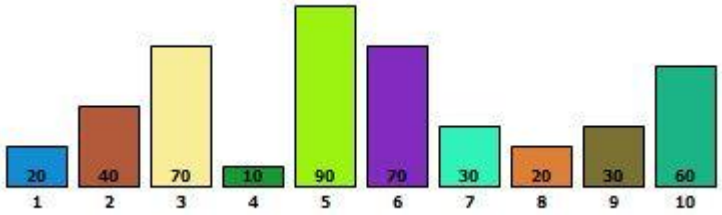
4.1.2 First-Fit

Heuristika First-Fit je základní a výchozí on-line heuristickým algoritmem pro řešení BPP.

U této heuristiky máme připraveny jednotlivé položky ve frontě. Postupně z jejího vrcholu odebíráme položky a vkládáme je do zásobníků podle níže uvedeného algoritmu.

V každém průchodu po odebrání položky z fronty, algoritmus položku vloží do prvního možného zásobníku za předpokladu, že výška položky nepřesahuje zbývající kapacitu aktuálního zásobníku. V opačném případě, kdy není nalezen ani jeden zásobník s dostatečnou kapacitou je vytvořen a zařazen do množiny zásobníků vložen nový, do kterého je položka vložena. V dalším průchodu, resp. při vložení následující položky, algoritmus opět začíná prvním zásobníkem v množině a tento rekurzivní postup opakujeme, dokud není fronta s položkami prázdná.

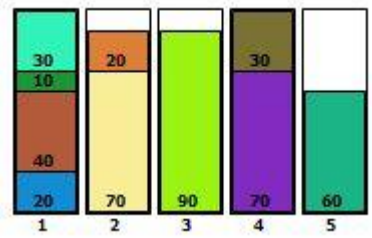
Pro přehlednější ilustraci algoritmu je na následujícím obrázku (Obrázek 1) uveden jednoduchý příklad.



Obrázek 1 - Vstupní položky pro heuristiku First-Fit

V tomto případě máme 10 vstupních položek o výšce $(0 - 100\rangle$. Dále máme k dispozici prázdné zásobníky o velikosti 100. Jednotlivé položky postupně odebíráme a umisťujeme do příslušných zásobníků. Položka o velikosti 20 je vložena jako první do prvního zásobníku. Následující položka o velikosti 40 je vložena také do prvního zásobníků, protože splňuje výše zmíněné pravidlo této heuristiky, tj. výsledná velikost po vložení $60\ (20 + 40) \leq 100$. První zásobník má tedy aktuální velikost 60. Dále vezmeme třetí položku o velikosti 70, pokusíme se jí vložit do prvního zásobníku. Spočteme aktuální velikost zásobníku a porovnáme s maximální velikostí zásobníků $130(60 + 70) > 100$. Z toho plyne vytvoření nového zásobníku a položka je vložena do druhého zásobníku a takto pokračujeme dále.

Následující obrázek (Obrázek 2) reprezentuje výsledný stav heuristiky First-Fit po vyprázdnění vstupní fronty a zařazení jednotlivých položek.



Obrázek 2 - Stav zásobníků po použití heuristiky First-Fit

V konečném stavu tohoto jednoduchého případu jsme využili 5 zásobníků. Pro tuto heuristiku lze určit horní hranici použitých zásobníku vztahem (viz [1]):

$$b^{FF}(L) <= \frac{17}{10} b^{*}(L),$$

kde $b^{FF}(L)$ – je počet zásobníků, které byly použity pro řešení,
$b^{*}(L)$ – je nejmenší počet zásobníků, které jsou potřeba pro řešení.

4.1.3 First-Fit Decreasing

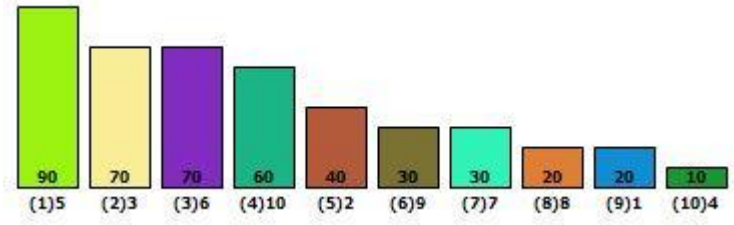
Tato metoda patří do skupiny off-line metod a vychází ze základní metody First-Fit. Liší se pouze tím, že vkládané položky nejdříve řadíme dle příslušné výšky. Poté jednotlivé položky postupně vkládáme do prázdných zásobníků dle stejného algoritmu jako u metody First-Fit.

Cílem této metody je nejprve vkládat do zásobníků větší položky a poté hledat místo pro menší položky, které nám zaplní zbylou kapacitu v zásobnících. Pro tuto heuristiku lze také určit horní hranici použitých zásobníku a to vztahem (viz [1]):

$$b^{FF}(L) <= \frac{11}{9} b^{*}(L) + 3,$$

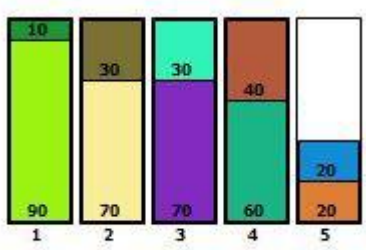
kde $b^{FF}(L)$– je počet zásobníků, které byly použity pro řešení, $b^{*}(L)$ – je nejmenší počet zásobníků, které jsou potřeba pro řešení

Pro přehlednější ilustraci algoritmu je na následujícím obrázku (Obrázek 3) uveden jednoduchý příklad.



Obrázek 3 - Sestupně setříděné položky pro heuristiku First-Fit Decreasing.

Setříděné položky vkládáme do zásobníku podle stejného algoritmu jako u heuristiky First-Fit. Na následujícím obrázku (Obrázek 4) je uveden cílový stav této metody.



Obrázek 4 - Stav zásobníků po použití heuristiky First-Fit Decreasing.

Z obrázku je také zřejmé, že bylo použito 5 zásobníků. Můžeme si také všimnout, že jsou nejdříve vkládány položky o velké velikosti a po té zbylou kapacitu zásobníků zaplníme menšími. V tomto případě se nám také podařilo první 4 zásobníky plně využít.

4.1.4 Last-Fit

Heuristika Last-Fit je opačnou metodou heuristiky First-Fit.

Algoritmus nehledá první možný hodící se zásobník, ale poslední hodící se. Hodícím se zásobníkem je myšlen takový zásobník, který po vložení aktuální položky nepřesahuje maximální možnou kapacitu.

4.1.5 Best-Fit

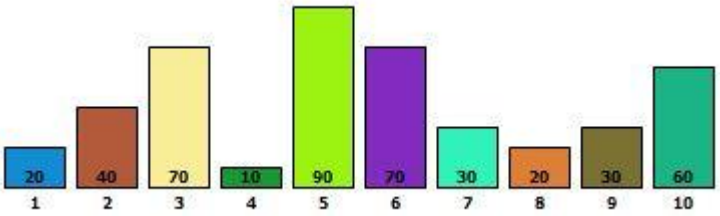
Heuristika Best-Fit je mírně pozměněnou metodou First-Fit a také spadá do skupiny on-line metod.

U této heuristiky máme jednotlivé položky ve frontě. Postupně z jejího vrcholu odebíráme položky a vkládáme je do zásobníků podle níže uvedeného algoritmu.

V každém průchodu po odebrání položky z fronty, algoritmus položku vloží do nejzaplněnějšího zásobníku za předpokladu, že výška položky nepřesahuje zbývající kapacitu vybraného zásobníku. V opačném případě rekurzivně hledáme následující nejzaplněnější zásobník. Pokud se nám nepodařilo vyhledat zásobník, který disponuje volnou kapacitou pro vkládanou položku, je vytvořen nový zásobník a zařazen do množiny zásobníků. Následně je položka vložena do tohoto aktuálně posledního zásobníku. V dalším průchodu, resp. při vložení následující položky, algoritmus opět vyhledává vhodný zásobník v množině a tento rekurzivní postup opakujeme, dokud není fronta s položkami prázdná.

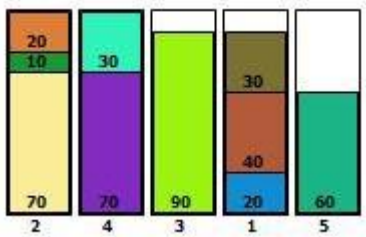
U tohoto algoritmu lze docílit lepších výsledků za použití některých metod, které časově zkrátí vyhledávání vhodného zásobníku. Jednoduchou optimalizací může být sestupné seřazení zásobníků podle obsazené kapacity. Dále pro efektivní vyhledávání můžeme použit složitějších datových struktur pro uložení zásobníků jako je např. 2-3 strom.

Pro přehlednější ilustraci algoritmu je na následujícím obrázku (Obrázek 5) uveden jednoduchý příklad.



Obrázek 5 - Vstupní položky pro heuristiku Best-Fit.

Následující obrázek (Obrázek 6) pak reprezentuje výsledný stav heuristiky Best-Fit po vyprázdnění vstupní fronty a zařazení jednotlivých položek.



Obrázek 6 - Stav zásobníků po použití heuristiky Best-Fit.

V konečném stavu tohoto jednoduchého případu jsme využili 5 zásobníků. Horní hranici použitých zásobníku lze určit vztahem (viz [1]):

$$b^{FF}(L) <= \frac{17}{10} b^{*}(L)$$

kde $b^{FF}(L)$– je počet zásobníků, které byly použity pro řešení,
$b^{*}(L)$ – je nejmenší počet zásobníků, které jsou potřeba pro řešení.

4.1.6 Best-Fit Decreasing

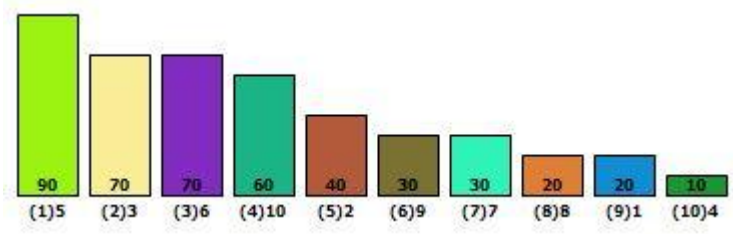
Tato metoda vychází ze základní metody Best-Fit, liší se pouze tím, že vkládané položky nejdříve řadíme podle jejich výšky, z čehož plyne, že tato metoda spadá do skupiny off-line metod. Poté jednotlivé položky postupně vkládáme do prázdných zásobníků podle stejného algoritmu jako u metody Best-Fit.

Cílem této metody je nejprve vkládat do zásobníků větší položky a poté hledat místo pro menší položky, které nám zaplní zbylou kapacitu v zásobnících. Pro tuto heuristiku lze také určit horní hranici použitých zásobníku a to vztahem (viz [1]):

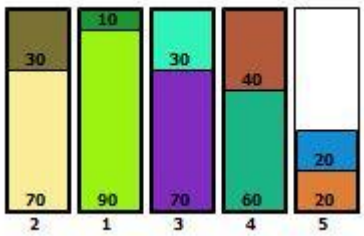
$$b^{FF}(L) <= \frac{11}{9} b^{*}(L) + 3,$$

kde $b^{FF}(L)$– je počet zásobníků, které byly použity pro řešení,
$b^{*}(L)$ – je nejmenší počet zásobníků, které jsou potřeba pro řešení

Pro přehlednější ilustraci algoritmu následující obrázky (Obrázek 7, Obrázek 8) demonstrují jednoduchý příklad.



Obrázek 7 - Sestupně setříděné položky pro heuristiku Best-Fit Decreasing.



Obrázek 8 - Stav zásobníků po použití heuristiky Best-Fit Decreasing.

4.1.7 Worst-Fit

Heuristika Worst-Fit je opačnou metodou heuristiky Best-Fit.

Algoritmus nehledá nejvíce hodící se zásobník, ale poslední hodící se. Hodícím se zásobníkem je myšlen takový zásobník, který po vložení aktuální položky nepřesahuje maximální možnou kapacitu zásobníku a zároveň je nejvíce zaplněný, resp. nejméně zaplněný. V případě, že takový zásobník neexistuje je inicializován nový zásobník a vložen do množiny.

4.1.8 Almost-Worst-Fit

Heuristika Almost Worst-Fit vyhází z heuristiky Worst-Fit.

Algoritmus pouze nehledá poslední hodící se zásobník, ale druhý poslední hodící se. Hodícím se zásobníkem je myšlen takový zásobník, který po vložení aktuální položky nepřesahuje maximální možnou kapacitu zásobníku a zároveň je druhým nejméně zaplněným zásobníkem z množiny inicializovaných zásobníků. V případě, že takový zásobník neexistuje je inicializován nový zásobník a vložen do množiny.

4.2 BPP v dvourozměrném prostoru (2D-BPP)

Dvoj dimenzionální typ BPP se zaměřuje na optimální využití plochy. S tímto typem se setkáme všude tam, kde je cílem efektivně vkládat malé obdélníkové oblasti, které představují vkládané položky, do co nejmenšího počtu velkých standardizovaných obdélníkových oblastí, které představují zásobníky.

Praktickým příkladem může být zpracování materiálu v dřevařském nebo sklářském průmyslu, kde je našim cílem rozdělit velké části výrobního materiálu na menší části, které pak slouží k výrobě daného produktu.

V dopravě se s tímto problémem můžeme setkat při skladovaní zboží, kde je zboží umísťováno na paletách do vysokých skladovacích prostor, které jsou přizpůsobeny násobkům rozměrů standardizované palety.

