UNIVERZITA PALACKÉHO V OLOMOUCI
PŘÍRODOVĚDECKÁ FAKULTA
KATEDRA MATEMATICKÉ ANALÝZY A APLIKACÍ MATEMATIKY

BAKALÁŘSKÁ PRÁCE

VRP Problém

Vedoucí bakalářské práce:
Mgr. Jaroslav Marek, Ph.D.
Rok odevzdání: 2010

Vypracoval:
Tomáš Talášek
MAP, III. ročník

Prohlášení

Prohlašuji, že jsem vytvořil tuto bakalářskou práci samostatně za vedení Mgr. Jaroslava Marka, Ph.D. a že jsem v seznamu použité literatury uvedl všechny zdroje použité při zpracování práce.

V Olomouci dne 2. dubna 2010

Poděkování

Rád bych na tomto místě poděkoval vedoucímu bakalářské práce Mgr. Jaroslavu Markovi, Ph.D. za vedení a spolupráci při tvorbě této bakalářské práce i za čas, který mi věnoval při konzultacích. Zároveň mu děkuji za předané zkušenosti a návrhy.

Obsah

1 Úvod

Dnešní doba je charakteristická obrovskou nabídkou zboží na trhu a proto každý, kdo se chce na trhu uchytit musí být konkurence schopný. Z tohoto důvodu se logistika dostává do popředí zájmu většiny společností.

V této bakalářské práci se budeme zabývat významnou logistickou úlohou, zvanou *Vehicle Routing Problem (VRP)*. Tato úloha si klade za cíl minimalizovat náklady spojené s přepravou zboží z firemního skladu k zákazníkům. I když se tento problém může jevit jako triviální záležitost, ve skutečnosti může jeho řešení přinést úspory v řádu několika procent.

Nyní si představíme jednotlivé kapitoly, na které v textu narazíme a nastíníme si nejdůležitější části této práce.

Nejprve se v kapitole *Logistika a NP-úplné úlohy* budeme zabývat pojmem logistika, řekneme si co jsou to NP–úplné problémy, ukážeme si co jsou to heuristické metody a jak lze určovat výkonnostní poměr jednotlivých heuristik.

Kapitola *Problém naplňování zásobníku* se zaobírá samotnou přípravou zboží předtím, než ho začneme distribuovat směrem k zákazníkům. Přípravou je zde myšleno roztřídění zboží do kontejnerů (případně krabic), ve kterých bude zboží převáženo.

Následuje kapitola *Problém obchodního cestujícího*, která má za cíl nalezení nejkratší cesty auta přes množinu zákazníků. Toto je jeden z nejznámějších problémů současné logistiky.

V poslední kapitole se budeme zabývat samotným *VRP* problémem. Zde bude naším cílem efektivně rozdělit naše zákazníky do oblastí, které budou následně obslouženy.

Všechny tyto problémy mají společnou tu vlastnost, že ani pomocí dnešních počítačů nelze nalézt jejich přesné řešení. Díky tomu je zde velký prostor pro hledání nových postupů, pomocí kterých by bylo dosaženo lepších výsledků. Tato práce byla napsána tak, aby umožňovala nahlédnout do oblasti logistických problémů i lidem, kteří se předtím o logistiku nezajímali.

2 Cíl práce

Cílem této bakalářské práce je prostudovat problematiku VRP problému a vytvořit programy, pomocí kterých lze tuto problematiku řešit.

V této práci se budeme zabývat třemi významnými problémy současné logistiky. Ke každému z těchto problémů bude třeba naprogramovat několik programů, pomocí kterých bude možné tyto problémy studovat, ověřovat naše předpoklady a zjišťovat nové skutečnosti. Je vhodné podotknout, že všechny tyto kapitoly jsou natolik komplexní, že by se mohly stát podkladem pro samostatnou bakalářskou práci.

V kapitole *Problém naplňování zásobníku* bude naším cílem popsat nejpoužívanější známé heuristiky a poté je porovnat jak z pohledu výkonnosti, tak z pohledu použitelnosti. K tomuto účelu bude potřeba naprogramovat software, který nám aplikuje všechny heuristiky a následně umožní tyto heuristiky porovnat. Navíc musíme zajistit grafický výstup pro přehlednější interpretaci výsledků.

Při studiu *Problému obchodního cestujícího* chceme popsat komplexnost celé úlohy, představit několik různých heuristických algoritmů pro jeho řešení a dodat software, pomocí kterého lze tuto úlohu řešit jak v čistě výpočetní podobě, tak i na skutečné mapě.

V kapitole *Vehicle routing problem* bude naším cílem najít heuristiky, pomocí kterých lze tento problém řešit a zpracovat je softwarově včetně grafického výstup. Následně budeme chtít tyto programy propojit s programy pro řešení obchodního cestujícího.

Posledním cílem bude pokusit se celou tuto bakalářskou práci vytvořit pomocí open source softwaru a následně popsat, jaké programy k tomu byly použity.

3 Logistika a NP-úplné úlohy

V této bakalářské práci se budeme zabývat různými problémy, na které můžeme v logistice narazit. Z tohoto důvodu by bylo vhodné si upřesnit, co se za pojmem logistiky skrývá. Následující definici logistiky používá Evropská logistická asociace.

Definice 3.1. *Organizace, plánování, řízení a výkon toků zboží vývojem a nákupem počínaje, výrobou a distribucí podle objednávky finálního zákazníka konče tak, aby byly splněny všechny požadavky trhu při minimálních nákladech a minimálních kapitálových výdajích.*

A takto vypadá nové pojetí logistiky dle British Institute of Logistics.

Definice 3.2. *Logistika je rozmístění zdrojů v čase, logistika je strategické řízení celého dodavatelského řetězce.*

Při „optimalizaci“ řešení logistických úloh zjišťujeme, že některé z nich vedou na takzvané *NP–úplné problémy*. Tyto úlohy jsou specifické tím, že sice jsme schopni projít všechna řešení a najít mezi nimi to nejlepší, nicméně časová náročnost je neúnosná. Pro zadefinování NP–úplných problémů si nejprve budeme muset zadefinovat *nedeterministicky polynomiální* (NP) problémy.

Definice 3.3. *NP je množina problémů, které lze řešit v polynomiálně omezeném čase na nedeterministickém Turingově stroji – na počítači, který umožňuje v každém kroku rozvětvit výpočet na n větví, v nichž se posléze řešení hledá současně.*

Definice 3.4. *NP-úplné (NP-complete, NPC) problémy jsou takové nedeterministicky polynomiální problémy, na které jsou polynomiálně redukovatelné všechny ostatní problémy z NP. To znamená, že třídu NP–úplných úloh tvoří v jistém smyslu ty nejtěžší úlohy z NP.*

Je zřejmé, že tento fakt nám způsobuje velké problémy při snaze nalézt řešení daných úloh a proto jsme nuceni užít speciální postupy – heuristiky.

Pojem heuristika lze přeložit jako „teorie řešení problémů“, případně jako „neobvyklé řešení“.

Definice 3.5. *Heuristika je postup získání řešení problému, které však není přesné a nemusí být nalezeno v krátkém čase. Nejčastěji slouží jako metoda rychle poskytující dostatečné a dosti přesné řešení, které však nelze obecně dokázat. Nejčastější použití heuristického algoritmu nalezneme v případech, kde není možné použít jiného lepšího algoritmu, poskytujícího přesné řešení s obecným důkazem.*

Z předešlé definice plyne, že i když nalezneme heuristiku, která funguje na velké množství úloh řešeného problému, nemusíme mít zaručeno, že nenarazíme na úlohu, na kterou nám naše heuristika dá řešení, které je velmi vzdálené od optimálního řešení. Z tohoto důvodu je po použití heuristické metody vždy vhodné se nad výsledkem zamyslet a tím se vyvarovat případným problémům. U některých heuristik lze určit horní závoru, čímž máme zaručeno, jak nejdál může být naše řešení vzdáleno od optimálního řešení.

Kromě horní závory se také užívá *absolutní výkonnostní poměr heuristiky*, který nám vyjadřuje, jak moc se naše heuristika vzdaluje od optimálního řešení.

Definice 3.6. *Uvažujme problém I. Nechť $Z^*(I)$ je hodnota cenového funkcionálu pro optimální řešení tohoto problému. Označme $Z^H(I)$ hodnotu cenového funkcionálu pro řešení získané heuristikou H. Absolutním výkonnostním poměrem heuristiky H (viz [1]) nazveme*

$$R^H = \inf\left\{r \geq 1 \mid \frac{Z^H(I)}{Z^*(I)} \leq r,\ \forall I\right\}.$$

Nedostatkem této definice ovšem je, že hodnota absolutního výkonnostního poměru bývá dosažena pro úlohy malého rozsahu. S pomocí definice je nemožné měřit výkonnost pro problémy velkého rozsahu. Proto se zavádí jiná míra výkonnosti heuristiky, zvaná asymptotický výkonnostní poměr.

Definice 3.7. *Asymptotický výkonnostní poměr pro heuristiku H (viz [1]) je*

$$R_\infty^H = \inf\left\{r \geq 1 \mid \exists n \text{ takové, že } \frac{Z^H(I)}{Z^*(I)} \leq r,\ \forall I, Z^*(I) \geq n\right\}.$$

Poznámka 3.1. *$R_\infty^H \leq R^H$ (viz [1]).*

4 Problém naplňování zásobníku

Naplňování zásobníku (Bin packing problem – BPP) je jeden ze známých problémů, na které můžeme v logistice narazit. Všeobecně by se dal tento problém popsat následujícími slovy.

„Jak nejlépe uspořádat zboží nachystané k přepravě do kontejnerů, aby bylo kontejnerů potřeba co nejmenší množství?“

Tento problém je ovšem velmi komplexní, poněvadž zde uvažujeme trojrozměrné předměty v trojrozměrném prostoru, takže zde neřešíme pouze to, které předměty v zásobníku budou, ale i to, jak v něm budou umístěny. Z tohoto důvodu se omezíme na jednodušší úlohu.

V této úloze budeme uvažovat, že předměty mají stejnou šířku i délku jako náš kontejner a liší se pouze svou výškou. Například vydavatel knih má knihy stejného formátu, ale liší se počtem stran (tedy svou výškou). Knihy má rozeslat do knihkupectví a pro rozesílání používá stejné krabice. Jeho snahou je odeslat co nejmenší počet krabic (jeho cílem je, aby každá krabice byla pokud možno úplně plná). Pro lepší pochopení bude vhodně si úlohu matematicky zadefinovat.

Definice 4.1. *Mějme seznam n reálných čísel $L = (\omega_1, \omega_2, \ldots, \omega_n)$, kde $0 < \omega_i \leq 1$ nazveme velikostí položky i. Naším úkolem je umístit jednotlivé položky do zásobníků tak, aby součet položek v zásobníku nepřesáhl 1 a současně minimalizovat počet použitých zásobníků.*

K řešení tohoto úkolu bylo navrženo několik heuristik, které se liší jak možností použití, tak výsledným uspořádáním předmětů. Nyní si postupně tyto metody popíšeme a poté se pokusíme určit, jak rozdílné výsledky nám dávají a kdy je vhodné je použít.

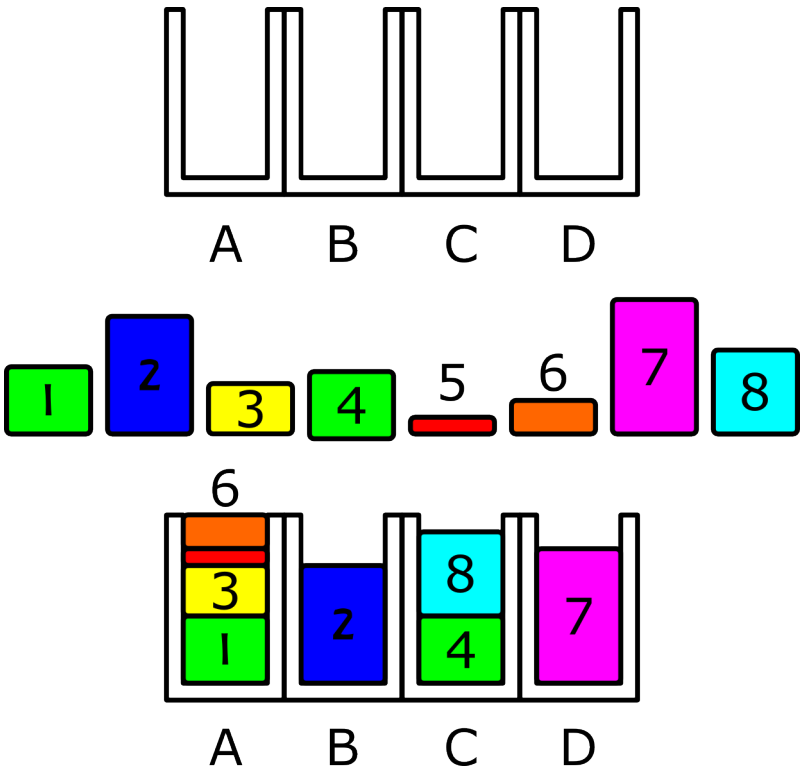
4.1 Heuristika First-Fit

Heuristika First-Fit (zkráceně FF) je nejjednodušší metodou naplňování zásobníku, přesto se jedná o velmi oblíbenou a používanou metodu.

Při užití této metody vkládáme jednotlivé položky do zásobníku přesně v tom pořadí, jak jsou uvedeny v seznamu. Algoritmus je zadán rekurentně.

První položku seznamu vložíme do zásobníku číslo jedna. Předpokládejme, že do zásobníků již bylo vloženo $j-1$ položek. Položku j vložíme do zásobníku s nejnižším číslem, který bude splňovat podmínku, že jeho objem spolu s objemem položky j nepřesáhne číslo 1.

Pro lepší pochopení si algoritmus ilustrujme na následujícím obrázku.



