Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Vehicle Routing Problém
Metódy riešenia

(Diplomová práca)

2007 Dušan Salaj

Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Vehicle Routing Problém
Metódy riešenia

(Diplomová práca)

Dušan Salaj

---

doc. RNDr. Juraj Procházka, CSc.
Konzultant

Máj 2007, Bratislava

Čestne prehlasujem, že som prácu vypracoval samostatne s použitím uvedenej literatúry a pod vedením diplomového konzultanta.

.....

Dušan Salaj

Poďakovanie

Ďakujem doc. RNDr. Jurajovi Procházkovi CSc., vedúcemu diplomovej práce, za cenné rady a pomoc pri tvorbe diplomovej práce a tiež rodine a priateľom za ich podporu.

Abstrakt

Hlavným cieľom tejto diplomovej práce poskytnúť čitateľovi bližší pohľad na triedu kombinatorických optimalizačných problémov, známu pod menom Vehicle Routing Problems. Konkrétne pôjde o variantu problému známu ako *Capacited Vehicle Routing Problem*.

Uvedieme základné definície tejto triedy problémov a jej varianty. Predstavíme metódy a techniky riešenia na exaktné vyhodnotenie problému. Hlavne sa zameriame na aproximačné a heuristické metódy. Tieto implementujeme a experimentálne vyhodnotíme na definovanej testovacej množine. Na základe nameraných výsledkov sa pokúsime vyhodnotiť jednotlivé charakteristiky skúmaných metód a vyvodiť závery.

Obsah

Kapitola 1

Úvod

Pojem NP-ťažký problém tak ako ho definuje dnešná informatika a teda pomocou Turingových strojov, určite nepatrí medzi najstaršie pojmy. No napriek tomu tieto problémy tu boli vždy a ich riešeniu sa venujú vedci už podstatne dlhšie. Jedným z dôvodov je ich prirodzená prítomnosť v praxi. Ich výskyt nie je zďaleka ohraničený na informatiku, ale môžme sa s nimi stretnúť v geometrii, matematika a v neposlednom rade logistike. Dá sa povedať, že už každý z nás sa stretol s nijakou inštanciou ťažkého problému aj keď to nezaregistroval.

Problém je v tom, že nájdenie optimálneho riešenia pre niektorý z ťažkých problémov môže už pri malej veľkosti problému trvať príliš dlho, teda aj roky. Samozrejme snaha vyriešiť niektorý z týchto problémov k optimalite v rozumnom čase nepramení iba z ľudskej súťaživosti. Napríklad vyriešením mnohých inštancií problémov môžme ušetriť obrovské množstvo zdrojov, a to si už každý obchodník dokáže ľahko zrátať.

My sa v tejto práci budeme venovať Vehicle Routing problému. Jeho cieľom je navrhnúť cesty pre zásobovacie vozidlá, ktoré zásobujú zákazníkov tovarom. Tento problém je z kategórie ťažkých. Na riešenie VRP problému, sa využíva niekoľko tried heuristík. Tie však môžeme rozdeliť na dve základné množiny. Prvou sú takzvané *klasické heuristiky*, ich vývoj sa dial prevažne medzi rokmi 1960 a 1990. A druhou triedou sú *meta-heuristiky*,

ktorých početnosť rastie najmä v posledných desaťročiach. Väčšina dnes používaných konštrukčných ako aj vylepšovacích metód patrí do prvej triedy heuristík. Tieto metódy prehľadávajú iba pomerne obmedzený priestor riešení problému, ale produkujú relatívne dobré riešenia a nepotrebujú dlhý čas na výpočet. A naviac veľa z nich môže byť jednoducho rozšírená o nové obmedzenia ktoré plynú z potreby ich nasadenia v reálnych situáciách. To je teda aj dôvod prečo sú používané v mnohých komerčných produktoch. V metaheuristikách je kladený dôraz na hlbšie preskúmavanie priestoru riešení a najmä tých jeho časti, ktoré sľubujú výborné riešenia. Tieto metódy väčšinou kombinujú sofistikované prehľadávanie množiny takzvaných susedných riešení a premiešavanie vytvorených spojení s cieľom vylepšiť známe riešenie. Kvalita riešení získaných metaheuristikami je oveľa lepšia ako tá získaná klasickými heuristikami, ale za cenu dlhšieho času výpočtu. Základný rozdiel je ale hlavne, že meta-heuristiky sú viac zviazané s konkrétnym problémom a často potrebujú byť presne vyladené na daný prípad.

Skôr ako sa pustíme do hlbšieho analyzovania VRP problémov a metód na jeho riešenie, treba mať na pamäti, že ide o optimalizačný problém. Konkrétne ide o minimalizáciu celkových nákladov na jednotlivé cesty. Preto treba brať v úvahu, že ak sa v texte vyskytuje spojenie hovoriace o rozumnom , prípadne vhodnom postupe, riešení.. a nie je exaktne napísane vzhľadom na akú vlastnosť je tým pre cesty myslené znižovanie ceny cesty. V prípade výpočtového času je rovnako myslené jeho znižovanie.

Kapitola 2

Vehicle Routing problém

V tejto kapitole uvedieme triedu Vehicle Routing optimalizačných problémov. Zadefinujeme si základné definície problému, a zhrnieme jeho vlastnosti. Ďalej predstavíme rôzne varianty daného problému a bližšie sa budeme venovať hlavne variante známej ako Capacited Vehicle Routing Problému. Na konci kapitoly ešte uvedieme motiváciu pre snahu hľadania metód a techník na riešenie daného problému v lepšom ako exponenciálnom čase a jeho obrovský význam pre prax.

2.1 Základy

Vehicle Routing Problem (VRP) je všeobecné meno pre celú triedu kombinatorických optimalizačných problémov, v ktorej ide o nájdenie množiny ciest pre skupinu vozidiel (*vehicle*), sústredených v jednom alebo viacerých skladiskách (*depot*), ktoré zásobia určený počet miest alebo zákazníkov geograficky rozložených na mape podľa potreby. Cieľom pritom je obslúžiť požiadavky celej množiny zákazníkov s minimalizáciou nákladov prípadne času na uskutočnenie všetkých ciest, ktoré začínajú aj končia v sklade.

Definícia 2.1.1 *Trieda NP-Ťažkých rozhodovacích problémov je trieda problémov ktorých rozhodnutie je ťažšie ako rozhodnutie problémov na nedeterministických Turingových strojoch v polynomiálnom čase.*

VRP je dobre známy celočíselný optimalizačný problém spadajúci do kategórie NP-Ťažkých problémov, čo znamená že výpočtové úsilie potrebné na vyriešenie daného problému k optimalite exponenciálne rastie s veľkosťou problému. Preto na hľadanie riešení daných problémov používame aproximačné metódy a heuristiky. Tieto síce často krát nenachádzajú optimálne riešenia ale zato riešenia nachádzajú často v polynomiálnom čase. To ako je dané riešenie blízko k optimálnemu riešeniu a čas potrebný na výpočet tohto riešenia sú niektoré z parametrov hodnotiacich kvalitu danej aproximačnej metódy alebo heuristiky. Na návrh kvalitných heuristík, potrebujeme dobre poznať charakteristické vlastnosti a črty riešeného problému. My sa preto teraz pozrieme bližšie na charakter VRP problému.

Zložitosť VRP problému vo veľkej miere vyplýva z faktu, že VRP problém leží niekde na priesečníku dvoch dobre známych a študovaných problémov Bin Packing Problém (BPP) a Traveling Salesman Problém (TSP). A pre vyriešenie VRP potrebujeme vyriešiť obidva tieto problémy. treba podotknúť, že tieto problémy tiež patria do triedy NP-Ťažkých problémov. A teda už nájdenie riešenie pre každý z nich osobitne vyžaduje veľkú námahu a je časovo náročné.

Definícia 2.1.2 *Bin Packing Problém je problém kde treba rozhodnúť ako vložiť čo najviac objektov do čo najmenšieho počtu zásobníkov (bin) s konštantnou veľkosťou, resp. nech je daný zásobník s kapacitou* $C > 0$ *a zoznam objektov* $\{p_1, p_2, p_3, ..., p_i\}$*, aký je najmenší počet daných zásobníkov potrebných na uskladnenie všetkých objektov? Pre všetky* p_i *s veľkosťou* s_i *platí* $0 \leq s_i \leq C$*. Teda žiadny objekt nie je väčší ako veľkosť zásobníka. Formálnejšie ide o nájdenie rozdelenia a priradenia množiny objektov tak aby všetky obmedzenia boli dodržané a funkcia cieľu bola minimalizovaná (maximalizovaná).*

Existujú rôzne varianty BPP problému ako napríklad, že zásobník môže byť lineárny ako aj 2D či 3D, a tiež objekty môžeme ukladať podľa ich váhy, veľkosti alebo tvaru... Rozhodovacia verzia BPP problému je obsahovo ekvivalentná s modelom VRP pre ktorý platí, že ohodnotenie všetkých hrán je

nulové a teda každé možné riešenie má cenu rovnú nule. Zaujíma nás teda iba rozdelenie vozidiel medzi zákazníkov tak aby boli splnené požiadavky na veľkosť nákladu a samotné vytváranie minimálnych ciest je vypustené. Ak teda existuje možné riešenie pre danú inštanciu VRP problému, je to aj riešenie pre tú istú inštanciu BPP problému.

Definícia 2.1.3 *Traveling Salesman Problém je problém, kde treba nájsť cestu v ohodnotenom grafe, ktorá začína aj končí v tom istom vrchole, obsahuje každý vrchol práve raz a minimalizuje celkovú cenu hrán v ceste. Graf môže byť orientovaný ako aj neorientovaný.*

TSP patri rovnako medzi jeden z najštudovanejších ťažkých problémov a obsahuje množstvo variant a obmien. My ho ale budeme spomínať v hore definovanej verzii, ktorá je súčasťou VRP. Konkrétne ak sa nám podarí vyriešiť BPP pre našu inštanciu VRP tak, získame množiny zákazníkov spĺňajúce kapacitu vozidiel, čo je predpoklad na vytvorenie ciest. Ďalej potrebujeme pre každé vozidlo vytvoriť cestu začínajúcu a končiacu vo vrchole skladu, prechádzajúcu cez všetkých zákazníkov v danej množine a dbáme na minimalizáciu ceny cesty. A to je presne inštancia pre TSP problém. Ako vidno VRP a TSP sú pomerne príbuzné a preto mnohé metódy a heuristiky na ich riešenie majú spoločné charakteristiky. Ich bližšiemu porovnaniu a možnosti využitia techník riešiacich TSP na riešenie VRP bude venovaná samostatná časť ďalej v texte.

2.2 Varianty vehicle routing problému

Ako sme už spomenuli pod VRP sa skrýva celá trieda problémov. V každom z nich síce ide o nájdenie ciest pre zásobenie zákazníkov tovarom zo skladu, ale postupným študovaním týchto problémov a hlavne skutočnými požiadavkami, ktoré prichádzali z prostredia praxe, vznikali nové prídavné obmedzenia. Pridávanie týchto obmedzení a rozširovaní dalo za vznik rôznym variantom VRP problému. Teraz si predstavíme niektoré najznámejšie

a pridáme aj pár viet k ich hlavným špecializačným črtám a príčinám ich vzniku. V zozname uvádzame ich anglické názvy z dôvodu často nevhodného prekladu, ale po prečítaní krátkej charakteristiky bude každému hneď jasné o čo v danej variante ide.

- Capacited Vehicle Routing Problem (CVRP)
- Multiple Depot VRP (MDVRP)
- Split Delivery VRP (SDVRP)
- Stochastic VRP (SVRP)
- VRP with Backhauls (VRPB)
- VRP with Pick-Up and Delivering (VRPPD)
- VRP with Time Windows (VRPTW)

Prejdime teraz k samotným charakteristikám jednotlivých variant klasického VRP problému.

Capacited Vehicle Routing Problem je variant VRP s pevnou a jednotnou kapacitou zásobovacích vozidiel. Rovnako ide o zásobenie všetkých zákazníkov s tovarom uloženým v sklade s minimálnymi nákladmi na cestu. To znamená, že CVRP je rovnaký ako všeobecný VRP s pridanou požiadavkou aby každé vozidlo malo rovnakú kapacitu pre náklad. Tejto variante sa budeme venovať podrobne v celej práci a preto je jej popisu venovaná samostatná kapitola, takže sa jej tu nebudeme bližšie venovať.

Multiple Depot VRP je rozšírenie o prirodzenú požiadavku z praxe a to, že každá spoločnosť môže mať viac ako jeden sklad z ktorých môže zákazníkov zásobovať tovarom. V každom sklade máme potrebné vozidlá na zásobovanie. Ďalšou podmienkou je, že každé vozidlo, ktoré sa vydá zo skladu na zásobenie zákazníkov sa musí vrátiť do toho istého skladu. Ak každému zákazníkovi priradíme sklad z ktorého bude zásobení, môžme sa dá tento distribučný problém pozrieť ako na množinu nezávislých problémov klasického

VRP typu. Vyriešením jednotlivých VRP problémov a následným zmiešaním výsledkov dostaneme riešenie pre MDVRP. Preto je zrejme, že metódy a heuristiky pre VRP sú použiteľné aj na tento variant.

Split Delivery VRP je oslabená varianta VRP, čo pri optimalizačných a kombinatorických problémoch znamená, že niektoré z obmedzení je vynechané prípadne môže byť čiastočne porušené. Pri tejto variante konkrétne ide o obmedzenie, že každý zákazník je navštívený práve raz. Pri SDVRP je teda dovolené aby jeden zákazník bol zásobený rôznymi vozidlami ak to vedie k zníženiu celkových nákladov na uskutočnené cesty. Toto oslabenie sa stalo dôležité najmä pre inštancie z praxe, kde objednávka zákazníkov často presahovala kapacitu reálnych vozidiel a teda bolo nutné urobiť dané oslabenie. Pre náš matematický model ale opäť existuje jednoduchá transformácia na klasický VRP a to rozdelenie objednávky do viacerých fiktívnych objednávok, pre ktoré je poloha zákazníka rovnaká. Opäť sú tu teda účinné metódy a heuristiky pre VRP.

Stochastic VRP môžme po preklade jeho názvu do slovenčiny označiť ako náhodný VRP problém. Tu ale nemôžme hovoriť o konkrétnej variante, lebo SVRP v seba ukrýva celú množinu VRP problémov, v ktorých jedna alebo viacero častí obsahuje náhodnú premennú. Čo je jednoducho premietnuté z praxe, kde sa mnoho okolitých vplyvom (dopravná situácia, odrieknutie objednávky, zlá logistika spolupracujúcej spoločnosti,..) priamo alebo nepriamo dotýka distribúcie tovaru k zákazníkom. Rozlišujeme základné tri častí problému s náhodnou premennou.

- **Náhodný zákazník :** Každý zákazník v_i sa nachádza v probléme s pravdepodobnostnou hodnotou p_i a nezúčastňuje sa s pravdepodobnosťou $1 - p_i$.

- **Náhodná objednávka :** Veľkosť objednaného tovaru d_i pre každého zákazníka je náhodná premenná.

- **Náhodné časy :** Časy na doručenie tovaru δ_i a časy na presun t_{ij} sú náhodne premenne.