Na tento problém také narazíme v novinovém průmyslu, kde je nutné řešit rozložení jednotlivých článků a reklam na stránce s cílem efektivního zaplnění jednotlivými informacemi a reklamou.

S tímto problémem také úzce souvisí Strip Packing Problem, dále označován jako SPP. U tohoto problému vkládáme obdélníkové položky pouze do jednoho zásobníku s neomezenou výškou.

Praktickým příkladem SPP může být zpracování textilních látek, plechových tabulí nebo papírových rolí v příslušné průmyslové výrobě. Vždy se jedná o roli výrobního materiálu a cílem při výrobě cílových produktů je zpracovat co nejkratší

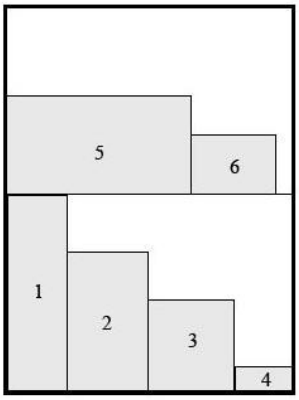
možnou délku role. Tento cíl nám ušetřit nemalé finanční prostředky při výrobě konečných produktů.

2D – BPP může být definován takto, mějme n položek obdélníkového tvaru, kde každá má šířku w_j, výšku h_j a hloubku d_j $(j \in J = \{1, \dots n\})$ a mějme také neomezený počet stejných zásobníků, které disponují šířkou W, výškou H a hloubkou D. Můžeme pak říci, že vstupní položka by měla mít rozměry $w_j \leq W$, $h_j \leq H$ a $d_j = D$ $(j \in J)$ (viz [4]).

Pokud bychom chtěli tyto praktické problémy řešit pomocí aplikací, které řídí výrobní proces, budeme donuceni použít konstruktivních nebo heuristických algoritmů.
V následujících kapitolách budou stručně uvedeny některé heuristické algoritmy.

4.2.1 Bottom Left Fill

Tato metoda je jedna ze základních konstruktivních strategií. Cílem tohoto heuristického algoritmu je seskupit položky do jednotlivých úrovní neboli skupin vkládaných položek. Poté může být pro každou skupinu použita vhodná heuristika jednodimenzionálního BPP, např. FF[4.1.2]. Tento přístup má ovšem jednu nevýhodu a to tu, že může v každé úrovni docházet k neefektivnímu využití prostoru zásobníku. Následující obrázek (Obrázek 9) ilustruje neefektivní využití prostoru zásobníku.



Obrázek 9 - Ukázka plýtvání prostoru metody Bottom Left Fill.

Při aplikování této metody musí být dodržena následující pravidla:

- V každé úrovní je nejlevější položkou ta, které má v dané úrovni největší výšku,
- V každém zásobníku je nejníže a nejlevěji položená položka ta, která disponuje největší výškou z celé množiny vkládaných položek,
- Při vkládání jsou položky seřazeny sestupně podle výšky a následně vkládány do zásobníků (viz 5).

4.3 BPP v třírozměrném prostoru (3D-BPP)

BPP v třírozměrném prostoru opět spadá jako předchozí problémy do NP-těžkých úloh[3] a v praxi je velmi těžce řešitelný. Protože se jedná o třírozměrný prostor, je tento typ nejsložitější.

3D – BPP může být definován takto, mějme *n* položek obdélníkového tvaru, kde každá má šířku w_j, výšku h_j a hloubku d_j $(j \in J = \{1, \dots n\})$ a mějme také neomezený počet stejných tří rozměrných zásobníků, které disponují šířkou W, výškou H a hloubkou D. Cílem je kolmé skládání všech vstupních položek do co nejmenšího počtu zásobníků. Pokud bychom uvažovali, že položky nemohou rotovat, to znamená, že hrana položky je paralelní hraně zásobníku. Můžeme říci, že vstupní položka by měla mít rozměry $w_j \leq W$, $h_j \leq H$ a $d_j \leq D$ $(j \in J)$ (viz [4]).

5 Traveling Salesman Problém (TSP)

5.1 Úvod

TSP je dalším velmi známým optimalizačním problémem v logistice. Optimálního řešení v přiměřené době lze dosáhnout pouze v případě malých instancí. Tento problém se řadíme mezi typy úloh NP-těžké úlohy[3] z čehož plyne, že při velkých instancích budou heuristické algoritmy časově velmi náročné a nezaručují nám nejvýhodnější řešení.

Problém u těchto úloh spočívá v nalezení nejkratší Hamiltonové kružnici v ohodnoceném úplném grafu.

Zjednodušeně lze TSP popsat jako hledání nejkratší možné cesty obchodního cestujícího, který musí pouze jednou navštívit postupně *n* měst (vrcholů) a poté se vrátit do výchozího města (vrcholu) za předpokladu, že obchodní cestující cestuje po silnicích (hranách) mezi jednotlivými městy. Je nutné také zmínit podmínku, že každé město obchodní cestující navštíví právě jednou (viz [1]).

Existuje mnoho heuristických algoritmů pro nalezení optimálního řešení. Ovšem ne každý je proveditelný pro velké instance, protože složitost roste exponenciálně na základě počtu vrcholů v grafu. Proto se u těchto metod zabýváme převážně rychlostí výpočtu za cenu kvality výpočtu. TSP dělíme podle jednotlivých vlastností grafu:

- *Symetrický* – vzdálenost je symetrická, tj. z města A do města B je rovna vzdálenosti z města B do města A,
- *Asymetrický* – opak symetrického, vzdálenost mezi vrcholy není symetrická,
- *Metrický* – pro každá tři města platí trojúhelníková nerovnost,
- *Euklidovský* – vzdálenosti měst odpovídají vzdálenosti v rovině.

V této kapitole jsou popsané metody pro řešení symetrického TSP, dále STSP. Některé lze ovšem modifikovat na řešení asymetrického TSP. Heuristiky lze rozdělit podle typu přístupu na dva následující typy. Celkové množství možných tras lze pro případ STSP vyjádřit tímto vztahem (viz [1]):

$$e = (n-1)!/2,$$

kde e – počet možných výsledných tras,

n – je počet měst, které má obchodní cestující navštívit.

Z uvedeného vztahu plyne, že s rostoucím počtem měst se řešení TSP stává velice komplikovanou a časově náročnou úlohou.

5.2 Heuristiky zaměřené na konstrukci trasy

Společnou vlastností těchto metod je, že končí po nalezení cesty a není prováděna následná dodatečná optimalizace těchto cest.

5.2.1 Nearest Neighbor (NN)

Název této heuristiky lze přeložit jako heuristika využívající nejbližších sousedů. Tato heuristika spočívá ve vybrání libovolného vrcholu, který je jako počáteční a zároveň se stane i cílovým vrcholem. Dalším krokem je nalezení nejbližšího sousedního vrcholu, který ještě nebyl navštíven. Poté se přesuneme do tohoto vrcholu a postup opakujeme do té doby, než jsou navštíveny všechny vrcholy. Na závěr se vrátíme z aktuálního vrcholu do počátečního (cílového).

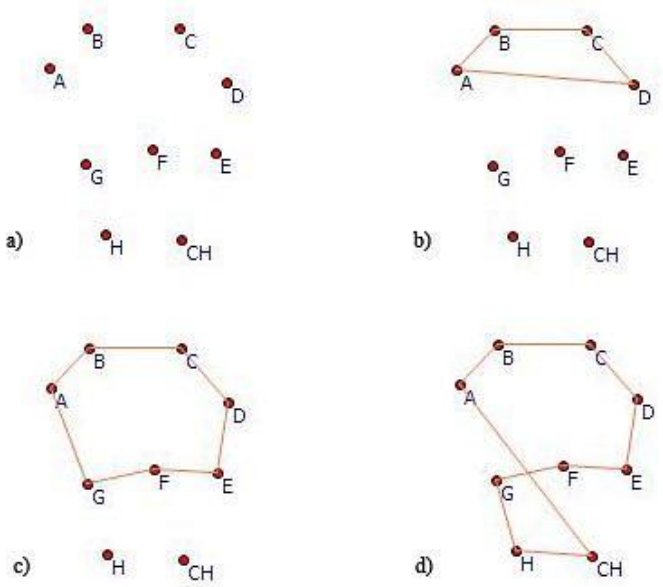
Tato metoda při výpočtu optimální trasy dosahuje složitosti (viz [1]):

$$O(n^2),$$

kde n – je počet vrcholů.

Metoda má tendenci zpočátku vkládat krátké cesty do cílové trasy a na konec zbývají velmi dlouhé. Tento jev se projeví při vkládání poslední hrany, která spojuje koncový a počáteční bod pro uzavření kruhu.

Na následujícím obrázku (obrázek 10) je naznačen postup průchodu algoritmu 9 vrcholy. Můžeme zde taky vidět jev, kdy jako poslední zbyla dlouhá cesta z bodu *CH* do bodu *A*.



Obrázek 10 – Průchod vrcholy heuristikou Nearest Neighbor

5.2.2 Nearest Insertion (NI)

Inicializační částí této heuristiky je vložení podmnožiny měst. Tato podmnožina může být tvořena třemi a více městy, které v případě třech tvoří trojúhelník resp. n-úhelník. Jedním ze způsobů jak vybrat tato inicializační města může být ten, že vybereme město položené nejvíce vlevo, vpravo, nahoře a dole.

V prvním kroku nalezneme nejkratší cesty mezi jednotlivými městy a vložením tato města do výsledné cesty. V dalších krocích postupně vybíráme a vkládáme zbylá města, která ještě nebyla vložena do výsledné cesty. V každém kroku vždy vybíráme takové město, které leží nejblíže k jednomu z již vložených měst. Pro aplikaci výběru je možné také použít město, které leží nejdále od již vložených měst nebo město, které nejméně prodlouží výslednou cestu. Tento postup opakujeme, dokud nezbývají města, která ještě nebyla vložena do výsledné cesty.

5.3 Heuristiky zaměřené na optimalizaci trasy

Předchozí heuristiky docílí pouze prohlídky všech měst. Heuristiky zaměřené na optimalizaci se snaží předchozí nalezené řešení optimalizovat. Předchozí heuristiky při velkém množství vstupních dat nedosahují uspokojujících výsledků a proto je snahou těchto heuristik optimalizovat nalezené řešení. Základními optimalizačním algoritmem je algoritmus k-opt.

5.3.1 K - optimalizace (k-opt)

Tento optimalizační algoritmus zlepšuje cestu mezi dvěma městy, tak že rekurzivně odebírá k cest a nahradí je jinými k cestami. Nahrazení aplikujeme pouze v případě, že nahrazující cesta má menší vzdálenost a je i nadále ve výsledku dodržena Hamiltonová kružnice. Tento postup opakujeme, dokud není výsledná trasa k-optimální. K-optimální trasou rozumíme trasu, kterou jíž nelze dále optimalizovat podle předchozího pravidla. S touto metodou se můžeme také často setkat pod názvem k-optimalizační posun.

Algoritmus k-opt dosahuje složitosti v čase (viz [1]):

$$O(n^k),$$

kde n – je počet vrcholů,

k – stupeň optimalizace, resp. počet hran, které mají být zaměněny za vhodnější.

Z tohoto důvodu se v praxi používají především 2 – optimalizace a 3-optimalizace.

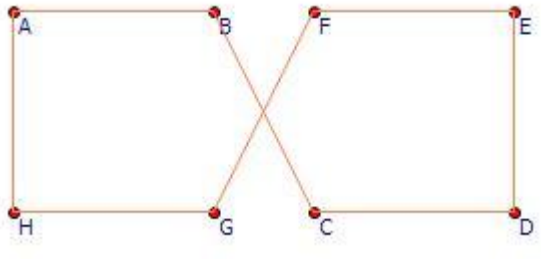
5.3.2 2 - optimalizace (2-opt)

Nejjednodušší variantou k-optimalizace je 2-optimalizace. Algoritmus spočívá v odebrání 2 cest, které nám nejprve výslednou trasu (Hamiltonovou kružnici) rozdělí na 2 trasy. Dalším krokem je vyhledání 2 cest, které nahradí odebrané. Hrany

nahrazujeme pouze za předpokladu, že nová výsledná trasa bude mít kratší vzdálenost než výchozí trasa před odebráním hran. Takto nahrazení cest opakujeme, dokud už není možné lepší optimalizace výsledné trasy. Výsledek na konci v tomto případě považujeme za 2-optimální.

V následujícím příkladu je vidět jednoduchá výměna hran. Cílem je navštívit 8 vrcholů co nejkratší trasou mezi vrcholy.

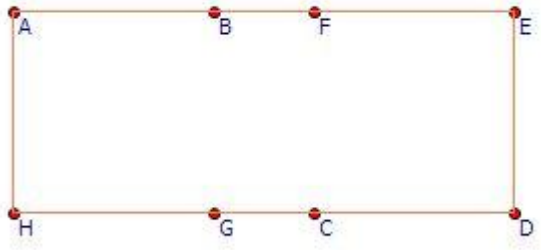
Města jsou označena identifikátory A, B, C, D, E, F, G, H a na následujícím obrázku (Obrázek 11) je vidět počáteční stav trasy, kterou bude optimalizovat.



Obrázek 11 - Výchozí stav před aplikování 2 – optimalizací

Z obrázků je na první pohled zřejmé, že je možné docílit kratší trasy záměnou hran {B, C} a {G, F}. Aktuální pořadí měst v trasy je A, B, C, D, E, F, G, H.

Na následujícím obrázku (Obrázek 12) je zobrazen konečný stav po aplikování 2-optimalizace, která byla aplikována na předchozí případ.



