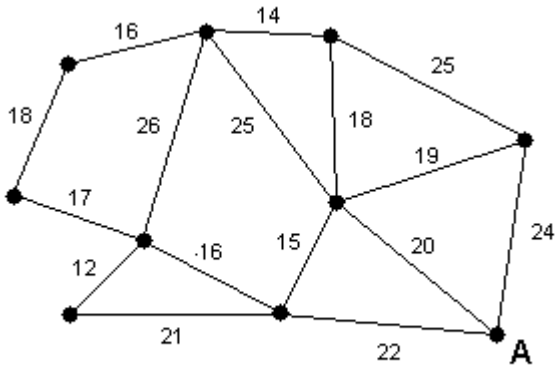
Obsah

KAPITOLA 1

Úvod

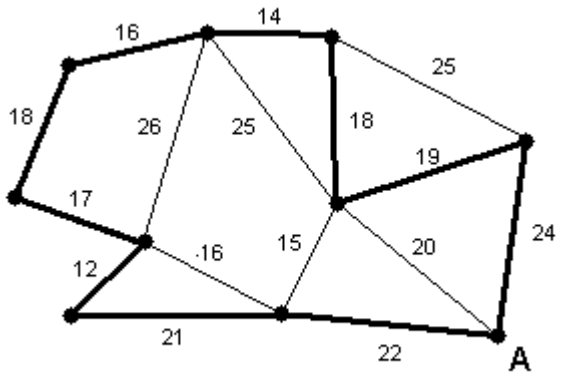
Ako mnoho ďalších úloh z teórie grafov aj problém obchodného cestujúceho (Traveling Salesman Problem, TSP) je definovaný veľmi jednoducho. Obchodný cestujúci má za úlohou precestovať každým mestom vo svojom regióne práve raz, a vrátiť sa domov. Zároveň musí túto cestu absolvovať, čo najkratším možným spôsobom.

Pre ilustráciu uvediem jeden príklad, ktorý ilustruje základnú úlohu ako aj jej optimálne riešenie. Predstavme si situáciu znázornenú na obrázku číslo 1:



Obrázok 1.

Je to mapa oblasti, ktorá je pridelená obchodnému cestujúcemu, prekreslená do grafu, kde vrcholy grafu predstavujú mestá, ktoré má navštíviť a hrany reprezentujú cesty, ktorými sú jednotlivé mestá pospájané. Ceny hrán zodpovedajú dĺžkam jednotlivých cestných úsekov. Obchodný cestujúci začína svoju púť v meste A. V každom meste sa musí rozhodnúť, ktoré mesto navštívi ako ďalšie, aby celková precestovaná dĺžka bola, čo najmenšia. Optimálne riešenie tejto úlohy je zobrazené na obrázku číslo 2 (hrubou čiarou).



Obrázok 2.

Po uvedení ilustratívneho príkladu môžeme pre daný problém uviesť jednoduchú všeobecnú definíciu. Vypustíme pevný začiatok a úloha vyzerá nasledovne: Nech je daný graf $G = (V, E)$ s dĺžkou hrán c_{ij} . Úloha nájsť najkratší hamiltonovský cyklus sa nazýva úlohou obchodného cestujúceho.

Ako úloha obchodného cestujúceho sa dajú formulovať mnohé problémy z praxe, väčšinou sú to úlohy, ktoré riešia cyklické presuny pozície po dopredu známych cestách pri minimalizovaní účelovej funkcie (obvykle dĺžky, času, ceny a pod.). Pre zaujímavosť ich niekoľko uvediem: *vŕtanie otvorov na matičnú dosku* (pohyb vŕtačky, ktorá mení veľkosť vrtákov po vyvŕtaní dier jednej veľkosti môže byť modelovaný ako TSP) [1], *röntgenová kryštalografia* (detektor meria intenzitu odrazu lúčov z rôznych pozícií a presun do pozície môže byť časovo význačný a preto sa minimalizuje časová náročnosť celkového pohybu detektora) [?], *kreslenie plošných spojov* (mechanický plotter zakresľuje spoje pred leptaním, treba minimalizovať celkový čas potrebný na vykreslenie celej dosky), *klasifikácia organických zlúčením* (J. Lederberg v roku 1958 preukázal na základe minimalizácie hamiltonovského cyklu v zlúčeninách isté charakteristické vlastnosti) [3]. Tieto a podobné úlohy v praxi predstavujú grafové úlohy veľkých rozmerov, a preto nájdenie optimálneho riešenia môže znamenať obrovskú úsporu.

Nájdenie optimálneho riešenia však nie je vôbec jednoduché. Úloha obchodného cestujúceho totiž patrí do triedy NP-ťažkých problémov, teda nie je známy žiadny polynomiálny algoritmus na ich vyriešenie. Z týchto dôvodov sa pozornosť riešiteľov tejto úlohy zamerala na približné riešenie problému. Samozrejme toto riešenie by malo mať určité výhody ako kompenzáciu istej odchýlky od optimality. Touto výhodou je časová úspora a výpočtová jednoduchosť. Tieto podmienky spĺňajú aproximačné algoritmy (heuristiky) riešenia úlohy obchodného cestujúceho, ktoré zaručujú pomerne dobré výsledky v prijateľnom výpočtovom čase.