Pre dosiahnutie riešenia pre SVRP sú za potreby dve fázy. Najskôr musíme riešenie vypočítať skôr ako sú známe hodnoty jednotlivých náhodných premenných. A potom v druhej fáze, keď sú už náhodné premenne známe, aplikujeme opravné postupnosti krokov na získanie riešenia aktuálneho pre hodnoty priradené náhodnými premennými.

VRP with Backhauls je varianta klasického VRP, kde je zákazníkovi umožnene okrem objednávania a prijímania tovaru zo skladu, časť doručeného tovaru aj vracať späť, prípadne zrušiť celú objednávku. Pre túto vrátenú časť tovaru musíme zabezpečiť jej doručenie do skladu. Treba brať v úvahu, že vrátený tovar zaberá vo vozidlách rovnaký priestor ako keď bol doručovaný. Jedna z ďalších obmien tejto varianty je, že skôr ako môže zásobovacie vozidlo začať zbierať vrátený tovar musí rozviesť všetky objednávky pre ktoré má naložený tovar. To vychádza opäť z faktu v praxi, že vozidla sú zväčša nakladané podľa toho v akej postupnosti bude robení rozvoz, a pre usporiadanie nákladu nie je u každého zákazníka možné, či už z ekonomických alebo technických dôvodov. Pri inštanciách VRPB sú množstva tovaru, ktoré majú byť rozvezené a dovezené dopredu známe.

VRP with Pick-Up and Delivering je v preklade VRP s vyzdvihnutím a doručením. Tu je opäť možnosť pre zákazníka aby vrátil nijakú časť doručeného tovaru späť. Takže ako v VRPB aj v VRPPD musíme brať v úvahu to že navrátený tovar sa musí do vozidla zmestiť. Je tu však pridaná ešte ďalšia požiadavka a to, že vrátený tovar nie je určený pre sklad ale môže byť určený pre iného zákazníka. Toto obmedzenie robí plánovanie oveľa ťažšie a tiež vedie k zlému využitiu kapacity vozidiel, čo v konečnom dôsledku zvyšuje prejdenú vzdialenosť zásobovacích vozidiel a tiež počet vozidiel nutných na zásobovanie. Práve preto sa väčšinou berie v úvahu obmedzená situácia, kde všetky požiadavky sú vybavené tovarom zo skladu a všetok navrátený tovar je doručený do skladu, takže žiadny tovar nie je vymieňaný medzi zákazníkmi. Ďalšia alternatíva je, že vylúčime obmedzenie na to aby bol každý zákazník navštívený práve raz. Pripadne určíme aby každé vozidlo najskôr rozviezlo celý náklad a až potom mohlo nakladať navrátený tovar a ten roz-

vážať podľa potreby.

VRP with Time Windows je jedna z asi najzaujímavejších VRP variant. A to hlavne Čo sa týka jej významu pre prax. Je to teda opäť klasický VRP s pridanými obmedzeniami. Konkrétne ide o pridanie požiadavky, že každý zákazník $v \in V$ má priradený aj časový rozvrh $[e_v, l_v]$, definujúci intervaly v ktorých musí byť zásobený tovarom. Musíme teda navrhnúť cesty tak aby zásobovacie vozidlo prišlo k zákazníkovi počas jeho rozvrhu, ináč zákazník tovar neprijme a vozidlo musí čakať, čo sa prejaví na výslednom čase. Je zrejme, že pri tejto variante nám nejde len o minimalizáciu nákladov na prejdenú vzdialenosť ale berieme v úvahu aj časove náklady spojené s distribúciou tovaru. Časový interval $[e_0, l_0]$ v sklade je nazývaný harmonogram časového plánu. Spojenie tejto varianty s inštanciou z praxe určite nie je problém, podotkli by sme len, že pridanie časovej veličiny do problému so sebou nesie množstvo obmien danej varianty. Niektoré ďalšie požiadavky sú napríklad vodiči vozidiel musia po určitom časovom intervale odpočívať, a preto na zabezpečenie súvislej distribúcie je potreba náhradných vodičov, čo sa samozrejme prejaví na cene.

Toto sú len stručné charakteristiky jednotlivých variant, pre viac informácií viď. použitú literatúru. Treba však pripomenúť, že spomenuté varianty sú len akousi všeobecnou formou pre definovanie niektorých základných obmedzení pre klasický VRP a preto ich rôznou kombináciou dostávame nové varianty. Takisto existuje veľké množstvo druhotných obmedzení ako sú napríklad opotrebovanie vozidiel, technická zručnosť vodičov, obmedzenia niektorých ciest ako sú mosty,.. ,ktoré opäť upravujú finálnu podobu VRP problému.

2.3 Capacited vehicle routing problém

Ako sme už spomenuli CVRP je jedna z mnohých variant klasického VRP problému. Jej hlavnou črtou je, že kapacita vozidiel, ktoré sa zúčastňujú na distribúcií je konštantná a pre všetky rovnaká. Teraz si zadefinujeme tento

problém formálnejšie a pozrieme sa naň z pohľadu grafov.

$$G(V, E) \text{ je graf}$$

$$V = \{v_0, v_1, .., v_n\} \text{ je množina vrcholov}$$

$$v_0 \text{ reprezentuje sklad}$$

$$V' = V \setminus \{v_0\} \text{ je množina zákazníkov}$$

$$E = \{(v_i, v_j)\}, v_i, v_j \in V, i \neq j \text{ je množina hrán}$$

$$C = \{c_{ij}\} \text{ je matica vzdialeností pre } v_i, v_j \in V, i \neq j$$

$$R_i \text{ reprezentuje cestu pre vozidlo } i$$

$$d_i \text{ reprezentuje náklad pre vozidlo } i$$

$$Q \text{ reprezentuje kapacitu vozidiel}$$

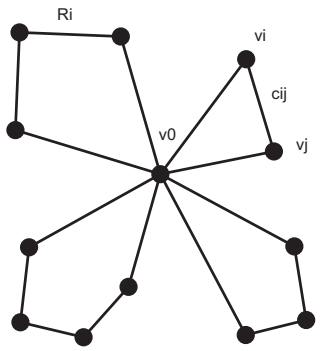
$$m \text{ reprezentuje počet vozidiel}$$

$$\sum_{i=1}^{m} d_i \leq Q \text{ reprezentuje podmienku kapacity}$$

Pokiaľ je graf orientovaný môžme maticu vzdialeností zredukovať iba na trojuholník pod diagonálou a definovať vzdialenosť vrcholov v_i a v_j iba pre $i < j$ a zároveň platí $(v_i, v_j) = (v_j, v_i)$.

Každá konkrétna inštancia CVRP problému je teda orientovaný, prípadne neorientovaný graf s doplňujúcimi informáciami o kapacite vozidiel. Treba nám ešte dodať, že matica vzdialeností C nám tvorí cenu výsledného riešenia a preto hodnoty tejto matice môžu mať rôzny charakter podľa potreby inštancie. Môže isť o Euklidovskú vzdialenosť vrcholov v rovine alebo priestore, ale rovnako môžme používať inú funkciu na výpočet tejto hodnoty. V reálnych inštanciách sa často definuje exaktne podľa charakteristiky problému. Teda váhy ciest zohľadňujú reálny stav ciest na mape a počítajú so skutočnou cenou potrebnou na prechod vozidla od zákazníka v_i k zákazníkovi v_j. V takejto cene je často zohľadnené okrem vzdialenosti v km aj cena za diaľničnú známku, prípadne mýtne za mosty, v niektorých prípadoch sa dokonca

počíta s opotrebovaním vozidiel na neudržovaných cestách, či potreba špeciálnych úprav vozidiel v horských oblastiach. Na obrázku 2.1 môžme vidieť jednoduchú inštanciu aj s riešením problému reprezentovanú grafom.



Obr. 2.1: Inštancia CVRP

2.4 Motivácia

V predchádzajúcich kapitolách sme si predstavili triedu VRP a jej známe inštancie. Skôr ako sa pustíme do samotného štúdia metód a techník zaoberajúcich sa VRP problém, by sme si mali uvedomiť aký význam pre nás z nájdenia riešení plynie. V charakteristike rôznych variant sme sa čiastočne pokúsili poukázať na paralelu grafového prístupu k problému a samotnej praxe. Ale ak by to predsa ešte nebolo úplne zrejmé tak na záver tejto kapitoly pripomenieme niektoré fakty, skryté za matematickou reprezentáciou problému. V závislosti od charakteru spoločností, hra v obrate každej z nich určitú rolu, zásobovanie, prípadne distribúcia tovaru. Zoberieme si napríklad pre Slovensko teraz aktuálnu automobilovú továreň. Niektoré uvádzajú až tisíc vyrobených áut denne, čo okrem toho, že tieto autá treba rozdistribuovať k veľkoodberateľom znamená aj doviezť od subdodávateľov množstvo jednotlivých častí potrebných na výrobu tisíc automobilov. A práve tieto distribúcie sú konkrétne inštancie rôznych variant VRP problému. A teda získanie riešenia, čo najbližšie k optimalite predstavuje zníženie nákladov

na túto distribúciu na minimum. Konkrétne sa dá hovoriť asi o 5 až 15% v závislosti od charakteru distribúcie, čo pri veľkých firmách ako napríklad už spomenuté automobilky alebo aj pivovary znamená úsporu v miliónoch. Niekto by mohol namietať, že predsa existujú exaktné metódy na výpočet optimálnych riešení a aj keď sú výpočtovo náročne veľké firmy by ich mohli prepočítať na výkonných systémoch a aplikovať s cieľom najväčších možných úspor. Tu si treba ale uvedomiť, že každý výrobný proces je dynamická záležitosť. Čím viac subjektov sa na výrobnom procese zúčastňuje, tým samozrejme rastie aj počet elementov, ktoré ho ovplyvňujú a rovnako jeho náchylnosť na zmeny. Preto potrebujeme získavať stále aktuálne riešenia distribúcie, vzhľadom na aktuálny stav. Lebo rovnako pozastavenie výrobného procesu čo i len na pár hodín z dôvodu nedostatku výrobných dielcov opäť produkuje obrovské straty, ktoré si firmy nemôže dovoliť. Preto je cieľom pri vypadnutí jedného zásobovateľa z hry okamžite nájsť iného a zabezpečiť dodanie chýbajúcich častí. Samozrejme by sme mohli pokračovať ďalej s možnosťami aplikácie riešenia VRP problémov v praxi a potrebou získavania dobrých riešení dostatočne rýchlo ale to je nad rámec tejto práce.

Kapitola 3

Experimentálne vyhodnocovanie

V tejto kapitole si predstavíme našu testovaciu množinu navrhnutú na experimentálne porovnanie jednotlivých heuristík. A tiež si povieme pár slov o softvérovom diele, ktoré sme vytvorili práve pre potrebu experimentálneho vyhodnotenia.

3.1 Testovacia množina

Našim cieľom je skúmať metódy a techniky na riešenie VRP problému, a konkrétne sa zameriame na variantu CVRP tak ako bola spomenutá v predošlom texte. Nejde nám však len o porovnanie jednotlivých prístupov akým metódy k hľadaniu riešenia pristupujú, a ich charakteristické črty ale je našim cieľom aj ich experimentálne vyhodnotenie. Teda aké výsledky v akých podmienkach dané metódy produkujú a kde sú úskalia danej techniky, prípadne aké sú jej výhody. Prostredím v predchádzajúcej vete myslíme samotnú konkrétnu inštanciu problému. V CVRP prípade ju teda charakterizuje počet zákazníkov, ich poloha a ich objednávka. Ďalej samotná poloha skladu a kapacita zásobovacích vozidiel. Pri niektorých metódach je aj počet týchto vozidiel pevná súčasť inštancie.

Riešeniu VRP problému sa venovalo a venuje viacero vedcov a výskumníkov. Takže rovnako počas ich výskumu vzniklo niekoľko testovacích množín pre rôzne varianty VRP problémov. Každá z nich je niečím zväčša špecifická. Ide o to čo daný vedec sledoval a na základe toho si aj tú danú testovaciu množinu navrhol. Za spomenutie stoja niektoré množiny pre CVRP, ale nebudeme sa im nijako bližšie venovať, lebo svojou povahou sú často krát rovno tvárne, prípadne pri väčších hodnotách zákazníkov, je ťažko analyzovať, čo danou množinou bolo sledované. Nasledujúci zoznam obsahuje niektoré známe množiny spolu s ich stručnou charakteristikou. Pre bližšie informácie viď. použitú literatúru, prípadne sú voľne dostupné na internete.

- **Augerat et al.** Ide o tri rôzne pomerne veľké testovacie množiny pre CVRP. Každá z nich ma asi 25 samotných inštancií. A počet zákazníkov je medzi 16 a 100. Počet zásobovacích vozidiel sa pohybuje od 5 pre inštancie s menším počtom zákazníkov po 10 pre väčšie inštancie.

- **Christofides a Eilon** Ide o 15 rôznych inštancií pre CVRP s počtom zákazníkov medzi 13 a 101. Počet vozidiel je rovnako úmerný počtu zákazníkov.

- **Fisher** Iba tri inštancie postupne s 45,72 a 135 zákazníkmi a počet zásobovacích vozidiel je pre prvé dve rovnaký 4 a pre poslednú inštanciu 7.

- **Gillet a Johnson** Tu je predstavená iba jedna inštancia s 262 zákazníkmi a maximálnym počtom vozidiel 25.

- **Rinaldi a Yarrow** Rovnako iba jediná inštancia s 48 zákazníkmi a vozidlami s kapacitou 15.

My sme si pre naše potreby vytvoríme vlastnú testovaciu množinu. Dôvodov vytvorenia vlastnej množiny a nie použitie hore spomenutých je viacero. Prvotný je ten, že sa chceme bližšie pozrieť ako jednotlivé metódy obstoja na inštanciách s väčšou množinou zákazníkov a väčšina spomenutých má

rádovo do 100 zákazníkov. Ďalším dôvodom je ohraničenie veľkosti objednávok jednotlivých zákazníkov. Opäť sme sledovali u mnohých inštanciách, že rozdiel medzi maximálnou a minimálnou objednávkou sa pohyboval iba v desiatkach. Prípadne ďalším špecifikom bolo niekoľko objednávok s hodnotou rádovo 50 až 100 krát vyššou ako všetky ostatné objednávky. Išlo teda o simuláciu akýchsi veľkoodberateľov. To síce nebolo v rozpore s našou predstavou o niektorých navrhovaných inštanciách ale často bola pre nás nevyhovujúca kombinácia tohto a iných dôvodov. A v neposlednom rade hrali úlohu aj kapacita zásobovacích vozidiel, ktorá je veľmi úzko spätá s veľkosťami objednávok a samotné rozmiestnenie zákazníkov. Pri rozmiestnení išlo hlavne o polohu skladu vzhľadom na väčšinu zákazníkov.

Naša testovacia množina obsahuje trinásť samostatných inštancií, ktoré teraz predstavíme. Treba si uvedomiť, že inštancie sme navrhovali takmer na konci práce, teda po štúdiu, spracovávaní a implementovaní metód a techník, ktoré budú ďalej v texte spomenuté. A teda nám boli mnohé aspekty riešenia VRP problémov známe. Preto niektoré početnosti, veľkosti objednávok, rozmiestnenia zákazníkov, či už prvky náhodností majú svoj význam aj keď to v tejto časti práce ešte nemusí byť exaktne vidieť. A presnejšie naň upozorníme pri analýze výsledkov získaných experimentálnym vyhodnotením konkrétnych metód a techník.