Obrázek 12 - Stav po aplikování 2 – optimalizace

Z obrázku je zřejmé, že došlo k záměně právě těchto dvou hran, {B, C} byla zaměněna s {B, F} a hrana {F, G} s {C, G}. Aktuálním pořadím měst v trase je A, B, F, E, D, C, G, H.

6 Vehicle Routing Problém (VRP)

6.1 Úvod

VRP je známým komplexním problémem v logistice, kde je cílem hledat optimální seskupení zákazníků pro distribuční síť tak, abychom minimalizovali náklady spojené s doručením zboží k zákazníkovi.

Cílem této metody je vytvořit minimální a optimalizovaný počet skupin zákazníku pro jednotlivé dopravní prostředky. Při vytváření skupin zákazníků je potřeba brát v potaz kapacitu dopravního prostředku, čas dodání zboží k zákazníkovi, možnosti zpětné dopravy do skladu a další možné ovlivňující faktory. Tyto faktory mohou ovlivnit výsledný výpočet rozložení. Na základě těchto ovlivňujících faktorů můžeme tento problém dělit do jednotlivých skupin (viz[1]):

- VRP with pick up and delivery – v první řadě je třeba zboží dopravit ze skladu k jednotlivým zákazníkům, ale zákazník také může odeslat zboží zpět do skladu,
- VRP with time windows – každý zákazník má být obsloužen v daném časové horizontu,
- Capacited VRP – dopravní prostředky mají omezenou přepravní kapacitu na zboží, které je třeba doručit zákazníkům,
- Split Delivery VRP – zákazníci mohou být obslouženi i více dopravními prostředky zároveň,
- Multiple Depot VRP – dodavatel má k dispozici více skladů, které jsou umístěny na rozdílných regionech,
- Periodic VRP – zboží musí být pravidelně dodáváno v daném časovém intervalu,
- Stochastic VRP – jednotlivé předhozí požadavky mohou být náhodné.

Obecný cílem heuristik pro řešení VRP je rozdělit výchozí oblast, která obsahuje N zákazníků na menší podoblasti, které obsahují j zákazníků z původní množiny zákazníků $Z = \{Z_1, Z_2, .. Z_N\}$. Oblasti jsou vytvářeny tak, aby každá oblast obsahovala Q zákazníků, krom jedné oblasti, kde se může vyskytnout menší počet zákazníků. Ovšem každá vytvořená oblast nesmí obsahovat více jak Q zákazníků.

Každé menší vytvořené podoblasti je pak v případě VRP přiděleno jedno vozidlo, které obslouží všechny zákazníky v jedné oblasti. Z pravidla je prvním obslouženým zákazníkem z podmnožiny $N\ (j)$, ten který je nejblíže k centrálnímu skladu. Pro vytvoření optimální trasy každého vozidla pak dále ve vybrané oblasti aplikujeme vhodný heuristický algoritmus řešící TSP[5], který nám vytvoří cílovou trasu pro jednotlivá vozidla v přidělených podoblastech. Po navštívení všech zákazníků se vozidlo vždy vrací zpět do centrálního skladu, tak jak je tomu v TSP.

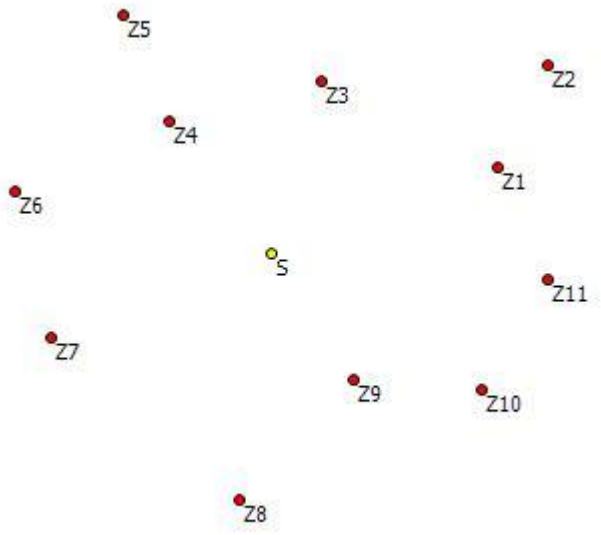
Dále budou představeny dvě základní heuristiky pro vytvoření podoblastí, které obsahují maximálně Q zákazníků.

6.2 Polar Region Partitioning (PRP)

U této heuristiky nejprve definujeme kruh, který uvnitř obsahuje N zákazníků a 1 sklad, z kterého je v daném případě nutné obsloužit zákazníky. Střed kruhu se nachází na pozici skladu, tedy ve vrcholu, ve kterém leží sklad obsahující zboží pro přepravu k jednotlivým zákazníkům..

Pokud budeme uvažovat Euklidův prostor, tak poloměrem kruhu se stává vzdálenost mezi středem kruhu a nejvzdálenějším zákazníkem od středu. Kruh v dalších krocích postupně dělíme na kruhové výseče (regiony) pomocí příček o délce poloměru r, které svírají úhel α s osou x. Nejprve spočítáme úhel α pro každého zákazníka a ty vzestupně seřadíme podle vypočítaného úhlu α. Po té zvolíme libovolně počáteční vrchol a postupně procházíme seřazené vrcholy dle α. Kruhové výseče postupně tvoříme tak, aby obsahovaly maximálně Q zákazníků.

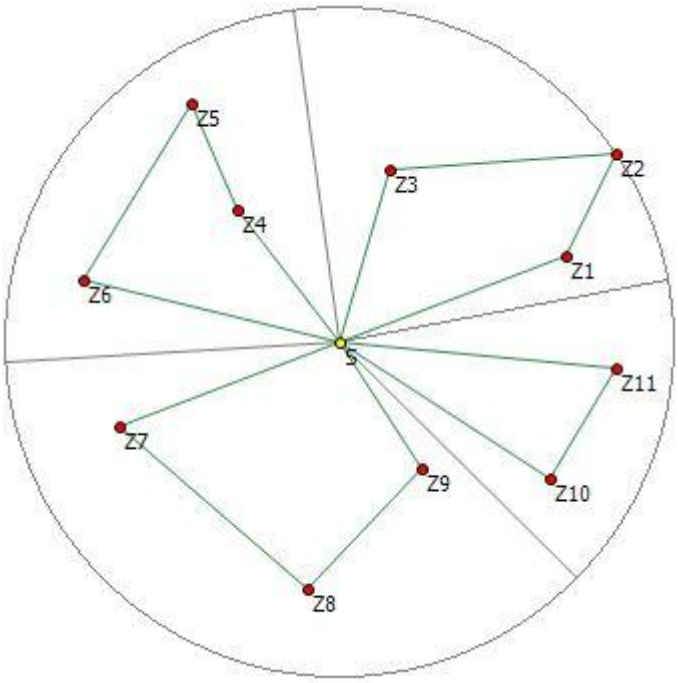
Na následujícím obrázku (Obrázek 13) je vidět počáteční stav před aplikací heuristiky PRP.



Obrázek 13 - Výchozí stav před použitím heuristiky PRP

Z obrázku je patrné, že v tomto případě máme 1 centrální sklad S a 11 zákazníků $Z = \{Z_1, Z_2, \cdots Z_{11}\}$, kterým má být doručeno zboží z centrálního skladu.

Na následujícím obrázku (Obrázek 14) je vidět výsledný stav po aplikaci heuristiky PRP, kde je 1 vozidlo schopné rozvést zboží maximálně 3 zákazníkům.



Obrázek 14 - Stav rozdělení zákazníků po aplikaci heuristiky PRP

Z předchozího obrázku je zřejmé, že zákazníci byli rozděleni do čtyřech regionů $R = \{R_1, R_2, R_3, R_4\}$. Z nichž každý region obslouží právě 1 vozidlo. V tomto případě byl zvolený jako počáteční bod Z1, který svírá nejmenší uhel s osou *x*.

Pro dosažení minimální celkové vzdálenosti je nutné určit přesně počáteční bod, od kterého tvoříme jednotlivé výseče kruhu. V případě že *N* je dělitelné *k*, lze jednoduše vyhodnotit možné kombinace PRP a určit právě tu, která dosahuje minimální vzdálenosti.

Při velkém počtu zákazníku muže vznikat nevýhodné dělení kruhu na výseče, neboť při stoupajícím počtu zákazníku, klesá svíraný uhel α dané oblasti. V takovémto případě je možné tuto heuristiku vylepšit o prstencové dělení, kde počáteční kruh nejprve rozdělíme na prstence a po té aplikujeme algoritmus PRP na jednotlivé prstencové oblasti.

6.3 Rectangular Region Partitioning (RRP)

Tato heuristika obsahuje výchozí obdélník se stranami *a* a *b*, který obsahuje množinu *N* zákazníků a je následně dělen vertikálními a horizontálními příčkami. V prvním kroku dělíme původní obdélník *t* vertikálními přímkami, tak že každý vzniklý region obdélníku obsahuje právě *(h+1)Q* zákazníků, krom jednoho, kde jich může být méně. Každý z těchto *t+1* regionů je pak rozdělen *h* horizontálními přímkami na *h +1* podoblastí, kde každá obsahuje právě *Q* zákazníků, krom jedné, ve které může být počet zákazníků opět menší.

Pro výpočet horizontálních příček h platí následující vztah (viz [1]):

$$h = ceil\left(\sqrt{\frac{n}{Q}} - 1\right),$$

kde n – je celkový počet zákazníků

Q – je maximální počet zákazníků v právě 1 výsledné oblasti.

Dále lze odvodit následující vztah pro výpočet vertikálních příček t (viz [1]):

$$t = ceil\left(\frac{n}{(h+1)Q}\right) - 1,$$

kde n – je celkový počet zákazníků

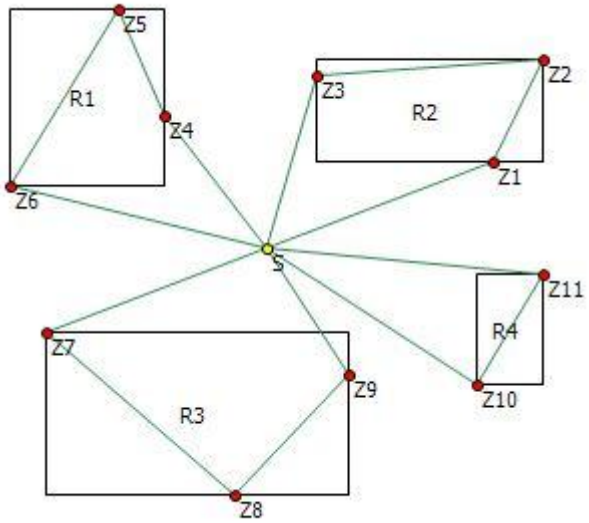
Q – je maximální počet zákazníků v právě 1 výsledné oblasti,

h – je počet horizontálních příček 1 vertikální podoblasti.

Dále je nutnou podmínkou, aby počet horizontálních příček h splňoval následující kritérium (viz [1]):

$$t(h+1)Q < n \leq (t+1)(h+1)Q$$

Na následujícím obrázku (Obrázek 15) je zobrazen výsledný stav po aplikaci heuristiky RRP. Heuristika byla aplikována na stejnou množinu jedenácti zákazníku jako v případě jednoduchého příkladu z kapitoly[6.2], popisující heuristiku PRP.



Obrázek 15 - Stav rozdělení skupin zákazníků po aplikaci heuristiky RRP

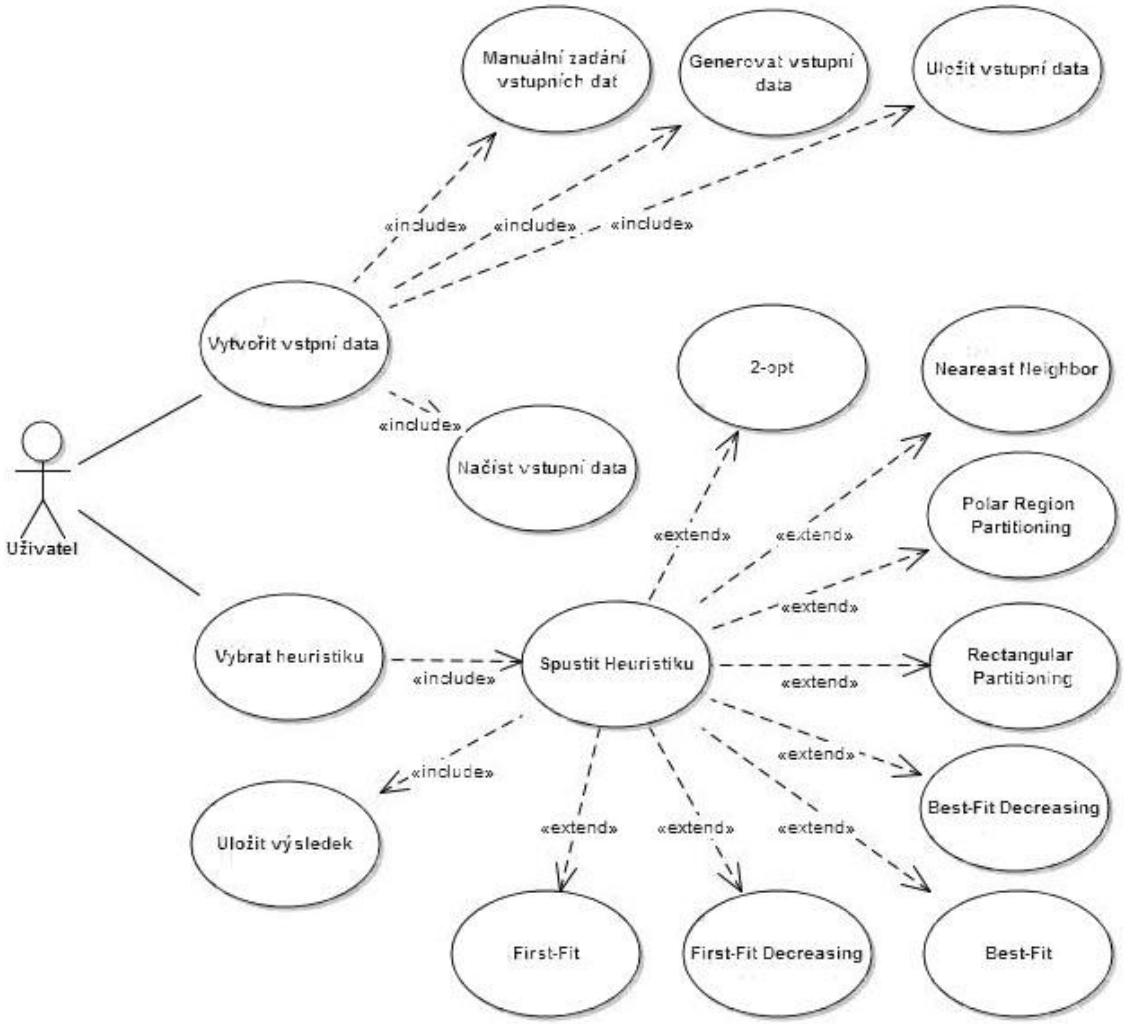
Z předchozího obrázku je zřejmé, že zákazníci byli také rozděleni do 4 třech regionů $R = \{R_1, R_2, R_3, R_4\}$. Z nichž každý region obslouží právě 1 vozidlo.