A práve tieto heuristiky sú predmetom tejto práce. Hlavnými cieľmi diplomovej práce boli: preskúmanie heuristík na riešenie úlohy obchodného cestujúceho, poskytnúť prehľad týchto heuristík v slovenskom jazyku, vzájomné experimentálne porovnanie a vyhodnotenie algoritmov. V porovnávaní sa budeme spoliehať na štatistické výsledky, ktoré nám poskytnú naprogramované heuristiky.

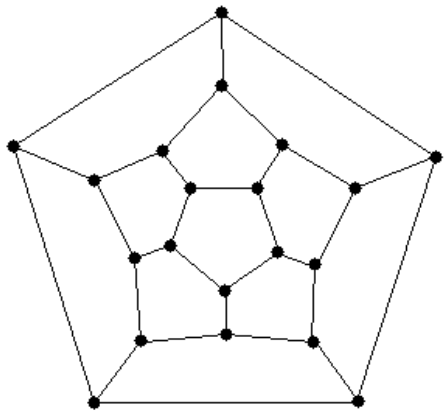
Skôr ako sa pustíme do samotného experimentálneho porovnávania, v 2. kapitole podáme všeobecný teoretický úvod do problému obchodného cestujúceho a rovnako aj vysvetlenie úpravy vstupných grafov na grafy kompletné. V 3. kapitole jednotlivé heuristiky podrobne vysvetlíme, podchytíme hlavné myšlienky a algoritmy budeme ilustrovať na názornom príklade. Kapitola bude rozdelená do častí, v ktorých budú zoskupené myšlienkovo príbuzné heuristiky. V 4. kapitole porovnáme heuristiky na základe štatistických podkladov naprogramovaných heuristík. Heuristiky budeme skúmať na náhodne nagenerovanom súbore príkladov a tiež aj na niekoľkých príkladoch s doteraz najlepšími známymi riešeniami z internetu. V prílohe poskytneme výsledky heuristík, vo forme výpisov výpočtových experimentov na počítači, a rovnako uvedieme aj popis programov, pomocou ktorých boli tieto výpočty zrealizované.

KAPITOLA 2

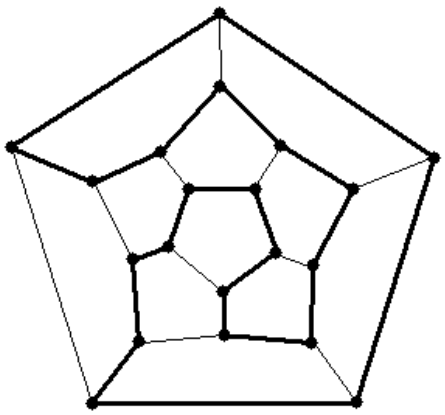
Teória

2.1. Hamiltonovské grafy

Slávny írsky matematik W. R. Hamilton v roku 1859 navrhol komerčnú hádanku "cesta okolo sveta". Príklad je zobrazený na obrázku č. 3. Úloha znie precestovať všetky hlavné mestá (vrcholy grafu) a vrátiť sa domov (ľubovolný východzí vrchol). Jedno riešenie je na obrázku č. 4.



Obrázok 3



Obrázok 4

Hra pre svoju jednoduchosť nemala veľký úspech, ale pomenovanie pre cyklus, ktorý prechádza všetkými vrcholmi na počesť Hamiltona zostal. Graf, ktorý obsahuje *hamiltonovský cyklus* sa nazýva *hamiltonovský*. Ak obsahuje len hamiltonovskú cestu, t.j. cestu spájajúcu dva vrcholy a prechádzajúcu všetkými vrcholmi práve raz, je označený ako *polo-hamiltonovský*. Zistenie, či graf $G = (V, E)$ je hamiltonovský, môže viesť k úlohe obchodného cestujúceho na kompletnom grafe, kde všetky hrany originálneho grafu sú nahradené hranami dĺžky 1 a všetky ostatné hrany majú dĺžku 2. Po nájdení optimálneho hamiltonovského cyklu z dĺžky

cyklu môžeme vypovedať o pôvodnom grafe. Ak dĺžka optimálneho riešenia je rovná n potom pôvodný graf G je hamiltonovský.

Z dôvodov jednoznačnosti teraz uvedieme najčastejšie používané pojmy a označenie používané v tejto práci.

2.2. Definície a používaná terminológia

Majme *graf G = (V, E). V* je konečná množina jeho vrcholov a *E* je množina jeho hrán (dvojprvkových podmnožín množiny *V*), spájajúce dvojice vrcholov. Počet prvkov jednotlivých množín označujeme ako $|V| = v$, $|E| = e$. Hrana $e \in E$ spájajúca vrcholy v_i a v_j je označená ako (v_i, v_j), alebo ij. Vrcholy v_i a v_j spojené hranou (v_i, v_j) sa nazývajú *susedné*. Základné pojmy z teórie grafov možno nájsť v [2, 3]. Vrchol v_i nasledujúci v hamiltonovskom cykle ako ďalší vrchol po vrchole v_j nazývam vo svojej práci *pravým susedom vrcholu* v_i. Hrany hamiltonovského cyklu, ktoré majú spoločný jeden vrchol nazývajú *susedné*. Ohodnotený graf $G = (V, E, c)$ je definovaný množinou vrcholov *V*, množinou hrán *E* a funkciou $c:E \rightarrow R$, ktorá každej hrane (v_i, v_j) priradí jej cenu označenú ako c_{ij}. Táto cena môže predstavovať dĺžku hrany, a preto v práci nerozlišujeme medzi cenou hrany a dĺžkou hrany.