Na vstupu máme 8 položek, které mají po řadě tyto velikosti $0,4; 0,7; 0,3; 0,4; 0,1; 0,2; 0,8; 0,5$. Nejprve vezmeme první položku o velikosti 0,4 a vložíme ji do zásobníku A. Druhá položka o velikosti 0,7 se nám do zásobníku A nevejde a proto ji vložíme do zásobníku B. Položka, která je třetí v pořadí má velikost 0,3, takže jí můžeme vložit do zásobníku A. Objem zásobníku A je nyní 0,7. Tímto postupem pokračujeme, dokud nevyčerpáme všechny položky. Pro větší názornost byly jednotlivé položky očíslovány tak, jak se vkládají do zásobníku.

Zajímavý problém nastane, pakliže se snažíme odhadnout, kolik zásobníků

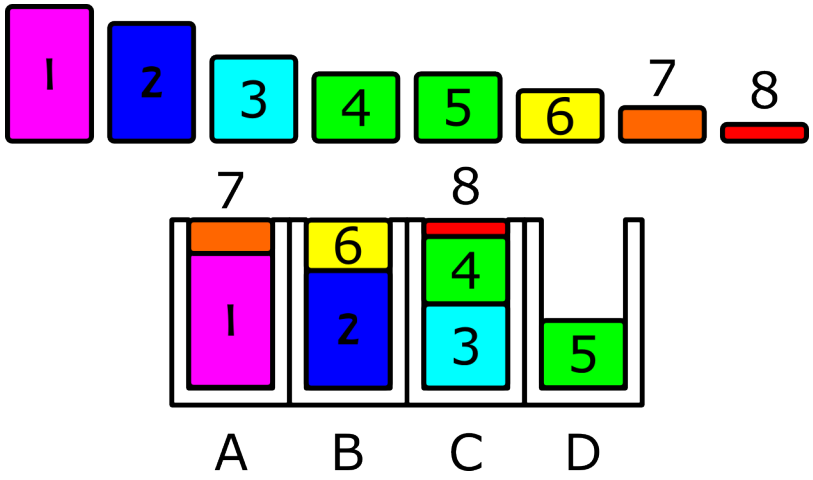
budeme potřebovat, abychom měli jistotu, že nám při užití tohoto algoritmu nebude žádný zásobník scházet. V takovémto případě je dobré znát takzvanou horní závoru. Horní závora nám udává maximální počet zásobníků, který bychom mohli potřebovat, ovšem neříká nám, kolik jich ve skutečnosti užijeme. Na první pohled je zřejmé, že horní závora nemůže přesáhnout počet položek, které se snažíme do zásobníků uspořádat, ovšem je vhodné ji určit blíže. Při užití heuristiky First-Fit je horní závora dána následující větou.

Věta 4.1. *Nechť $b^{FF}(L)$ udává počet zásobníků, které byly použity při užití heuristiky First-Fit a $b^*(L)$ udává nejmenší počet zásobníků, které jsou potřeba při řešení problému naplňování zásobníku. Horní závora pro algoritmus First-Fit je pak dána vzorcem*

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

Důkaz lze nalézt v časopise Combinatorial Theory [3].

Z heuristiky First-Fit je odvozena heuristika First-Fit Decreasing (FFD), která se od původní heuristiky liší tím, že seznam L nejprve seřadíme tak, že první položka bude mít největší objem a poslední položka bude mít objem nejmenší. Tato metoda má tu výhodu, že nejprve vkládáme ty „nejméně skladné“ položky a až poté hledáme vhodné umístění pro položky s menším objemem.



Pro tuto heuristiku lze také určit horní hranici, a ta je uvedena v následující větě.

Věta 4.2. *Nechť $b^{FFD}(L)$ udává počet zásobníků, které byly použity při užití heuristiky First-Fit Decreasing a $b^*(L)$ udává nejmenší počet zásobníků, které jsou potřeba při řešení problému naplňování zásobníku. Horní závora pro algoritmus First-Fit Decreasing je pak dána vzorcem*

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

Důkaz lze nalézt v časopise Algorithms [4].

4.2 Heuristika Best-Fit

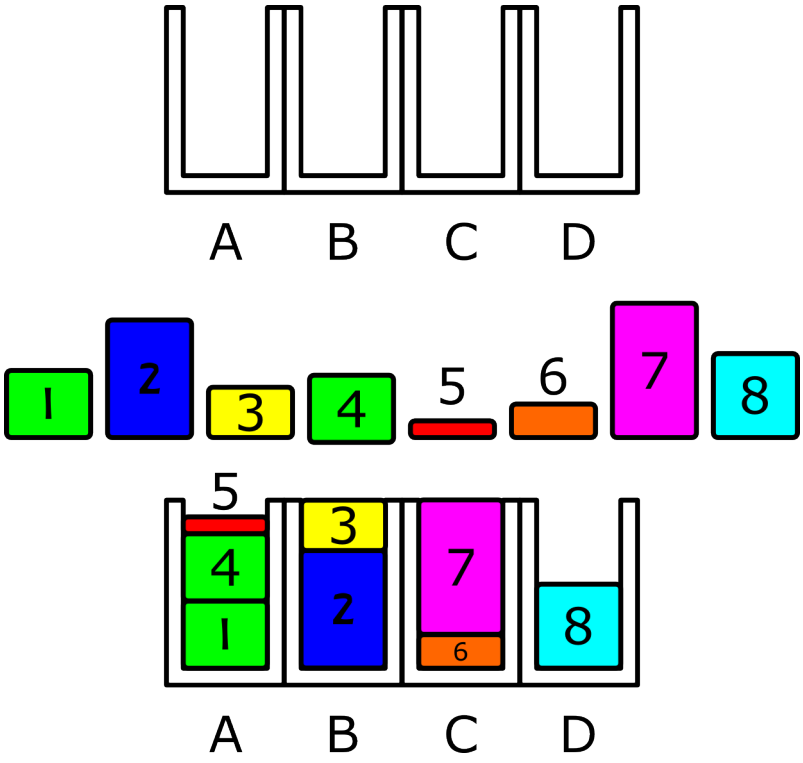
Heuristika Best-Fit (zkráceně BF) je obdobou metody First-Fit, ale metodika je mírně pozměněná.

Při užití této metody vkládáme jednotlivé položky do zásobníku přesně v tom pořadí, jak jsou uvedeny v seznamu. Algoritmus je zadán rekurentně.

První položku seznamu vložíme do zásobníku číslo jedna. Předpokládejme, že do zásobníků již bylo vloženo $j-1$ položek. Položku j vložíme do toho zásobníku, jehož současný objem je největší, nicméně nepřesahuje hodnotu $1 - \omega_j$ (objem zásobníku spolu s objemem položky j je menší nebo roven 1). V případě, že najdeme více zásobníků, které splňují danou podmínku lze užít libovolný zásobník, ovšem většinou volíme zásobník s nejnižším číslem.

Aby byl algoritmus efektivně řešen, je vhodné po každém vložení položky do zásobníku, tyto zásobníky seřadit od nejzaplněnějšího po nejprázdnější. V takovémto případě můžeme vložit následující položku do prvního zásobníku, do kterého se vleze, aniž bychom museli kontrolovat další zásobníky v pořadí.

Pro lepší pochopení si algoritmus ilustrujeme na následujícím obrázku. Hodnoty, které zde užijeme jsou stejné, jako v případě First-Fit plnění.



Na vstupu máme 8 položek, které mají po řadě tyto velikosti $0,4; 0,7; 0,3; 0,4; 0,1; 0,2; 0,8; 0,5$. Nejprve vezmeme první položku o velikosti 0,4 a vložíme ji do zásobníku A. Druhá položka o velikosti 0,7 se nám do prvního zásobníku nevejde a proto ji vložíme do zásobníku B. Položka, která je třetí v pořadí má velikost 0,3. Nejvíce zaplněný je zásobník B a splňuje podmínku, že objem tohoto zásobníku spolu s objemem třetí položky nepřesahuje číslo 1. Proto třetí položku vložíme do zásobníku B. Tímto postupem pokračujeme, dokud nevyčerpáme všechny položky. Pro větší názornost byly jednotlivé položky očíslovány tak, jak se vkládají do zásobníku.

Pro řešení, která byla nalezena pomocí algoritmu Best-Fit je také známa horní závora, která je uvedena v následující větě:

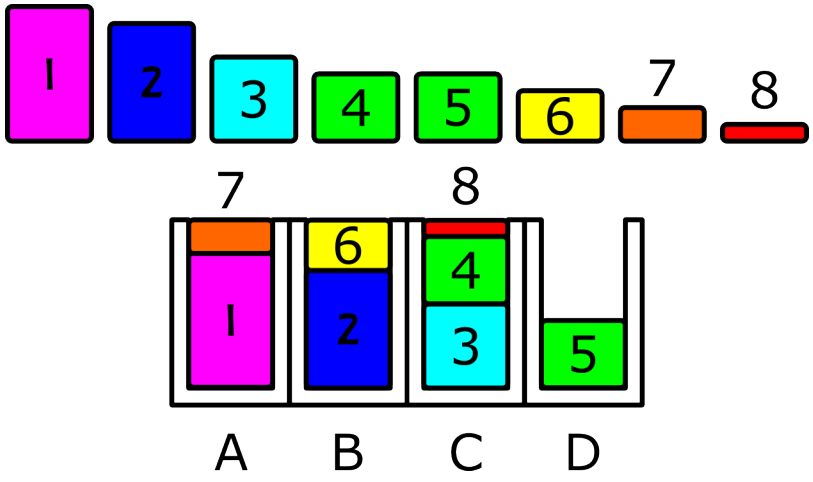
Věta 4.3. *Nechť $b^{BF}(L)$ udává počet zásobníků, které byly použity při užití heuristiky Best-Fit a $b^*(L)$ udává nejmenší počet zásobníků, které jsou potřeba při řešení problému naplňování zásobníku. Horní závora pro algoritmus Best-Fit je*

pak dána vzorcem

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

Důkaz lze nalézt v časopise Combinatorial Theory [3].

Stejným způsobem, jako jsme z heuristiky First-Fit odvodili heuristiku First-Fit Decreasing, odvodíme z heuristiky Best-Fit heuristiku Best-Fit Decreasing (BFD). Opět tedy stačí před užitím heuristiky First-Fit seřadit položky od největší po nejmenší.



I pro takto nově vzniklou heuristiku známe horní závoru, která je uvedena v následující větě.

Věta 4.4. *Nechť $b^{BFD}(L)$ udává počet zásobníků, které byly použity při užití heuristiky Best-Fit Decreasing a $b^*(L)$ udává nejmenší počet zásobníků, které jsou potřeba při řešení problému naplňování zásobníku. Horní závora pro algoritmus Best-Fit Decreasing je pak dána vzorcem*

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

Důkaz lze nalézt v časopise Algorithms [4].

4.3 Porovnání Heuristik

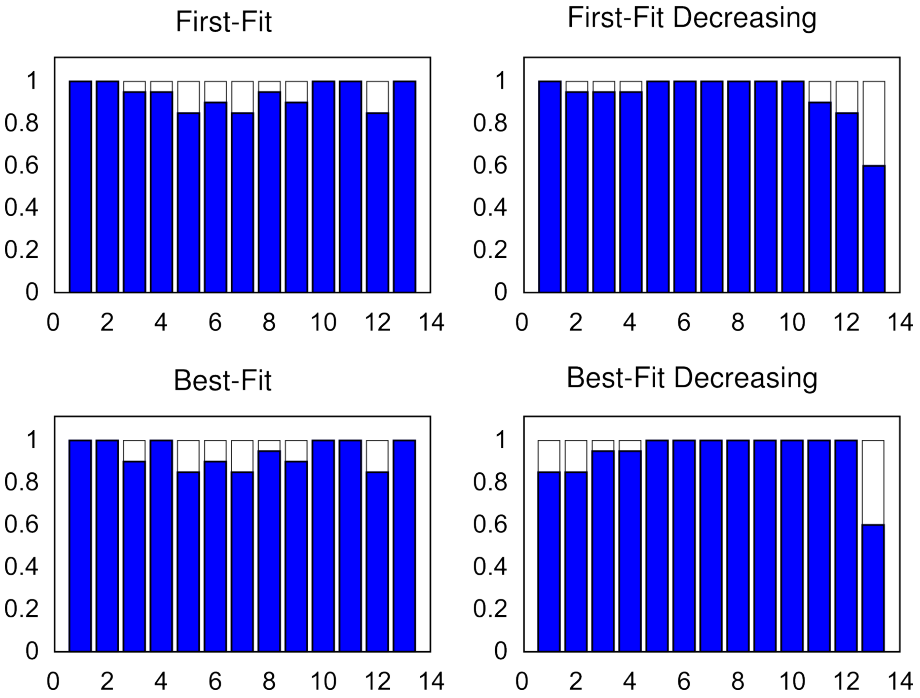
Pro úlohu naplňování zásobníků jsme si ukázaly celkem čtyři různé heuristiky, ovšem není vůbec snadné určit, kterou z nich užít v různých situacích, které mohou nastat. Na tyto otázky se nyní zaměříme a pokusíme se určit, jaký je v jednotlivých heuristikách rozdíl.

První čeho bychom si měli všimnout je fakt, že ne vždy můžeme použít heuristiky First-Fit Decreasing a Best-Fit Decreasing. Ne vždy totiž máme možnost si jednotlivé položky seřadit od největší po nejmenší. To může být dáno například tím, že daných položek je velké množství (například v řádu tisíců) nebo položky přibývají postupně a není možné čekat, až se objeví všechny. V takovém případě jsme odkázáni pouze na zbývající dvě heuristiky. Jindy může nastat případ, kdy není možné užít heuristiku Best-Fit (případně Best-Fit Decreasing). Důvodů může být několik, například špatné rozmístění zásobníků.