Prvé štyri inštancie, sa medzi sebou líšia iba početnosťou zákazníkov. Konkrétne ide o 50, 100, 150 a 200 zákazníkov. Kapacita zásobovacích vozidiel je stanovená na 350. Rozmiestnenie zákazníkov ako aj skladu bolo urobené náhodným generovaním na mape s veľkosťou $1000x1000$ logických jednotiek. A nakoniec veľkosť objednávok je tiež výsledkom náhodného generovania. Ide o hodnoty z intervalu $[1, 200]$. V týchto inštanciách teda ide hlavne o rôznorodosť jednotlivých objednávok. Označenie týchto inštancií je postupne vzhľadom na počet zákazníkov nasledovný $R1-50-350$, $R2-100-350$, $R3-150-350$, $R4-100-350$. Toto označenie ako aj všetky ďalšie uvedené je pre dané inštancie jednoznačné a platné pre celý text.

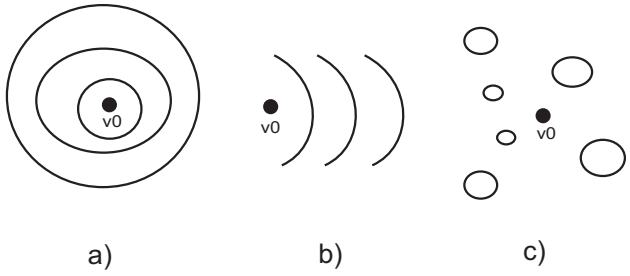
Ďalšie štyri inštancie sa rovnajú počtom zákazníkov predchádzajúcim šty-

rom. Teda ide o inštancie s 50, 100, 150 a 200 zákazníkmi. Kapacita zásobovacích vozidiel je však stanovená na 250. Veľkosť objednávok jednotlivých zákazníkov je rovnako vygenerovaná náhodne. Rozdiel je v tom, že hodnoty sú z oveľa menšieho intervalu a navyše interval neobsahuje malé jednotkové objednávky. Konkrétne sú hodnoty z intervalu $[25, 30]$. Rozmiestnenie skladu a zákazníkov je na ploche $1000x1000$ logických jednotiek a je generované náhodne. Označenie týchto inštancií je nasledovné podľa počtu zákazníkov vzostupne $R5-50-250$, $R6-100-250$, $R7-150-250$ a $R8-200-250$.

Nasledujúce dve inštancie sú posledné zo zástupcov náhodne rozmiestnených inštancií. Ide o inštancie s veľkými množinami zákazníkov. Hlavný rozdiel medzi nimi je v navrhnutí veľkosti kapacity vozidiel vzhľadom na veľkosti objednávok. V prvej inštancií pod označením $R9-500-1000$ ide o množinu 500 zákazníkov rozmiestnených na ploche $2500x2500$. Veľkosť objednávok je z intervalu $[10, 20]$ a kapacita vozidiel je 1000. V druhej inštancií máme rovnako 500 zákazníkov rozmiestnených na ploche $2500x2500$ logických bodov. Veľkosť objednávok je tiež z intervalu $[10, 20]$. Rozdiel oproti predchádzajúcej inštancií je v kapacite vozidiel, ktorá je rádovo menšia a to 150. Označenie je $R10-500-150$.

Posledné naše inštancie už nie sú náhodne generované ale presne rozmiestnené na ploche podľa presných špecifík. Prvá z nich je charakteristická tým, že ma centralizovaný sklad a zákazníci ležia rovnomerne rozmiestnení okolo neho na akýchsi pomyslených sústredných kružniciach. Kapacita vozidiel je 100 a hodnoty objednávok sú z intervalu $[1, 10]$ a sú rovnomerne zastúpené. Označenie je $C1-100-100$. Ďalšia inštancia má rovnako 100 zákazníkov. Rozdiel je v tom, že sklad je umiestnený na kraji a zákazníci sú rozmiestnení na pomyslených výsekoch sústredných kružníc. Kapacita vozidiel je 100 a hodnoty objednávok sú rovnako z intervalu $[1, 10]$ a sú rovnomerne zastúpené. Označenie je $C2-100-100$. Posledná inštancia má označenie $C3-100-100$. Sklad je centralizovaný a zákazníci sú rozmiestnený po mape v akýchsi zhlukoch. Môžme si to predstaviť ako mapu s mestami, kedy jedno mesto obsahuje niekoľko objednávok relatívne blízko, vzhľadom na vzdia-

lenosť miest. Počet zákazníkov je opäť 100 a kapacita vozidiel je tiež 100. Hodnoty objednávok sú z intervalu $[1, 10]$ a sú rovnomerne zastúpené.



Obr. 3.1: Schéma testovacích množín a) C1-100-100 b) C2-100-100 c) C3-100-100

3.2 Softvérové dielo

Pre potrebu testovania jednotlivých heuristík na navrhnutej testovacej množine vznikol popri diplomovej práci aj program, ktorý testované heuristiky implementuje. Jeho účel je teda hlavne na experimentálne vyhodnocovanie jednotlivých inštancií problému. Program je napísaný v jazyku C++ v prostredí Visual C++ 6.0 a využíva knižnicu grafických komponentov MFC.

Architektúra programu je založená na Dokument/Pohľad architektúre tak ako je predstavená vo vývojovom prostredí Visual C++. Program obsahuje základnú triedu, ktorá reprezentuje graf a obsahuje množstvo operácií nad týmto grafom. Ďalej je tu trieda všeobecná trieda algoritmov, ktorá pracuje nad s inštanciami triedy grafu a on ktorej dedia jednotlivé konkrétne triedy reprezentujúce heuristiky.

Samotné dielo vznikalo ako potreba diplomovej práce a čisto pre interné potreby. Tomu svedčí aj jeho charakter. Rovnako program neplánuje byť nijako ďalej distribuovaný, a preto mu nebudeme v tejto práci viac venovať.

Kapitola 4

Exaktné algoritmy

Riešenie NP-Ťažkých problémov až k optimálnym riešeniam v polynomiálnom čase je pre výskum výzva už od začiatkov počítačovej histórie (Dokonca dlho predtým ako bol vôbec pojem NP-ťažkosť objavený). Teraz už vieme, že VRP patrí do NP-ťažkých problémov, a teda aj tu bude nájdenie optimálneho riešenia veľmi výpočtovo náročné. V tejto kapitole si teda predstavíme niekoľko spôsobov na exaktné riešenie.

4.1 Backtracking

Základnou metódou na výpočet optimálneho riešenia je použitie hrubej sily. Na to nám slúžia backtrack algoritmy, ktoré postupne prechádzajú celý strom riešení a po skončení je najlepšie riešenie na ktoré narazili vyhlásené za optimálne. Ono v skutočnosti aj optimálne je, ale výpočtový čas, ktorý sme na prehľadávanie celého stromu riešení potrebovali je neprípustne veľký a rozhodne sa za optimálny nepovažuje. Je to samozrejme spôsobené aj tým, že VRP je zložený z dvoch ťažkých problémov a už len na výpočet BPP problému je potrebný exponencionálny čas. A keď chceme pracovať s pomerne veľkými inštanciami táto metóda nepripadá v úvahu.

4.2 Branch and Bound

Branch and Bound je metóda pre návrh algoritmov pre optimalizačné problémy. Algoritmus B&B využíva stratégiu rozdeľuj a panuj (*divide and conquer*) na rozdelenie priestoru riešení na pod problémy a nasledovne hľadá optimálne riešenie v každom pod probléme zvlášť. Samotné prehľadávanie priestoru riešení je založené na backtracingu, čo je hlavný problém, pretože množina možných riešení je zväčša exponenciálne veľká vzhľadom na veľkosť problému.

Základná idea B&B techniky je zrýchlenie backtracingu vynechaním prehľadávania niektorých častí stromu možných riešení na základe toho, že vieme v danom vrchole povedať ,že sa žiadne optimálne riešenie v danej časti nenachádza skôr ako daný pod strom začneme prehľadávať.

Aby sme vedeli túto vlastnosť rozhodnúť využíva algoritmus hraničnú hodnotu. Keďže ide o optimalizačné problémy, hľadáme maximálne resp. minimálne optimálne riešenia. Vzhľadom na to využívame dolnú resp. hornú hranicu pre optimálne riešenie. Väčšinou je počiatočná hraničná hodnota vypočítaná ešte pred samotným algoritmom. Na jej výpočet môže byť použitá niektorá z aproximačných metód alebo heuristiky. Keď máme hraničnú hodnotu s_0 pre riešenie problému S, rozdelíme problém na n pod problémov $S_1, ..., S_n$. Niekedy sa tieto pod problémy nazývajú aj deti problému S. Následne všetky tieto problémy pridáme do zoznamu kandidátov na optimálne riešenie. Sú to tie pod problémy ktoré ešte čakajú na prehľadanie. Táto časť sa nazýva vetvenie(*branching*). Algoritmus pokračuje spracovávaním vybraného pod problému zo zoznamu kandidátov. Počas spracovávanie môžeme dospieť k štyrom možným výsledkom. Nájdeme možné riešenie problému lepšie ako naša hraničná hodnota s_0, a tak zmeníme s_0 na nové riešenie a pokračujeme v spracovávaní zoznamu kandidátov. Ďalšia možnosť je že zistíme že daný pod problém nemá riešenie a teda ho vymažeme zo zoznamu(*prune*). Iná možnosť je ,že porovnáme hraničnú hodnotu pod problému s našou globálnou hraničnou hodnotou s_0, ktorá je doposiaľ najlepšie možné riešenie a pokiaľ hraničná hodnota pod problému je rovná prípadne väčšia (ak hľadáme

minimum), vieme že sa lepšie riešenie v pod probléme nenachádza a preto ho môžme vymazať zo zoznamu(*prune*). Posledná možnosť je, že na základe informácií, ktoré sme získali výpočtom, nemôžme pod problém vymazať, a preto sme nútení prejsť ku prvej fáze algoritmu a rozdeliť pod problém a jeho deti pridať do zoznamu ako aktívnych kandidátov. Iteratívne pokračujeme v algoritme až kým nie je zoznam kandidátov prázdny. Keď algoritmus skončí vieme, že naše doteraz najlepšie riešenie je v skutočnosti aj optimálne riešenie pre daný problém. Daná metóda je síce o niečo rýchlejšia, v závislosti na kvalitu pred počítanej hornej hranice na optimálne riešenie, ako klasický backtracking ale stále je použiteľná iba na inštancie s pomerne málo zákazníkmi. Pre veľké inštancie je výpočtový čas stále pre dynamické potreby nevhodný.

4.3 Lineárne programovanie

Mnohé algoritmy pre exaktný výpočet, ktoré dosahujú pomerne dobré výsledky, sú založené práve na technike lineárneho programovania. Pri tejto technike ide o vytvorenie polytopu za pomoci pomocou množiny rovností a nerovností charakterizujúcich daný problém. A následne v danom polytope hľadať bod aby naša definovaná cenová funkcia dosahovala minimum. Ak sa nám to podarí získali sme optimálne riešenie. Aby sme získali, čo najlepšie vlastnosti algoritmu, je dôležité navrhnúť množinu nerovností tak aby vzniknutý polytop bol čo najmenší. Musíme však dbať aj na to aby sme prílišným "priťahovaním" nerovností nevytesnili optimálne riešenie z polytopu.

Vzhľadom na to, že táto technika vykazuje dobré výsledky pri hľadaní exaktných riešení, tak sa tejto metóde venovalo mnoho vedcov a výskumníkov, čo dalo za vznik niekoľkým množinám nerovností, založených na rôznych charakteristikách. Tiež na následne hľadanie vhodného bodu existuje niekoľko techník. Technika lineárneho programovania je však natoľko obsiahla, že by si zaslúžila minimálne priestor samostatnej práce a my máme za cieľ sa venovať hlavne aproximačným a heuristickým metódam. Preto predstavíme na ilustráciu iba základnú množinu nerovností pre CVRP a viac sa tejto metóde

nebudeme venovať.

Predtým ako popíšeme samotné nerovnosti musíme si zadefinovať niektoré symboly. Takže pre každú hranu $e \in E$ vyjadruje celočíselná x_e počet koľkokrát bola daná hrana v riešení použitá. Ďalej $r(S)$ je minimálny počet vozidiel potrebný na obslúženie zákazníkov z podmnožiny S. Hodnota množiny $r(S)$ môže byť vypočítaná vyriešením BPP problému pre množinu S so zásobníkmi veľkosti kapacity vozidiel Q alebo použiť niektorú heuristiku. Nakoniec pre $S \subset V$, majme $\delta(S) = \{(i,j) : i \in S, j \notin S \text{ alebo } i \notin S, j \in S\}$. Ak $S = \{i\}$ tak píšeme $\delta(i)$ namiesto $\delta(\{i\})$. Teraz už môžme zapísať nerovnosti tak ako ich predstavili Laporte, Nobert a Desrochers.

$$\text{Minimalizujme} \sum_{e \in E} c_e x_e \text{ vzhľadom na} \tag{4.1}$$

$$\sum_{e \in \delta(i)} x_e = 2 \ (i \in V \setminus \{0\}) \tag{4.2}$$

$$\sum_{e \in \delta(0)} x_e = 2m \tag{4.3}$$

$$\sum_{e \in \delta(S)} x_e \geq 2r(S) \ (S \subseteq V \setminus \{0\}, S \neq \emptyset) \tag{4.4}$$

$$x_e \in \{0,1\} \ (e \notin \delta(0)) \tag{4.5}$$

$$x_e \in \{0,1,2\} \ (e \in \delta(0)) \tag{4.6}$$

Nerovnosť 4.2 zabezpečuje aby, každý zákazník bol navštívený práve raz. Nerovnosť 4.3 hovorí o vytvorení m ciest. Kapacitná nerovnosť 4.4 zabezpečuje súvislosť riešenia a obmedzuje početnosť hrán aby vyhovovala riešeniu. A nakoniec 4.5 a 4.6 zabezpečujú aby každá hrana medzi zákazníkmi bola použitá práve raz, prípadne pri sklade dvakrát.

Na záver treba iba podotknúť, že nakoľko sú tieto množiny nerovností pomerne početné, na ich vyhodnotenie sa používajú špeciálne optimalizované nástroje vytvorené pre účely lineárneho programovania.

Kapitola 5

Heuristiky

Ako sme spomenuli v predchádzajúcich kapitolách množstvo kombinatorických a optimalizačných problémov spadá do triedy NP-Ťažkých problémov. Pri ich riešení sa dostávame do situácie, že s narastajúcou veľkosťou vstupných hodnôt problému nám veľmi rýchlo, často exponenciálne, rastie aj priestor riešení problému. Na vypočítanie optimálneho riešenia musíme tento priestor prehľadať, čo sa stáva vysoko časovo náročným. Ak si však tento čas v našej aplikácií nemôžme dovoliť, musíme upustiť od exaktných metód a na rad prichádzajú heuristiky.