Problém obchodného cestujúceho pozostáva z nájdenia hamiltonovského cyklu, ktorého *c - dĺžka* (suma dĺžok prejdených hrán) je minimálna (niekedy sa môže žiadať, aby bola maximálna). Bez straty na všeobecnosti však môžeme brať do úvahy len minimalizačnú verziu úlohy.

V diplomovej práci venujeme zvýšenú pozornosť úlohe obchodného cestujúceho na *euklidovských grafoch*. Takýto graf je zhodný s bodmi (vrcholmi) rozloženými na 2-rozmernej rovine, kde vzdialenosť medzi vrcholmi (dĺžka hrany) sa rovná euklidovskej vzdialenosti príslušných bodov. Euklidovský graf je kompletný a zaručuje platnosť trojuholníkovej nerovnosti, t.j. $c_{ij} + c_{jk} \geq c_{ik}$ pre ľubovoľné tri vrcholy *i, j,* a *k*.

2.3. Úprava vstupných grafov

Ako vstupné grafy používame, kompletné grafy s kladne ohodnotenými hranami. Kladné ohodnotenie grafu nie je obmedzujúcim predpokladom, lebo pripočítaním dostatočne veľkej konštanty W ku všetkým dĺžkam hrán si túto podmienku môžeme zaručiť a táto úprava nemá vplyv na spracovanie úlohy. Keďže konštantu *W* sme pripočítali ku všetkým hranám a výsledný

cyklus obsahuje práve *n* hrán, kde *n* je rozmer úlohy (počet vrcholov), od výslednej hodnoty dĺžky hamiltonovského cyklu stačí odrátať hodnotu *W*n*. Takto získaný výsledok korešponduje s riešením pôvodného nekladne ohodnoteného grafu.

Dôvodom pre použitie kompletného grafu ako základného prostredia úlohy obchodného cestujúceho je to, že kompletný graf nám zaručuje existenciu prípustného riešenia, kým dokázať existenciu hamiltonovského cyklu na všeobecnom grafe je NP-ťažký problém.

Ľubovoľný nekompletný súvislý graf *G = (V, E)* sa dá pre potreby úlohy obchodného cestujúceho „skompletizovať" pridaním hrán do pôvodného grafu. Každá novo pridaná hrana bude mať priradenú dĺžku *M*, kde *M* je dostatočne veľká hodnota (napr. $M \geq \sum_{e \in E} c_e$) . Takýto graf môžeme riešiť algoritmami úlohy obchodného cestujúceho pre kompletné grafy. Ak optimálne riešenie neobsahuje žiadnu hrany s dĺžkou *M*, potom optimálne riešenie doplneného grafu je optimálnym riešením aj pôvodného grafu. Ak optimálne riešenie obsahuje hranu s dĺžkou *M*, graf neobsahuje hamiltonovský cyklus. Heuristiky však nemôžu zaručiť nájdenie hamiltonovského cyklu ani v prípade, že sa v pôvodnom grafe skutočne nachádza. To môže zaručiť len exaktný algoritmus.

Druhý spôsob ako „skompletizovať" vstupný graf znamená odstúpenie od striktného dodržiavania podmienky návštevy každého vrcholu práve jedenkrát.

Majme vstupný graf *G = (V, E)* a predpokladajme, že je súvislý. V opačnom prípade prípustné riešenie neexistuje. Upravená úloha obchodného cestujúceho pozostáva z nájdenia trasy, ktorá sa začína aj končí v tom istom vrchole, ale na rozdiel od štandardnej úlohy obchodného cestujúceho, minimalizuje celkovú precestovanú vzdialenosť, a to aj za cenu opätovnej návštevy niektorých miest. Definícia prípustného riešenia takto upravenej úlohy obchodného cestujúceho je teda nasledovná: ľubovoľný uzavretý sled prechádzajúci všetkými vrcholmi daného grafu. Aby sme vylúčili neohraničenosť vzhľadom na účelovú funkciu, žiadame podmienku nezáporného ohodnotenia hrán.

Upravenú úlohu obchodného cestujúceho môžeme transformovať na štandardnú úlohu obchodného cestujúceho nasledovne: skonštruujeme nový kompletný graf K_n s rovnakým počtom vrcholov, v ktorom dĺžka hrany (v_i, v_j), či už existujúcej, alebo neexistujúcej v pôvodnom grafe, je nahradená dĺžkou najkratšej možnej cesty z vrcholu v_i do vrcholu v_j v pôvodnom grafe. Riešením úlohy obchodného cestujúceho na grafe K_n dostaneme hamiltonovský cyklus *H*. Riešenie pôvodnej upravenej úlohy obchodného cestujúceho dostaneme nahradením hrán v cykle *H*, sledom hrán najkratších ciest, ktoré spájajú jednotlivé vrcholy v grafe G.