Jindy se může ukázat, že ne vždy ta nejefektivnější heuristika je pro nás nejvhodnější. Typickým příkladem může být situace, kdy pomoci algoritmu First-Fit potřebujeme 26 zásobníků, ale při užití metody Best-Fit Decreasing jich je potřeba pouze 25. Na první pohled se nám může zdát, že druhá metoda je pro nás výhodnější, nicméně pakliže si započítáme náklady (v podobě času), které vzniknou nutností výrobky setřídit a pak ještě správně založit do příslušných zásobníků, zjistíme, že prvně jmenovaná metoda je pro nás výhodnější. Proto je výhodné se nad zvolením metody vždy zamyslet.

Nyní se zaměříme na porovnání jednotlivých heuristik, co do výkonnosti. Jistý náhled na efektivitu jednotlivých heuristik nám udává již horní závora, nicméně ta dělí naše heuristiky jen na dvě skupiny: na heuristiky bez seřazení a se seřazením. Podstatně lepší přehled získáme podrobným zkoumáním výsledků pro úlohy naplňování zásobníku.

Nejlépe si rozdílnost jednotlivých heuristik znázorníme, když na jednu množinu položek užijeme všechny čtyři heuristiky a výsledky následně vykreslíme.



Na tomto obrázku je vidět, že každá z námi použitých heuristik nám určí jiný výsledek, i když je počet potřebných zásobníků stejný. Nicméně i přes tento fakt jsou zde zřejmé rozdíly v jednotlivých heuristikách. Heuristika First-Fit, u které se dá očekávat, že bude vykazovat nejhorší výsledky, nám ze 13 zásobníků pouze 5 zaplnila úplně. Druhá nejhorší se jeví heuristika Best-Fit, které se povedlo zcela zaplnit 6 zásobníků. Heuristika First-Fit Decreasing se jeví jako druhá nejlepší, protože počet zcela zaplněných zásobníků u ní dosáhl čísla 7. Zcela očekávaně dopadla nejlépe heuristika Best-Fit Decreasing, která zcela zaplnila 8 zásobníků.

Pořadí výkonnosti heuristik zde ovšem bylo vztaženo k jednomu konkrétnímu případu. Ne vždy to ovšem dopadá stejně. Občas lze narazit na takovou množinu položek, že heuristika Best-Fit dosáhne lepších výsledků (pakliže bereme v potaz počet zcela zaplněných zásobníků), než heuristiky First-Fit Decreasing a Best-Fit Decreasing. Nicméně tento případ nastává spíše výjimečně a to ještě za předpokladu, že počet položek není příliš veliký.

Otázkou ovšem zůstává, jak moc velký význam má pro nás počet zcela zaplněných zásobníků. V rámci logistiky je pro nás podstatně důležitější, abychom minimalizovali celkový počet použitých zásobníků. Proto by nás při snaze určit,

která heuristika je efektivnější, měl primárně zajímat tento aspekt. Z tohoto důvodu bylo nasimulováno několik výpočtů, které jsou shrnuty v následující tabulce.

Počet položek	Průměrný počet zásobníků			
	FF	FFD	BF	BFD
50	23,20	22,19	23,11	22,19
100	45,68	43,58	45,43	43,58
500	223,43	215,42	222,72	215,42
1000	444,94	430,04	443,78	430,04

Při této simulaci byly jednotlivé položky generovány z multinomického rozdělení při kterém bylo předpokládáno, že v reálném životě nejčastěji narazíme takové položky, jejichž velikost odpovídá zhruba polovině velikosti zásobníku, kdežto na položky, které jsou velmi malé nebo pro změnu velmi velké narazíme spíše výjimečně. Pro každý počet položek byly vždy položky generovány stokrát. Na takto vygenerovaných položkách byly vždy použity všechny 4 heuristiky a výsledný počet použitých krabic zaznamenán pro každou heuristiku zvlášť. Nakonec byla vypočítána střední hodnota počtu potřebných zásobníků pro každou heuristiku.

Z tabulky je na první pohled zřejmé, že heuristiky, při kterých jsme napřed položky seřadili dosahují podstatně lepších výsledků, než heuristiky bez seřazení. Tento výsledek se dal očekávat a potvrzuje nám informace, které plynou z vět o horních závorách pro jednotlivé heuristiky.

Dále je z tabulky vidět, že rozdíl mezi heuristikami First-Fit a Best-Fit sice je, ale není nikterak markantní. Tento rozdíl se zvětšuje s větším počtem položek. Tento výsledek je poměrně zajímavý, protože z konstrukce heuristiky Best-Fit by se dalo očekávat, že rozdíl bude podstatně větší.

Nejvíce ovšem překvapí, že heuristiky Firts-Fit Decreasing a Best-Fit Decreasing dosáhli naprosto totožných výsledků. Z tohoto důvodu vznikly obavy, že tento výsledek je důsledkem užití multinomického rozdělení. Proto byla provedena druhá simulace, při které se jednotlivé položky generovali z náhodného rozdělení.

Počet položek	Průměrný počet zásobníků			
	FF	FFD	BF	BFD
50	22,21	21,27	22,02	21,27
100	43,71	41,95	43,43	41,95
500	212,54	206,65	212,08	206,65
1000	421,97	411,49	421,42	411,49

Při druhé simulaci jsme dosáhli obdobných výsledků jako prvně, i když rozdíl mezi heuristikami First-Fit a Best-Fit již nebyl tak markantní. Ovšem opět se ukázalo, že pakliže položky nejprve seřadíme, je jedno, kterou z heuristik následně užijeme.

Celkově vzato při výběru vhodné heuristiky je pro nás nejdůležitější zjistit, zda máme možnost si položky setřídit dříve, než je začneme pokládat do zásobníků. Jestliže to lze, je výhodné užít metodu First-Fit Decreasing. Pokud to ovšem není možné, je důležité se rozhodnout, jestli použít metodu First-Fit, při kterém možná užijeme zásobníků o trochu více, ale se znatelnou úsporou času, nebo si zvolit metodu Best-Fit, při které sice můžeme ušetřit pár zásobníků, ale za cenu delší časové náročnosti.

4.4 Programy pro úlohu naplňování zásobníku

Pro zpracování úlohy naplňování zásobníku bylo potřeba naprogramovat několik programů pro Octave[1]. Všechny tyto programy naleznete na přiloženém CD ve složce `problem_naplnovani_zasobniku` .

Postupně si probereme jednotlivé programy a ukážeme si, jak pracují.

Nejprve potřebujeme znát nějaká vstupní data pro další zpracování. Těmito daty se rozumí řádkový vektor, který obsahuje hodnoty z intervalu od 0 do 1. Pakliže nemáme žádná reálná data, můžeme si je vygenerovat pomocí programů `generator.m` a `generatorm.m` . Oba tyto programy žádají na vstupu pouze počet položek, které chceme vygenerovat a výsledkem je vektor těchto položek. Generátory jsou zde dva, protože první z nich generuje položky z náhodného rozdělení,

[1]Matematický software, který vychází z programu Matlab, viz kapitola 7.5.

kdežto druhý užívá multinomické rozdělení. První program si ukážeme na příkladu.

```
octave:1> polozky = generator(10)
polozky =0.10 0.40 0.45 0.70 0.25 0.70 0.45 0.25 0.60 0.20
```

Jak je vidět, po spuštění jsme do proměnné `polozky` uložili deset náhodně vygenerovaných položek. Obdobný výsledek bychom dostali užitím programu `generatorm.m`

Nyní je potřeba setřídit jednotlivé položky do zásobníků. K tomuto zde slouží programy `ff.m, ffd.m, bf.m, bfd.m` . Všechny tyto algoritmy mají stejné užití. Na vstup vložíme vektor položek a na výstup dostáváme matici, která říká do jakého zásobníku každá položka patří, a vektor, který podává informaci o zaplnění jednotlivých zásobníků. Nyní si ukážeme, jak bychom setřídili naše položky pomocí metody First-Fit Decreasing.

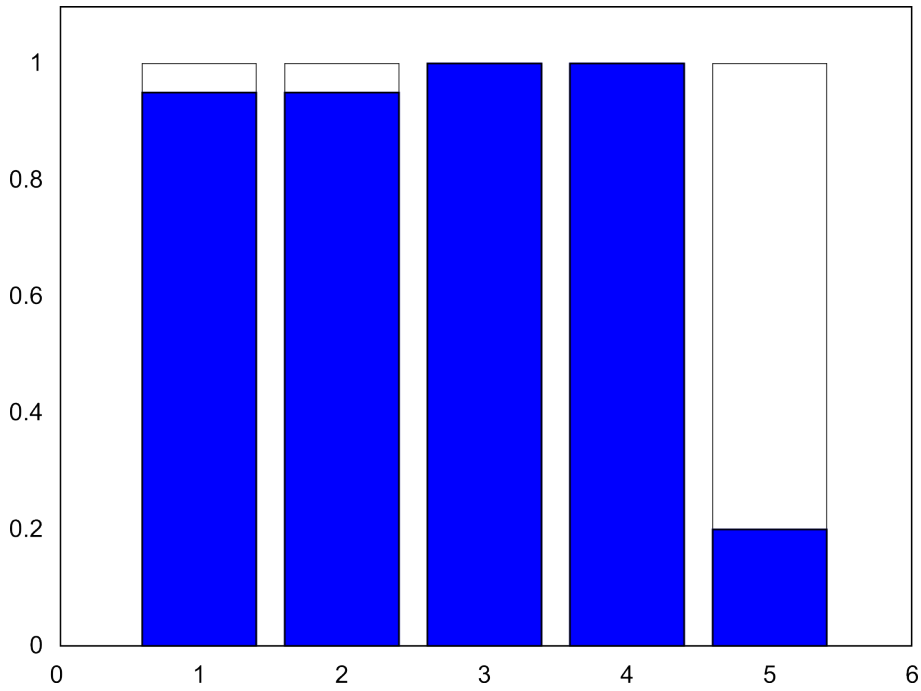
```
octave:2> [rozdeleni_polozek , obsazeni_zasobniku] = ffd(polozky)
rozdeleni_polozek =
0.70 0.70 0.60 0.45 0.45 0.40 0.25 0.25 0.20 0.10
1.00 2.00 3.00 4.00 4.00 3.00 1.00 2.00 5.00 4.00
obsazeni_zasobniku = 0.95 0.95 1.00 1.00 0.20
```

Proměnná `rozdeleni_polozek` obsahuje informace o tom, že položka o velikosti 0.70 patří do zásobníku číslo 1, další položka o velikosti 0.7 patří do zásobníku 2 atd. V proměnné `obsazeni_zasobniku` je vidět jednotlivé obsazení našich zásobníků.

Pro lepší znázornění (zvlášť pro větší počet položek) je vhodné si výsledek graficky znázornit. K tomuto účelu slouží programy `graf.m` a `grafr.m` . Oba tyto programy žádají na vstup matici `rozdeleni_polozek` a vektor `obsazeni_zasobniku`. Program `grafr.m` by se sice bez matice `rozdeleni_polozek` obešel,

nicméně pro sjednocení syntaxe jsem se rozhodl ho i tak vyžadovat. První program je animovaný a postupně vykresluje, jak se zásobník plní jednotlivými položkami, kdežto druhý graf slouží pouze pro rychlou orientaci a ukáže nám rovnou výsledné zaplnění jednotlivých zásobníků. Použití programu `grafr.m` můžete vidět na následujícím příkladě.

```
octave:3> grafr(rozdeleni_polozek , obsazeni_zasobniku)
```



Pakliže je potřeba porovnat jednotlivé heuristiky řazení, je vhodné užít program `porovnani.m` . Tento program vyžaduje na vstup pouze vektor položek a následně vykreslí všechna setřídění položek do zásobníků pomocí studovaných heuristik. Výsledek takového řazení je vidět na straně 15 .

Toto jsou všechny programy, které byly použity při studiu úlohy naplňování zásobníku. Programy byly v prvé řadě navrhovány tak, aby se dali velmi dobře kombinovat a umožnili tak různé experimenty při zkoumání vlastnosti jednotlivých heuristik, případně velmi snadnou úpravu.

5 Problém obchodního cestujícího

Pokud bychom hledali největší problémy současné logistiky, respektive matematiky obecně, určitě bychom brzy narazili na problém obchodního cestujícího (Traveling Salesman Problem – TSP), který lze zformulovat takto:

„Obchodní cestující potřebuje navštívit n měst. Jeho snahou je vybrat takové pořadí měst, aby cesta, kterou urazí byla co nejkratší, každé město navštívil právě jednou a nakonec se vrátil zpět do města, odkud vyrážel[2] (pro zjednodušení problematiky bývá uvažováno, že z každého města je možné přímo cestovat do všech dalších měst).“

Tato úloha byla prvně zformulována roku 1930[3], kdy se jí zabýval rakouský matematik Karl Merge [10] a od této doby trápí matematiky po celém světě. Samozřejmě, že se při řešení této úlohy nalezla velká řada postupů, jak se přiblížit nejlepšímu řešení, nicméně pořád neexistuje způsob, jakým by bylo možné najít nejlepší řešení v přijatelném čase.

Problém obchodního cestujícího se dá klasifikovat podle několika vlastností:

Symetrický - cesta z města A do města B je stejně dlouhá jako cesta z města B do města A.

Asymetrický - úloha obchodního cestujícího není symetrická.

Metrický - pro každá tři města platí trojúhelníková nerovnost.

Euklidovský - vzdálenosti měst odpovídají vzdálenostem v rovině.

Je zřejmé, že euklidovský problém je metrický a symetrický zároveň. Řešení tohoto problému je jednodušší než řešení ostatních verzí.

Kvůli zjednodušení úvah se v textu budeme zabývat pouze symetrickou úlohou.

[2] V praxi se můžeme setkat s modifikací úlohy obchodního cestujícího, kde se vypouští požadavek, aby každé město bylo navštíveno právě jednou.

[3] Už v 19. století se irský matematik William Rowan Hamilton a anglický matematik Thomas Kirkman zabývali problémy, které souvisely s problémem obchodního cestujícího, nicméně sama úloha byla zformulována později.