7 Aplikace demonstrující vybrané heuristické algoritmy

Pro demonstraci vybraných problémů byla naimplementována aplikace, která se skládá ze dvou částí:

- z knihovny obsahující jednotlivé heuristiky a algoritmy, která je navržena tak, aby jí bylo možné snadno využít pro jinou aplikaci prostřednictvím abstraktních tříd a rozhraní,
- z desktopové aplikace, která slouží k zobrazení průběhu a výsledků jednotlivých heuristik a algoritmů.

Požadavky na aplikaci jsou specifikovány na následujícím obrázku (Obrázek 16) pomocí Use- Case diagramu.



Obrázek 16 - Use Case diagram aplikace

7.1 Použité technologie

7.1.1 .Net Framework 4.0

.Net Framework je platforma pomocí, které je možné vyvíjet aplikace. Poskytuje řadu služeb pro kompilaci a nasazení aplikace, podporuje desktopové, mobilní a webové aplikace nebo služby. Skládá se z CLR, který poskytuje funkce pro správu paměti a ostatních systémových prostředků. Dále obsahuje knihovnu, která poskytuje testovatelný a znovupoužitelný kód pro hlavní oblasti při vývoji aplikaci.

Z pohledu uživatele, který nevyvíjí aplikace postavené na této platformě, není třeba znát specifické informace. Většina částí je pro uživatele transparentní a nemusí se o nic starat a nikterak uživatele neobtěžuje (viz [5]).

V případě, že uživatel využívá operační systém Windows, tak by již měl být .Net Framework součástí systému. V opačném případě je jej nutné doinstalovat.

Z pohledu vývojáře Net. Framework poskytuje tyto služby:

- *Správu paměti* - o alokaci a dealokaci místa v paměti se stará CLR,
- *Základní datové typy* - datové typy jsou definovány a zapouzdřeny v .Net Frameworku a nikoliv překladačem jak je tomu zvykem v jiných jazycích,
- *Rozsáhlá knihovna* - obsahuje třídy, které nahrazují nízko úrovňové programování
- *Vývojové oblasti a technologie* - zahrnuje knihovny pro specifické oblasti, jako je např. ASP.NET pro web, ADO.NET pro přístup k datům atd.
- *Přechodný kód (CIL)* - díky tomuto kódu je možné aplikace pro .Net Framework vyvíjet ve více jazycích, jako jsou Visual Basic, C#, F# a C++. Před kompilací jsou tyto kódy nejdříve kompilovány do přechodného kódu CLI a až poté se provede kompilace z přechodného kódu. Díky této funkcionalitě je možné kód aplikace psát v uvedených jazycích a ve výsledku může každý vývojář použít svůj preferovaný jazyk.
- *Kompatibilita* - až na některé výjimky aplikace vyvinuté na starších verzích lze bez problému spustit na novějších verzích,
- *Podpora pro více cílových zařízení* - pomocí knihovny Portable Class Library je možné vyvíjet knihovny a aplikace pro různé platformy, jako jsou. Net Framework, Silverlight, Windows Phone 7 nebo Xbox 360.Visual C# 2010 Express.

7.1.2 Visual C# 2010 Express

Visual C# je vývojové prostředí pro vývoj aplikaci založené na platformě .Net Framework. Toto prostředí umožnuje jednoduché, rychlé, typové a objektové programovaní aplikací (viz [6]).

7.1.3 NUnit 2.6.2

NUnit je testovací knihovna pro všechny dostupné jazyky .Net Framework. Základní myšlenkou pro použití testovacího knihovny je testování malých částí systému. Jednotlivé části můžeme chápat jako moduly systému. Moduly jsou navzájem izolované a vytváří jednotlivé části systému, které jsou zapouzdřeny a komunikují spolu pouze prostřednictvím předem specifikovaného rozhraní (viz [7]).

Použití Unit testování se skládá ze dvou základních částí. První je produkční kód, který provádí určitou funkcionalitu a ten je třeba otestovat. Druhou části unit testování je ovladač, který tyto malé funkční jednotky volá a porovnává skutečný s očekávaným výsledkem. Použitím testovacích knihoven roste časová náročnost při vyvíjení systému. Proto je někdy dobré zvážit, co je skutečně nutné v daném případě testovat. Z pravidla vypuštění některých testu bývá častou chybou. Pokud se jedná o rozsáhlý a složitý projekt prvotní časová investice se rychlé vrátí. Unit testy nám také umožňují vytvořit automatický proces testování a zamezit tak opakujícím se chybám. Již napsané testy zůstávají a kontrolují dosavadní funkcionalitu systému. Tím máme zaručenou menší možnost rozbití stávající funkcionality.

Úroveň testování je možné měřit pomocí Test Coverage neboli testového pokrytí jednotlivých modulů. Hodnota pokrytí je vyjádřena v procentech. Pro výpočet používáme nástroje, které jsou často distribuovány s testovací knihovnou.

7.1.4 Moq 4.0.10827

Moq je jedna z nejjednodušších a nejpoužívanějších knihoven pro mokovaní rozhraní a abstraktních tříd, které reprezentují jednotlivé moduly nebo části systému. Mokováním je myšleno obalení a nastavení jednotlivých metod rozhraní nebo abstraktní třídy. Při Unit testování často narážíme na problém neimplementovaného rozhraní, které pro daný test potřebujeme pro ověření funkčnosti našeho kódu. První intuitivní možností by bylo, vytvářet objekty jen pro testovací účely, ale to by bylo velmi neefektivní a časové náročné. Později by se touto metodou stal kód velice nepřehledný a zbytečně rozšířený o třídy, které nemají žádný význam pro výslednou funkčnost implementovaných modulů. Preferovanější a použitelnější možností v moderním programování, které klade důraz na kvalitu čistého a přehledného kódu, je použití zmiňované mokovací knihovny nebo i jiné. Hlavní výhodou mokovací knihovny je snadné a přímočaré použití, které nevyžaduje žádné speciální dovednosti (viz [8]).

7.2 Použité návrhové vzory

Použití návrhových vzorů je cestou k dobrému návrhu aplikace a jsou řešením často se vyskytujících problémů, které jsou společné pro všechny aplikace. Návrhové vzory jsou nezávislé na konkrétním objektově orientovaného přístupu a je nutné také zmínit, že nejde o teoretické vzory, ale jsou sbírkou dlouholetých zkušeností programátorů. Z čehož plyne, že vychází z reálných zkušeností a vzory jsou ověřené a osvědčené.

Dodržením a použitím návrhových vzorů docílíme:

- jednoduchého funkčního mechanizmu, který je ověřený z hlediska funkčnosti,
- vytvoření společného názvosloví pro všechny programátory a architekty, kteří se podílejí na vývoji produktu,
- možnosti kombinování vzorů,
- opakovaného použití architektury, návrhu a implementačních rozhodnutí.

Jednotlivé návrhové vzory doporučují vztahy a interakci mezi více objekty, ale neukazují konkrétní implementaci. Návrhové vzoru můžeme dělit do skupin dle užití jednotlivých vzoru. Slouží k vytváření objektů, rozdělení struktury programu, specifikaci chování a pro řešení problému, které mohou vzniknout při běhu aplikace (viz [9]).

7.2.1 Factory

Návrhový vzor Factory je velmi jednoduchým vzorem pro vytváření instancí objektů, kdy jeden abstraktní objekt slouží k vytváření jiného abstraktního objektu, resp. jeho potomků. Použitím tohoto vzoru nejsme omezení předem specifikovat konkrétní objekt, ale můžeme zde využít dědičnost, kde se abstraktní objekt stává předkem. Pro tyto konkrétní objekty poté implementujeme příslušné objekty Factory, které disponují metodou vytvoř, resp. create (viz [10]).

Tento vzor byl využit v praktické části pro dynamické vytváření zásobníků v metodách 1D-BPP[4.1] a pro vytváření jednotlivých instancí TSP[5] algoritmu v části VRP[6], kde pro každý vytvořený region aplikujeme jeden z TSP[5] algoritmů.

7.2.2 Decorator

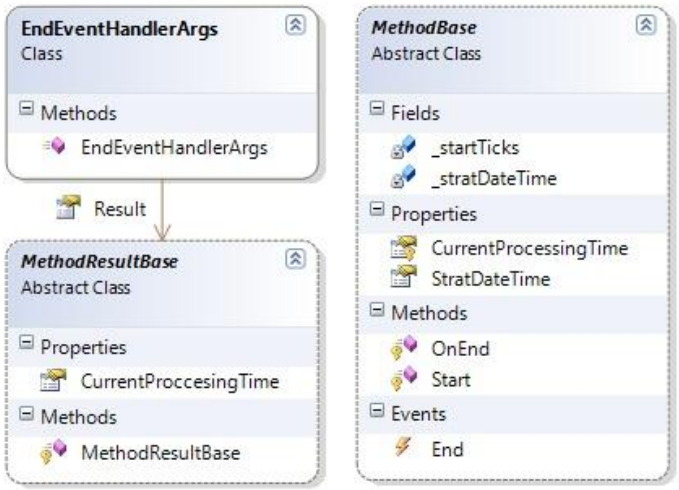
Návrhový vzor Decorator (ozdobení) lze použít v případě, že je potřeba některým objektům přidat dynamicky nové funkce. Na tento návrhový je znám taky pod názvem Wrapper (obal). Decorator tedy rozšiřuje (obaluje) objekt, nikoliv třídu, kde je využíváno klasické dědičnosti tříd. Pomocí tohoto vzoru tedy přidáváme a odebíráme povinnosti jednotlivým objektům (viz [11]).

Tento návrhový vzor byl použit u metod First-Fit Decreasing[4.1.3] a Best-Fit Decreasing[4.1.6], kde bylo nutné rozšířit objekty First-Fit[4.1.2], resp. Best-Fit[4.1.5] o metodu sestupného setřídění.

7.3 Knihovna obsahující heuristické algoritmy

Knihovna byla implementována tak, aby byly jednotlivé heuristické algoritmy oddělené od samotného uživatelského rozhraní. Uživatelské rozhraní muže tyto algoritmy využívat prostřednictví předem definovaných rozhraní a abstraktních tříd. Tímto byla dosažena nezávislost algoritmů od prezentační vrstvy a je tedy možné tyto algoritmy použít v libovolné aplikaci, popřípadě je dále rozšiřovat a optimalizovat.

Na následujícím obrázku (Obrázek 17) je uveden diagram tříd, které jsou společné pro všechny naimplementované heuristické algoritmy.



Obrázek 17 - Diagram základních a společných tříd v knihovně

Diagram obsahuje tyto tři následující třídy:

- *MethodBase* – tato abstraktní třída reprezentuje základní funkcionalitu pro heuristické metody. Obsahuje vlastnosti a metody spojené s aktuálním časem zpracování. Dále obsahuje událost *OnEnd*, pomocí niž je možné zaregistrovat vlastní funkce, které mají být zavolány po ukončení heuristické metody,
- *EndEventHandlerArgs* – třída obsahující *MethodResultBase*, který je poté dostupný při volání události *OnEnd*,
- *MethodResultBase* - abstraktní třída která reprezentuje výsledek heuristické metody, která disponuje aktuálním časem zpracování.

Knihovna byla rozdělena do čtyř následujících částí.

7.3.1 Část datové struktury

V této časti, jsou naimplementovány potřebné datové struktury, které jsou využity pro práci s daty jednotlivých heuristických algoritmů.

Jako strukturou uchovávající zásobníky u metod BPP byla zvolena struktura zřetězeného seznamu, kterou disponuje .Net Framework[7.1.1]. Pro optimalizaci těchto metod byla použita také datová struktura 2-3 Strom, která dosahuje menší složitosti při vyhledávání než zřetězený seznam.