Heuristické metódy a algoritmy sa využívajú na nájdenie riešenia, ktoré nemá moc ďaleko k optimálnemu riešeniu a jeho výpočtový čas nie je dlhý. Tieto metódy hľadajú riešenie len v relatívne malej podmnožine priestoru riešení. Toto obmedzenie im teda znižuje výpočtový čas na úkor možnosti nájdenia lepšieho riešenia ktoré sa nachádza mimo tejto podmnožiny. Mnoho heuristík sa dokonca uspokojuje s nesprávnymi riešeniami, ktoré samozrejme nie sú moc ďaleko od tých správnych. Heuristiky okrem toho, že pracujú samostatne sú často krát dopĺňané o vylepšovacie algoritmy. Tento typ algoritmov upravuje riešenie nájdené heuristikami s cieľom priblížiť sa k optimálnemu riešeniu.

My v tejto kapitole predstavíme základné triedy heuristík pre VRP problémy spolu s ich zástupcami. Pre každú skupinu urobíme aj experimentálne

testy na navrhnutej testovacej množine, ktorá je popísaná v predchádzajúcom texte. Na základe získaných výsledkov porovnáme jednotlivé algoritmy daných tried a následne zanalyzujeme ich charakter. Tiež spomenieme triedu vylepšovacích algoritmov, ktoré sú častou súčasťou mnohých heuristík.

5.1 Heuristiky pre Vehicle Routing Problem

Pre spomenuté vlastnosti heuristík a ich význam pri riešení NP-Ťažkých problémov je prirodzené, že sa využívajú aj pri riešení VRP problému. Je mnoho komerčných produktov, ktorých súčasťou je riešenie VRP problému, napr. rôzne logistické programy pre firmy na riadenie zásobovania materiálov a následného distribuovania tovaru. A práve v týchto produktoch je potreba získavať riešenie v reálnom čase, a preto sa v nich najčastejšie uplatňujú heuristické metódy. Pri heuristikách riešiacich VRP môžme rozoznať niektoré základné skupiny algoritmov. Základné tri triedy algoritmov, ktorým sa v texte budeme venovať sú :

- **Konštruktívne metódy**
- **Vkladacie algoritmy**
- **2-fázové algoritmy**

Každá skupina algoritmov je charakteristická svojím prístupom k hľadaniu riešenia problému. Prejdime teda ku konkrétnej charakteristike jednotlivých tried.

5.2 Konštruktívne metódy

Algoritmy tejto triedy sú špecifické tým, že vytvárajú riešenie postupne. Zväčša ide o vytvorenie priamych ciest pre každého zákazníka, prípadne skupinu zákazníkov a následne spájanie týchto ciest podľa nijakých kritérií do novej cesty. Tieto kritéria sú závislé na hodnotách ako cena spájaných ciest,

veľkosť požiadaviek zákazníkov na cestách, či kapacita vozidiel. Týmto postupným spájaním konštruujeme cesty do výsledného riešenia. Pri spájaní ciest musíme však dbať na to aby žiadna z novo vytvorených ciest nebola v rozpore s požiadavkami problému a zároveň dbáme na cenu konštruovaného riešenia.

5.2.1 Clark and Wright Savings Algoritmus

Určite jeden z najznámejších heuristických algoritmov pre VRP je práve *Clark and Wright - Savings algoritmus*. Ide o konštrukčnú metódu, kde kritériom spájania dvoch ciest je úspora (*saving*), ktorá týmto spojením vznikne. Algoritmus sa využíva pre inštancie problému kde nie je pevne daný počet zásobovacích vozidiel. A navyše funguje rovnako dobré pre VRP s orientovanými ako aj s neorientovanými hranami.

Rozlišujeme dve základné verzie algoritmu a to sekvenčnú a paralelnú. Pri sekvenčnej verzii budujeme vždy jednu aktuálnu cestu pokiaľ je to možne, teda nie je naplnená kapacita alebo neexistujú žiadne cesty ktoré by sa dali pripojiť. V paralelnej verzií naopak, spájame cesty podľa najväčšej úspory a teda vytvárame viac ciest súčasne.

Prácu algoritmu môžme popísať v dvoch základných krokoch. Prvý krok je rovnaký ako pre sekvenčnú tak aj pre paralelnú verziu. Vypočítame v ňom úspory pre všetky dvojice zákazníkov $i, j = 1, .., n$ $i \neq j$ podľa pravidla :

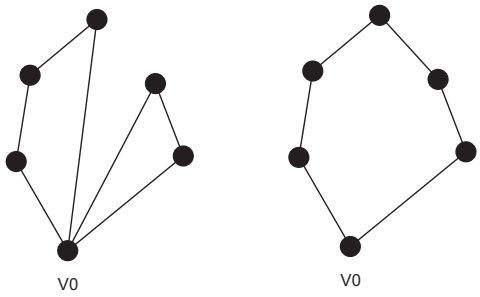
$$S(i,j) = 2d(0,i)+2d(0,j)-[d(0,i)+d(i,j)+d(0,j)] = d(0,i)+d(0,j)-d(i,j) \tag{5.1}$$

,kde $d(i,j)$ je cena cesty medzi zákazníkmi i,j a 0 reprezentuje sklad. Treba si uvedomiť, že $S(i,j)$ je úspora získaná pri kombinácií ciest obsahujúcich i a j do jednej cesty. Samozrejme cesty môžu byť skombinované iba ak sa tým neporuší žiadna požiadavka definície problému. Po vypočítaní sú úspory usporiadané v nerastúcom poradí. V tomto kroku si ešte vytvoríme n základných ciest $(0,i,0)$ $i = 1, .., n$, ktoré zásobujú jednotlivých zákazníkov.

Sekvenčná verzia pokračuje druhým krokom v budovaní aktuálnej cesty $(0,i,..,j,0)$ tak, že vyberie najväčšiu úsporu $S(k,i)$ alebo $S(j,l)$, takú aby

mohla byť budovaná cesta rozšírená pripojením cesty obsahujúcej hranu $(k, 0)$ alebo $(0, l)$. Po pripojení opakujeme výber úspor až, kým je možnosť rozširovať aktuálnu cestu. Inač vyberáme inú cestu za aktuálnu a pokračujeme jej rozširovaním. Ak neexistuje žiadna vhodná cesta algoritmus končí.

Druhý krok pre paralelnú verziu pozostáva z postupného prechádzania zoznamu úspor zvrchu nadol a pre každú úsporu $S(i, j)$ vykonáme nasledovné. Ak existujú dve cesty, pričom jedna obsahuje hranu $(i, 0)$ a druhá hranu $(j, 0)$ tak ich môžeme spojiť a vytvoriť novú cestu, pričom sme získali úsporu $S(i, j)$. Novú cestu však môžme vytvoriť iba ak množstvo objednávok zákazníkov na nej neprekračuje kapacitu zásobovacieho vozidla. Kapacita všetkých vozidiel je rovnaká preto nemusíme brať ohľad na to ktoré vozidlo je ktorej ceste priradené. Cesty spojíme odstránením hrán $(i, 0)$,$(j, 0)$ a pridaním hrany (i, j). Algoritmus končí ak prejdeme celý zoznam úspor.



Obr. 5.1: Savings algoritmus

Spomenuté algoritmy sú originálnou verziou tejto konštrukčnej metódy, tak ako bola predstavená. Časom však vznikli mnohé obmeny, za cieľom vylepšenia získaných riešení. Jednou z charakteristík, ktorú môžme pri originálnom riešení sledovať je, že na začiatku produkuje dobré cesty vzhľadom na ich cenu, lebo začína s maximálnymi úsporami. Postupným výpočtom však začne vytvárať cesty, ktoré nie sú pre výsledok až tak zaujímavé. Gaskell a Yellow predstavili algoritmus, ktorý sa snaží tomuto predísť. Tento algorit-

mus je založený na zovšeobecní úspor, zmenou ich výpočtu nasledovne :

$$S(i,j) = d(0,i) + d(0,j) - \lambda d(i,j) \tag{5.2}$$

,kde všetky členy majú rovnaký význam ako v originálnom výpočte úspor, a λ je parameter. Je zrejme, že pre hodnotu $\lambda = 1$ ide o rovnaký výpočet úspor ako v originálnom algoritme. Parameter λ zohľadňuje tvar budovaných ciest a to tak, že čím je λ väčšia, tým je aj väčší dôraz kladený na vzájomnú vzdialenosť vrcholov, ktoré majú byť spojené hranou pre vytvorenie novej cesty.

5.2.2 Experimentálne výsledky

V tejto časti venujeme priestor, porovnaniu a rozanalyzovaniu nameraných hodnôt pre spomenuté algoritmy. V našom softvérovom diele sme implementovali sekvenčný, paralelný Clark and Wright algoritmus a rovnako aj algoritmus s parametrom pre výpočet úspor pre obidve tieto verzie. Tieto algoritmy sme testovali na jednotlivých inštanciách našej testovacej množiny. Namerané hodnoty sme zaznamenali do tabuľky 5.1. Hodnota v tabuľke vyjadruje celkovú cenu riešenia pre danú inštanciu a použitú metódu riešenia a hodnota za hviezdičkou vyjadruje počet použitých vozidiel.

Už s prvého pohľadu je jasné, že paralelná verzia algoritmu dosahuje pri všetkých inštanciách lepšie výsledky ako verzia sekvenčná. Výraznejší rozdiel až do 10% je pri inštanciách s menším počtom zákazníkov, resp. keďže ide o rozmiestnenie na ohraničenej ploche, hrá tu významnú rolu aj hustota grafu. Hlavný dôvod pri sekvenčnej verzií je ten, že ako náhle vytvárame jednu cestu až do naplnenia kapacity vozidlá, tak zo začiatku máme veľký výber kvalitných úspor. Ak sa však blížime k hraničnej kapacite vozidla, tak získavame možnosť vložiť zákazníkov, ktorý sú veľmi nevýhodný ale vzhľadom na to, že je to jediná možnosť aj takýto zákazníci sú nakoniec do cesty vložený.

Čo sa týka algoritmu s parametrom pre váhu vzdialeností jednotlivých zákazníkov sme experimentálne zistili, že hodnoty nízke, teda $\lambda \leq 0.5$ nám riešenie len zhoršujú a preto sú nevhodné. Zaujímavejšie výsledky sme získali

pre hodnoty $\lambda = 0.5$ a hlavne $\lambda = 0.7$. Preto v tabuľke uvádzame výsledky práve pre tieto hodnoty parametra λ. Ak parameter stúpal vyššie k hodnote $\lambda = 1$, tak sme sa iba približovali k hodnotám nameraným pre algoritmy bez parametra rovnako pre sekvenčnú ako aj pre paralelnú verziu. Z tabuľky možno vidieť, že pre hodnotu $\lambda = 0.7$ sme dosahovali pre naše inštancie najlepšie výsledky. Nebolo to však pravidlom. Ale pri hodnote $\lambda = 0.5$ sme zväčša získavali riešenia o niečo horšie ako riešenia algoritmov bez parametra. Môžme teda povedať ,že najlepšie výsledky pre tento algoritmus sa dajú očakávať pri paralelnej verzií s parametrom, kde hodnota parametra je z intervalu $[0.6, 0.8]$.

Počas meraní sme ešte pozorovali úskalie parametrickej verzie hlavne pri inštanciách so zákazníkmi rozmiestnenými na sústredných kružniciach, prípadne výsekoch. Tu nám oslabenie významu na vzdialenosť medzi vrcholmi, spôsobovala, že vytváranie ciest preskakovalo medzi jednotlivými kružnicami namiesto toho aby smerovalo po obvode pomerne husto obsadených kružníc. Toto preskakovanie sa nám potom prejavilo na zvýšení ceny celkového riešenia. Preto na takomto type inštancií si počínali lepšie algoritmy s väčšou hodnotou λ.

5.3 2-fázové algoritmy

Ďalšou triedou algoritmov sú takzvané 2-fázové algoritmy. Hlavnou črtou týchto algoritmov je rozdelenie ich riešenia na dve fázy. Tieto prirodzene vyplývajú z charakteru VRP problémov. Vieme že ide o problém zahrňujúci v sebe dva samostatné ťažké problémy a to BPP problém a TSP problém. A teda každá fáza sa venuje riešeniu práve jedného z týchto problémov. A po skončení jednej sú sprostredkované jej výsledky druhej fáze a tá s nimi pracuje. Konkrétne ide o tieto dve základné fázy :

- Rozdelenie vrcholov do množín tak aby ich objednávka neprekročila kapacitu vozidiel.
- Vytvorenie samotných ciest pre zákazníkov z jednotlivých množín.

Testovacia množina	**Sekvenčný**	**Sekvenčný** $\lambda = 0.5$	**Sekvenčný** $\lambda = 0.7$	**Paralelný**	**Paralelný** $\lambda = 0.5$	**Paralelný** $\lambda = 0.7$
R1-50-350	18399*15	18169*14	18115*14	16899*15	17246*15	17007*15
R2-100-350	38995*28	39943*28	38638*28	36630*28	37262*27	36396*27
R3-150-350	58502*46	60730*46	57914*45	55327*48	56975*45	54702*45
R4-200-350	74391*63	77730*63	75327*63	70740*66	71091*64	70708*64
R5-50-250	11112*6	10946*7	10378*6	10229*6	10757*6	10681*6
R6-100-250	15979*12	16968*13	15958*12	14814*12	15646*12	15399*12
R7-150-250	23118*18	27686*18	23154*18	21391*17	22809*17	21933*17
R8-200-250	29313*23	30456*23	30050*24	28312*23	29038*23	28386*23
R9-500-1000	63787*8	70308	66371*8	58407*8	63843*8	60929*8
R10-500-150	202804*52	211315*52	207728*52	196797*51	201733*52	199306*51
C1-100-100	9259*13	9744*14	9283*13	8584*14	8880*13	8705*13
C2-100-100	14861*14	14876*14	15473*15	13756*13	13708*13	13704*13
C3-100-100	9950*14	10577*14	10312*14	9561*13	9672*14	9471*14

Tabuľka 5.1: Výsledky Clark and Wright heuristiky

Podľa poradia fáz ďalej rozlišujeme dve základné triedy 2-fázových algoritmov. Opäť uvádzame anglické pomenovanie pre jeho výstižnosť a pridávame krátky popis pre jasné pochopenie.

- **Cluster first, route second** Algoritmy tejto triedy v prvej fáze rozdeľujú zákazníkov do množín a v druhej tvoria cesty.
- **Route first, cluster second** Algoritmy tejto triedy naopak v prvej fáze vytvárajú cesty, a vo fáze druhej ich upravujú, tak aby objednávky zákazníkov na jednotlivých cestách neprekračovali kapacitu vozidiel.

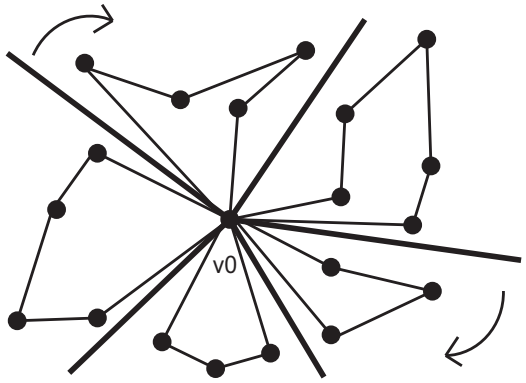
Existuje aj niekoľko hybridných algoritmov, kde sa jednotlivé fázy prelínajú, prípadne sú fázy oddelené ale je umožnený návrat k predošlej fáze.