Na první pohled se může zdát zvláštní, že by tato úloha měla činit problémy, přeci jenom působí docela nevinně. Nezainteresovaný člověk si po přečtení zadání úlohy většinou pomyslí, že se nejedná o těžkou úlohu, protože řešení se nabízí samo a není nikterak složité. Jednoduše si najdeme všechny kombinace pořadí měst (jejich počet je konečný), pro každou kombinaci si zjistíme vzdálenost, kterou by obchodní cestující musel urazit a nakonec vybereme nejkratší řešení. Ovšem tato úvaha opomíjí jednu důležitou věc. Počet kombinací možných cest roste exponenciálně. Nyní si odvodíme vzorec, pomocí kterého lze vypočítat počet všech možných cest.

Předpokládejme, že chceme projít celkem n měst. Počet všech permutací je roven $n!$. Protože naše úloha je symetrická, nezáleží na tom, zda celou trasu projdeme odpředu nebo odzadu. Díky tomu se nám počet všech možných cest zmenší na polovinu, tedy $\frac{n!}{2}$. Pro další úpravu vzorce je vhodné si prezentovat cesty pomocí posloupnosti čísel (ta nám reprezentují jednotlivá města). Pokud si pozorně prohlédneme následující posloupnosti

$$2 - \cdots - 6 - 15 - 3 - 11 - 19 - 17,$$
$$15 - 3 - 11 - 19 - 17 - 2 - \cdots - 6,$$
$$3 - 11 - 19 - 17 - 2 - \cdots - 6 - 15,$$

zjistíme, že všechny nám určují stejnou trasu, ovšem začínají pokaždé v jiném městě. Z toho plyne, že každá naše trasa se při permutaci všech cest objeví celkem n krát. Po zohlednění této vlastnosti do našeho vzorce je zřejmé, že počet všech možností, jak lze projít n měst je roven $\frac{(n-1)!}{2}$.

Právě tato skutečnost má za následek, že ani v době velmi výkonných počítačů není možné vyzkoušet všechna pořadí dostatečně rychle. Aby bylo lépe vidět, jak rychle roste náročnost výpočtu tzv. „hrubou silou“, byl vytvořen algoritmus [4], který zkouší všechny možné cesty a vybírá z nich tu nejlepší. Tento algoritmus byl testován na dvou různých počítačích. Výsledek porovnávání je vidět v následující tabulce.

[4] Podrobněji se jím budeme zabývat v kapitole 5.2.1 na straně 28.

počet měst	počet kombinací	PC 1	PC 2
6	60	0,0775 s	0,0156 s
7	360	0,2044 s	0,0624 s
8	2 520	1,1471 s	0,3120 s
9	20 160	9,9009 s	2,6988 s
10	181 440	101,52 s	26,754 s
11	1 814 400	1 059,4 s	296,51 s

Zvolená PC měla záměrně různou konfiguraci, aby bylo vidět, jak se díky rychlému vývoji v oblasti počítačů zkracuje doba nutná pro výpočet. Zde jsou uvedeny příslušné konfigurace.

označení	typ PC	procesor	frekvence	RAM	rok výroby
PC 1	notebook	Pentium 4 M	2 GHz	757 MiB	2004
PC 2	stolní PC	Core i7 920	2,67GHz	6 GiB	2009

Je vhodné podotknout, že PC 2 nebyl plně využit, poněvadž procesor je 64-bitový, ale program Octave je prozatím pouze 32-bitový. Přesto je výkonnostní rozdíl velmi zřetelný a jasně ukazuje pokroky ve výpočetní technice.

Pakliže se zaměříme na tabulku rychlosti výpočtů, zjistíme, že s rostoucím počtem měst se délka výpočtu velmi prodlužuje. Rozdíl mezi 10 a 11 městy je skoro jedenáctinásobek doby. Pakliže bychom se drželi toho, že s každým dalším městem se doba výpočtu zjedenáctinásobí, výpočet úlohy obchodního cestujícího by pro 20 měst trval více jak 71 000 let u PC 1 a více jak 22 000 let u PC 2.

Nyní už je zcela zřejmé, že úloha obchodního cestujícího je podstatně složitější, než se na první pohled může zdát a i když k výpočtu použijeme počítač, pořád nemáme zaručeno, že se v rozumném čase dobereme řešení. Pro bližší zkoumání bude tedy nutné si úlohu řádně matematicky zadefinovat.

Definice 5.1. *Mějme dánu množinu n měst $\{M_1, \ldots, M_n\}$ a symbolem $d(M_i, M_j)$ $\forall i, j = 1, \ldots, n$ rozumějme vzdálenost měst M_i a M_j, přičemž platí $d(M_i, M_j) = d(M_j, M_i), \forall i, j = 1, \ldots, n$. Cestou rozumíme posloupnost měst $M_{\pi(1)}, \ldots, M_{\pi(n)}$, kde π je permutace čísel $1, \ldots, n$. Délkou cesty potom rozumíme:*

$$\sum_{i=1}^{n-1} d(M_{\pi(i)}, M_{\pi(i+1)}) + d(N_{\pi(n)}, N_{\pi(1)}).$$

Optimální cestou pak rozumíme cestu, jejíž délka je ze všech nejmenší.

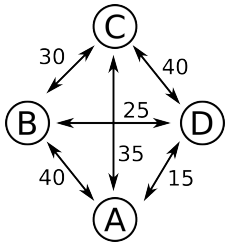
Z definice je zřejmé, že $d(M_i, M_i) = 0, \quad \forall i = 1, \ldots, n.$

Poznámka 5.1. *V definici 5.1 se hovoří o vzdálenosti měst. Ovšem bylo by vhodné si uvědomit, že vzdálenost zde není vzdáleností v pravém slova smyslu, nýbrž se jedná o pojem, který lze chápat různě podle potřeby. Kromě skutečné vzdálenosti to může znamenat například čas (cestu chceme urazit co nejrychleji), ekonomické náklady na cestu (snažíme se najít takovou cestu, aby nás vyšla co nejlevněji), faktor bezpečnosti (vybíráme cestu, u které chceme mít nejmenší pravděpodobnost nehody) případně jakákoliv jiná věc, která je pro nás v daném případě důležitá a jsme schopni ji matematicky zadefinovat.*

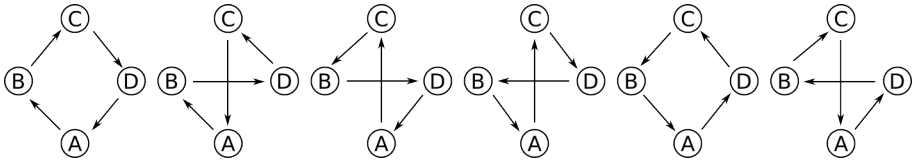
Například pokud převážíme vysoce výbušný materiál, je pro nás nejdůležitější najít takovou cestu, kde riziko havárie bude co nejmenší, nicméně na náklady spojené s cestou tolik nehledíme. Proto v takovémto případě pro nás pojem vzdálenost bude představovat již zmiňovaný faktor bezpečnosti.

Pro lepší pochopení problematiky ukážeme na následujícím příkladě všechny možnosti řešení úlohy pro 4 města a následně z nich vybereme nejlepší řešení.

Příklad 5.1. *Mějme dána města A, B, C, D a vzdálenosti mezi nimi. Naším úkolem je najít nejkratší cestu dle úlohy obchodního cestujícího. Vycházíme z města A a vzdálenosti jsou dány obrázkem.*



Nejprve tedy musíme určit všechny možné cesty. Těchto cest je 6 a lze je zobrazit například takto:



Pro větší názornost jsou zde uvedeny i ty cesty, které jsou díky symetrii stejné a liší se pouze tím, jestli procházejí danou cestu z jedné nebo z druhé strany.

Nyní stačí pro každou nalezenou cestu spočítat délku cesty a následně vybrat optimální cestu, která bude řešením naší úlohy. Pro větší přehlednost si výpočty zapíšeme do tabulky.

cesta	*výpočet vzdálenosti*	*vzdálenost*
1	$40 + 30 + 40 + 15$	125
2	$40 + 25 + 40 + 35$	140
3	$35 + 30 + 25 + 15$	105
4	$35 + 40 + 25 + 40$	140
5	$15 + 40 + 30 + 40$	125
6	$15 + 25 + 30 + 35$	105

Je zřejmé, že optimální pro nás budou cesty 3 a 6. Tyto cesty, jak již bylo řečeno, jsou stejné a liší se pouze směrem, kterým se po nich vydáme.

5.1 Heuristické algoritmy pro obchodního cestujícího

Jak jsme si ukázali v předchozí kapitole, úloha obchodního cestujícího je velice složitá a pro větší počet měst je prakticky neřešitelná v reálném čase. Proto je efektivnější tuto úlohu řešit pomocí heuristických algoritmů. Některé z těchto algoritmů si nyní představíme.

5.1.1 Konstruktivní řešení

Jedním z nejjednodušších algoritmů je tzv. konstruktivní řešení, při kterém cestu tvoříme tak, že k výchozímu městu najdeme nejbližší město. K tomuto městu opět hledáme nejbližší město. Tento algoritmus opakujeme, dokud neprojdeme všechna města. Tato metoda je velmi rychlá, nicméně její efektivita moc dobrá není. To je způsobeno tím, že ze začátku jsou jednotlivé cesty velmi krátké, ovšem ke konci se začínají velmi prodlužovat.

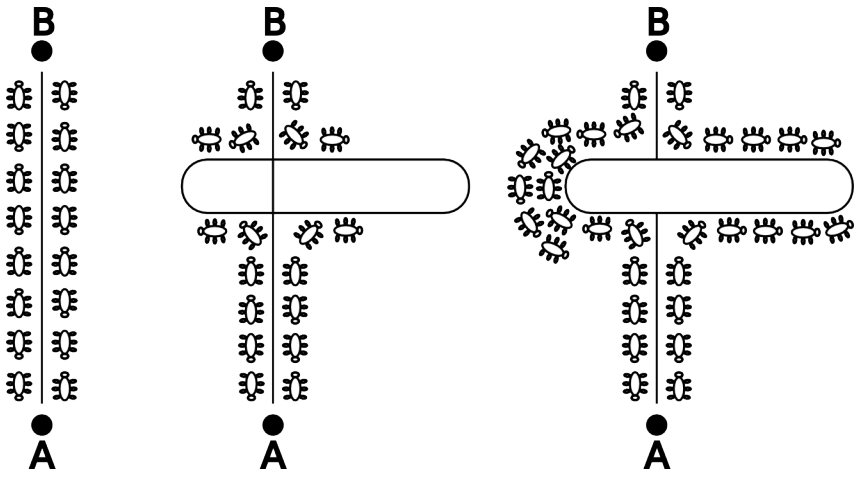
5.1.2 Algoritmus 2-opt

Podstatně výkonnější je algoritmus zvaný *2-opt*. Pro tento algoritmus je vhodné chápat města jako uzly a jednotlivé cesty jako hrany. Vybereme libovolné řešení úlohy (například pomocí předchozího algoritmu) a následně se snažíme libovolné dvě hrany nahradit jinými dvěma hranami tak, aby se výsledná vzdálenost zmenšila. Jakmile dostaneme řešení, které záměnou dvou libovolných hran nelze zlepšit, prohlásíme nynější řešení jako *2-optimální*.

Tento algoritmus lze analogicky převést na algoritmus *3-opt* případně zobecnit na *k-opt*, kde k se v průběhu výpočtu mění. V takovém případě lze dosáhnout vysoce výkonných algoritmů, které nám dávají řešení, jejichž odchylka od optimálního řešení je v řádu jednotek procent.

5.1.3 Algoritmus mravenčí kolonie

Jiné heuristické metody pro řešení úlohy obchodního cestujícího vychází ze sledování světa kolem nás, nejčastěji zvířat. Jednou z takovýchto metod je *mravenčí kolonie (Ant Colony)*. Tento algoritmus se chová obdobně jako živí mravenci. Ti se považují za sociální hmyz a tomu odpovídá jejich chovaní. Při honbě za potravou je pro ně důležité dostat se k potravě co nejkratší cestou. K tomuto účelu je matka příroda vybavila schopnosti vylučovat chemickou látku zvanou feromony. Ostatní mravenci feromony rozpoznávají a jsou schopni rozeznat i jeho koncentraci. Mravenec, který se vydává na cestu za potravou si s větší pravděpodobností zvolí cestu, na které je výskyt feromonů větší. Pro větší názornost si na následujícím obrázku ukážeme, co se stane, když mravencům položíme do cesty překážku.



Jak je vidět, mravenci se pokusí překážku zdolat oběma směry. Protože je ovšem cesta zleva výrazně kratší, projde jí rychleji větší množství mravenců. Díky tomu se výskyt feromonů na této cestě stane velmi výrazný a ostatní mravenci se proto začnou vydávat spíše touto kratší cestou.

Na tomto principu pracuje i algoritmus pro řešení obchodního cestujícího. Při této metodě se používá „virtuální feromon“, který v paměti udržuje dobrá řešení, pomocí kterých se snažíme najít řešení ještě lepší. Bohužel je nutné dávat pozor, aby se neobjevilo řešení, které se na první pohled jeví jako dobré, nicméně vede k brzkému zastavení algoritmu a tím i k horším výsledkům. Z tohoto důvodu se do algoritmu implementuje zpětná vazba, která nám simuluje vypařování feromonu v průběhu času. Tímto se do algoritmu přidává velice podstatná věc, kterou je časové měřítko. To musí být voleno velmi pečlivě, poněvadž pokud bude moc velké, hrozí že algoritmus „uvízne“ v lokálním extrému a zároveň pokud bude moc malé, dojde k brzkému ukončení algoritmu a díky tomu dosáhneme horších výsledků.

Podrobnější popis tohoto algoritmu by byl velmi obsáhlý a přesáhl by rámec této bakalářské práce. Případní zájemci naleznou podrobné rozpracování tohoto algoritmu na internetových stránkách Miloše Němce [13].