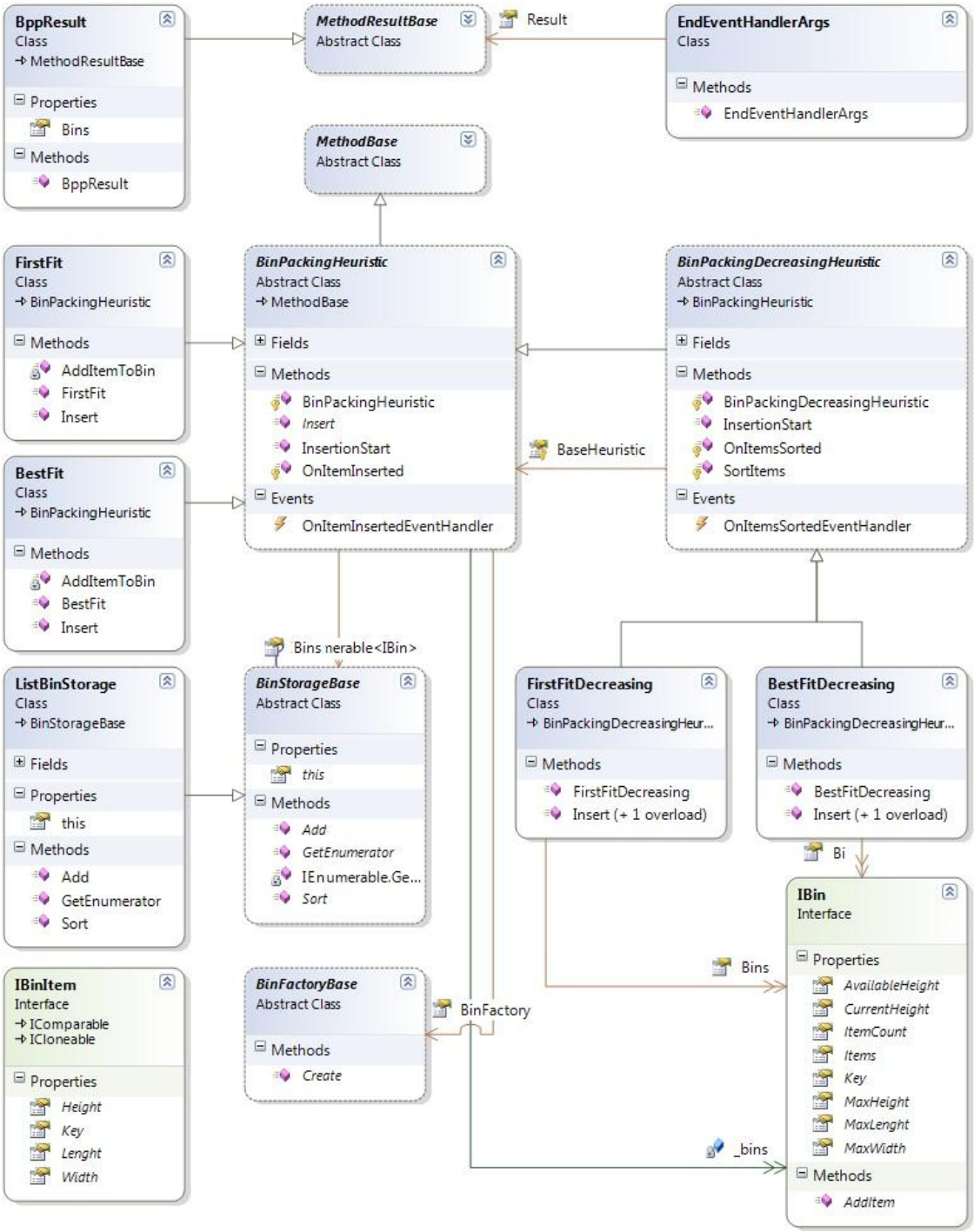
Pro uchování vrcholů grafu byla zvolena datová struktura graf typu tabulka (strom)- tabulka (seznam), kde strom je datová struktura dvojdimenzionální rozsahový strom. Tato stromová struktura byla zvolena pro efektivní jednoduché a intervalové vyhledávání vrcholů v dvourozměrném prostoru, které bylo využito pro výběr bodu.

7.3.2 Část BPP

V této části jsou naimplementovány třídy a rozhraní, které využívají heuristiky pro BPP[4]. Obsahuje tedy potřebné objekty a rozhraní pro metody First-Fit[4.1.2], First-Fit Decreasing[4.1.3], Best-Fit[4.1.5] a Best-Fit Decreasing[4.1.6].

Tato část využívá dědičnosti základních abstraktních tříd, které obsahují společné vlastnosti a metody pro všechny implementované metody.

Na následujícím obrázku (Obrázek 18) je uveden diagram tříd.



Obrázek 18 - Diagram tříd části BPP v naimplementované knihovně.

Diagram obsahuje tyto stěžejní třídy a rozhraní:

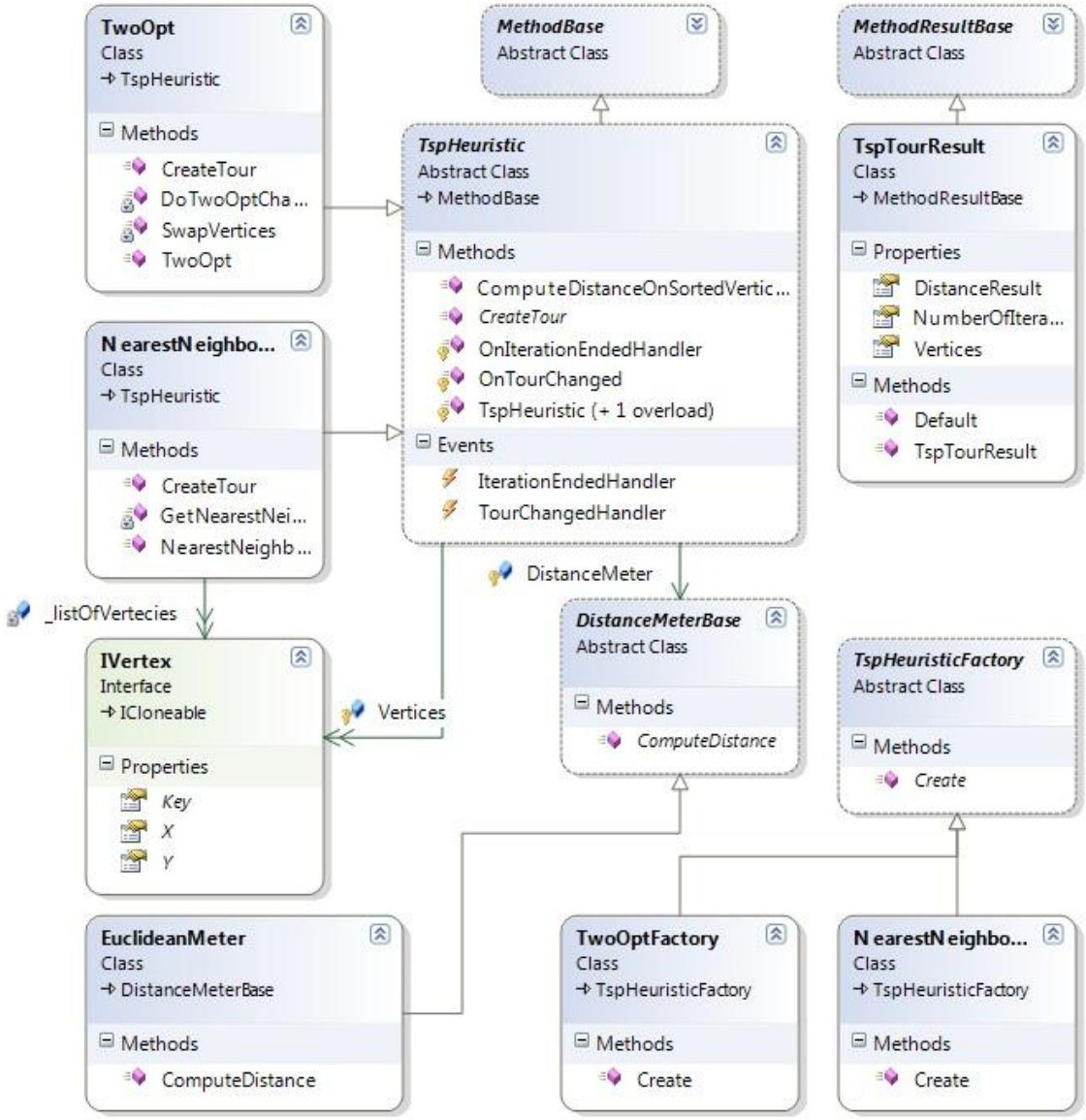
- *BppResult* – třída zapouzdřující výsledek heuristické metody, také je potomkem třídy *MethodResultBase*[7.3],
- *IBinItem* – rozhraní, které definuje vlastnosti položky, která může být vložena do zásobníku *IBin*,
- *IBin* – rozhraní definující vlastnosti a metody pro zásobník,
- *BinFactoryBase* – abstraktní třída, která slouží v jednotlivých algoritmech k vytváření prázdných zásobníků,
- *BinStorageBsae* – abstraktní třída, definující potřebné vlastnosti pro uložiště aktuálně použitých zásobníků,
- *ListBinStorage* – konkrétní implementace, která je založená na ukládání aktuálně využitých zásobníků v zřetězeném poli,
- *BinPackingHeuristic* – abstraktní třída, ve které jsou naimplementovány nebo definovány společné metody pro všechny heuristické algoritmy BPP[4.1],
- *FirstFit*- tato třída je potomkem třídy *BinPackingDecreasing* a v metodě Insert obsahuje implementaci algoritmus First-Fit[4.1.2],
- *BestFit* – tato třída je opět potomkem třídy *BinPackingDecreasing* a v metodě *Insert* implementuje algoritmus Best-Fit[4.1.5],
- *BinPackingDecreasingHeurisic* – u této třídy byl použit návrhový vzor Decorator[7.2.2], přičemž třída dekoruje přechozí třídu *BinPackingHeuristic* Tento návrhový vzor zde byl využit, protože u těchto metod je nejprve nutné vkládané položky seřadit dle jejich výšky. Proto disponuje metodou *SortItems*,
- *FirstFitDecreasing*- tato třída už je konkrétním dekorátorem metody First-Fit a potomkem třídy *BinPackingDecreasingHeurisic*. V metodě *Insert* jsou nejdříve položky sestupně setříděny a po té využívá metodu *Insert* dekorovaného algoritmu *First-Fit*. Tím je docílena implementace metody First-Fit Decreasing[4.1.3],
- *BestFitDecreasing* – tato třída je také konkrétním dekorátorem třídy *BinPackingDecreasingHeurisic*. V metodě *Insert* jsou nejdříve položky sestupně setříděny a po té využívá metodu *Insert* dekorovaného algoritmu Best-Fit. Tím je docílena implementace metody Best-Fit Decreasing[4.1.6].

7.3.3 Část TSP

Do této části byly zařazeny naimplementované třídy a rozhraní, které jsou potřebné pro funkčnost heuristických algoritmů při řešení TSP[5]. Konkrétně byly pro implementaci vybrány metody Nearest Neighbour[5.2.1] a 2 – opt[5.3.2].

Tato část také využívá dědičnosti základních abstraktních tříd, které obsahují společné vlastnosti a metody pro všechny implementované metody.

Na následujícím obrázku (Obrázek 19) je uveden diagram tříd pro tuto část knihovny.



Obrázek 19 - Diagram tříd částí Tsp v naimplementované knihovně.