5.3.1 Sweep algoritmus

Prvým zástupcom triedy 2-fázovych algoritmov, ktorému sa budeme venovať je Sweep algoritmus. Tento algoritmus patrí medzi *Cluster first, route second* algoritmy, a teda v prvej fáze sa rozdelia zákazníci do jednotlivých množín tak aby sa neporušila podmienka kapacity pre vozidlá, ktoré budú zákazníkov z

daných množín zásobovať. V druhej fáze sa vypočítajú cesty pre každé vozidlo tak, aby ich cena bola čo najoptimálnejšia. Sweep algoritmus sa využíva na planárne inštancie VRP, a to z dôvodu, že prvú fázu má založenú na geometrickej reprezentácií problému.

Rozdelenie vrcholov v prvej fáze je vytvorené tak, že rotujeme pomyslenú polpriamku so začiatkom v sklade a vytvárame oblasti ktorých celková objednávka nepresahuje kapacitu vozidla. Teda postupne ako pretíname polpriamkou zákazníkov tak im priraďujeme vozidlo, ktoré ich obslúži. Ak by bola kapacita vozidla prekročená, tak pokračujeme v rotácií ale zákazníkov pridávame do nového vozidla. Takto pokračujeme, až kým nie je každému zákazníkovi priradené vozidlo, ktoré ho obslúži. Niektoré implementácie môžu obsahovať aj následnú optimalizačnú fázu založenú na výmene zákazníkov v susedných oblastiach.



Obr. 5.2: Sweep algoritmus

Jednoduchá implementácia takéhoto delenia, je že si pre každého zákazníka v_i vypočítame jeho polárne súradnice (Φ_i, π_i) (*polar coordinate*), kde Φ_i je jeho uhol a π_i vzdialenosť od počiatku. Na základe týchto súradníc si zákazníkov zoradíme a postupným prechádzaním rozdelíme do množín pre jednotlivé vozidlá s určenou kapacitou.

Druhá fáza pozostáva z vytvorenia jednotlivých ciest pre naše rozdelenie. Ide teda o inštancie TSP problému. A na riešenie tohto problému množstvo exaktných ako aj heuristických metód. Ale rovnako ide o problém ktorý by si

vyžadoval samostatnú publikáciu a je nad rámec nášho textu. Aby sme však neostali čitateľovi nič dlžný uvedieme aspoň krátky prehľad TSP na konci tohto oddielu.

5.3.2 Fisher a Jaikumar algoritmus

Druhá heuristika z triedy 2-fázových algoritmov, ktorú si predstavíme patrí rovnako medzi *Cluster first, route second* algoritmy. Hlavný rozdiel oproti Sweep algoritmu je ale jej prístup k rozdeľovaniu zákazníkov. Nejde tu o geometrické rozdelenie ale vytvára množiny na základe riešenia problému generalizovaného problému priradenia (*Generalized Assignment Problem*). V tomto probléme ide o priradenie množiny úloh jednotlivým agentom, pričom každá úloha má svoju váhu a cenu. Je to opäť pomerne rozsiahly problém, a preto pre bližšie informácie viď. použitú literatúru. GAP problém spadá do množiny ťažkých problémov, a preto na jeho výpočet použijeme heuristiku. Rozdiel oproti predošlému algoritmu je aj vtom, že si vyžaduje ako vstupnú hodnotu aj konštantný počet vozidiel k. Spomínaný GAP potom rieši pre tento počet agentov/vozidiel.

Teraz priblížime prácu prvej fázy algoritmu, starajúcej sa o rozdelenie zákazníkov v troch základných krokoch. Prvý krok je inicializačný a vyberáme v ňom k zákazníkov $v_i i \in \{1, .., k\}$, ktorý budú slúžiť ako jadro (*seed*) pre vytvárané množiny pod označením j_k. V druhom kroku vypočítame pre zákazníkov cenu ich priradenia do každej z množín, reprezentovaných vybranými zákazníkmi z prvého kroku, podľa pravidla 5.3. Uvádzame pravidlo pre orientované grafy, lebo je všeobecnejšie a pokrýva aj neorientované grafy.

$$d_{ij_k} = \min\{c_{0i} + c_{ij_k} + c_{j_k 0}, c_{0j_k} + c_{j_k i} + c_{i0}\} - (c_{0j_k} + c_{j_k 0}) \tag{5.3}$$

Algoritmus pokračuje v tretím krokom v ktorom počíta inštanciu GAP problému s nasledujúcimi hodnotami. Pre cenu úlohy i pre daného agenta k berie hodnotu d_{ij_k}, ako váhu úlohy sa berie veľkosť objednávky zákazníka a kapacita agenta sa rovná kapacite vozidiel Q.

Druhá fáza opäť pozostáva z vytvorenia jednotlivých ciest pre naše rozdelenie a teda vyriešenia jednotlivých inštancií TSP. Pre TSP platí to čo bolo napísané pri prvom zástupcovi 2-fázových algoritmov.

5.3.3 Traveling Salesman problém

Definíciu TSP sme predstavili v predchádzajúcom texte a teda tu sa k nej už nebudeme vracať. V tejto časti iba vymenujeme základné prístupy k riešeniu tohto ťažkého problému a to z dôvodu, že sa tieto metódy priamo vyskytujú aj pri riešení VRP problémov, konkrétne pri 2-fázových algoritmoch.

Prvou množinou rovnako ako pre VRP sú exaktné algoritmy na výpočet optimálneho riešenia. Nasledujúci zoznam obsahuje základné prístupy aj s krátkou charakteristikou.

- **Branch and Bound** algoritmy sa úspešne používajú na riešenie TSP inštancií s počtom 40-60 miest.
- **Lineárne programovanie** vie dosiahnuť optimálne riešenia pre inštancie s počtom miest v intervale 120-200.
- **Branch, Bound and Cut** je metóda rovnako založená na lineárnom programovaní a vie si poradiť s inštanciami s počtom miest až 5000. Dokonca najnovšie implementácie tejto metódy sa vysporiadali s inštanciou obsahujúcou 33810 miest.

Tu môžme vidieť, že exaktné metódy pre VRP a TSP sú založené na rovnakých metódach, čo potvrdzuje blízku príbuznosť týchto dvoch ťažkých problémov.

V ďalšom zozname ešte spomenieme niektoré triedy heuristík, ktoré pri riešia obrovské inštancie TSP problému v dobrom čase a dosahujú s vysokou pravdepodobnosťou výsledky, ktoré sa od optimálneho riešenia často líšia iba o 2-3%.

- **Konštruktívne heuristiky** zväčša založené na greedy rozhodovaní sa pre najbližšie mesto, prípadne lokálne najlepšiu cestu. To že ide o greedy rozhodovanie im dáva výborný čas ale nie moc dobré riešenia.
- **Iteratívne vylepšenia** ide o obdobu vylepšovacích algoritmov, ktoré sú predstavené ďalej v texte.
- **Znáhodnené vylepšenie** sú obdobou vylepšovacích algoritmov ale obsahujú aj prvky lokálneho prehľadávania používaného pre meta-heuristiky.
- ...

Existuje množstvo ďalších metód pre riešenie TSP, ten ale nie je predmetom tejto diplomovej práce a nebudeme sa mu ďalej venovať.

5.3.4 Experimentálne výsledky

Opäť sa budeme venovať nameraným výsledkom pre našu testovaciu množinu a spomenuté algoritmy. Namerané hodnoty sú uvedené v tabuľke 5.2. Už na prvý pohľad nám udrú do oči veľmi zlé výsledky Fisher a Jaikumar algoritmu pre niektoré inštancie. Tu ale musíme podotknúť, že obidve fázy algoritmu sú samostatné ťažké problémy. Pri našej implementácií však za cieľom udržať nízky čas výpočtu algoritmu používame pre obidve časti len aproximačné techniky. Teda už rozdelenie do jednotlivých množín je vzdialené od optimálneho riešenia. A spustenie druhej fázy, ktorá je tiež aproximačná technika na už nie optimálnych hodnotách nám posunie riešenie opäť ďalej od optima. Musíme teda pripustiť, že výber našich aproximačných techník pre jednotlivé fázy nebol vôbec vhodný a preto aj význam samotnej metódy sa úplne stratil.

Prejdime teraz na hodnotenie Sweep algoritmu. Namerané hodnoty sa vyznačujú tým, že počet vozidiel potrebných na zásobenie všetkých zákazníkov oproti konštrukčným heuristikám vzrástol. Je to dôsledkom toho, že vytváranie množín zákazníkov je založený na geometrickej polohe, a teda najme pri objednávkach z veľkého intervalu je často kapacita vozidiel zlé využívaná.

Jemnú optimalizáciu prináša výmena zákazníkov medzi susednými množinami. To že vzrástol počet vozidiel sa prejavilo aj na zhoršení celkovej ceny riešenia. Dobré výsledky sme však namerali inštanciách s menším počtom zákazníkov, kedy graf reprezentujúci problém nie je taký hustý. A tiež pri inštancií reprezentujúcej mestá okolo skladu, čo pripisujeme tomu, že často spadalo celé mesto obsahujúce viacero zákazníkov geometricky do jedného sektora. A teda vozidlo, ktoré bolo vybrané pre daný sektor zväčša zásobilo zákazníkov v danom meste a nerobilo zbytočné presuny medzi vzdialenými mestami.

Testovacia množina	**Sweep alg.**	**F&J alg.**
R1-50-350	18526*16	19791*16
R2-100-350	43703*31	44092*31
R3-150-350	69229*55	63253*45
R4-200-350	91739*73	78600*63
R5-50-250	11134*6	14193*6
R6-100-250	16895*12	21682*12
R7-150-250	24898*18	32090*18
R8-200-250	33582*23	32937*23
R9-500-1000	67293*8	92103*8
R10-500-150	250176*53	300254*52
C1-100-100	8113*10	9345*14
C2-100-100	13595*10	15934*14
C3-100-100	8841*10	11068*14

Tabuľka 5.2: Výsledky Sweep a Fisher a Jaikumar algoritmov

5.4 Vkladacie heuristiky

Poslednou spomenutou triedou algoritmov sú vkladacie algoritmy. Tieto algoritmy sú založené na postupnom vyberaní vrcholov z množiny nepriradených vrcholov a ich vkladanie do jednotlivých ciest.

5.4.1 Základné vkladanie

Nasledujúci algoritmus je základným predstaviteľom triedy vkladacích heuristík. Ukážeme si na ňom základnú schému väčšiny vkladacích heuristík. Je to teda akýsi základný stavebný kameň pre algoritmy z tejto triedy. Rovnako ako pri konštrukčných heuristikách aj tu rozlišujeme sekvenčnú a paralelnú verziu algoritmu. Pripomeňme si teda, že pri sekvenčnej verzií začíname budovať novú cestu až vtedy, keď je konštrukcia predchádzajúcej cesty ukončená. Kdežto pri paralelnej verzií algoritmu sú konštruované viaceré cesty naraz.

Vkladacie algoritmy majú dve základné fázy a to výber vrcholu pre vloženie a samotné vloženie vrcholu do cesty. Obidve tieto fázy majú svoje špecifiká a kritéria. Pre výber zákazníka, ktorý má byť vložený, existuje niekoľko rôznych prístupov. My si niektoré základné teraz spomenieme.

- **Najbližší zákazník** Pri tomto prístupe je vybraný zákazník, ktorý je spomedzi nevložených zákazníkov najbližšie k skladu.
- **Najvzdialenejší zákazník** Pri tomto prístupe je vybraný zákazník, ktorý je spomedzi nevložených zákazníkov najďalej od skladu.
- **Najlepší zákazník** Pri tomto prístupe je vybraný zákazník, ktorý ma spomedzi nevložených zákazníkov najlepšiu cenu, vzhľadom na kritérium vkladania.

Rovnako druhá fáza a teda samotné vkladanie môže mať rôzne kritéria na základe ktorých je vybraný zákazník do cesty vkladaný. Tu sú niektoré z nich :

- **Najlepšie pridanie** Pri tomto kritériu je výpočet ceny zákazníka na vloženie rovný jeho vzdialenosti od zákazníkov, ktorí už v danej ceste sú.
- **Najlepšie vloženie** Pri tomto kritériu je výpočet ceny zákazníka na vloženie rovný jeho vzdialenosti od zákazníkov, ktorí už v danej ceste sú a navyše sa berie do úvahy jeho vzdialenosť od skladu.

- **Ľubovoľná funkcia** Tu môže ísť o rôzne funkcie ktoré zobrazujú zákazníkov na hodnoty, a vkladaný je zákazník s minimálnou prípadne maximálnou hodnotou.

My si teraz predstavíme paralelnú verziu základného vkladacieho algoritmu. Algoritmus pracuje s množinou zákazníkov N, ktorý ešte nie sú priradený v žiadnej ceste a s množinou ciest R. Na začiatku obsahuje množina ciest R iba prázdne cesty a množina N obsahuje všetkých zákazníkov. Musíme si uvedomiť, že pri vkladacím algoritme, musí množina ciest vždy obsahovať aspoň jednu prázdnu cestu. Táto podmienka je nutná preto aby sa nám podarilo vložiť všetkých zákazníkov, lebo ak nie je žiadna možnosť vložiť zákazníka do už vybudovaných ciest, napríklad jeho objednávka v spojení s objednávkami zákazníkov na danej ceste presahuje kapacitu vozidla, musíme mu priradiť cestu novú. Tu je vidno, prečo pri vkladacích heuristikách je počet vozidiel potrebných na zásobenie všetkých zákazníkov v danej inštancií problému voľný. Algoritmus pracuje tak, že vyberie zákazníka z množiny N podľa zvoleného prístupu a vloží ho do niektorej cesty, tak aby sa cena ciest zvýšila čo najmenej. Pokiaľ je vloženie zákazníka do novej cesty lacnejšie ako jeho vloženie do aktuálnych ciest, tak je vložený do cesty novej. Po každom vložení ešte algoritmus obnoví pre danú cestu veľkosť objednávok, ktoré na danej ceste sú a pokračuje, výberom a vkladaním ďalších ešte nevložených zákazníkov. Algoritmus končí ak sú všetci zákazníci priradení do niektorej z ciest. Sekvenčnú verziu algoritmu si vie každý určite predstaviť, a preto ju tu nebudeme popisovať.