Po ujasnení si základnej terminológie uvedieme prevod úlohy obchodného cestujúceho na úlohu celočíselného programovania.

2.4. Transformácia úlohy obchodného cestujúceho na úlohu celočíselného programovania

Jedným zo spôsobov ako vyriešiť úlohu obchodného cestujúceho až k optimalite, je pokus transformovať ju na úlohu, ktorej riešenie vieme nájsť exaktným postupom. Crowder a Padberg [4] sa pokúšali previesť úlohu obchodného cestujúceho na úlohu lineárneho programovania a jej riešenie upraviť na celočíselné. Túto transformáciu možno rozdeliť na dve fázy. Prvá fáza je prevod na úlohu lineárneho programovania, ktorého riešenie nám prisúdi len váhy s akým majú byť hrany do výsledného cyklu zapojené. Druhá fáza pozostáva z úpravy riešenia elimináciou hrán pomocou metódy rezov. Druhú fázu si vyžadujú podmienky, ktoré nie sú lineárne. Úprava na úlohu lineárneho programovania je realizovaná nasledujúcim spôsobom:

Majme kompletný graf $G = (V, E)$ majúci n vrcholov. Ďalej majme vektor c, so súradnicou c_e pre každú hranu obsahujúcu jej dĺžku (t.j. ak hrana e spája vrcholy v_i a v_j, tak zložka c_e sa rovná dĺžke hrany ij). Označme si maticu A typu n x m, (vrchol x hrana), napr. ak sa vrchol i nachádza na hrane e, tak v i-tom riadku a e-tom stĺpci bude 1 a inak 0. Pre každý vektor x so súradnicami x_e $e \in E$, a pre každú podmnožinu $W \subseteq V$ definujme:

$$x(W) = \sum_{e \in E(w)} x_e ,$$

kde E(W) je množina týchto hrán grafu G, ktoré majú oba konce v množine vrcholov W.

Úloha obchodného cestujúceho potom predstavuje nasledujúci minimalizačný problém:

Minimalizujte **cx**

za pomienok

$$\mathbf{Ax} = \mathbf{2}, \quad 0 \leq \mathbf{x} \leq \mathbf{1}$$

$$\sum_{i \in [0,..,k]} x(W_i) \leq |W_0| + \sum_{i \in [1,..,k]} (|W_i| - 1) - < 0{,}5 * k >$$

k nepárne,

kde **c** je vektor dĺžky *m*, **A** matica *n* x *m*, **2** je dvojkový vektor dĺžky *n*, |W| je kardinalitou množiny W, $< \bullet >$ označuje najbližšie väčšie celé číslo. Podmienka rovnosti požaduje, aby váhy hrán riešenia pri každom vrchole mali súčet 2. Druhá podmienka nám zaručuje, že trasy obsahujúce menej ako *n* hrán budú neprípustné. Výsledný vektor **x** nám určí s akou váhou máme zapojiť jednotlivé hrany do cyklu. V prípade, že riešenie je súvislé (hrany s kladnými x_e tvoria súvislý graf) a súradnice vektora **x** sú len 0 a 1 dostávame optimálne riešenie.

Prvým krokom je len lineárna časť, teda lineárna úloha typu:

$$\text{Min } \{ \mathbf{cx} \mid \mathbf{Ax} = \mathbf{2}, 0 \leq \mathbf{x} \leq \mathbf{1} \}$$

V druhej fáze sa pokračuje metódou eliminácie niektorých hrán pomocou rezov, ktorá ako uvádzajú autori, síce v niektorých prípadoch našla optimálne riešenie, ale aj napriek dodatočným úpravám riešenia, nedokáže zaručiť optimalitu výsledného riešenia (k optimálnemu riešeniu dospeli v menej ako 25%-ách skúmaných úloh).

Napriek tomu, že táto metóda nevedie vždy k optimálnemu riešeniu, jej prvá fáza môže poslúžiť ako dolný odhad pre optimálne riešenie úlohy obchodného cestujúceho. Takáto informácia môže poskytnúť aspoň približný odhad kvality riešenia iných algoritmov v prípade, že optimálne riešenie úlohy nie je známe.

KAPITOLA 3

Heuristiky

Ak chceme vyriešiť úlohu obchodného cestujúceho stretávame sa s niekoľkými problémami. Ako sme už spomenuli v úvode, úloha obchodného cestujúceho je NP-ťažký problém (Karp[4]). Z tohto dôvodu sa pozornosť matematikov sústreďuje na polynomiálne aproximačné algoritmy – heuristiky, ktoré síce nenájdu zaručene optimálne riešenie, ale riešenie blízke optimálnemu, v čase, ktorý môžeme považovať za prípustný. V niektorých prípadoch sa dokonca môže stať, že heuristiky dospejú k riešeniu optimálnemu.