Zobrazený diagram tříd obsahuje tyto třídy:

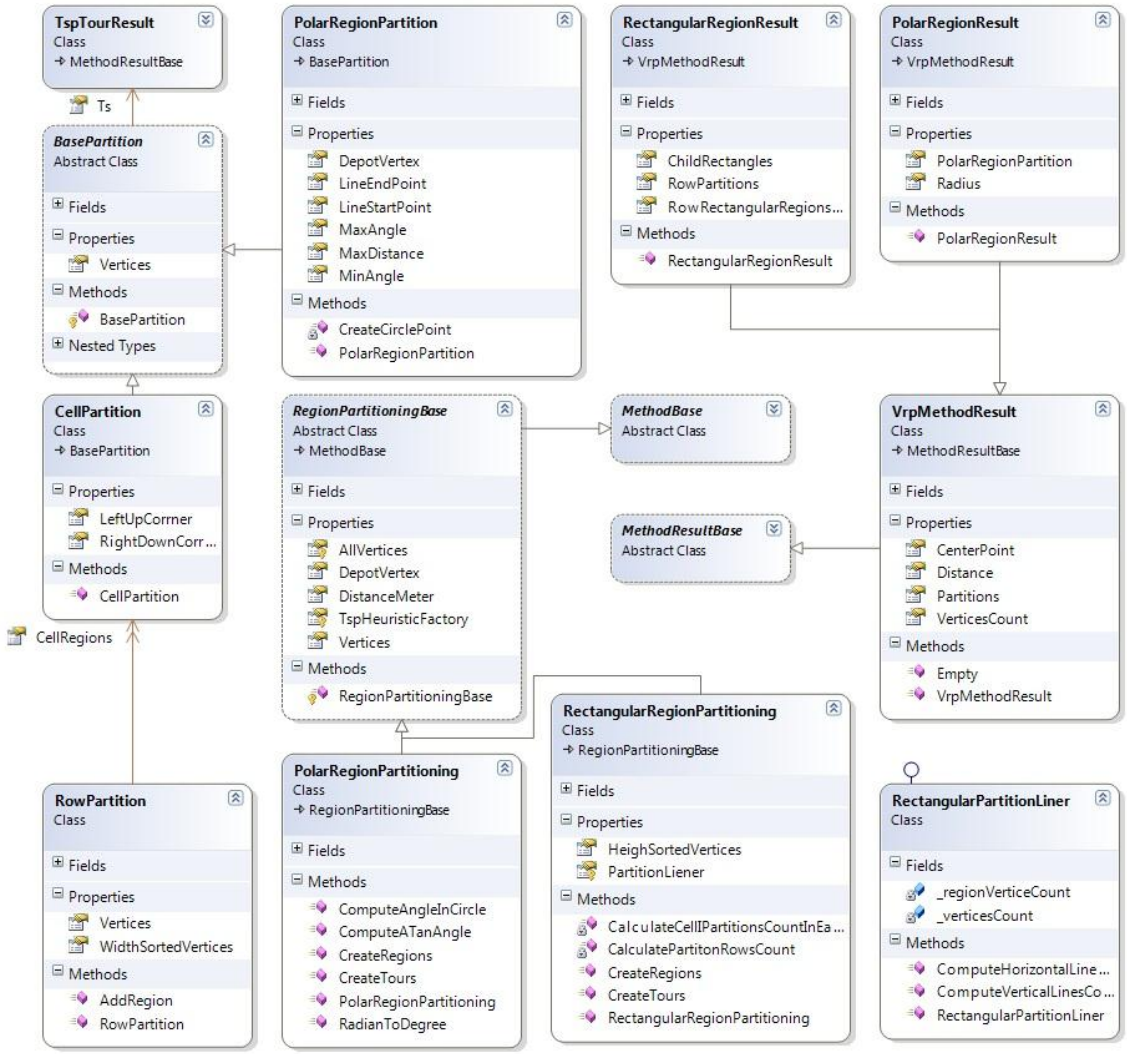
- *TspTourResult* – tato třída zapouzdřuje výsledek použitého heuristického algoritmu z problematiky TSP,
- *IVertex* – rozhraní definující vrchol grafu s kterými poté pracují heuristiky TSP,
- *DistanceMeterBase* – tato abstraktní třída definuje a reprezentuje prostředek pro zjištění délky mezi vrcholy,
- *EuclideanMeter* – tato třída implementuje definované metody z abstraktní třídy DistanceMeterBase a vzdálenost je počítaná v Euklidovým prostoru, který tvoří dvourozměrná rovina,

- *TspHeuristic* – tato abstraktní třída je potomkem obecné třídy *MethodBase*[7.3]. Definuje zejména metodu *CreateTour* a události, na které je možné zaregistrovat metody, které budou při dané události zavolány s předanými parametry,
- *NearestNeighbor* – třída implementující heuristický algoritmus Nearest Neighbor[5.2.1],
- *TwoOpt* – tato třída implemntuje konkrétní heuristický algoritmus 2-optimalizace[5.3.2],
- *TspHeuristicFactory* – abstraktní třída představující návrhový vzor Factory[7.2.1] pro vytváření instancí TSP heuristik,
- *TwoOptHeuristic,NearestNeighborFactory* – tyto třídy jsou potomkem abstraktní třídy *TspHeuristicFactory*, které slouží pro vytváření konkrétní instance TSP heuristiky *TwoOpt*, resp. *NearestNeighbor*.

7.3.4 Část VRP

V této části se nachází heuristické algoritmy pro řešení VRP[6]. Konkrétně obsahuje metody Polar Region Partitioning[6.2] a Rectangular Partitioning[6.3].

Na následujícím obrázku (Obrázek 20) je uveden diagram tříd pro tuto část knihovny.



Obrázek 20 - Diagram tříd částí Vrp v naimplementované knihovně

Zobrazený diagram tříd obsahuje tyto třídy:

- *RegionPartitioningBase* – je asbtraktní třída definující a obsahující základní metody pro VRP heuristiky a je potomkem obecné abstraktní třídy *MethodBase*[7.3],
- *BasePartition* – abstraktní třída, která obsahuje společné vlastnosti a metody jednotlivých regionů, které jsou vytvářeny VRP algoritmy,
- *VrpMethodResult* – je abstraktní třída, která je potomkem obecné třídy *MethodResultBase*[7.3] a rozšiřuje ji o společné vlastnosti výsledků VRP heuristik,
- *RectangularRegionPartitioning* – je třídou, která disponuje vlastnostmi pro výsledek algoritmu Rectangular Region Parttioning[6.3] Rectangular Region Parttioning[6.3]a je potomkem abstraktní třídy *VrpMethodResult*,
- *CellPartition* – třida, která reprezentuje konečný region vytvořený heuristickým algoritmem Rectangular Region Parttioning[6.3] a je potomkem *BasePartition*,

- *RowPartition* – tato třída zapouzdřuje pomyslný řádkový region, který obsahuje jednotlivé buňky ležící v jednom řádku,
- *RectangularPartitionLiner* – tato třída obsahuje pomocné metody pro výpočet počtu vertikálních a horizontálních čar pro metodu Rectangular Region Partitioning[6.2],
- *PolarRegionPartitioning* – v této třídě je naimplementován algoritmus Polar Region Parttioning[6.2] a jeho pomocné metody. Tato třída také implementuje abstraktní bázovou třídu *RegionBasePartitioning*,
- *PolarRegionPartition* – třída obsahující vlastnosti pro kruhovou výseč metody Polar Region Parttioning[6.2] a je potomkem abstraktní třídy *BasePartition*

7.4 Desktopová aplikace

Desktopová aplikace byla vytvořena pro prezentaci výsledků vybraných heuristik a algoritmů. Aplikace je založená na technologii WinForms, kterou obsahuje platforma .Net Framework.

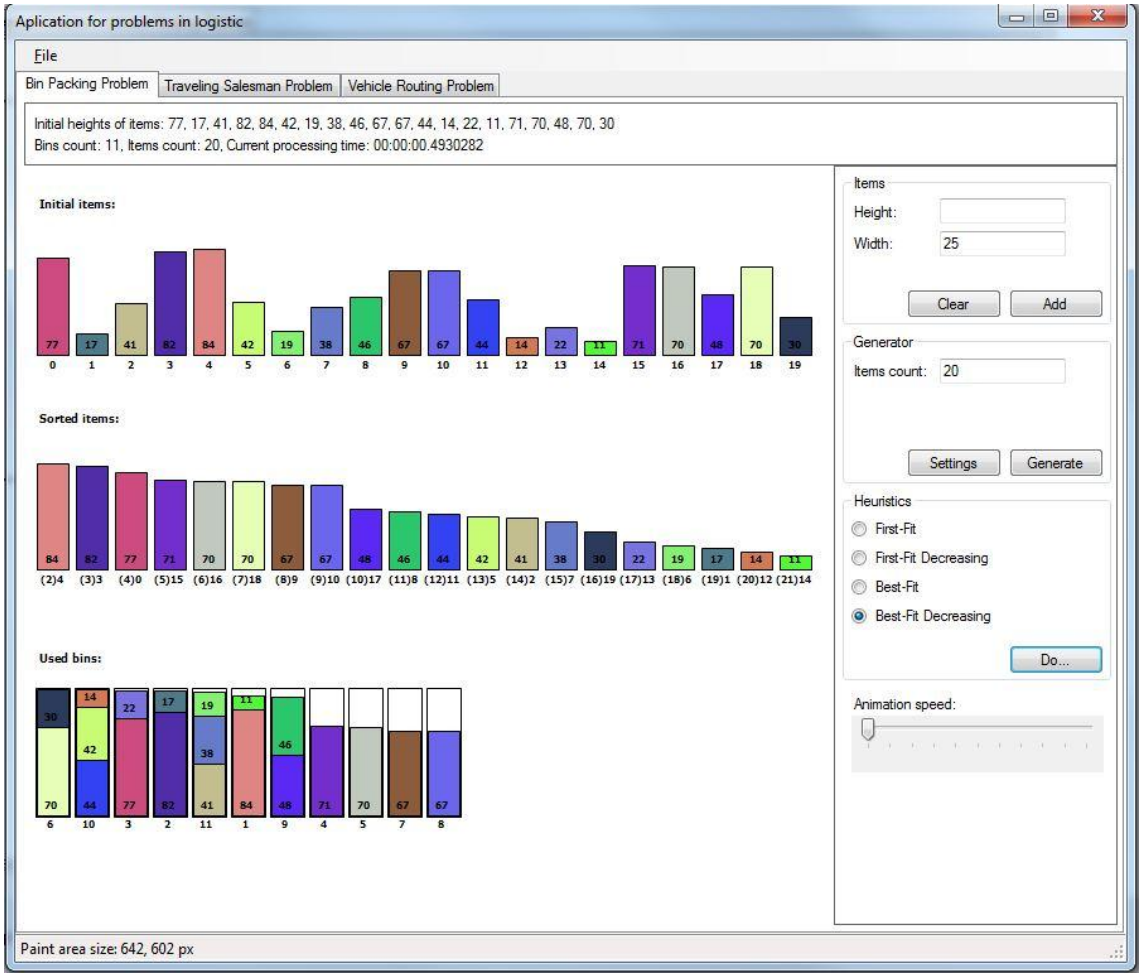
Aplikace byla rozdělena do třech hlavních záložek. Každá záložka obsahuje potřebné ovládací prvky tak, aby bylo možné demonstrovat funkčnost jednotlivých heuristik a algoritmů. Aplikace obsahuje také kontextové menu, které je pro všechny tři části společné. Menu obsahuje otevření a uložení vstupních dat a výsledků, které byly vytvořeny aktuálně zvolenou heuristikou.

Desktopová aplikace využívá knihovnu[7.3], která obsahuje implementaci jednotlivých heuristik. Byly využity odpovídající abstraktní třídy a rozhraní, aby bylo následně možné využít dostupné metody a zobrazit jejich výsledky.

7.4.1 Záložka BPP

Tato první a výchozí záložka po spuštění aplikace obsahuje dostupné metody pro řešení BPP[4].

Na následujícím obrázku (Obrázek 21) je vidět rozložení této záložky.



Obrázek 21 - Záložka pro BPP v desktopové aplikaci.

V horním panelu se nachází výpis výšek položek, které jsou voženy do fronty, ke zpracování. Dále obsahuje na druhém řádku počet použitých zásobníků, počet položek a aktuální čas zpracování, který se aktualizuje při každé iteraci vybraného heuristické algoritmu v pravém panelu.

Pravý panel tvoří čtyři skupiny ovládacích prvku. První skupina ovládacích prvku obsahuje zadávací textové pole pro hodnoty výšky a šířky vkládané položky, ale je nutné dodržet stejnou šířku položek a proto byla implementována validace pro vkládání hodnoty. Zároveň byly vytvořeny numerické validace tak, aby bylo předcházeno špatnému zadání hodnot. Tato skupina obsahuje také tlačítka smazatt a přidat položky.

Skupina ovládacích prvků pro generování položek obsahuje textové pole pro zadání množství generovaných položek. Dále obsahuje tlačítko nastavení generátoru a tlačítko generovat. V nastavení generátoru je možné nastavit specifické výšky položek a jejich četnost ve výsledné generované řadě položek. Tlačítko generovat spouští samotné generování položek, které se zobrazí na levé části obrazovky, která slouží k vykreslování vstupní fronty položek a výsledků jednotlivých heuristik.

Skupina heuristiky obsahuje přepínače pro zvolení BPP heuristiky, která bude použita pro výpočet potřebných zásobníků na vstupní frontu položek. Dále také obsahuje tlačítko pro spuštění vybrané heuristiky.

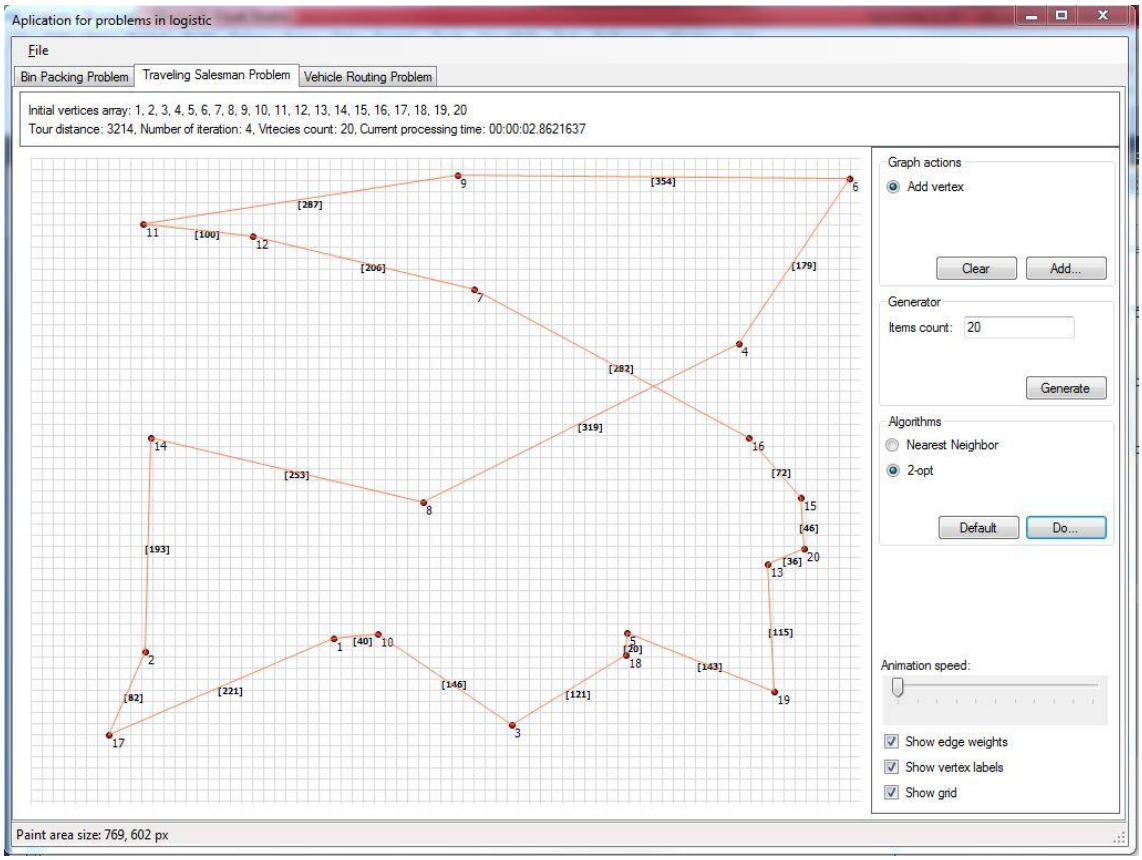
V dolní části pravého panelu je možné pomocí posuvníku zpomalit či zrychlit čas animace a je tak tedy možné kontrolovat přechodný stav mezi jednotlivými iteracemi vybraného algoritmu.