5.1.4 Metoda simulovaného žíhání

Existuje daleko více metod pro řešení obchodního cestujícího, které vycházejí ze světa kolem nás. Za všechny zde ještě zmíníme metodu *simulovaného žíhání*. Tato metoda je založena na principu náhodného hledání lepších cest, které by jsme následně použili. Aby se algoritmus nezastavil v nějakém lokálním minimu, je zde přidána náhodná funkce, která v jistých případech připustí přechod na horší cestu, než je ta, kterou momentálně považujeme za nejlepší. S postupem času se naše náhodná funkce „ochlazuje“, což znamená, že šance na přechod na horší cestu se zmenšuje. Zde je velmi důležité, obdobně jako u algoritmu mravenčí kolonie, vhodně zvolit rychlost, s jakou se naše náhodná funkce ochlazuje.

Ukázali jsme si několik heuristik pro řešení úlohy obchodního cestujícího. Porovnání těchto heuristik je ovšem velmi složité a je nad rámec této bakalářské práce, nicméně bude vhodné alespoň zmínit 2 přístupy, které se pro testování užívají.

První postup předpokládá, že města jsou generována náhodně, typicky z rovnoměrného rozdělení a jejich vzdálenost je určena Euklidovskou metrikou. V takovémto případě existuje empirický vztah pro očekávanou délku L^* minimální cesty obchodního cestujícího ve tvaru $L^* = k\sqrt{n \cdot R}$, kde n je počet měst, R je plocha čtverce, který náhodně generovaná města obsahuje a k je empirická konstanta. Očekávaná hodnota k je podle matematiků Helda a Karpa (viz [5]) pro $n \geq 100$ ohraničena uvedeným vztahem s konstantou

$$k = 0,70805 + \frac{0,52229}{\sqrt{n}} + \frac{1,13572}{n} - \frac{3,07474}{n\sqrt{n}}.$$

Matematici Ernesto Bonomi a Jean-Luc Lutton doporučují použít hodnotu $k = 0,749$ [6].

Druhý postup vychází ze známých úloh obchodního cestujícího, které jsou buď vyřešené nebo známe nejlepší zatím nalezené řešení. Takovéto úlohy lze nalézt například na stránkách Ruprecht Karl University of Heidelberg http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/ .

Tyto postupy patří mezi nejoblíbenější testovací kritéria algoritmů pro řešení úlohy obchodního cestujícího.

5.2 Programy pro úlohu obchodního cestujícího

V této kapitole si ukážeme dva programy pro řešení úlohy obchodního cestujícího. První z nich řeší naši úlohu „hrubou silou“ (vyzkouší všechny možnosti a z nich vybere optimální řešení), kdežto druhý je on-line aplikace, která řeší obchodního cestujícího na skutečných mapách celého světa.

5.2.1 Obchodní cestující řešený „hrubou silou“

Při studiu problému obchodního cestujícího bylo potřeba naprogramovat algoritmus, který tento problém řeší „hrubou silou“. Z tohoto důvodu je vhodné nastínit, jak celý algoritmus funguje. Tento algoritmus je navrhnut pro eukleidovskou úlohu obchodního cestujícího. Zdrojové kódy, které jsou psány v programu Octave naleznete na přiloženém CD ve složce `problem_obchodniho_cestujiciho`.

Na vstup algoritmu je přivedena *mapa měst*. Tu představuje matice, která každé město prezentuje pomocí x-ové a y-ové souřadnice. Tyto souřadnice je nutné zpracovat do matice vzdálenosti, do které uložíme délku všech cest mezi městy. Tato matice je zvláštní tím, že je souměrná podle hlavní diagonály (to je dáno symetrií úlohy) a samotná hlavní diagonála je tvořena nulami (toto je dáno tím, že vzdálenost z jednoho města do toho samého města je nulová). Takováto matice se vytváří vždy při řešení úlohy obchodního cestujícího.

Nyní je potřeba vytvořit matici, která obsahuje všechny možnosti, jak lze města projít. Jak již bylo řečeno, budeme muset projít celkem $\frac{(n-1)!}{2}$ cest. Abychom se vyhnuly opakování stejných tras, pouze s jiným počátečním městem, budeme permutovat pouze $(n-1)$ měst. Z takto vzniklé matice permutací odstraníme duplicitu cest, která je způsobena symetrii (nezáleží na tom, jestli cestu procházíme odzadu nebo odpředu). K této matici přidáme sloupcový vektor zbývajícího města, které jsme nepermutovali. Takto dostaneme matici všech možných cest. Zde bych rád upozornil, že příkaz `perms`, který nám právě tyto permutace za-

jišťuje, je v programu Octave naprogramován jiným způsobem než v programu Matlab. Z tohoto důvodu není možné můj algoritmus v programu Matlab použít.

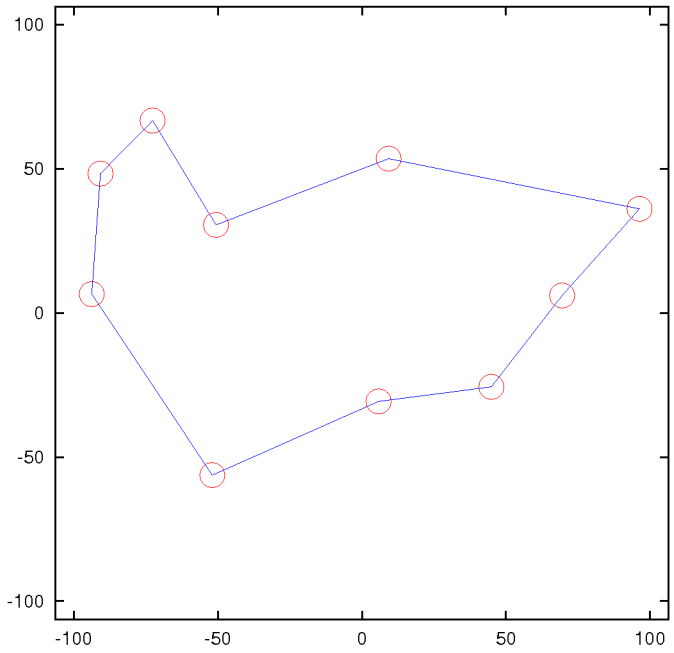
Poslední krok je časově nejnáročnější. Zde je potřeba počítat délku jednotlivých cest a hledat mezi nimi tu nejkratší.

Nyní se podíváme, jak tento program funguje. Nejprve potřebuje znát mapu měst, na kterou chceme náš program aplikovat. Jak již bylo naznačeno, tato mapa je tvořena dvouřádkovou maticí, kde každý sloupec představuje souřadnice jednoho města. Pakliže nemáme žádná reálná data, můžeme si je vygenerovat pomocí programu `generator.m`. Tento program požaduje na vstup počet měst, která chceme vygenerovat a na výstupu vrací námi požadovanou mapu. Příklad užití vypadá takto:

```
octave:1> mapa = generator(10)
mapa =
-93.81 69.51 -72.68 -90.78  9.28  44.96 -51.97   5.81 96.41 -50.65
  6.56  6.09  66.77  48.36 53.60 -25.65 -56.24 -30.68 36.14  30.60
```

Tímto jsme si vygenerovali mapu, která obsahuje 10 měst. Nyní přichází na řadu program `tsp.m`, který začne hledat optimální řešení, které následně i vykreslí. Tento program požaduje na vstup námi vygenerovanou mapu.

```
octave:2> tsp(mapa)
nejkratsi_cesta = 3 4 1 5 2 6 3
delka_nejkratsi_cesty = 521.72
delka_vypoctu = 296,21
```



Program nám vrací hned tři proměnné a jeden graf. První proměnná `nejkratsi_cesta` nám udává pořadí měst (podle matice `mapa`), jak máme naši trasu projít. Vzhledem k symetrii úlohy je možné tuto trasu projít odzadu. Proměnná `delka_nejkratsi_cesty` nám udává, jak dlouhá naše trasa bude a pro sledování rychlosti algoritmu zde přidáváme proměnnou `delka_vypoctu`, která je udávána v sekundách. Graf zde figuruje spíše orientačně, aby bylo názorně vidět, jak budeme celou mapu procházet.

Bohužel tento algoritmus je velice náročný jak časově (toto je řešeno na straně 21), tak i na paměť počítače. Při řešení úlohy pro 11 měst je potřeba vytvořit matici, která obsahuje více než milion řádků. Pakliže se pokusíme řešit úlohu pro více měst, zjistíme, že Octave tuto úlohu nevyřeší a vypíše chybovou hlášku `error: memory exhausted or requested size too large for range of Octave's index type`, což značí, že Octave není schopno pracovat s tak velkými maticemi, které požadujeme. Pro 12 měst by tento problém šel velmi snadno ošetřit například tak, že bychom napřed uvažovali pouze 11 měst, vytvořili matici všech možných cest a až poté bychom ke každému řádku přidávali zbývající město a vytvářeli s ním všechny možné kombinace. Takto by bylo

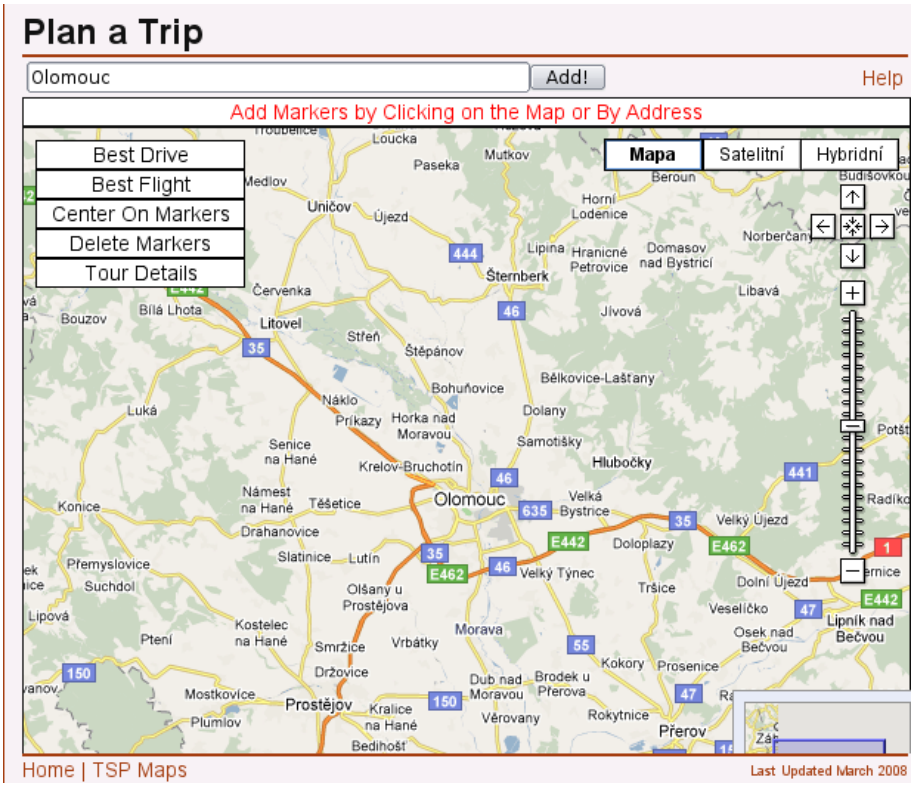
možné tuto úlohu dále řešit metodou „hrubé síly“.

Zde by bylo vhodné podotknout, že při řešení VRP problému nám velmi často tento program bude plně dostačovat pro většinu našich výpočtů. Tento fakt bude podrobněji rozebrán v kapitole 6.

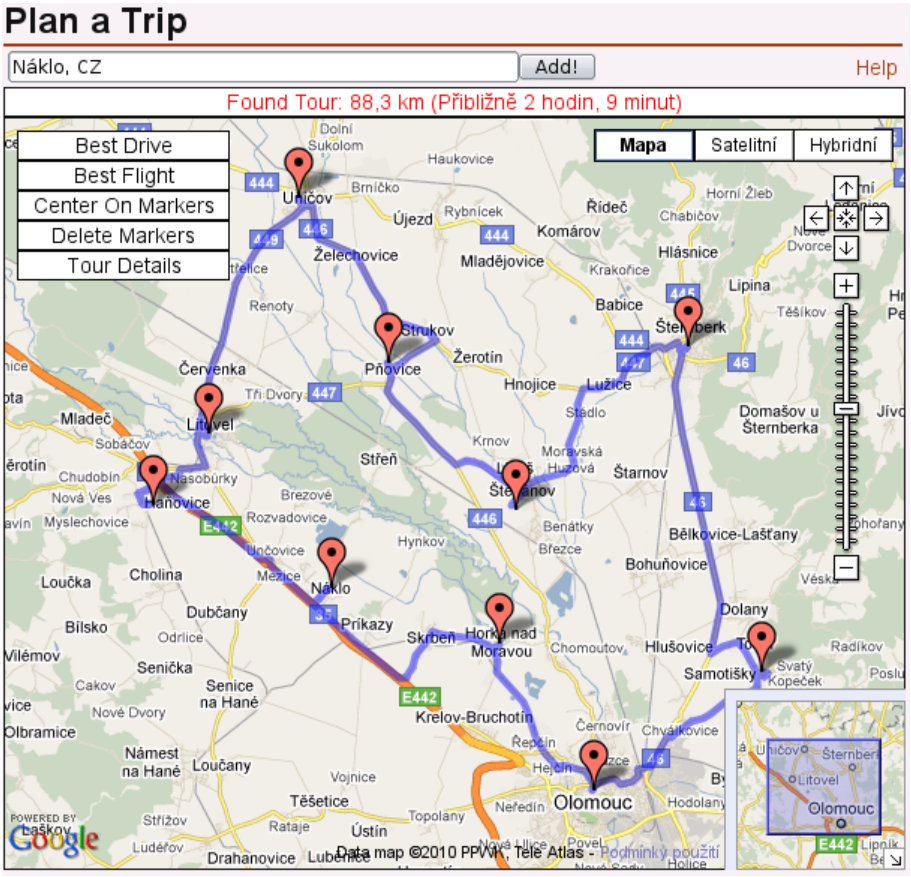
5.2.2 On-line řešení obchodního cestujícího

Úloha obchodního cestujícího se stala natolik zajímavou, že se objevilo několik on-line aplikací (placených i neplacených), které se snaží tuto úlohu nějak efektivně řešit. Velmi zajímavě se jeví aplikace, která se nalézá na stránce `http://www.tsp.gatech.edu/maps/index.html`. Zde stačí kliknout na položku `Plan a Trip` a aplikace je připravena k použití.