V tejto kapitole uvedieme systematický prehľad týchto heuristík, ako aj hlavné myšlienky, na ktorých sú postavené. Podrobný popis algoritmov uvádzame formou pseudokódov. Začneme konštrukčnými heuristikami, ktoré pristupujú k problému skôr intuitívne, ďalej rozoberieme heuristiky založené na minimálnej kostre a nakoniec sa budeme venovať heuristikám zlepšovacím (improvement heuristics). Teda algoritmom, ktoré hľadajú zlepšenia už známych hamiltonovských cyklov. U niektorých heuristík uvedieme aj príklad ilustrujúci ich činnosť, alebo časovú zložitosť. Aj keď názvy heuristík uvádzame v slovenskom preklade, vždy uvedieme aj anglický názov. Ako základné prostredie pre heuristiky používame kompletné kladne ohodnotené grafy $G = (V, E, c)$, a to z dôvodov, ktoré sme uviedli v predchádzajúcej kapitole.

3.1. Pomocné algoritmy

Pri realizovaní heuristík sme sa stretli s troma problémami: hľadania najlacnejšej cesty medzi ľubovoľnými dvoma vrcholmi grafu G, nájdenia najlacnejšej kostry grafu G a nájdenia euklidovského sledu. Keďže sa jedná o problémy dobre známe a dopodrobna opísané v

literatúre, nebudeme uvádzať ich riešenie (napr. Plesník [2]). Bude postačujúce, keď uvedieme algoritmy, pomocou ktorých sme spomínané problémy vyriešili.

Na nájdenie najlacnejšej cesty medzi dvoma vrcholmi grafu G, sme použili Dijsktrov algoritmus (1959). Keďže sme potrebovali poznať najkratšie vzdialenosti pre každú dvojicu vrcholov, vyriešili sme to opakovaným spustením tohto algoritmu.

Pre problém nájdenia najlacnejšej kostry sme použili známy Kruskalov algoritmus (1959).

Nájdenie euklidovského sledu sme vyriešili pomocou upraveného Tarryho (1895) algoritmu. Jednalo o špeciálny prípad úlohy, kde máme zaručené, že sled pozostávajúci zo spätných prechodov hranami pomocou Tarryho algoritmu je eulerovský ťah.

3.2. Konštrukčné heuristiky

Konštrukčné heuristiky sú založené na optimalizácii vznikajúceho hamiltonovského cyklu z hľadiska určitého lokálneho minima. Keďže na kompletnom grafe vieme skonštruovať hamiltonovský cyklus veľmi jednoducho (napr. náhodným pripájaním vrcholov na jeden koniec cesty a koncovým spojením začiatočného a konečného vrcholu), líšia sa tieto heuristiky len spôsobom vyberania ďalšieho vrcholu vhodného na zapojenie.

3.2.1. Heuristika najbližšieho suseda

(Nearest Neighbour heuristic)

Jedna z najjednoduchších heuristík pre úlohu obchodného cestujúceho, tzv. *metóda najbližšieho suseda*, konštruuje hamiltonovský cyklus, ako už názov napovedá pripájaním najbližších susedov. Štandardná verzia vyzerá nasledovne.

N A J B L I Ž Š Í_S U S E D

(1) Vyberte ľubovoľný počiatočný vrchol j, položte $k = j$ a $W = \{1,2,...,n\} - \{j\}$.

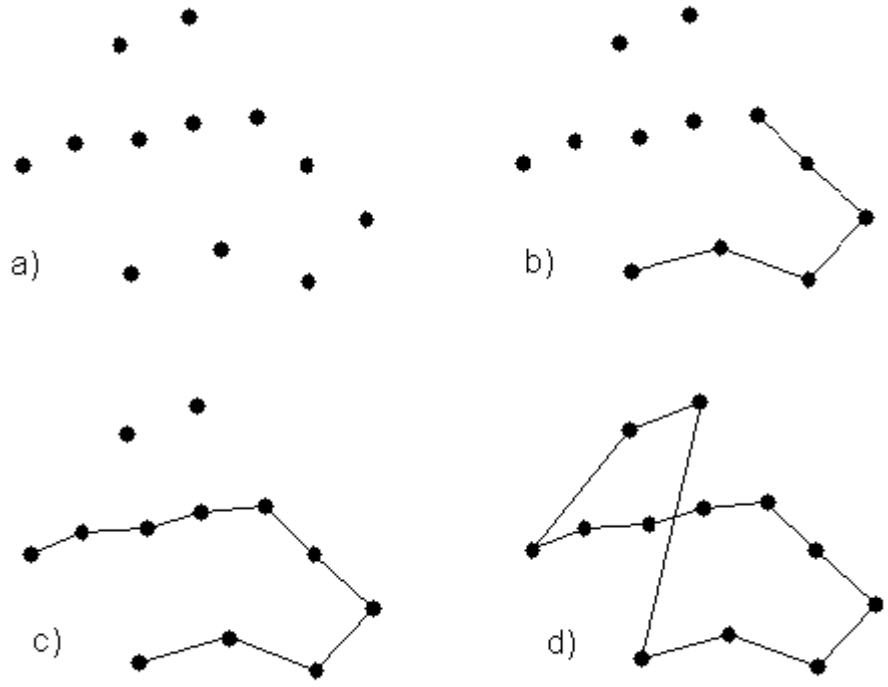
(2) Kým $W \neq \emptyset$ opakujte:

(2.1) Vyberte $j \in W$ také, že $c_{kj} = min\ \{c_{ki} / i \in W\}$.