5.4.2 Mole & Jameson vkladací algoritmus

Ďalším zástupcom vkladacích algoritmov je *Mole & Jameson* algoritmus. Ide o sekvenčný algoritmus, teda jednotlivé cesty sú vytvárané jedna po druhej. A rovnako ako pri predchádzajúcom zástupcovi aj tu je počet vozidiel na zásobenie všetkých zákazníkov nešpecifikovaný. Hlavnou črtou tohto algoritmu je že na výber miesta pre vloženie zákazníka používa dva parametre. Rozširovanie každej cesty je teda závislé na kritériách 5.4 a 5.5 v ktorých sú λ a μ

parametre :

$$\alpha(i,k,j) = c_{ik} + c_{kj} - \lambda c_{ij} \tag{5.4}$$

$$\beta(i,k,j) = \mu c_{0k} - \alpha(i,k,j). \tag{5.5}$$

Prácu algoritmu môžme popísať v troch základných krokoch. Prvý krok je inicializačný a zároveň aj terminačný. A teda ak sú všetci zákazníci priradený k nijakej ceste algoritmus končí. Inak zostrojíme počiatočnú cestu (v_0, k, v_0),ktorá sa stane aktuálnou, kde v_0 je sklad a k je niektorý nepriradený zákazník. V druhom kroku je hlavným cieľom priradiť ďalšieho zákazníka. A to sa deje nasledovným spôsobom. Pre každého nepriradeného zákazníka k vypočítame cenu, ktorú stojí jeho vloženie ako $\dot{\alpha}(i_k, k, j_k) = min\{\alpha(r,k,s)\}$ pre všetky susedné vrcholy r a s v aktuálnej ceste, kde i_k a j_k sú zákazníci medzi ktorých treba vložiť zákazníka k, aby výsledkom bola cesta s najlepšou možnou cenou. Teraz vyberieme spomedzi nepriradených zákazníkov zákazníka $\dot{k}$, ktorý je pre nás najvýhodnejší a to nasledovne $\dot{\beta}(i_{\dot{k}}, \dot{k}, j_{\dot{k}}) = max\beta\{(i_k, k, j_k)\}$, kde k je z množiny zatiaľ nepriradených zákazníkov, ktorý môžu byť vložený. Ak neexistuje žiadny vhodný adept pokračujeme krokom jeden. Ak adepta $\dot{k}$ máme vložíme ho do aktuálnej cesty medzi zákazníkov $i_{\dot{k}}$ a $j_{\dot{k}}$. Posledný krok je optimalizačný krok. Zavoláme teda optimalizačný algoritmus na aktuálnu cestu a pokračujeme krokom dva.

5.4.3 Christofides, Mingozzi a Toth vkladací algoritmus

Pri našom treťom zástupcovi vkladacích algoritmov si predstavíme postupne sekvenčnú a potom aj jeho paralelnú verziu. Tento algoritmus rovnako využíva dva nastavovateľné parametre λ a μ ale v princípe ide o akúsi sofistifikovanejšiu dvoj fázovú vkladaciu heuristiku. To že ma algoritmus dve fázy ho rovnako radí aj medzi dvojfázové heuristiky. My ho spomíname v tejto časti heuristík lebo na budovanie ciest používame vkladanie a svojimi parametrami je podobný s predchádzajúcim vkladacím algoritmom.

Najskôr sa pozrieme sekvenčné konštruovanie ciest. Pre algoritmus popíšeme jeho základné kroky. Prvý krok bude inicializačný a vytvoríme v ňom

prvú prázdnu cestu, ktorá sa stane aktuálnou. V druhom kroku priradíme ľubovoľný ešte nepriradeného zákazníka i_k do aktuálnej cesty a tým ju nastavíme na cestu k. Následne vypočítame cenu vloženia pre všetky nepriradené vrcholy i ako $\delta_i = c_{0i} + \lambda_{cii_k}$. V treťom kroku vyberieme zákazníka $\dot{i}$ spomedzi všetkých nepriradených zákazníkov S_k, ktorý môžu byť vložené, nasledovne $\delta_{\dot{i}} = min(\{\delta_i\}); i \in S_k$. Vložíme zákazníka $\dot{i}$ do cesty k a optimalizujeme ju pomocou 3-Opt algoritmu. Opakujeme tretí krok pokiaľ je možne vložiť nových zákazníkov do cesty k. Štvrtý krok ukončí algoritmus ak sú už všetci zákazníci priradený k nijakej ceste. Ak nie tak začneme budovať novú cestu $k = k + 1$ a pokračujeme krokom dva.

Teraz sa pozrieme na verziu s paralelnou konštrukciou ciest. Ju treba pripomenúť, že ako sme spomenuli ide o dvojfázový algoritmus. Nie je to ale presný zástupca dvojfázových algoritmov ako boli spomenuté v predchádzajúcom texte a teda, že prvá fáza rozdelí zákazníkov do skupín a druhá konštruuje cesty pomocou riešenia TSP pre jednotlivé skupiny. Tento algoritmus využíva ako prvú fázu svoju sekvenčnú verziu, popísanú v krokoch jedna až štyri. A pokračuje druhov fázou ktorú teraz popíšeme. V piatom kroku inicializujeme k ciest $R_k = (0, i_t, 0), t = 1, .., k$, kde k získame ako počet ciest vytvorených v poslednom kroku prvej fázy. Označme $J = \{R_1, .., R_k\}$. Pokračujeme šiestym krokom, v ktorom cenu priradenia zákazníkov. Pre každú cestu $R_t \in J$ a pre každého ešte nepriradeného zákazníka i vypočítame $\epsilon_{ti} = c_{0i} + \mu c_{ii_t}$ a $\varepsilon_{t^*i} = \min_t\{\varepsilon_{ti}\}$. Následne priradíme zákazníka i do cesty R_{t^*} a opakujeme krok šesť pokiaľ nebudú všetci zákazníci priradený do nijakej cesty. Krok sedem pokračuje vo výpočte ceny vloženia zákazníka. Vezmeme ľubovoľnú ceste R_t a upravíme $J = J \backslash \{R_t\}$. Pre každý vrchol i priradený k ceste R_t vypočítame $\varepsilon_{t'i} = \min_{R_t \in J}\{\varepsilon_{ti}\}$ a $\rho_i = \varepsilon_{t'i} - \varepsilon_{ti}$. V kroku osem vložíme do cesty R_t vrchol i^* pre ktorý platí $\rho_{i^*} = \max_{i \in S_t}\{\rho_i\}$, kde S_t je množina nepriradených zákazníkov asociovaných s cestou R_t a môžu byť úspešne do nej vložený. Následne optimalizujeme cestu optimalizačným algoritmom a opakujeme krok osem pokiaľ môžme do cesty R_t vkladať zákazníkov. Deviaty krok je terminačný a teda ak platí $|J| \neq \emptyset$ pokračujeme

krokom šesť. Ak sú všetci zákazníci priradený skonči. Ak nám ostali nijaké nepriradené vrcholy, tak pokračujeme vytváraním ciest v kroku jeden prvej fázy.

5.4.4 Experimentálne výsledky

Rovnako aj pre vkladacie algoritmy sme urobili výpočty na našej testovacej množine. Keďže jednou zo základných a charakteristických vlastnosti týchto algoritmov je výber zákazníkov tak sa pozrieme na vkladacie algoritmy s rôznym kritériom výberu. Testy sme teda robili na šiestich variantoch vkladacieho algoritmu. Konkrétne išlo o paralelne a sekvenčné verzie vkladacieho algoritmu, ktorý pre má kritérium vkladania najlepšie pridanie. Pre výber zákazníka sme postupne testovali najbližší výber (NEAR), najvzdialenejší (FAR) výber a nakoniec najlepší výber (BEST). Namerané výsledky sú v tabuľke **??**.

Testovacia množina	**Sekvenčný BEST**	**Sekvenčný NEAR**	**Sekvenčný FAR**	**Paralelný BEST**	**Paralelný NEAR**	**Paralelný FAR**
R1-50-350	22377*14	25637*14	25885*14	21041*15	24513*14	25139*14
R2-100-350	47366*27	58097*27	49874*27	45488*27	48611*27	45856*28
R3-150-350	72626*46	85829*45	80606*45	67721*46	75329*45	74244*46
R4-200-350	93252*62	110766*61	100990*61	85773*63	98998*63	92842*61
R5-50-250	12074*6	16083*6	15569*6	12074*6	16083*6	15569*6
R6-100-250	18164*12	28465*12	29537*12	18164*12	28387*12	29537*12
R7-150-250	26474*18	44267*17	42778*17	26474*18	43763*17	42778*17
R8-200-250	33609*23	55938*23	57220*23	33609*23	55938*23	57115*23
R9-500-1000	74230*8	103698*8	99487*8	74230*8	103698*8	98684*8
R10-500-150	211775*51	344023*52	347440*52	211775*51	339399*52	346216*52
C1-100-100	8112*10	15685*10	15886*10	8112*10	15685*10	15886*10
C2-100-100	12132*10	14729*10	15809*10	12132*10	14729*10	15809*10
C3-100-100	9501*10	11771*10	11824*10	9501*10	11771*10	11824*10

Tabuľka 5.3: Výsledky pre vkladacie heuristiky

Z nameraných výsledkov je jasné, že najlepší spôsob pre výber zákazníka je podľa kritéria vkladania. Tento výber si síce vyžaduje prídavné výpočty ale tie sa veľmi efektné prejavia na cene riešenia. Najbližší a najvzdialenejší

výber je síce jednoduchý ale vyberá si spoju daň v oveľa horších výsledkoch na všetkých inštanciách testovacej množiny. Ďalej treba ešte podotknúť, že pri vkladacích algoritmoch nehrá až takú rolu rozdiel medzi sekvenčnými a paralelnými verziami. Ale ďaleko najdôležitejšia je funkcia pre vkladanie a výber na základe hodnôt tejto funkcie.

V druhej časti sa ešte pozrieme na výsledky pre Mole & Jameson vkladací algoritmus. Tu sme spravili niekoľko výpočtov pre rôzne hodnoty parametrov λ a μ. Výsledky sú v tabuľke 5.4.

Testovacia množina	$\lambda = 0.3$ $\mu = 0.7$	$\lambda = 0.7$ $\mu = 0.2$	$\lambda = 0.6$ $\mu = 0.4$
R1-50-350	20015*14	21081*15	21634*14
R2-100-350	45276*28	44542*27	45371*28
R3-150-350	65554*45	70732*46	69127*47
R4-200-350	82976*64	83008*62	85174*62
R5-50-250	12186*6	12289*6	12050*6
R6-100-250	17598*10	17922*12	18212*12
R7-150-250	25503*17	25324*17	27050*17
R8-200-250	32759*23	32940*23	33048*23
R9-500-1000	80672*8	79624*8	76467*8
R10-500-150	235156*52	218081*51	221333*52
C1-100-100	9425*10	8808*10	8911*10
C2-100-100	13001*10	12796*10	12888*10
C3-100-100	8819*10	8697*10	8740*10

Tabuľka 5.4: Výsledky pre Mole & Jameson vkladací algoritmus

Pri výsledkoch Mole & Jameson vkladacieho algoritmu je vidno, že voľba parametrov , závisí od inštancii k inštancií a kde niektoré hodnoty dosahujú najlepšie výsledky inde môžu máličko zaostať. Zaujímavejšie je ale pre nás všimnúť si že ako samotné kritéria pre vloženie ovplyvňujú výsledok. Pri množstve inštancii sme dostali slušné výsledky oproti jednoduchému vkladaniu. Dokonca pri poslednej inštancií sme dostali zatiaľ najlepšie namerané hodnoty. Ale na druhú stranu pri niektorých inštanciách sa dá povedať, že dané kritéria zlyhali. Na záver vkladacích algoritmov teda treba povedať len toľko, že správnym výberom kritérií na vkladanie a vkladaním podľa najlep-

šej možnej hodnoty pre konkrétnu inštanciu problému vieme dosiahnuť dobré výsledky. Nájsť však univerzálne kritéria pre rozmanité prípady z praxe nie je vôbec jednoduché. Na zjemnenie výrazných odchýlok v kvalite riešenia, môžme vložiť do kritérií vhodné parametre a pred počítať si výsledky pre niektoré ich rôzne nastavenia.

5.5 Zhrnutie experimentálnych výsledkov

Skôr ako sa predstavíme vylepšovacie algoritmy nedá nám napísať ešte pár slov k nameraným výsledkom všeobecne. Pozrieme sa teda na dané výsledky nielen v rámci jednotlivých tried algoritmov ale ako na výsledky pre heuristiky riešiace VRP problém.

Hneď na začiatok treba povedať, že toto všeobecné porovnanie nemôžme urobiť tak, že postavíme namerané výsledky jednotlivých tried vedľa seba zhodnotíme. Dôvod je ten, že my sme sa snažili jednotlivé metódy nechať čo najviac čisté, teda bez nasledujúcich optimalizačných krokov, ktoré spomenieme v ďalšej sekcií. Cieľom tohto nášho úmyslu bolo sledovanie základných čŕt daných tried a ich význam pre rôzne inštancie. Teda vyhodnotiť, kde majú slabé miesta a naopak v akých prípadoch je výhodné sa nimi zaoberať. Treba si uvedomiť, že aj keď po samotnom vyhodnotení niektoré heuristiky dali horšie výsledky ako iné tak ich prístup k riešeniu je nevhodný. Mnohé z nich majú horšie hodnoty z dôvodu, že mali horší prístup napríklad k tvoreniu samotných ciest, no rozdelenie zákazníkov majú oveľa lepšie ako tie heuristiky s lepšími výsledkami. A práve to ich vysoko favorizuje po aplikovaní niektorej z optimalizačných technik na konkrétne cesty. Kdežto ak niektoré heuristiky majú nie príliš vhodné rozloženie zákazníkov tak ich optimalizácia neprinesie takmer žiadny posun k lepším riešeniam. Samozrejme môže ísť aj o opačný prípad, kedy sa snažíme optimalizovať rozloženie zákazníkov medzi cestami a tým sa približujeme k lepším riešeniam.

Teraz ešte v krátkosti zhrnieme niektoré základné črty. Ako prvé si porovnáme sekvenčný a paralelný prístup. Z výsledku je jasné, že paralelný prístup k tvorbe ciest tvorí cesty lacnejšie. Tu si ale treba uvedomiť, že ide

o cenu vzhľadom na prejdenú vzdialenosť. Tu teda paralelný prístup získava ale na úkor úplného vyťaženia vozidiel a teda stúpajú nároky na ich počet. Ak by sme si to premietli do praxe, tak si treba uvedomiť, že v mnohých prípadoch nás vide lacnejšie prevádzka jedného vozidla, ktoré síce nabehá viac kilometrov ako prevádzka dvoch vozidiel a ich neúplne vyťaženie.

Ďalej sa treba pozrieť pri riešení problémov aj na rozhodovanie medzi výberom najlepšieho rozhodnutia, ktoré nás stojí dodatočný výpočet a teda aj čas alebo zvoliť možnosť bez výpočtu. Tu je dôležité nájsť rozumný kompromis, teda hlavne ak sa jedná o kritéria, ktorých výpočet je skutočne časovo náročný a výrazne predlžuje čas heuristiky. Ako príklad slúži prítomnosť ťažkého problému GAP vo Fisher a Jaikumar algoritme. Teraz už ale prejdeme k vylepšovaním algoritmom lebo ich význam je nezanedbateľný.