Celá aplikace je úzce propojená s Google maps, na kterých hledáme místa, která chce obchodní cestující navštívit. Díky tomuto je zabezpečen velmi kvalitní výstup, protože všechny výsledky se zobrazují na mapě včetně cest. Aplikace je schopna řešit úlohu obchodního cestujícího nejvýše pro 12 měst, což není mnoho, ale na druhou stranu si většina lidí s tímto počtem vystačí. Program je zajímavý i tím, že máme na výběr, jestli bude úloha brát v potaz pozemní vzdálenosti měst (Best Drive) nebo vzdušné vzdálenosti (Best Flight). Po výpočtu je možné si nechat vypsat velice detailní informace o nalezené trase včetně toho, jak máme cestovat. Nyní si celý postup ukážeme názorně:



Jak je vidět, okno je velmi podobné klasickému oknu v Google maps, akorát zde přibyl řádek na přidávání měst (momentálně je zde zadáno Olomouc) a pět tlačítek vlevo nahoře. Funkci těchto tlačítek si budeme průběžně ukazovat. Ze všeho nejdříve musíme přidat nějaká města, přes která chceme úlohu řešit. Město lze zadat dvěma způsoby. Buď klikneme kdekoliv do mapy a bod se nám automaticky přidá nebo napíšeme název města a stiskneme tlačítko `Add!`. Města můžeme zadávat velmi přesně včetně adres a čísel ulic. Pakliže zadáme nějaké město špatně, stačí na něj kliknout a ono zmizí ze seznamu. Tlačítko `Delete Markers` slouží ke smazání všech měst ze seznamu. Pokud delší dobu brouzdáte po mapě, je vhodné stisknout tlačítko `Center of Markers`, které vám vycentruje mapu tak, aby byla vidět všechna zvolená města. Poté co zadáte všechna města, stačí stisknout `Best Drive` a počkat, až se výsledek vykreslí. V případě zájmu lze použít i tlačítko `Best Flight`, které úlohu vyřeší vzdušnou čarou. Takto vypadá úloha vyřešená pro 10 měst z okolí Olomouce:



Z výsledné mapy lze velmi přehledně vyčíst, jak máme cestovat. Pakliže požadujeme přesnější informace o trase, klikneme na `Tour Details` a dostaneme se na stránky Google maps, kde je naše cesta velmi podrobně popsána.

Bylo by vhodné upozornit, že Google maps jsou zde použity pouze pro zjištění vzdáleností různých měst a finální vykreslení. Vše ostatní je řešeno přes `http://www.tsp.gatech.edu/` .

6 Vehicle Routing Problem

Vehicle routing problem (VRP) je známá úloha z logistiky, která si klade za cíl najít nejefektivnější způsob rozvozu zboží k zákazníkům. Úlohu lze zformulovat takto:

„Máme centrální skladiště, ze kterého rozvážíme různé zboží různým zákazníkům. Naším úkolem je najít nejkratší množinu cest, které začínají ve skladišti a vrací se zpět poté, co uspokojí všechny zákazníky.“

Poprvé byla tato úloha zformulována v roce 1959 matematiky G.B. Dantzigem a R.H. Ramserem v článku The Truck Dispatching Problem publikovaným v Management Science [11]. Úloha vznikla převážně proto, že v některých firemních sektorech tvoří náklady na dopravu velkou část výdajů, které se promítnou ve výsledné ceně produktu a tudíž je výhodné snížit náklady na dopravu co nejvíce. Jako většina úloh v logistice má i tato úloha několik variant. Mezi nejznámější patří tyto:

Capacitated VRP - každé auto má omezenou kapacitu.

VRP with time windows - každý zákazník má být obsloužen v určitou dobu.

Multiple Depot VRP - prodejce používá několik skladů, které jsou umístěny na různých místech.

VRP with Pick-Up and Delivering - zákazník může odeslat nějaké zboží zpět do skladu.

Split Delivery VRP - zákazníci mohou být obslouženi různými auty.

Stochastic VRP - některé požadavky (počet zákazníků, nároky zákazníků, doba cesty, čas doručení, ...) jsou náhodné.

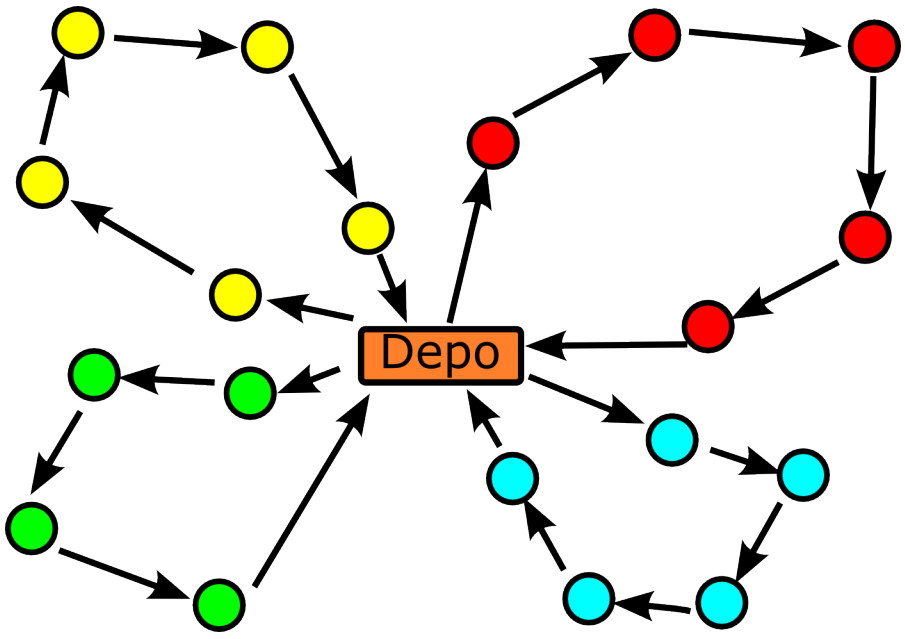
Periodic VRP - rozvoz má být vykonán v určitých dnech.

Velmi důležítou částí VRP úloh je požadavek na pokud možno nejkratší cestu jednotlivých aut, kterou řešíme pomocí problému obchodního cestujícího. Vzhledem k tomu, že obě tyto úlohy jsou NP-úplné, bude se chyba výsledného řešení

odvíjet od chyb řešení obou dílčích úloh. To je jeden z důvodů, proč patří VRP k velmi složitým úlohám.

My se budeme zabývat modelem *Single-depot capacitated vehicle routing problem*, který je poměrně zjednodušený, nicméně se od něj odvíjí řešení složitějších modelů. V této úloze uvažujeme pouze jedno skladiště a požadavky všech zákazníků jsou stejné (všichni si objednali tu samou věc ve stejném množství). Hledáme nejkratší množinu cest, které začínají ve skladišti a vrací se zpět po uspokojení požadavků všech zákazníků. To, že není hledaná jediná nejkratší cesta je limitováno množstvím nákladu, které lze přepravit jediným dopravním prostředkem.

Počet zákazníků, které musíme obsloužit označíme Q a platí, že $Q \geq 1$. Skladiště budeme značit x_0 a množinu zákazníků $N = \{x_1, x_2, \ldots, x_n\}$. Množinu zákazníků spolu se skladištěm budeme značit $N_0 = N \cup \{x_0\}$. Zákazníka, skladiště a cesty lze reprezentovat pomocí grafu $G = (N_0, E)$. Vzdálenost mezi i-tým zákazníkem a skladištěm označíme d_i, kdežto vzdálenost mezi i-tým a j-tým zákazníkem budeme značit $d_{i,j}$. Vzdálenost $d_{\max} = \max_{i \in N} d_i$ Optimální řešení označíme Z^* a řešení získané naší heuristikou budeme značit Z^H.



Je důležité si uvědomit, že množinu zákazníků je třeba rozdělit na s skupin o Q zákaznících a jednu skupinu o $n - sQ$ zákaznících (v případě, že by n bylo

dělitelné Q, budou všechny skupiny obsahovat Q zákazníků). Počet cest je roven $ceil(\frac{n}{Q})$, kde $ceil$ je celočíselné zaokrouhlování nahoru.

Nejčastější heuristikou, která se užívá při řešení VRP úloh je Region Partitioning heuristic (RP), která je založena na konstrukci j oblastí obsahujících $N(j)$ zákazníků. Na těchto oblastech budeme hledat nejkratší cesty pomocí metody obchodního cestujícího.

Je patrné, že výkonnost této heuristiky z velké části záleží právě na výkonnosti algoritmu obchodního cestujícího. Budeme předpokládat, že tato výkonnost je rovna α.

V takovém případě lze dokázat, že

$$Z^{RP(\alpha)} \leq \frac{2}{Q} \sum_{i \in N} d_i + 2d_{\max} + \alpha \sum_{j} L^*(N(j)),$$

kde $L^*(S)$ je délka optimální řešení obchodního cestujícího pro množinu S měst. Důkaz tohoto tvrzení lze nalézt v knize The Logic of Logistics [1].

Zde je důležité uvážit, zda je lepší použít algoritmus, který úlohu obchodního cestujícího řeší pomocí heuristických metod, nebo použít metodu „hrubé síly“. Pakliže budeme VRP problém řešit například pro zásilkovou službu, kde každé auto musí rozvézt několik desítek balíků, jsme nuceni využít nějakou vhodnou heuristickou metodu, protože výpočet hrubou silou by byl nemožný.

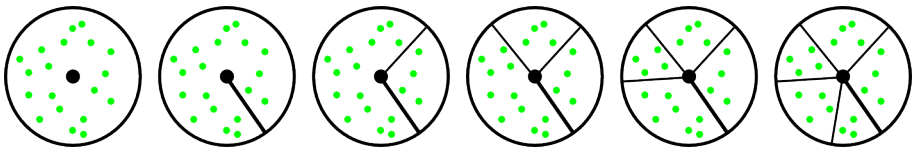
Na druhou stranu je velmi běžné, že firmy rozváží objemnější zásilky a tudíž každé auto musí obsloužit méně jak 10 zákazníků. V takovémto případě už je velmi výhodné použít výpočet hrubou silou, poněvadž časová náročnost je v přípustných mezích a navíc máme jistotu, že nalezená cesta bude nejkratší.

Nyní se budeme zabývat otázkou, jak rozdělit množinu zákazníků na vhodné oblasti. Nejčastěji můžeme narazit na následující dvě metody.

6.1 Polar Region Partitioning – PRP

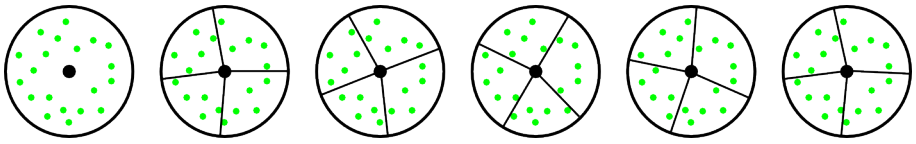
Tato metoda je navržena tak, že body množiny N vepíšeme do kružnice o poloměru $d_{\max}$ se středem v bodě x_0. Kružnici budeme následně dělit do paprskovitých oblastí.

Při programování tohoto úkolu se jako největší problém jeví zvolení počáteční bodu našeho dělení (první paprsek). Následně už je úloha triviální, poněvadž stačí napočítat Q zákazníků a následně za nimi zvolit bod dalšího dělení (další paprsek). Takto postupujeme, dokud nerozdělíme všechny zákazníky do nějakých oblastí.

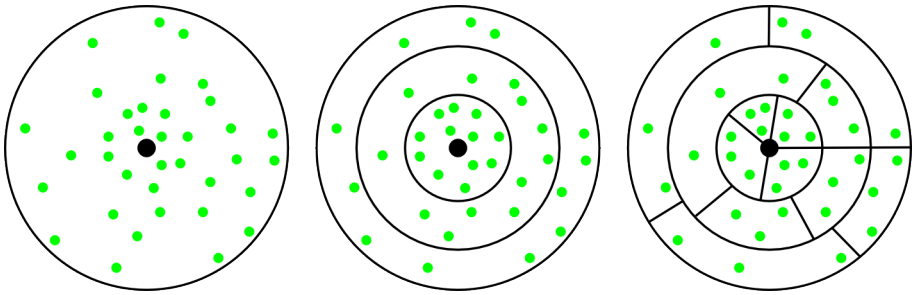


Hledání počátečního bodu dělení se na první pohled zdá velmi složité, poněvadž zde není z čeho vyjít. Ukázalo se, že nejjednodušší způsob je využít náhodnosti. Počáteční bod náhodně vygenerujeme. Tímto krokem se vyhneme pokusům o nalezení optimálního počátečního bodu. Místo toho je vhodnější provést několik dělení, vždy s jiným náhodně zvoleným počátečním bodem, nalézt cesty na těchto děleních pomocí obchodního cestujícího a následně vybrat to dělení, které se jeví jako nejvhodnější.