(2.2) Spojte k s j a položte $W = W - \{j\}$, $k = j$.

(3) Na vytvorenie hamiltonovského cyklu spojte k s počiatočným vrcholom vybratým v prvom kroku.

Znázornenie fungovania algoritmu je vidieť na obrázku č.4. Pre názornosť som vybral príklad euklidovského grafu. Graf samozrejme kompletný.



Obrázok 4

Ďalšia možná variácia tohto algoritmu, je *dvojstranná metóda najbližšieho suseda*, kde nové vrcholy zapájame podľa potreby na výhodnejší koniec trasy.

Štandardná verzia beží v čase $O(n^2)$. Pre kvalitu riešenia však nemáme žiadnu konštantu, ktorá by ohraničovala najhorší možný výsledok (najhorší chybový pomer). Pre úlohy menších rozmerov síce táto metóda poskytuje uspokojivé výsledky, ale pre väčšie úlohy výrazne zaostáva za ostatnými algoritmami. Napriek tomu je často využívaná pre jednoduchosť svojho prevedenia. Táto metóda funguje veľmi efektívne na začiatku, keď zapája skutočne tie najlepšie vrcholy. V priebehu realizácie algoritmu sú však niektoré vrcholy „zabudnuté“ a sú zapojené neskôr na pozíciu značne horšiu ako je optimálna. Záverečné spojenie začiatočného a posledne pridaného vrchola za účelom získania hamiltonovského cyklu je vo väčšine prípadov tiež veľmi „drahé“. Spomenuté nedostatky algoritmu sú viditeľné na ilustratívnom príklade.

Nedostatkom heuristiky najbližšieho suseda je drahé zapájanie vrcholov, ktoré sa zapájajú neskôr. Keďže spočiatku je zapájanie pomerne dobré, jedno z možných zlepšení je zapojiť niekoľko prvých vrcholov (Jünger, Reinelt a Rinaldi [1] uvádzajú 10 vrcholov). Získaný podgraf K potom zadefinovať ako jediný bod k a v ďalšej iterácii začať odznova na pôvodnom

grafe zmenšenom o body podgrafu K_1 s pridaným novým bodom *k*. Po zredukovaní pôvodného grafu na menej ako 20 vrcholov zapojiť zostávajúce štandardným spôsobom. Výpočtový čas algoritmu zostane $O(n^2)$.

Myšlienkou druhého spôsobu vylepšenia je využiť citlivosť algoritmu na výber počiatočného vrcholu, spustiť ho zo všetkých možných štartovacích vrcholov a vybrať tú najlepšiu alternatívu. Oproti predchádzajúcemu vylepšeniu však časová náročnosť výpočtu vzrastie na $O(n^3)$.

3.2.2. Heuristika vkladania

(Insertion heuristic)

Táto heuristika je ďalšia z radu intuitívnych prístupov k danému problému. Začína s malou podmnožinou vrcholov tvoriacich hamiltonovský cyklus, ktorý sa zväčšuje priberaním ďalších a ďalších vrcholov. Týmto spôsobom nakoniec zapojí do výsledného hamiltonovského cyklu všetky vrcholy grafu.

V K L A D A N I E

(1) Vyberte štartovný cyklus s *k* vrcholmi $v_1,v_2,...,v_k$ $(k \geq 1)$, $W = V - \{ v_1,v_2,...,v_k \}$.

(2) Kým $W \neq \emptyset$ opakujte:

(2.1) Vyberte $j \in W$ vyhovujúce dohodnutým podmienkam (vysvetlíme neskôr).

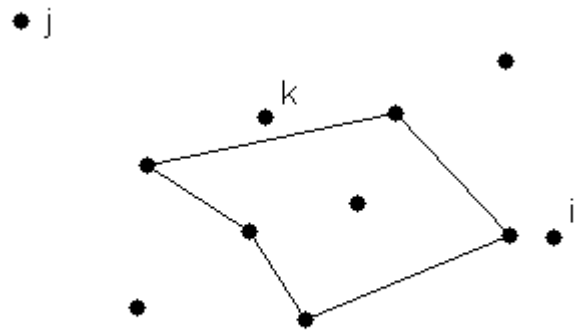
(2.2) Zapojte *j* na nejakú pozíciu v cykle, položte $W = W - \{j\}$

Samozrejme je niekoľko možností ako realizovať takéto vkladanie. Základné rozdiely sú ukotvené v podmienkach výberu vrcholu *j* v kroku (2.1). Štartovný cyklus môže pozostávať z troch ľubovoľných vrcholov grafu $G = (V, E)$, ale v degenerovaných prípadoch to môže byť aj hrana (2 vrcholy), alebo bod (1 vrchol). Teraz uvediem niekoľko podmienok, ktoré môžu slúžiť ako výberové kritériá v kroku (2.1). Najprv si však definujme minimálnu vzdialenosť vrcholu *j* od na množine vrcholov $V - W$: $d_{min}(j) = min\{c_{ij} \mid i \in V - W\}$.