Levá část slouží jako panel pro vykreslení aktuálního stavu. U metod First-Fit a Best-Fit obsahuje dvě sekce a to položky ve frontě a položky v zobrazené v jednotlivých zásobnících. V případě, že je použita jedna z heuristik First-Fit Decreasing nebo Best-Fit Decreasing, tak je v prostřední části zobrazen mezi výsledek položek po sestupném seřazení vstupních položek ve frontě.

7.4.2 Záložka TSP

Záložka TSP obsahuje potřebné ovládací prvky pro demonstraci vybraných heuristik pro řešení TSP[5]. Skládá se ze tří panelů, které děli ovládací prvky dle příslušné funkcionality.

Na následujícím obrázku (Obrázek 22) je vidět rozložení této záložky.



Obrázek 22 - Záložka pro TSP v desktopové aplikaci.

V horním panelu se nachází klíče všech aktuálně vložených vrcholů a informace o aktuální trase. Údaje o aktuální trase obsahují délku trasy, počet iterací vybraného algoritmu, počet vrcholů v grafu a čas zpracování.

Levá část záložky slouží k zobrazení aktuálních dat.

Pravá část slouží jako panel pro kontrolní prvky, které jsou roztříděny dle své funkcionality do skupin.

Skupina akce obsahuje přepínače pro akci nad grafem, tlačítko vyčistit a vložit. Tlačítko vyčistit smaže všechny vstupní data, které jsou zobrazeny v levém panelu. Tlačítko vložit vyvolá dialogové okno pro vložení vrcholu do grafu.

Skupina algoritmy obsahuje přepínače pro vybrání algoritmu. Zde jsou na výběr dva algoritmy Nearest Neighbor a 2-opt. Dále skupina obsahuje tlačítka výchozí a spustit. Tlačítko výchozí vynuluje již optimalizovanou trasu, která byla zobrazena pro vybraný algoritmus a zobrazí výchozí stav grafu. Tlačítko názvem spustit spouští vybraný algoritmus.

Skupina generátor obsahuje dva prvky. Prvním je textové pole, kde je možné vložit počet prvků, které mají být vygenerovány. Druhým je tlačítko generuj, které spouští vygenerování zadaného množství vrcholů. Vrcholy jsou generovány k stávajícím datům, které jsou zobrazeny v levé části.

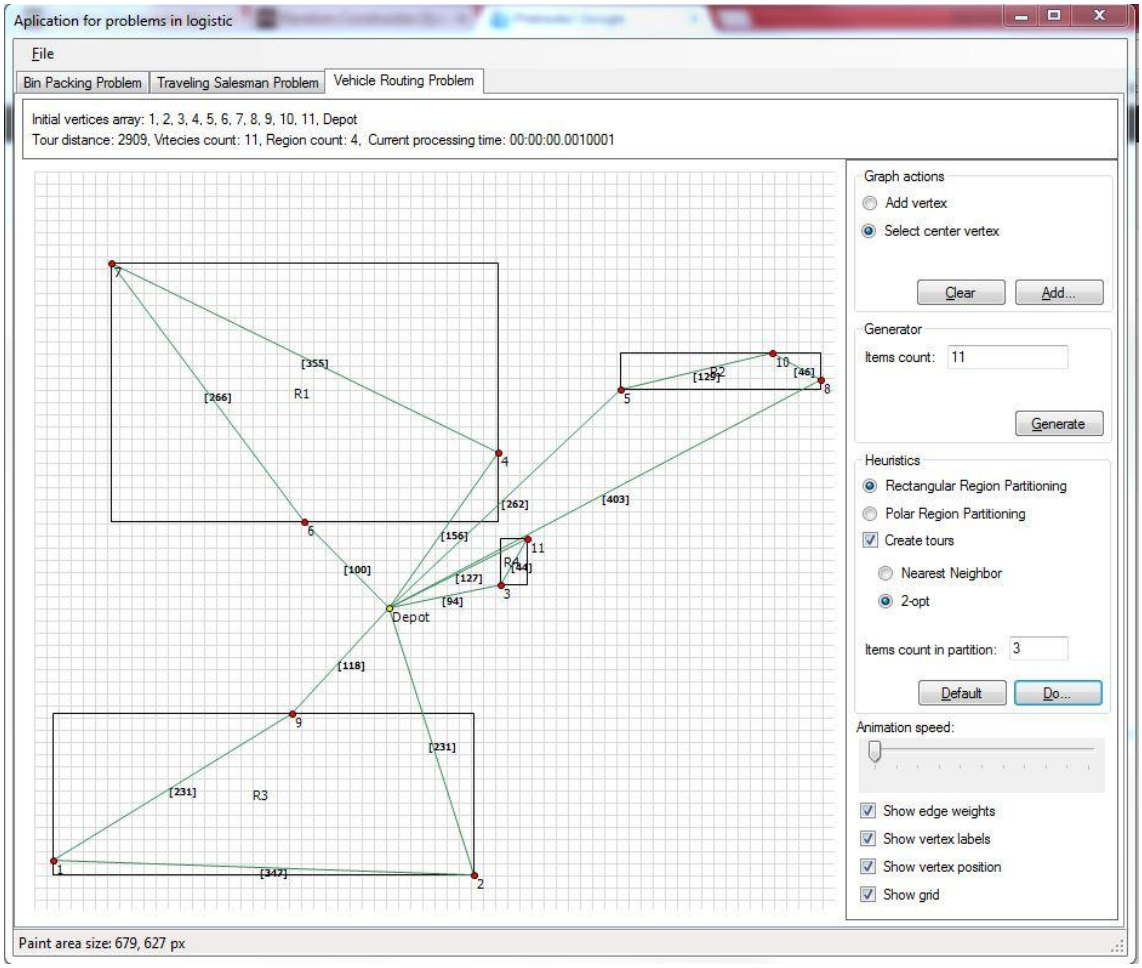
Pod předešlými skupinami je umístěn posuvník, kterým je možné nastavit rychlost animace. Rychlostí animace je myšleno, za jaký časový interval má být vypočítán následující krok algoritmu tak, aby bylo možné pozorovat jednotlivé kroky vybraného algoritmu.

Poslední skupinou jsou zaškrtávátka, která ovlivňují zobrazení v levé části záložky. Zobrazit či vypnout je možné klíče vrcholů, váhu hran a mřížku v pozadí.

7.4.3 Záložka VRP

Záložka VRP disponuje ovládacími prvky pro demonstraci řešení problému VRP[6] pomocí vybraných heuristik.

Na následujícím obrázku (Obrázek 23) je vidět rozložení této záložky.



Obrázek 23 - Záložka pro VRP v desktopové aplikaci

Jak je vidět tato záložka částečně vychází ze záložky TSP a je rozdělena opět do třech základních částí.

V horním panelu nalezneme základní informace k vytvořenému grafu, který je zobrazen v levé části obrazovky na panelu, kde je vykreslován aktuální stav grafu popřípadě výsledek vybraného heuristického algoritmu.

V pravé části nalezneme ovládací prvky pro vytvoření grafu, vybrání centrálního bodu, z kterého jsou vytvářeny cesty pro jednotlivé regiony a generátoru vrcholů v grafu.

Skupina heuristiky obsahuje ovládací prvky pro výběr metody VRP a dostupných algoritmů pro řešení TSP. V této skupině se také nachází textové pole pro zadání počtu regionů.

Poslední skupinou je posunovátko a zaškrtávátka, které nám umožní ovlivnit zobrazený výstup vybrané heuristiky v levé části.

8 Porovnání metod

Pro následující testování byl použit stolní počítač, který disponuje těmito základními vlastnostmi:

- procesor – Intel(R) Core(TM)2 Quad CPU Q9300, 2.50GHz,
- operační paměť 8GB,
- typ operačního systému – Windows 7 64bt.

Pro následující experimenty s naimplementovanými metodami byly vytvořeny jednotlivé testy pomocí knihovny NUnit[7.1.3] a je tak tedy možné tyto experimenty automaticky opakovat. Vstupní data jednotlivých testů je také možné zpětně zobrazit v desktopové aplikaci a manuálně experimenty zopakovat.

8.1 BPP

Pro ověření a porovnání heuristických algoritmů v oblasti 1D-BPP[4.1] byly provedeny následující experimenty.

V prvním experimentu bylo postupně vygenerováno 50, 100, 200, 300, 500 a 1000 položek, které mohou nabývat výšky v rozmezí 1 – 100 jednotek. Jednotlivé zásobníky disponují maximální výškou 100 jednotek. Následně byly použity metody pro FF, FFD, BF, BFD. Pro každý počet byl experiment proveden 100 krát a následně byla vypočítána střední hodnota použitých zásobníků a počet zcela zaplněných zásobníků.

Následně byl vytvořen souhrn dosažených hodnot a tyto hodnoty jsou zobrazeny v následující tabulce (Tabulka 1).

Tabulka 1 - Souhrn výsledků 1. experimentu při porovnání BPP heuristik.

Počet položek	Celkový počet použitých zásobníků*				Počet zaplněných zásobníků*			
	FF	FFD	BF	BFD	FF	FFD	BF	BFD
5	3	3	3	3	0	0	0	0
10	6	6	6	5	0	0	0	0
50	28	26	28	27	2	8	4	10
100	55	53	54	53	6	25	13	26
200	108	104	106	103	18	63	34	65
300	162	154	159	154	30	106	59	110
500	265	255	262	253	61	191	114	198
1000	527	506	520	507	148	418	267	424

* průměrné hodnoty počítané ze 100 iterací

Z uvedených hodnot v tabulce vyplývají tyto závěry:

- s rostoucím potem položek se jeví jako efektivnější metody FFD a BFD, jak v celkovém počtu zásobníků, tak v počtu zcela zaplněných zásobníků,
- v počtu zcela zaplněných zásobníků metody FFD a BFD výrazně převyšují metody FF a BF. Ovšem je potřeba zmínit, že před vkládáním je třeba položky setřídit dle jejich výšky. Z této skutečnosti plyne, že jsou z hlediska času náročnější na zpracování. V praxi je třeba zvážit cenu ušetřeného zásobníku a cenu setřídění. Cílem je tedy zvolit kompromis dle celkových nákladů vzhledem k reálné situaci.

V druhém experimentu byly požity stejné výchozí podmínky jako v prvním experimentu. Položky byly pouze vygenerovány s nabývající velikostí 25, 50, 75 a 100 jednotek, tedy ¼ násobcích maximální výšky zásobníku, s četností výskytu položek 40%, 25%, 25% a 10%. V praxi se setkáme spíše s tímto typem úlohy, kde velikost položek odpovídá normovaným velikostem, které jsou uváděny v násobcích zásobníků. V reálném případě se může jednat o skladovací prostory palet v logistickém centru.

V následující tabulce (Tabulka 2) jsou zobrazeny souhrnné výsledky tohoto experimentu.

Tabulka 2 - Souhrn výsledků 2. experimentu při porovnání BPP heuristik.

Počet položek	Celkový počet použitých zásobníků*				Počet zaplněných zásobníků*			
	FF	FFD	BF	BFD	FF	FFD	BF	BFD
5	3	2	3	2	1	1	1	1
10	5	5	5	5	3	4	3	4
50	26	26	26	26	23	25	24	25
100	52	52	51	51	49	51	49	50
200	102	102	103	102	99	102	101	102
300	154	153	154	154	151	153	151	153
500	256	256	257	256	253	255	254	255
1000	513	512	512	513	510	511	510	512

* průměrné hodnoty počítané ze 100 iterací

Z uvedených hodnot v tabulce vyplývají tyto závěry:

- v případě že použijeme toto rozdělení pro četnosti položek, dle výšky, je zřejmé, že se mažou rozdíly mezi metodami FF a FFD, resp. BF a BFD,
- v tomto případě by bylo výhodnější využít metod FF a BF, které disponují menší náročností na výpočet.

Výsledky pozorování jsou přiloženy na CD - *(CD://Vysledky_testů/BPP)*.

8.2 TSP

Pro ověření a porovnání heuristických algoritmů v oblasti TSP[5] byl proveden následující experiment.