Poznámka 6.1. *Je dobré si uvědomit, že v případě, kdy n je dělitelné Q, lze velmi snadno vyzkoušet všechna dělení (pomocí PRP), poněvadž jich může nastat právě Q. Toto je velmi dobře vidět na následujícím obrázku.*



Toto dělení je velmi výhodné, pakliže potřebujeme rozdělit množinu našich zákazníků na relativně menší počet oblastí. S narůstajícím počtem zákazníků se totiž zmenšují úhly, dané paprsky dělícími naši kružnici. V takovýchto případech dostáváme dělení, které je pro nás nevýhodné. Z tohoto důvodu je v takovýchto případech vhodnější užít takzvané prstencové dělení, při kterém naši kružnici nejprve rozdělíme na několik prstenců a tyto prstence následně dělíme na oblastí.



Z obrázku je zřejmé, že užitím prstencového dělení se zbavíme všech neduhů PRP dělení. Prstencové dělení lze považovat za přechod od PRP dělení k dělení RRP, kterým se budeme zabývat nyní.

6.2 Rectangular Region Partitioning – RRP

Při této metodě body z množiny N vepíšeme do nejmenšího obdélníku o stranách a, b. Obdélník rozdělíme t vertikálními čarami tak, že každá oblast obsahuje přesně $(h+1)Q$ zákazníků, s případnou výjimkou jedné oblasti. Každou z těchto $t+1$ oblastí rozdělíme h horizontálními úsečkami na $h+1$ menších oblastí tak, že každá obsahuje přesně Q bodů s případnou výjimkou jedné oblasti. Tímto postupem jsme náš původní obdélník o stranách a, b rozdělili na $ceil(\frac{n}{Q})$ menších obdélníků a na každém z těchto obdélníků nyní použijeme obchodního cestujícího pro nalezení nejkratší cesty.

Pro výpočet hodnot t a h byla zjištěna následující tvrzení:

$$t = ceil\Big(\frac{n}{(h+1)Q}\Big) - 1 \quad \text{a} \quad t(h+1)Q < n \leq (t+1)(h+1)Q.$$

Tyto podmínka splní hodnota $h = ceil\Big(\sqrt{\frac{n}{Q}} - 1\Big)$ (viz [1]).

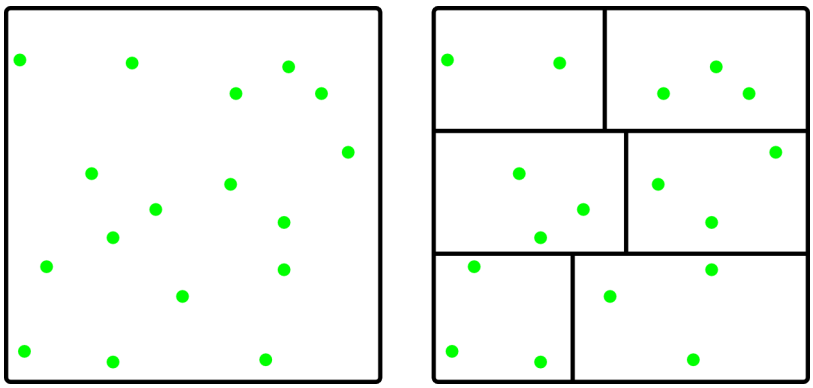
Pro lepší názornost si tento postup ukážeme na příkladě.

Příklad 6.1. *Firma musí rozvést zboží 17 zákazníkům a jedno auto je schopno rozvést zboží nejvýše pro 3 zákazníky. Rozmístění zákazníků je dáno obrázkem. Rozděl zákazníky do oblastí pomocí RRP.*

$$h = ceil\Big(\sqrt{\frac{n}{Q}} - 1\Big) = ceil\Big(\sqrt{\frac{17}{3}} - 1\Big) = ceil(\sqrt{5,\overline{6}} - 1) = 2.$$

$$t = ceil\Big(\frac{n}{(h+1)Q}\Big) - 1 = ceil\Big(\frac{17}{(2+1)3}\Big) - 1 = 1.$$

Rozdělení na oblasti pomocí RRP pak vypadá takto:



Poznámka 6.2. *Je dobré si uvědomit, že v případě, kdy n je dělitelné Q, může nastat pouze jedno dělení, poněvadž všechny oblasti obsahují stejný počet zákazníků. V ostatních případech tomu tak ovšem není, protože zde vznikne jedna oblast, která obsahuje menší počet zákazníků. Problém je ovšem rozhodnout, která oblast to bude. Nejčastěji za tuto oblast volíme jednu z těch, které se nacházejí v rohu našeho obdélníku, ovšem není to nutností. Pakliže bychom se pokusili najít všechny možnosti, které při RRP dělení mohou nastat, hledali bychom celkem* $ceil\frac{n}{Q}$ *možných rozdělení na oblasti.*

6.3 Programy pro vehicle routing problem

Pro zkoumání VRP problému bylo potřeba vytvořit programy, které nám umožňovali rozdělit množinu všech zákazníků do oblastí a následně na tyto oblastí aplikovat program pro řešení obchodního cestujícího. Všechny programy lze nalézt na přiloženém CD ve složce `vrp_problem`.

Pro testování algoritmů byl navrhnut generátor, který generuje mapu našich zákazníků (z rovnoměrného rozdělení). Je důležité si uvědomit, že mapa je dána

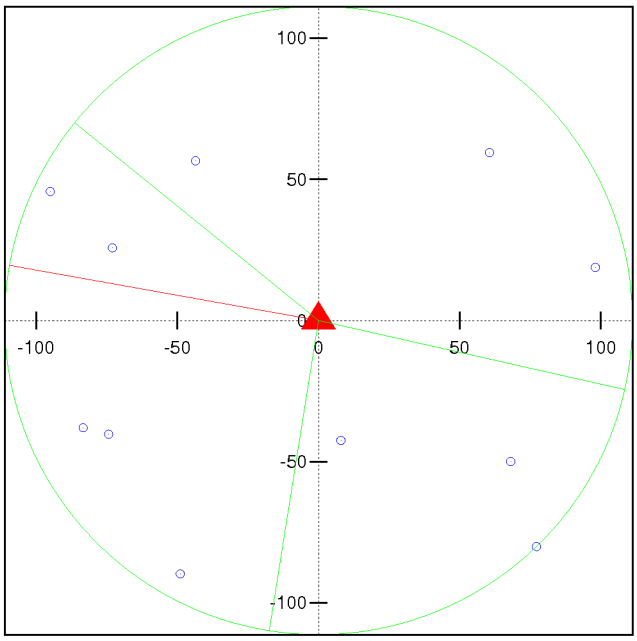
kartézskými souřadnicemi a naše centrální skladiště se nachází v jejím počátku. Generátor požaduje na vstup pouze počet zákazníků, které chceme vygenerovat. Používá se následujícím způsobem:

```
octave:1> mapa=generator(11)
mapa =
 68.09 60.56 -43.53   7.94 -73.00 98.05 -95.05 -48.99 -83.37  77.20
-74.32
-49.95 59.43  56.56 -42.51  25.73 18.79  45.66 -89.74 -37.97 -80.04
-40.28
```

Takto jsme si vygenerovali mapu 11 zákazníků, které budeme chtít nyní rozdělit do oblastí. Nejprve si ukážeme použití programu pro PRP dělení. Na vstup požaduje mapu zákazníků a počet zákazníků v jedné oblasti. V případě, že celkový počet měst bude menší než počet zákazníků v jedné oblasti, program nás na to upozorní.

```
octave:2> [serazena_mapa,Q]=prp(mapa,3)
pocatecni_uhel = -2.96
serazena_mapa =
-83.37 -74.32 -48.99   7.94  77.20  68.09 98.05 60.56 -43.53 -95.05
-73.00
-37.97 -40.28 -89.74 -42.51 -80.04 -49.95 18.79 59.43  56.56  45.66
25.73
Q = 3.00
```

Na vstup programu jsme dali naši mapu a požadavek, aby dělení do jednotlivých oblastí bylo prováděno po třech. Program nám vypíše počáteční úhel (`pocatecni_uhel`), který byl náhodně zvolen pro začátek dělení. Na výstupu programu dostáváme seřazenou mapa zákazníků (`serazena_mapa`) pomocí dělení PRP a počet zákazníků v jedné oblasti `Q`. Kromě těchto informací se nám navíc naše dělení i vykreslí. Počáteční dělení je zobrazeno červenou barvou pro lepší orientaci.



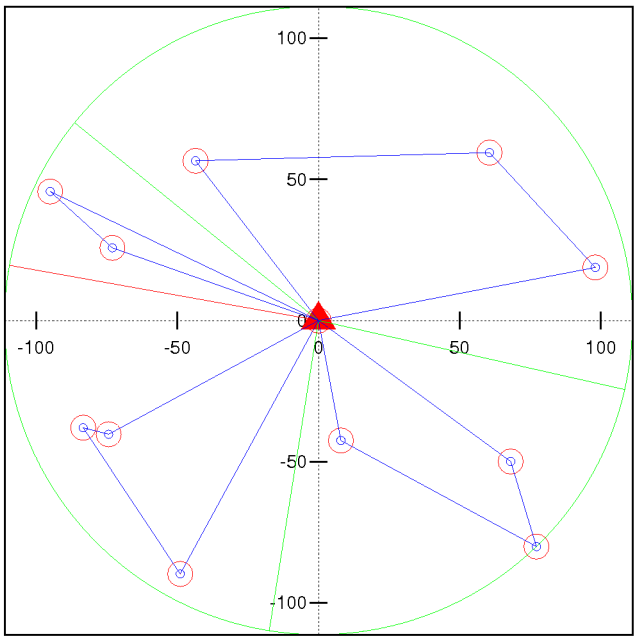
Nyní již máme rozdělenou mapu na jednotlivé oblasti. Posledním krokem při řešení VRP problému je nalezení optimálních cest v jednotlivých oblastech. K tomuto účelu slouží program `vrp`, který pro svůj běh využívá mírně upravený program pro výpočet obchodního cestujícího „hrubou silou“. Jak již bylo řečeno, pro menší počet měst je vhodné tento algoritmus použít. Program na vstup žádá již seřazenou mapu zákazníků a dále počet zákazníků v dané oblasti.

Je vhodné si před spuštěním programu promyslet, jaká data dáváme na vstup. Poněvadž pokud máme VRP problém řešen pro 220 měst a tato města dělíme do oblastí po 11 zákaznících, může výpočet trvat i několik hodin. Proto je vždy nutné si úlohu řádně promyslet.

Samotný program se používá takto:

```
octave:3> vrp(serazena_mapa,Q)
celkova_delka = 1039.38
```

Program nám pro orientaci vypíše celkovou délku trasy. Zobrazení podrobnějších informací o jednotlivých děleních lze dosáhnout mírnou úpravou zdrojového kódu souboru `tsp.m`. Nejdůležitější je ovšem vykreslený graf, který zachovává dělení na oblasti a navíc vykreslí jednotlivé cesty pomocí obchodního cestujícího.

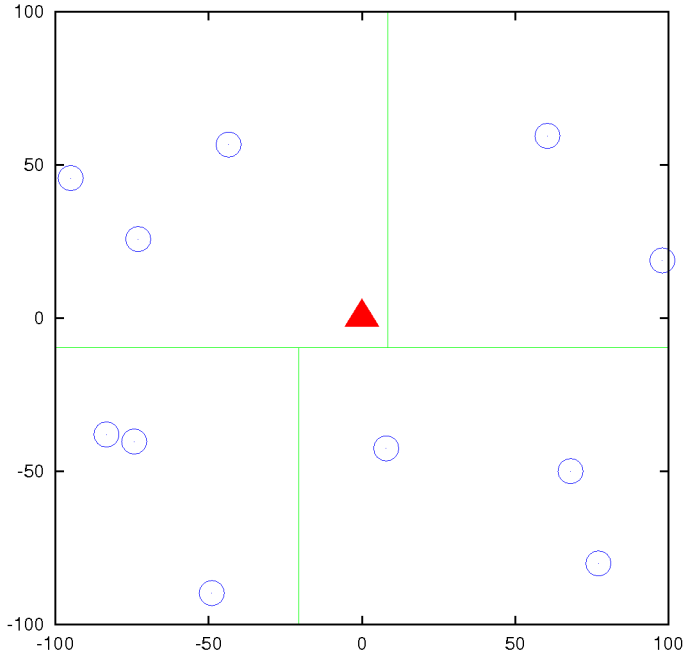


Velmi podobně pracuje program pro RRP dělení. Vstupem programu je mapa zákazníků a počet zákazníků v jedné oblasti. Při použití tohoto programu mohou nastat celkem tři různé situace.

Pokud celkový počet měst bude menší než počet zákazníků v jedné oblasti, program nás upozorní, že není třeba provádět dělení na oblasti. Další situace nastává v okamžiku, kdy je celkový počet měst větší než počet měst v jedné oblasti, nicméně oblast se má dělit pouze pomocí horizontálních čar. V tomto případě budeme opět programem upozorněni. Nejčastější ovšem narazíme na situaci, kdy bude nutné provádět jak horizontální, tak i vertikální dělení.

```
octave:4> [serazena_mapa,Q]=rrp(mapa,3)
h = 1.00
t = 1.00
serazena_mapa =
 -83.37 -74.32 -48.99   7.94  68.09  77.20 -95.05 -73.00 -43.53 60.56
98.05
 -37.97 -40.28 -89.74 -42.51 -49.95 -80.04  45.66  25.73  56.56 59.43
18.79
```

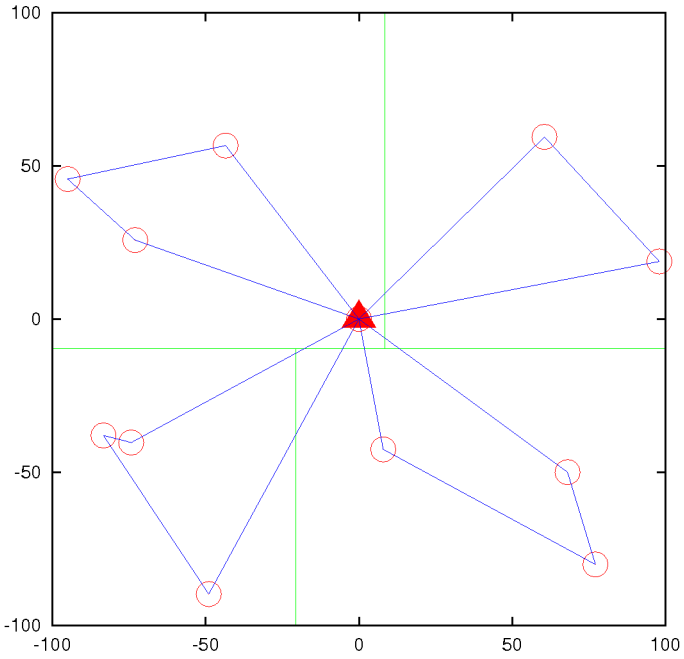
```
Q = 3.00
```



Program jsme spustili s požadavkem, aby dělení do jednotlivých oblastí bylo prováděno po třech. Program nám vypíše počet horizontálních (`t`) a vertikálních (`h`) dělení na oblasti. Na výstup se dostane již seřazená mapa zákazníků `serazena_mapa` a počet měst v jedné oblasti `Q`. Kromě těchto informací se nám naše dělení opět vykreslí.