- Najbližšie vkladanie: vloží vrchol, ktorého minimálna vzdialenosť od vrcholov cyklu je najmenšia, t.j. vyberie $j \in W : d_{min}(j) = min\{d_{min}(l) \mid l \in W\}$.
- Najvzdialenejšie vkladanie: vloží vrchol, ktorého minimálna vzdialenosť od vrcholov cyklu je najväčšia, t.j. vyberie $j \in W : d_{min}(j) = max\{d_{min}(l) \mid l \in W\}$.
- Náhodné vkladanie: náhodne vyberie vrchol, ktorý vloží na najlepšiu možnú pozíciu

- Najlacnejšie vkladanie: vyberie vrchol, ktorého vloženie do cyklu predĺži jeho dĺžku, čo najmenej

  Na obrázku č. 5 ilustrujeme rozdiely medzi výberovými kritériami.



Obrázok 5

Najbližšie vkladanie by zvolilo v načrtnutej situácii vrchol *i*, najvzdialenejšie vrchol *j* a najlacnejšie vkladanie by zvolilo vrchol *k*.

Všetky spomínané heuristiky, okrem najlacnejšieho vkladania, majú náročnosť výpočtu $O(n^2)$. Najlacnejšie vkladanie má časovú náročnosť $O(n^2 \log n)$. Pre niektoré typy vkladacích heuristík sa dajú určiť najhoršie chybové pomery. Rosenkrantz, Stearns a Lewis [5] uvádzajú, že hamiltonovské cykly vypočítané najbližším a najlacnejším vkladaním nemajú väčšiu dĺžku ako dvojnásobok dĺžky optimálneho cyklu.

3.2.3. Metóda úspor

(Savings method)

Táto metóda bola pôvodne odvodená z dopravnej úlohy. Jej autormi sú Clark a Wright [6]. Dá sa však úspešne použiť aj pre úlohu obchodného cestujúceho, keďže sa jedná o špeciálny prípad dopravnej úlohy zahrňujúci jediné auto bez obmedzenia nosnosti. Heuristika začína zo špeciálneho hviezdicovitého zapojenia dvoma hranami do jedného z vrcholov a hľadá možné vylepšenia. Algoritmus vyzerá nasledovne:

Ú S P O R Y

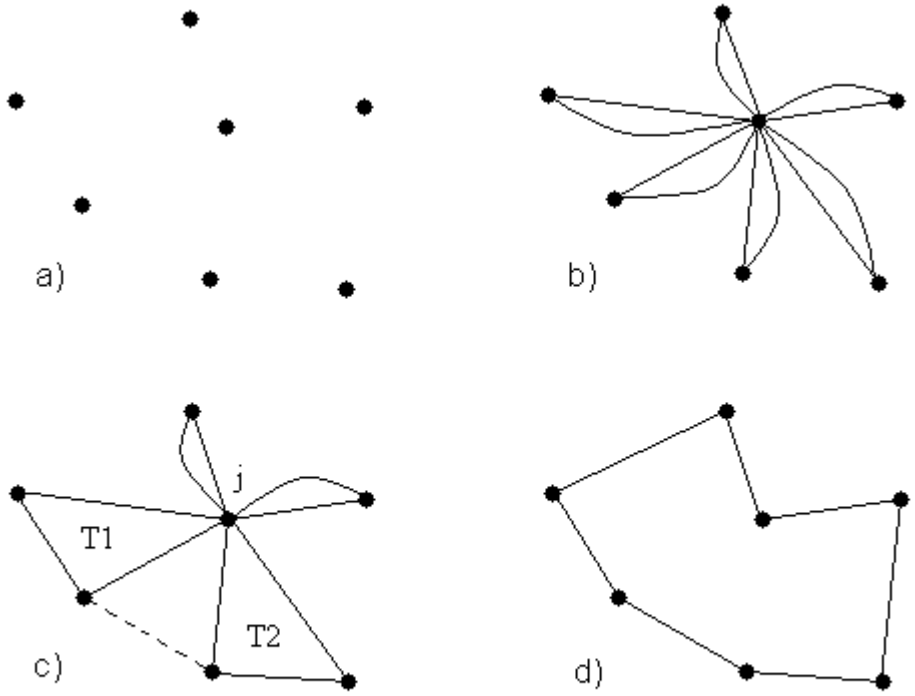
(1) Vyberte ľubovoľný počiatočný vrchol *j*, zapojte $n - 1$ sledov (j, v, j), $v \in V-\{j\}$, (každý sled pozostáva z dvoch rovnakých hrán).

(2) Kým zostáva viac ako jeden sled (j, v, j), $v \in V-\{j\}$ opakujte:

(2.1) Pre každý pár *j-j* sledov T_1 a T_2 vypočítajte úsporu, ktorú by sme získali zmazaním jednej hrany pri *j* v každom slede a pridaním hrany spájajúcej zvyšné voľné konce každého z nich.

(2.2) Vyberte dvojicu *j-j* sledov s najväčšou úsporou, ktorá po zapojení dáva cyklus a zapojte ju uvedeným spôsobom.