Připomeňme, že obecným cílem metod řešících TSP[5] je nalezení co nejkratší možné cesty za těchto pravidel:

- cílem cestujícího je navštívit všechna města pouze jednou,
- cestující se po navštívení všech měst vrátí zpět do města, z kterého vyjel.

Pro následné pozorování bylo postupně vygenerováno 10, 20, 30, 40, 50, 100 a 200 vrcholů, resp. měst. Na každou množinu vrcholů byla aplikována metoda Neareast Neighbor[5.2.1] a 2-opt[5.3.2]. Výsledky těchto metod byly uloženy do formátu CSV. Následně byly výsledky zpracované do tabulky (Tabulka 3), ve které je vidět výsledná délka cesty, počet iterací a čas potřebný pro zpracování konkrétní metody.

Tabulka 3 - Souhrn výsledků při porovnání TSP heuristik.

Počet vrcholů	NN Délka cesty (km)	NN počet iterací	NN čas zpracování (hh:mm:ss)	2-opt délka cesty (km)	2-opt počet iterací	2-opt čas zpracování (hh:mm:ss)	Rozdíl délky cesty	Rozdíl počet iterací	Rozdíl času zpracování (hh:mm:ss)
10	3931,16	10	0:00:00	2484,43	243	00:00:00.0070004	-1446,73	233	00:00:00.0070004
20	9883,40	20	00:00:00.0010001	4213,35	1444	00:00:00.0630036	-5670,05	1424	00:00:00.0620035
30	14511,38	30	00:00:00.0010000	7246,37	4205	00:00:00.2540145	-7265,01	4175	00:00:00.2530145
40	20552,31	40	00:00:00.0030001	8164,04	6084	00:00:00.4780273	-12388,27	6044	00:00:00.4750272
50	27496,50	50	00:00:00.0020001	7338,60	12005	00:00:01.1320647	-20157,89	11955	00:00:01.1300646
100	49488,22	100	00:00:00.0200012	12263,45	49005	00:00:08.7455002	-37224,77	48905	00:00:08.7255090
200	103305,73	200	00:00:00.0400023	23184,22	316808	00:01:50.4523175	-80121,51	316608	00:01:50.4123152

Z uvedených hodnot v tabulce vyplývají tyto závěry:

- se zvyšujícím se počtem vrcholů je v porovnání celkové délky, kterou cestující urazí výhodnější použít metodu 2-opt, za předpokladu, že není kladen důraz na dobu výpočtu,
- se zvyšujícím se počtem vrcholů rapidně narůstá výpočetní doba trasy pro metodu 2-opt, která roste společně s provedenými iteracemi algoritmu,
- se zvyšujícím se počtem vrcholů se stává metoda NN nepoužitelnou pro výpočet délky, kdy přesahuje až čtyřnásobek vypočítané trasy metodou 2-opt.

Výsledky pozorování jsou přiloženy na CD - *(CD://Vysledky_testů/TSP)*.

8.3 VRP

Pro o testování metod VRP[6] byl vybrán praktický příklad rozvozu pečiva z pekárny do regionů, kde uvažujeme, že v každém regionu může být umístěn stejný počet pekáren.

Našim cílem je tedy jednotlivé pekárny rozdělit do regionů a vypočítat náklady spojené s rozvozem pečiva z pekárny.

Pro experiment je uvažována rozloha o velikosti 10 000 km^2. Na této rozloze bylo vygenerováno v jednotlivých krocích 10, 20, 30, 40, 50, 60 vrcholů, resp. pekáren. Vrchol, v kterém sídlí pekárna, byl uvažován jako těžiště vygenerovaných bodů.

Následně byly aplikovány metody PRP[6.2] a RRP[6.3]. Pro jednotlivé regiony byl poté aplikován algoritmus 2-opt[5.3.2]. Výsledky těchto metod byly porovnány v následující tabulce (Tabulka 4).

Tabulka 4 - Souhrn výsledků při pozorování VRP heuristik.

Počet vrcholů	Počet vrcholů v regionu	PRP počet regionů	PRP délka cesty (km)*	PRP čas zpracování (hh:mm:ss)*	RRP počet regionů	RRP délka cesty (km)*	RRP čas zpracování (hh:mm:ss)*	Rozdíl délky cesty (PRP-RRR)*	Rozdíl času zpracování (PRP-RRR)
10	5	2	350,64	00:00:00.0208311	2	369,20	00:00:00.0035602	-18,57	00:00:00.0172709
20	5	4	565,25	00:00:00.0843748	4	576,96	00:00:00.0058603	-11,71	00:00:00.0785145
30	5	6	767,60	00:00:00.2204326	6	786,20	00:00:00.0092805	-18,59	00:00:00.2111521
40	5	8	972,29	00:00:00.3756314	8	995,87	00:00:00.0099605	-23,58	00:00:00.3656709
50	5	10	1171,89	00:00:00.6050146	10	1200,15	00:00:00.0124507	-28,26	00:00:00.5925639
60	5	12	1373,43	00:00:00.8888108	12	1371,05	00:00:00.0150808	2,39	00:00:00.8737300
100	5	20	2197,62	00:00:02.6299804	20	2118,15	00:00:00.0243313	79,47	00:00:02.6056491
150	5	30	3215,29	00:00:06.2759989	30	2999,88	00:00:00.0360020	215,41	00:00:06.2399969
200	5	40	4238,07	00:00:11.7037794	40	3882,69	00:00:00.0477227	355,37	00:00:11.6560567

* hodnota zahrnuje součet pro všechny regiony

Z uvedených hodnot v tabulce vyplývají tyto závěry:

- metoda PRP, oproti metodě RRP, je z hlediska času zpracování výrazně pomalejší a s rostoucím počtem vrcholů a tím i počet regionů, čas zpracování výrazně narůstá,
- délka výsledné cesty pro všechny regiony se zpočátku, pro počet vrcholů 10 - 50, jeví metoda PRP výhodnější. Ovšem se stoupajícím počtem vrcholů se stává efektivnější na výpočet celkové délky metoda RRP.

Na základě předchozích výsledků se celkově jeví jako výhodnější metoda RRP, jak z vypočtené celkové délky, tak času zpracování. Především s narůstajícími počty

vrcholů, resp. regionů. Metoda PRP se jeví jako použitelná pouze pro malý počet vrcholů (v našem případě do 50, resp. 60 vrcholů).

Pozorované výsledky jsou přiloženy na CD - *(CD://Vysledky_testů/VRP).*

Závěr

Hlavním cílem této práce bylo prostudování vybrané problematiky v oblasti dopravní logistiky, jejichž řešení vede k NP-těžkým úlohám. Následně byly tyto vybrané metody naimplementovány a použity k demonstraci a otestování výsledků.

Nejdříve byla vytvořena teoretická část, která studuje a popisuje jednotlivé algoritmy. Po prostudování vybraných algoritmů, byla vytvořena aplikace. Prostřednictvím naprogramované aplikace je možné studovat jednotlivé algoritmy. Aplikace byla již od začátku tvořena tak, aby obsahovala kvalitní čistý a testovatelný kód. Jádrem aplikace je knihovna obsahující vybrané algoritmy, které je možné jednoduše dále rozšiřovat a použít v jiné aplikaci zabývající se obdobnou problematikou.

Praktická část může dále posloužit pro testovací nebo edukativní účely této rozsáhle problematiky. Při spojení naimplementované knihovny s mapovými portály by také mohla být vytvořena webová aplikace, která by posloužila jako vnitropodnikový systém zabývající se logistikou.

Na závěr byly naimplementované metody otestovány a porovnány dle dosažených výsledků. Vstupní a výstupní data těchto experimentů byly uloženy na přiložené CD.

Práce na tomto projektu pro mě byla přínosem a příležitostí rozšířit již nabyté dovednosti v tvorbě desktopových aplikací. Také při tvorbě teoretické části této práce jsem si ujasnil některé důležité pojmy této problematiky v oblasti dopravní logistiky.

Literatura

[1] **Simchi-Levi D., Bramel J., Chen X.:** The Logic of Logistics, Springer-Verlag, New York, (2004)

[2] **Garey M. R., Graham R. L., Johnson D.S., Yao A.:** Resource constrained scheduling as generalized bin packing, J.,Combinatorial Theory Ser. A, 21 (1976)

[3] **Baker B. S.:** A New Proof for the First-Fit Decreasing Bin Packing Algorithm. J. Algorithms 6 (1985)

[4] **The Three-Dimensional Bin Packing Problem:** [online] 2013 [cit. 1.8.2013] Dostupné na: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.47.2512&rep=rep1&type=pdf>

[5] **.Net Framework 4:** [online] 2013 [cit. 3.8.2013] Dostupné na: <http://msdn.microsoft.com/en-us/library/vstudio/w0x726c2(v=vs.100).aspx>

[6] **Microsoft Visual Studio 2012:** [online] 2013 [cit: 3.8.2013] Dostupné na: <http://www.microsoft.com/visualstudio/cze/products/visual-studio-2010-express>

[7] **NUnit -Home:** [online] 2007 [cit: 4.8.2013] Dostupné na: <http://www.nunit.org/>

[8] **Moq - The simplest mocking library for .NET and Silverlight - Google Project Hosting:** [online] 2013 [cit. 3.8.2013] Dostupné na: <https://code.google.com/p/moq/>

[9] **Design Patterns:** [online] 2013 [cit. 6.8.2013] Dostupné na: <http://msdn.microsoft.com/en-us/library/ff647483.aspx>

[10] **Exploring the Factory Design Pattern:** [online] 2013 [cit. 6.8.2013] Dostupné na: <http://msdn.microsoft.com/en-us/library/ee817667.aspx>

[11] **Decorator Design Pattern:** [online] 2013 [cit. 6.8.2013] Dostupné na: <http://sourcemaking.com/design_patterns/decorator>

Příloha A – Zdrojový kód metody First-Fit

```
using System;
using System.Linq;
using Logistics.Heuristics.Bpp.Bins;

namespace Logistics.Heuristics.Bpp.Methods
{
	public class FirstFit : BinPackingHeuristic
	{
		public FirstFit(BinFactoryBase binFactory)
			: base(binFactory)
		{
		}

		public override void Insert(IBinItem item)
		{
			if (!Bins.Any())
				Bins.Add(BinFactory.Create());

			if (item.Height > Bins.First().MaxHeight)
				throw new ArgumentException("Item height cannot
be higher then bin height.", "item");

			var currentBinIndex = 0;
			do
			{
				var currentBin = Bins[currentBinIndex++];
				var newHeight = currentBin.CurrentHeight +
item.Height;
				if (newHeight <= currentBin.MaxHeight)
				{
					AddItemToBin(item, currentBin);
					break;
				}

				if (Bins.Last().Equals(currentBin))
					Bins.Add(BinFactory.Create());

			} while (true);

OnEnd(new EndEventHandlerArgs(new BppResult(CurrentProcessingTime,
Bins)));
		}

		private void AddItemToBin(IBinItem item, IBin currentBin)
		{
			var beforeInsertHeight = currentBin.CurrentHeight;
			currentBin.AddItem(item);
			OnItemInserted(this, new
OnItemInsertedEventHandlerArgs(CurrentProcessingTime, Bins,
beforeInsertHeight, item, currentBin));
		}
	}
}
```

Příloha B – Zdrojový kód metody Best-Fit

```
using System.Collections.Generic;
using Logistics.Heuristics.Bpp.Bins;

namespace Logistics.Heuristics.Bpp.Methods.Decreasing
{
      public class FirstFitDecreasing : BinPackingDecreasingHeuristic
      {
            public FirstFitDecreasing(BinFactoryBase binFactory,
IDecreasingBinItemComparer comparer)
                  : base(binFactory, new FirstFit(binFactory),
comparer)
            {
                  BaseHeuristic.OnItemInsertedEventHandler +=
OnItemInserted;
            }

            public new BinStorageBase Bins
            {
                  get { return BaseHeuristic.Bins; }
            }

            public override void Insert(IEnumerable<IBinItem> items)
            {
                  InsertionStart();

                  var binItems = SortItems(items);

                  foreach (var binItem in binItems)
                        Insert(binItem);

                  OnEnd(new EndEventHandlerArgs(new
BppResult(CurrentProcessingTime, Bins)));
            }

            public override void Insert(IBinItem item)
            {
                  BaseHeuristic.Insert(item);
            }
      }
}
```

Příloha C – Zdrojový kód testů pro vypočet regionů v metodě RRP

```
using Logistics.Heuristics.Vrp.Rrp;
using NUnit.Framework;

namespace Logistics.Heuristics.Test.Vrp
{
	public class RectangularPartitionLinerTest
	{
		[TestCase(17, 3, 2)]
		[TestCase(4, 4, 0)]
		[TestCase(8, 4, 1)]
		[TestCase(9, 4, 1)]
		public void ComputesHorizntalCount(int verticesCount, int
regionVerticeCount, int expectedCount)
		{
			var liner = new
RectangularPartitionLiner(verticesCount, regionVerticeCount);

			var count = liner.ComputeHorizontalLinesCount();

			Assert.That(count, Is.EqualTo(expectedCount));
		}

		[TestCase(17, 3, 1)]
		[TestCase(4, 4, 0)]
		[TestCase(5, 4, 1)]
		public void ComputesVerticalCount(int verticesCount, int
regionVerticeCount, int expectedCount)
		{
			var liner = new
RectangularPartitionLiner(verticesCount, regionVerticeCount);

			var computeVerticalCount =
liner.ComputeVerticalLinesCount();

			Assert.That(computeVerticalCount,
Is.EqualTo(expectedCount));
		}
	}
}
```