Na takto rozdělené oblasti můžeme opět aplikovat program `vrp`. Jeho použití je stejné jako v předchozím případě.

```
octave:5> vrp(serazena_mapa,Q)
celkova_delka = 967.31
```



Toto jsou všechny programy, které byly naprogramovány a použity pro studium VRP problému. Je zajímavé, že ve většině případů bylo složitější vykreslit dělení na oblasti než samotný výpočet dělení.

7 Použitý software

Tato bakalářská práce je specifická jednou věcí, která sice v zadání není zmíněna, nicméně si zaslouží pozornost. Velmi často se v současné době setkáváme s *Open Source Softwarem (OSS)*, ovšem ne každý je tomuto softwaru nakloněn. Velmi často je to z důvodu neznalosti problematiky nebo prosté nedůvěry. Z tohoto důvodu celá tato bakalářská práce vznikala za pomocí open source. V této kapitole si ho přiblížíme a ukážeme si, jaký má potenciál jak pro matematiky, tak pro veřejnost obecně.

Definice 7.1. *Open source software je počítačový software s otevřeným zdrojovým kódem. Otevřenost zde znamená jak technickou dostupnost zdrojových kódu, tak legální dostupnost – licenci software, která umožňuje, při dodržení jistých podmínek, uživatelům zdrojový kód využívat, například prohlížet a upravovat.*

Tento software je ve většině případů distribuován zcela zdarma a jediné co po uživatelích žádá je dodržování licencí. Pakliže chce uživatel vývojáře finančně podpořit, je mu to na stránkách těchto aplikací většinou umožněno, ovšem nic jej k tomu nenutí.

Open source software se zpravidla pozná podle licence, pod kterou je software vydáván. Jednou z nejčastěji používaných licencí je *GNU General Public License (GNU GPL)*, která momentálně existuje ve třech verzích[5]. Tato licence je specifická tím, že kdokoliv může stávající program pozměňovat, ovšem odvozená díla musí být vydána pod stejnou licencí. Oproti tomu licence *BSD*, nenutí uživatele téměř k ničemu a upravený kód může distribuovat pod jakoukoliv licencí. Tato licence je jednou z nejsvobodnějších licencí na světě. Vzhledem k tomu, že rozdíl mezi těmito svobodnými licencemi je veliký, vznikla licence *GNU Lesser General Public License (GNU LGPL)*, což je volnější verze licence GNU GPL. Kromě těchto licencí existuje spousta jiných, které mají různé využití. Například licence Creative Commons je zaměřena na autorská práva a převážně se používá v hudebním průmyslu. Vzhledem k tomu, že problematika licencí je velmi

[5]Pro upřesnění je vhodné zmínit, že GNU GPL spadá do kategorie Svobodného softwaru, což je podmnožina open source softwaru.

obsáhlá, naleznou případní zájemci podrobnější informace na stránkách české wikipedie http://cs.wikipedia.org/, kde jsou licence velmi podrobně probrány (samotná wikipedia patří mezi nejznámější open source projekty).

Nyní se podíváme na jednotlivé programy, které byly při práci použity.

7.1 Operační systém Kubuntu

Základem veškeré práce na počítači je operační systém. Nejznámější open source operační systém je beze sporu GNU/Linux. Přestože na poli domácích počítačů linux nijak nekraluje (jeho penetrace se v ČR pohybuje kolem 2%), ve světe serverových počítačů, DVD rekordérů, Set-Top Boxů a podobně patří k nejpoužívanějším operačním systémům.

Při výběru vhodné linuxové distribuce byla vybrána distribuce *Kubuntu 8.04*, která je sice starší, nicméně se jedná o LTS verzi (Long time support - dlouhodobá podpora). Tato volba zaručila velkou stabilitu použitého systému. Operační systém Kubuntu je odvozen od momentálně nejznámější linuxové distribuce *Ubuntu*, ovšem liší se od ní grafickým prostředím. Grafické prostředí je KDE 3.5, které je velmi svižné i na starších počítačích, díky čemuž nevznikali problémy, způsobeném zpomalováním počítače.

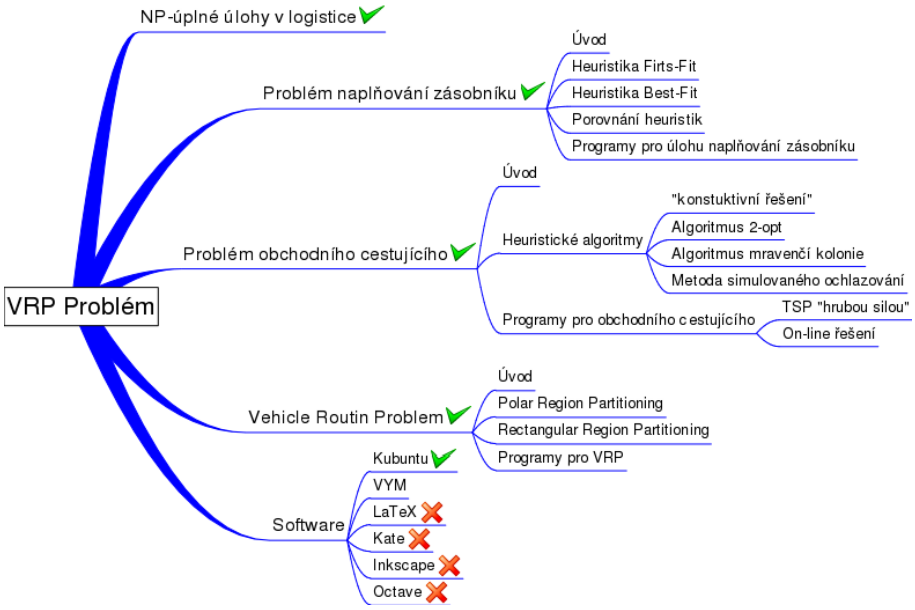
Při práci lze velmi výhodně využít konzoli (obdoba příkazového řádku ve Windows), která je uzpůsobena pro velmi efektivní práci, díky čemuž je člověk schopen urychlit práci až o desítky procent. Typicky se s ní setkáváme při práci s programem LaTeX (viz kapitola 7.3) nebo Octave (viz kapitola 7.5).

České stránky věnované této distribuci včetně instalačních souborů naleznete na http://www.kubuntu.cz/ .

7.2 VYM – editor myšlenkových map

VYM, neboli View your mind je program, který dělá přesně to, co říká jeho název. Tento program umožní uspořádat si své myšlenky a následně podle nich zpracovat například bakalářskou práci. Na rozdíl od obyčejných seznamů je velice názorný a přehledný.

Příklad myšlenkové mapy, která byla použita při tvorbě této bakalářské práce je vidět na následujícím obrázku:



Zpočátku se může použití myšlenkové mapy jevit jako zbytečné, ne–li přímo nepříjemné, nicméně pokud člověk takovou mapu několikrát použije, zjistí, že je to nepostradatelný pomocník.

Domovská stránka programu je na adrese http://www.insilmaril.de/vym/ a je šířen pod licencí GPL.

7.3 LaTeX– software pro sazbu textů

LaTeXje balík maker pro sázecí software TeX, který se pojí se jménem Donald Knuth. Tento sázecí program je velmi oblíbený převážně v akademických kruzích, zvláště v oborech, jakými jsou matematika a fyzika.

Mezi hlavní výhody tohoto softwaru patří velmi propracovaný systém sazby textu, který si klade za cíl co nejlepší čitelnost vysázeného textu a vyznačuje se vynikající podporou pro sazbu matematických vzorců. Navíc je možné poměrně jednoduchým způsobem dodefinovávat další makra a tím si usnadnit práci.

Jeho zásadní nevýhodou pro většinu potenciálních uživatelů je, že se nejedná o WYSIWYG (What you see is what you get) editor. Z tohoto důvodu se uživatel

musí nějakou dobu s programem učit pracovat, než ho může používat.

Domovská stránka projektu je http://www.latex-project.org/ a je šířen pod licencí LPPL. Z tuzemských webových stránek je vhodné zmínit stránky Československého sdružení uživatelů TeXu, které naleznete na http://www.cstug.cz/.

7.4 Inkscape – vektorový grafický editor

Protože se v této práci nachází poměrně velké množství ilustrací, bylo potřeba zajistit, aby tyto ilustraci byly dostatečně kvalitní. Z tohoto důvody byla většina obrázků vytvářena pomocí vektorového editoru Inkscape. Vektorová grafika je velmi vhodná při sazbě knih, poněvadž nám zajistí požadovanou ostrost ilustrací. Navíc Inkscape umí exportovat grafiku do formátu EPS a PNG, což jsou formáty, které jsou potřeba při sazbě textu pomocí LaTeXu.

Inkscape patří mezi plnohodnotné editory vektorové grafiky a na internetu se nalézá obrovské množství tutoriálů, pomocí kterých se s tímto programem lze velmi rychle naučit pracovat.

Jedinou drobnou nevýhodou je, že nativně užívá formát SVG, který stále není všeobecně podporován, nicméně v poslední době se situace velmi zlepšuje.

Domovská stránka programu je na adrese http://www.inkscape.org/ a je šířen pod licencí GPL.

7.5 Octave – software pro matematické výpočty

Octave je matematický software, který obsahuje rozsáhlý soubor nástrojů pro numerické řešení problémů lineární algebry, hledání řešení nelineárních rovnic, integrování funkcí, práci s polynomy a integrování diferenciálních rovnic. Tento software lze považovat za alternativu k matematickému softwaru MatLab, s kterým je vesměs kompatibilní.

Vzhledem ke kompatibilitě syntaxí je tento software vhodný i pro studenty, kteří potřebují pro svou práci MatLab, ale nemají potřebné finance na jeho zakoupení.

Octave se ovládá pomocí příkazové řádky, což nevyhovuje úplně každému.

Z tohoto důvodu lze nalézt několik grafických nádstaveb (GUI) k tomuto programu, které vesměs kopírují vzhledem programu MatLab.

Všechny programy, které jsou v bakalářské práci použity jsou psány pro tento software.

Domovská stránka programu je na adrese http://www.gnu.org/software/octave/ a je šířen pod licencí GPL.

Závěr

V této bakalářské práci jsem popsal základy VRP problému spolu s problémy, které s ním souvisí. Při psaní jsem se snažil, aby byl text čitelný i pro čtenáře, který se touto problematikou nikdy předtím nezabýval.

Při studiu jednotlivých problémů jsem se vždy snažil čerpat z většího počtu zdrojů a následně vše ověřovat pomocí programů, které jsem programoval.

Pro každou úlohu jsem naprogramoval několik programů, které umožňují její řešení. Tyto programy jsou psány pro matematický software Octave a byly testovány na větším množství úloh. U každého programu popisuji v textu způsob jeho použití, díky čemuž si jej případný zájemce může sám vyzkoušet.

Až při psaní této práce jsem plně pochopil, jak složité jsou jednotlivé úlohy. Navíc jsem si uvědomil, že je velmi náročné vymyslet nějaký heuristický algoritmus, který by tyto úlohy řešil. Nejsložitější částí se stali situace, kdy jsem se snažil porovnávat jednotlivé heuristické metody. Každá metoda má totiž své klady i zápory, a není vůbec lehké se rozhodnout, která je pro nás v dané situaci vhodnější.

Navíc jsem si ověřil, že je možné si při psaní bakalářské práce plně vystačit pouze s open source softwarem.

Literatura

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

[2] Michalewicz Z., Fogel D.B.: How to Solve It: Modern Heuristics Second Edition, Springer-Verlag Berlin Heideberg New York (2004)

[3] 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)

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

[5] Held M., Karp M.: The Traveling Salesman Problem and Minimum Spanning Trees, Operations Research (1970)

[6] Bonomi E., Lutton J.L.: The N-City Travelling Salesman Problem: Statistical Mechanics and the Metropolis Algorithm, SIAM review, Vol. 26 (1984)

[7] Marek, J.: Cvičení z logistiky, Přírodovědecká fakulta Univerzity Palackého v Olomouci

[8] Bláhová L.: Matematické modely v logistice, Univerzita Palackého (2008) (Bakalářská práce)

[9] Maslák S.: Problém obchodného cestujúceho, Univerzita Palackého (2008) (Bakalářská práce)

[10] TSP web : http://www.tsp.gatech.edu/ (on-line 10.3.2010)

[11] The VRP web: http://neo.lcc.uma.es/radi-aeb/WebVRP/main.html (on-line 4.3.2010)

[12] Tuháček J. : Problém obchodního cestujícího
http://www.volny.cz/jtuhacek/school/paa_tsp/index.htm (on-line 3.3.2010)

[13] Němec M. : Optimalizace pomocí mravenčích kolonií
http://www.milosnemec.cz/clanek.php?id=78 (on-line 12.3.2010)