5.6 Vylepšovacie algoritmy

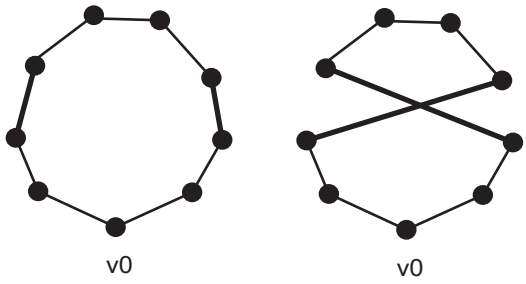
Už z názvu vylepšovacie algoritmy môžme vytušiť ,že ide o algoritmy ,ktoré budú zlepšovať aktuálne riešenie. Teda ak už máme nijaké riešenie, ktoré sme získali heuristikami alebo inými metódami, môžme ich použiť na jeho vylepšenie získanie tak riešenia bližšieho k optimálnemu riešeniu. V tejto časti predstavím niektoré základné vylepšovacie algoritmy. Dôvod nie je len to že sú často používané ako samostatný člen nasledujúci po mnohých konštrukčných heuristikách, kde dosahujú uspokojivé zlepšenia, ale sú aj súčasťou mnohých metaheuristík používaných na riešenie VRP. Napríklad genetické algoritmy využívajú 2-Opt vylepšovací algoritmus na simulovanie mutácie jednej populácie.

5.6.1 k-Opt

Vylepšovacie algoritmy k-Opt spadajú do kategórie algoritmov, ktoré pracujú vždy na jednej ceste. Konkrétne ide o zmenu k hrán na danej ceste, s cieľom získať lepšiu cenu cesty, vzhľadom na optimalizačnú podmienku. Samozrejme základná vlastnosť týchto algoritmov je aby aj po zmene hrán tvorili tieto

hrany cestu a aby spĺňala podmienky problému. V našom prípade teda máme na vstupe cestu prechádzajúcu množinou vrcholov S_v a začína a aj končí vo vrchole v_0 resp. v sklade. A výstupom je cesta , ktorá tiež prechádza všetkými vrcholmi z množiny S_v, začína aj končí vo vrchole v_0 resp. v sklade a jej cena je menšia ako cena pôvodnej cesty. Najčastejšie aplikovanými k-Opt algoritmami sú 2-Opt a 3-Opt algoritmy.

2-Opt algoritmus odstraňuje zo zadanej cesty dve hrany a nahradí ich inými dvoma. Odstránením dvoch nesusedných hrán z cesty začínajúcej aj končiacej vo vrchole v_0 (ide o cyklus) ju rozdelíme na dve samostatné cesty. Na to aby sme vytvorili opäť cyklus musíme pridať dve hrany. Existuje iba jedna možnosť pridania týchto dvoch hrán aby sme vytvorili cyklus 5.3. Táto výmena sa často nazýva ako 2-Opt posun. Prechádzame teda celú cestu a ak nájdeme dve hrany, ktoré po 2-Opt posune vytvoria cestu s menšou cestou tak tento posun aplikujeme. Pokračujeme v analýze cesty až pokiaľ neexistujú žiadne 2-Opt zlepšujúce posuny. Cestu bez možných zlepšujúcich 2-Opt posunov nazývame 2-Optimálna.



Obr. 5.3: Výmena hrán pre 2-Opt

Na rovnakej báze ako 2-Opt algoritmus je aj algoritmus 3-Opt. Rozdiel je vtom , že neodstraňujeme z danej cesty dve hrany ale hrany tri. Rozdiel vzniká aj pri spájaní vzniknutých ciest do jednej cesty s počiatkom aj koncom vo vrchole v_0, lebo na rozdiel od 2-Opt algoritmu, kde existovala len jedna možnosť tu existujú možnosti dve. Na 3-Opt algoritmus sa môžme teda pozrieť aj ako na dva alebo tri 2-Opt posuny. Rovnako ako pri 2-Opt algoritme

hľadáme vhodnú množinu troch hrán tak aby po ich odstránení a spojení vytvorených ciest do jednej, pridaním nových hrán bola cena vzniknutej cesty menšia ako cena pôvodnej. Ak taká množina troch hrán neexistuje, cesta je 3-Optimálna. Z jednoduchého pozorovania je jasné, že každá 3-optimálna cesta je aj 2-Optimálna.

Po porozumení práce spomenutých dvoch k-Opt vylepšovacích algoritmov, je jasné, že nemusíme zákonite skončiť na 3-Opt ale môžme pokračovať s 4-Opt, 5-Opt, ... Musíme si ale uvedomiť že odstránením väčšieho počtu hrán vzniká aj väčší počet možností na spojenie vzniknutých ciest do výslednej cesty. A keďže všetky tieto možnosti musíme analyzovať aby sme zistili cenu cesty ktorá danou výmenou vznikne, stúpa zložitosť a algoritmus vyžaduje viac a viac výpočtového času. A zároveň prináša iba malé zlepšenia ceny cesty v porovnaní s 2-Opt a 3-Opt algoritmami. Samozrejme je na konkrétnom prípade sa rozhodnúť pre daný problém ktorý vylepšovací algoritmus typu k-Opt a za akú cenu použijeme. Ďalšou možnosťou je preskúmavať iba niektoré možnosti spojenia ciest do výslednej cesty. Zástupca často používaného 4-Opt posunu je napríklad posun "krížiacich sa mostov". Jeho osobitosť a význam je ten, že nemôže byť postupne skonštruovaný za pomoci 2-Opt posunov a preto ma veľkú pravdepodobnosť priniesť vylepšenie oproti iným 4-Opt posunom a teda aj na už 2-optimálnej ceste.

5.6.2 Experimentálne výsledky

Na ukázanie významu vylepšovacích algoritmov sme experimentálne vyhodnotili našu testovaciu množinu vybranými heuristikami a následne použili 2-Opt vylepšovací algoritmus. Namerané výsledky sú v tabuľke **??**.

Keď sa pozrieme na výsledky a porovnáme ich s výsledkami heuristík nemôžme hovoriť o obrovských zlepšeniach. Ale dá sa povedať, že takmer každé riešenie sa zlepšilo o 0.5 - 1% ,ale v niektorých inštanciách išlo o zlepšenie 5 až 10%, čo už rozhodne nie je zanedbateľné. Navyše 2-opt algoritmus pracuje rýchlo, takže nijako výrazne nepredlžuje výpočet heuristík.

Testovacia množina	**Sekvenčný C&W**	**Paralelný C&W**	**Sweep**
R1-50-350	18399*15	16899*15	18426*16
R2-100-350	38881*28	36630*28	43596*31
R3-150-350	58498*46	55308*48	69156*55
R4-200-350	74387*63	70739*66	70739*66
R5-50-250	10907*6	10154*6	10774*6
R6-100-250	15848*12	14789*12	16758*12
R7-150-250	22954*18	21288*17	24598*18
R8-200-250	29175*23	28265*23	32799*23
R9-500-1000	62005*8	57985*8	67144*8
R10-500-150	201830*52	196247*51	250015*53
C1-100-100	9101*13	8569*14	7959*10
C2-100-100	14747*14	13749*13	13304*10
C3-100-100	9897*14	9546*13	8523*10

Tabuľka 5.5: Výsledky pre heuristiky s 2-Opt optimalizáciou

Kapitola 6

Meta-heuristiky

Na záver sa ešte pozrieme na ďalšiu triedu algoritmov riešiacich optimalizačné problémy a to meta-heuristiky. Konkrétne pre VRP sú meta-heuristiky veľmi významnou triedou a algoritmy spadajúce do nej dosahujú veľmi dobré výsledky. Rovnako ako heuristiky aj meta-heuristiky prehľadávajú len obmedzený priestor riešení. Rozdiel je však v tom, že prehľadávaný priestor je väčší ako pri heuristikách, čo ma v konečnom dôsledku dopad na to že aj riešenia nájdené meta-heuristikami sú k optimálnemu riešeniu bližšie. Prehľadávanie väčšieho priestoru riešení sa samozrejme prejaví na výpočtovej zložitosti, a preto pri meta-heutistikách je aj výpočtový čas primerane dlhší.

Z pomedzi metaheuristík pre VRP sme sa konkrétne zamerali, na triedu Tabu Prehľadávacích *Tabu search* algoritmov. Dôvod je, že algoritmy založené na tejto metóde dosahujú výborné výsledky pre VRP. V nasledujúcom texte predstavíme koncept Tabu prehľadávacích algoritmov, základné vlastnosti a aspekty, ktoré formujú riešenie.

6.1 Tabu prehľadávanie (Tabu Search)

Metódu Tabu prehľadávania (TS) predstavil Glover v roku 1986 a následne sa rýchlo stala jednou z najlepších a najrozšírenejších metód založených na metóde lokálneho prehľadávania (*local search*) pre kombinatorické optimalizačné problémy. Metóda lokálneho prehľadávania je v skutočnosti iteratívna prehľadávacia procedúra, ktorá začína v nijakom základnom možnom riešení

problému, a postupne vylepšuje toto riešenie modifikovaním ho za pomoci lokálnych zmien a posunov na dané riešenie. V každej iterácií, sa prehľadávanie posunie z jedného riešenia na druhé vylepšené, ktoré sa len jemne líši od toho pôvodného. V podstate to o koľko sa budú riešenia od seba líšiť závisí na množine zmien a posunov, ktoré daný prehľadávací algoritmus v danej iterácií na pôvodné riešenie použil. Prehľadávanie končí v momente, keď narazíme na lokálne minimum vzhľadom na modifikácie, ktoré uskutočňujeme. Teda pomocou dovolených modifikácií sa nevieme posunúť na žiadne lepšie akceptovateľné riešenie. Pri tejto metóde je dôležité si uvedomiť, že aj keď budeme mať rozumnú inštanciu problému, tak naše lokálne optimum, ktoré akceptujeme ako riešenie bude často od optimálneho riešenia veľmi vzdialené. Treba si uvedomiť, že kvalita a výpočtový čas riešenia nájdeného pomocou lokálneho prehľadávania je vysoko závislý na množine povolených modifikácií a posunov, ktoré sa vykonávajú v každej iterácií.

Vráťme sa ale späť k metóde tabu prehľadávania (TS). Keďže ide o rozšírenie LS metódy tak aj TS pracuje v iteráciách. Táto metóda teda prehľadáva priestor riešení problému posunom z riešenia x_t získaného v iterácií t do najlepšieho možného riešenia x_{t+1} v podmnožine $N(x_t)$ nazývanej ako množina susedov (*neighborhood*) riešenia x_t. Rozdiel oproti LS je ten, že riešenie x_{t+1} nemusí zákonite vylepšovať riešenie x_t. Tu ale vzniká problém možnosti zacyklenia iterovania nad nijakou sekvenciou riešení. Jednoduchým riešením ako zacykleniu predísť je zakázať výpočtovej postupnosti možnosť vrátiť sa späť na riešenie, ktoré už predtým bolo spracovávané. Tento spôsob si ale vyžaduje stále zapamätávanie celej výpočtovej postupnosti. Namiesto toho radšej využívame takzvaný mechanizmus krátkodobej pamäti (*short-term memory*). Tu ide o to, že niektoré vlastnosti riešenia sú zapamätané a žiadne riešenie majúce niektorú zo zapamätaných vlastností nie je prípustné pre ϕ iterácií. V našom prípade VRP problému daná vlastnosť môže byť presun vrcholu, hrany v danej ceste, prípadne medzi rôznymi cestami. Dá sa teda povedať, že TS je jednoduchá kombinácia lokálneho prehľadávania a krátkodobej pamäti.

Niektoré ďalšie povšimnutia hodné vlastnosti, ktoré bývajú implementované, sú Rôznotvárnosť (*diversification*)a Zosilňovanie (*intensification*). Cieľom rôznotvárnosti je zaistiť aby proces prehľadávania nebol obmedzený iba na obmedzenú časť priestoru riešení. To býva docielené zapamätávaním postupnosti posledných riešení a následným penalizovaním často uskutočňovaným zmenám. Tento mechanizmus je nazývaný ako dlhodobá pamäť (*long-term memory*). Zosilňovanie pozostáva z výraznejšieho prehľadávania priestoru okolo najlepších známych riešení.

6.1.1 Množina susedov

Z predchádzajúceho popisu tabu prehľadávacej (TS) heuristickej metódy je jasné, že základný prvok pre každú konkrétnu implementáciu založenú na TS je definícia prehľadávaného priestoru a charakteristika prvkov množiny jeho susedov. Prehľadávací priestor je jednoducho množina všetkých riešení do ktorých sa počas prehľadávania môžme dostať. Konkrétne pre VRP problém ide o množinu možných riešení problému, pričom každý bod v tomto priestore reprezentuje množinu vozidiel a ich ciest sĺňajúcich všetky definované obmedzenia. Aj keď pre VRP problém je priestor riešení celkom prirodzený, nie je to pravidlo pre všetky metódy založené na TS.

Keď teraz prejdeme k definícií množiny susedov, zistíme že je úzko spätá s priestorom riešení. Definujeme teda $N(S)$ množinu susedou v danom prehľadávanom priestore riešení, ako množinu riešení ktoré môžme získať aplikáciou lokálnych zmien alebo posunov na aktuálne riešenie S. Formálne teda $N(S)$ je podmnožina prehľadávaného priestoru riešení definovaná ako $N(S) = \{$ riešenia získané aplikáciou jednoduchých lokálnych zmien na S $\}$.

Samozrejme vo všeobecnosti existuje množstvo iných definícií konkrétnych pre vybraný problém a tiež pre jeho definíciu priestoru riešení. Keďže nás zaujíma VRP pozrieme sa bližšie na množinu susedov pre túto triedu problémov.

Prvá jednoduchá štruktúra množiny susedov $N(S)$ pre CVRP obsahuje riešenia, ktoré získame v iterácií, presunom jedného zákazníka z jeho pô-

vodného miesta v danej ceste v riešení problému S. Tento vybraný zákazník môže byť následne vložený do tej istej cesty alebo do inej cesty s dostatočnou kapacitnou rezervou vo vozidle, ktoré je k danej ceste priradené, pre pridanú objednávku. Dôležitou vlastnosťou tejto štruktúry je okrem spôsobu výberu zákazníka hlavne spôsob rozhodovania na aké miesto bude zákazník vložený. Tento spôsob môže byť čisto náhodný s takmer nulovou výpočtovou náročnosťou, prípadne môže isť o najlepšiu pozíciu, vzhľadom na cenu, v ceste do ktorej má byť vložený. K týmto jednoduchým vloženiam ale existujú aj zložitejšie vloženia, ktoré v sebe zahŕňajú aj čiastočné reoptimalizačne zmeny na danej ceste (Gendeau, Hertz and Laporte, 1994). Tu si treba uvedomiť, že naša množina susedov obsahuje všetky možné konfigurácie ciest, ktoré vzniknú z pôvodného riešenia, získaného z predchádzajúcej iterácie, vložením jedného vybraného zákazníka podľa zvolených pravidiel pre vkladanie. Takže aj keď vyberáme iba jedného zákazníka na presun, výpočet množiny susedou je zložitejšia operácia.

Ďalšou zložitejšou štruktúrou pre množinu susedov je štruktúra známa pod označením λ-medzivýmeny (λ-*interchanges*), ktorú prezentoval Osman. Táto množina obsahuje riešenia, ktoré vznikli presunom až λ zákazníkov medzi dvoma cestami v riešení S. Pretože so zväčšujúcou sa λ sa prirodzene zväčšuje aj počet možností na následné vloženie zákazníkov zvykneme obmedziť λ na hodnoty 1 alebo 2, a tým aj rozumne ohraničiť mohutnosť množiny susedov. Samozrejme opäť si treba uvedomiť, že sa týmto obmedzením oberieme o množinu možných riešení, ktorých prehľadávaním sa môžme dostať k riešeniam ktoré sú bližšie k optimálnemu riešeniu. Pre lepšiu reprezentáciu uskutočňovaných výmen pri λ-medzivýmenách používame dvojice (λ_1,λ_2) , kde $\lambda_1 \leq \lambda$ a $\lambda_2 \leq \lambda$. Daná dvojica reprezentuje operáciu, v ktorej λ_1 zákazníkov bolo presunutých z prvej cesty do druhej cesty, a λ_2 zákazníkov bolo presunutých z druhej cesty do prvej. Ak teda zanedbáme symetriu tak sú možné nasledujúce pohyby: (2,2), (2,1), (2,0), (1,1), (1,0). Tu si môžme ľahko predstaviť ako by rástla mohutnosť množiny susedou s rastúcou λ a tiež stojí za povšimnutie, že λ-medzivýmeny v sebe zahrňujú aj jednoduché vý-

meny (1,1) medzi dvoma cestami a jednoduché presuny (1,0) v rámci jednej cesty, ktoré tvorili množinu susedou v prvom spomenutom prípade. Obdobne aj pri tejto štruktúre existuje niekoľko možných variant, založených napríklad na obmedzením množiny ciest alebo vrcholov, aby sa zúčastňovali na medzi výmenách alebo pridaním lokálnej reoptimalizácie na cestách ktorých sa odoberanie a vkladanie zákazníkov dotklo.

Poslednou štruktúrou, ktorú predstavíme je vlastne rozšírenie λ-medzivýmien na viac ako dve cesty. Túto štruktúru nazývame vyhadzovacie reťaze (*ejection chains*) a bola zadefinovaná vo viacerých publikáciách (Xu and Kelly, 1996; Rego and Roucairol, 1996; Rego, 1998). Hlavnou myšlienkou je nájdenie množiny ciest a následne presúvaním zákazníkov v cyklickom poradí a teda z cesty 1 do cesty 2, z cesty 2 do cesty 3, ... Pretože ide o rozšírenie predchádzajúcej štruktúry tak postrehy a vlastnosti týkajúce sa λ-medzivýmien sa dajú jednoducho preniesť aj na spomínanú štruktúru.

Na záver by sme si mali uvedomiť, že navrhnutie efektného TS algoritmu je nájdenie správnej hranice rovnováhy medzi kvalitou množiny susedov a jej výpočtovou náročnosťou. Teda čim zložitejšie množiny susedou si zostrojíme a pripravíme na prehľadanie, tým viac riešení sa nám dostane v každej iterácií, ale rovnako aj čas výpočtu porastie.

6.1.2 Tabu posuny a krátkodobá pamäť

Ako sme už spomenuli v základnej charakteristike TS metódy, pripustenie krokov, ktoré nezlepšujú riešenie, môže viesť k zacykleniu iterácii na nijakej množine riešení. Tu prichádza na rad deklarácia, takzvaných tabu (zakázaných) posunov. Odtiaľ pochádza aj názov metódy tabu prehľadávanie. Napríklad pre našu inštanciu CVRP problému ide o to, že ak zákazník v_1 bol práve presunutý z cesty $R1$ do cesty $R2$, tak si zadefinujeme tabu posun pre návrat zákazníka v_1 späť z cesty $R2$ do cesty $R1$ pre špecifikovaný počet ϕ iterácií. Hodnotu určujúca dĺžku trvania tabu posunu označujeme ako trvanie (*tenure*) tabu posunu (*tabu move*). Keď sa vrátime späť k nášmu problému, tak to znamená, že ak bol presun zákazníka v_1 v iterácií t, tak

potom je zakázané aby bol presunutý späť do cesty $R1$ aspoň $t + \phi$ iterácií, prípadne nesmie byť presunutý preč z cesty $R2$ rovnako $t + \phi$ iterácií. Taillard (1991) predstavil náhodný výber hodnoty ϕ, pre trvanie tabu posunu z intervalu $[\underline{\phi},\bar{\phi}]$, vzhľadom na povahu problému. Pri tejto implementácií si ale musíme okrem trvania tabu posunu pamätať aj iteráciu kedy bol daný posun do tabu listu pridaný. Stále mnohé implementácie algoritmu používajú konštantnú hodnotu ϕ. Je ale preukázané, že konštantná hodnota trvania tabu posunu nedokáže zakaždým predísť zacykleniu!

Počas prehľadávania máme zoznam všetkých aktuálnych tabu posunov uložený v takzvanej krátkodobej pamäti. Zvyčajne však iba pevne a pomerne obmedzené množstvo informácií je zapamätávané. Treba si uvedomiť ,že pre každý iteračný posun navrhnutý prehľadávaním musí byť skontrolované, či sa nenachádza medzi tabu posunmi a to je operácia priamo závislá od veľkosti zoznamu tabu posunov. A teda podľa našej potreby pre konkrétnu implementáciu TS algoritmu existuje niekoľko možností, čo sa týka špecifikácie informácií, ktoré budú zapamätávané. Jednoduchý prístup je zapamätávanie si celé riešenie, čo ale so sebou nesie jednak potrebu pomerne veľkého priestoru na ukladanie informácií ale pre nás hlavne zvýšenie náročnosti na overenie, či sa nijaké riešenie nachádza v tabu zozname. Najviac používane riešenie tabu listu je zapamätávanie si posledných pár posunov uskutočnených na danom riešení a zakázanie posunov ktoré sú k týmto inverzné a vracajú sa späť k už preskúmavaným riešeniam, tak ako bolo ukázané v predchádzajúcom príklade. Na tomto príklade môžme tiež pekne ukázať, ako je dôležite zvoliť správnu reprezentáciu. Jednoducho môžme do krátkodobej pamäti zaznamenať tabu posun vrcholu v_1 z cesty $R2$ späť do cesty $R1$, ako trojicu $(v_1, R2, R1)$. Treba si ale uvedomiť, že tento zákaz nám naše prehľadávanie moc neobmedzí, a zacyklenie môže jednoducho nastať tak, že zákazník v_1 je presunutý do novej cesty R3 a následne v ďalšej iterácií späť do cesty R1. Prirodzeným rozšírením je zakázanie posunu zákazníka v_1 späť do cesty $R1$ a to bez ohľadu na to v akej ceste sa zákazník aktuálne nachádza. Prirodzene je vidieť, že tento tabu posun zapamätaný ako $(v_1, R1)$ je silnejší ako náš

prvý zakázaný posun. Ak by sme chceli ešte viac zosilniť, môžme jednoducho zákazníkovi v_1 zakázať akýkoľvek ďalší posun a jednoducho si zapamätať (v_1). Mnoho ďalších charakteristík riešenia krátkodobej pamäti je založených na niektorých kľúčových vlastnostiach riešení daného problému.

Tabu posuny sú tiež veľmi užitočne pre lokálne prehľadávanie aj na posun z už prehľadávanej časti priestoru riešení do inej, čo pomáha rozsiahlejšiemu prieskumu.

6.1.3 Dlhodobá pamäť

Pojem dlhodobá pamäť sme už spomenuli skôr. Ide teda o penalizáciu často sa uskutočňujúcich posunov za cieľom rozšíriť prehľadávanú oblasť a tým získať väčšiu rôznotvárnosť. Rôznotvárnosti je venovaná samostatná časť, a preto ju tu nebudeme popisovať. Teraz spomenieme základné tri faktory ovplyvňujúce dobu penalizácie.

- **Početnosť** vypovedá o tom ako často sa daný posun vyskytoval v predchádzajúcom výpočte.
- **Veľkosť problému** je faktor závislý na veľkosti inštancie daného problému. Teda zohľadňuje počet zákazníkov v probléme.
- **Užívateľský parameter** je užívateľom voľne nastavovateľný faktor.

Nastavenie penalizácie je teda pridanie k cene $c(x_{t+1})$ potrebnej na priradenie zákazníka x_{t+1} prídavnú hodnotu, ktorú získame vynásobením jednotlivých faktorov. V praxi je implementácia tohto mechanizmu efektívna a jednak nie je nijako výpočtovo náročná. Tento prístup predstavil Glover(1989) a do konečnej podoby vylepšil Taillard(1992).

6.1.4 Zosilňovanie

Pri zosilňovaní nám ide o to aby sme prehľadávali viac oblasti, ktoré sľubujú dobré riešenia, ako ostané oblasti. Jednou z možností je vytvorenie väčšej množiny susedov okolo riešenia, ktoré je ďaleko lepšie ako predchádzajúce

riešenia. Musíme však aj zabezpečiť aby sa výpočet vrátil k tomuto riešeniu, a to vieme zabezpečiť napríklad zmenšením zoznamu tabu posunov na niekoľko najbližších iterácií. Zväčšením množiny susedov zabezpečíme aby algoritmus prehľadával väčšiu oblasť okolo každého riešenia.

Teraz spomenieme asi najlepšiu metódu zosilňovania pre Tabu Search, ktorá bola predstavili Rochat and Taillard (1995). Ide o to, že každé riešenie môže byť rozložené do menších komponentov. Každému komponentu môžme priradiť jeho hodnotu vzhľadom na jeho význam v nijakom dobrom riešení. Hlavná myšlienka je taká, že si udržujeme v pamäti množinu dobrých komponentov riešení a snažíme sa z dobrých komponentov poskladať nové lepšie riešenie. Je to niečo podobné ako pri genetických algoritmoch, tie sú však mimo rozsah nášho textu. Každú takúto množinu nazývame populácia. V každom kroku si udržujeme v populácií tie komponenty ktorých hodnota je menšia ako hodnota niektorých riešení a z populácie vyhadzujeme riešenia so zlými hodnotami. Ide o akýsi proces učenia populácie. Táto metóda je známa pod označením adaptívna pamäť (*adaptive memory*). Ak by sme sa vrátili k VRP problému, tak môžme povedať, že nové riešenia získavame kombináciou neprekrývajúcich sa ciest z viacerých dobrých riešení. Zväčša nám však tieto cesty nepokryjú všetkých zákazníkov z dôvodu, že sa niektoré prekrývajú a teda nemôžu byť súčasne v riešení, a preto pokračujeme prehľadávanie so získanými cestami a nepriradenými zákazníkmi.

6.1.5 Rôznotvárnosť

V krátkosti ešte spomenieme ďalšiu vlastnosť TS metódy a to rôznotvárnosť. Úlohou je zabrániť aby nám v priestore riešení ostala pomerne veľká vôbec nepreskúmaná podmnožina k sebe blízkych riešení. Takýto stav by mohol nastať ak by naša prehľadávacia metóda bola príliš jednotvárna a dostávala by sa stále do rovnakých oblastí priestoru riešení.

Najjednoduchšou cestou ako tomu zabrániť je niekoľko náhodných reštartov. Tie nás vždy dostanú do nijakej novej oblasti a tam môžme pokračovať v prehľadávaní. Ďalšou sofistifikovanejišiou metódou je penalizovanie riešení

a posunov za využitia dlhodobej pamäti ako to bolo spomenuté. Ďalšou možnosťou je oslabovanie obmedzení na problém. Každé naše riešenie musí spĺňať určité obmedzenia, a preto aj prehľadávanie na riešeniach je držané týmito obmedzeniami v nijakej oblasti, môžme si to predstaviť ako kopec s neprekročiteľnou výškou. Tu môžme využiť oslabenie týchto požiadaviek a tak znížiť kopec na výšku cez ktorú vieme prejsť a výsledkom bude, že sa pár krokmi dostaneme do novej oblasti riešení. Počas tejto fázy zrôznotvárňovania nemuseli byť všetky riešenia, cez ktoré sme prechádzali akceptovateľné z dôvodu oslabenia požiadaviek. Jedna z možností ako sa dostať späť k akceptovateľným riešeniam je vysoká penalizácia posunov, ktoré porušujú požiadavky.

6.1.6 Návrh algoritmov

Na základe spomenutých vlastností a charakteristík vzniká množstvo meta-heuristík založených na TS. Ale rovnako ako pri metaheuristikách aj tu majú rôzne metódy rôzne výhody vzhľadom na testovacie množiny. A preto treba pri každom návrhu brať do úvahy určité kompromisy. Na úplný záver ešte pridáme aspoň zoznam niektorých známych meta-heuristík založených na TS.

- **Osmanov algoritmus**
- **Taillard's algorithm**
- **Gendreau, Hertz a Laporte tabu algoritmus**
- **Xu a Kelly algoritmus**
- ...

Kapitola 7

Záver a ďalšia práca

Po preštudovaní tejto práce, nikto nemôže pochybovať o robustností triedy VRP problémov. A rovnako aj o množstve techník, či už ide o exaktné metódy, heuristiky alebo metaheuristiky. Rovnako aj význam hľadania stále nových a lepších metód pre daný problém je jasný. V dnešnej dobe sa ukazuje hlavne význam meta-heuristík a spomenutá tabu metóda je len jedna z množstva. Preto ďalší posun v hľadaní riešení pre tento a aj iné ťažké problémy rozhodne smeruje práve k meta-heuristikám.

Literatúra

[1] Alexander Poot,A Savings Based Method for Real-Life Vehicle Routing problems, Econometric Institute Report EI9938/A

[2] Ann Melissa Campbell, Martin Savelsbergh, Efficient Insertion Heuristics for Vehicle Routing and Scheduling Problems, Transportation science 2004

[3] Martin Savelsbergh, Transportation science, The Logistic Institute, Georgia Institute of Technology

[4] Gilbert Laporte, Frédéric Semet, Classicall Heuristic for the Vehicle Routing Problem, Les Cahiers du GERAD, G-98-54, October 1998

[5] Yannis Marinakis, Athanasios Migdalas, Heuristic Solutions of Vehicle Routing Problems in Supply Chain Management

[6] Juraj Hromkovič, Algorithmics for Hard Problems, March 2001

[7] Cristina Bazgan, Refael Hassin, Jérôome Monnot, Differential approximation for some routing problems, Research paper

[8] Luca Maria Gambardella, Vehicle Routing Problems, Technische Universiteit Eindhoven, Short Course; November 28-29 2000

[9] Ulrich Blasum, Winfried Hochstättler, Aplication of the Branch and Cut Method to the Vehicle Routing Problem

[10] T.K. Ralphs, L. Kopman, W.R. Pulleyblank, L.E. Trotter, Jr. , On the Capacited Vehicle Routing Problem, Revised December 17, 2001

[11] Christian Nilsson, Heuristics for the Traveling Salesman Problem, Linkoping University, chrni794@student.liu.

[12] Alain Hertz, Eric Taillard, Dominique de Werra, A tutorial on tabu search

[13] Jean-Francois Cordeau, Gilbert Laporte, Canada Research Chair in Distribution Management and GERAD, Tabu Search Heuristics for the Vehicle Routing Problem

[14] Bruno De Backer, Vincent Furon, Philip Kilby,Patrick Prosser, Paul Shaw, Solving Vehicle Routing Problems using Constraint Programming and Metaheuristics, Journal of Heuristics, Volume No., 116 (1997),Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.