Na obrázku č. 6 je znázornené ako metóda úspor pracuje. Prerušovanou čiarou je vyznačený kandidát na nasledujúcu úsporu (obrázok c)).



Obrázok 6

Časová náročnosť výpočtu je $O(n^3)$. Dá sa však upraviť na $O(n^2 \log n)$ zavedením matice úspor, ktorá by si pamätala úspory pre každé dva vrcholy a po každom zapojení hrany sa aktualizuje.

Modifikácie tejto metódy spočívajú vo vytvorení niekoľkých sledov, ktoré sa na začiatku algoritmu zapoja do štartovného vrcholu koncovými vrcholmi, a tak sa beh algoritmu značne urýchli. Pričom výsledky sú porovnateľné s pôvodnou verziou.

3.3. Heuristiky založené na nájdení najlacnejšej kostry

Tieto heuristiky generujú hamiltonovský cyklus z najlacnejšej kostry grafu. Pôvodne boli obidva nižšie uvedené algoritmy navrhnuté pre grafy spĺňajúce trojuholníkovú nerovnosť, ale v princípe fungujú aj na všeobecných grafoch.

Skôr ako uvediem heuristiky riešiace úlohu obchodného cestujúceho, uvediem algoritmus, ktorým z ľubovoľného uzavretého sledu, vytvorí hamiltonovský cyklus cez vrcholy, ktorými daný sled prechádzal, nie dlhší ako dĺžka pôvodného sledu. Majme sled (v_{i0} , v_{i1} ,..., v_{i0}) (vrátane opakovaní).

Z Í S K A N I E _ C Y K L U

(1) Položte $T = \{ j \}$ a $v = v_{i0}$, $l = 1$.

(2) Kým $|T| < n$ opakujte:

(2.1) Ak $v_{i1} \notin T$ potom $T = T \cup \{ v_{i1} \}$, spojte v s v_{i1} a položte $v = v_{i1}$.

(2.2) $l = l + 1$.

(3) Na vytvorenie hamiltonovského cyklu spojte v do v_{i1} .

Každá hrana zapojená algoritmom je hranou sledu, alebo jej skratka, ktorá spája dva koncové vrcholy. Je teda zrejmé že získaný hamiltonovský cyklus bude nanajvýš rovnako dlhý ako pôvodný sled.

3.3.1. Heuristika zdvojenia kostry

(Doubletree heuristic)

Je to najjednoduchší algoritmus, v ktorom nám existenciu hamiltonovského cyklu zaručí zdvojnásobenie všetkých hrán najlacnejšej kostry.

D V O J S T R O M

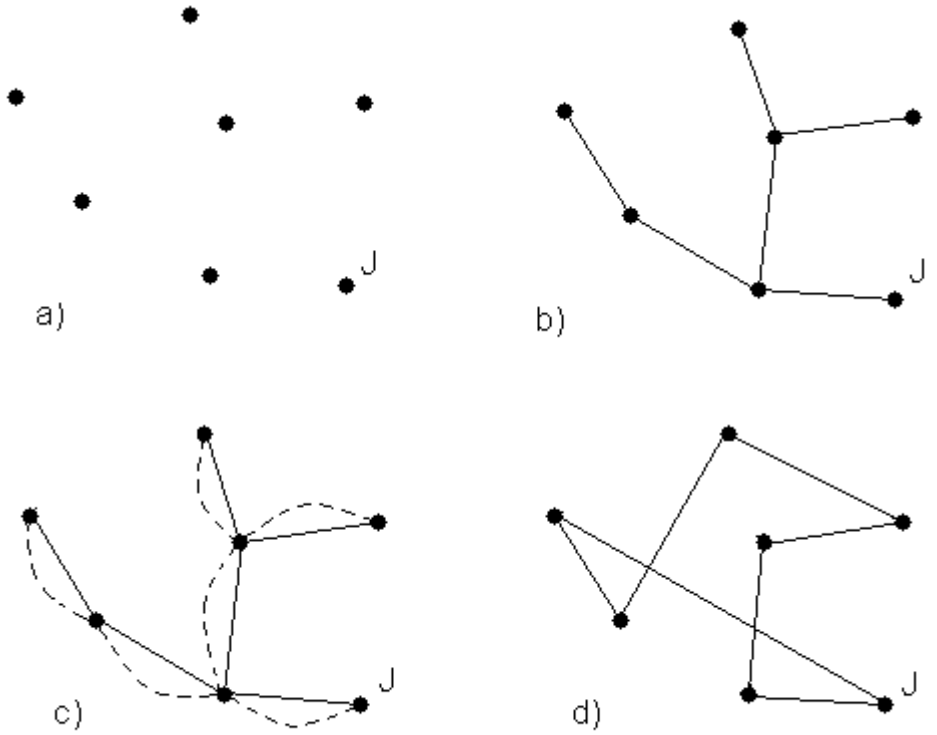
(1.) Vypočítajte najlacnejšiu kostru grafu.

(1.) Zdvojnásobte všetky hrany kostry.

(2.) Nájdite eulerovský ťah.

(3.) Použite ZÍSKANIE_CYKLU na vytvorenie hamiltonovského cyklu.

Časová náročnosť algoritmu je podmienená časovou náročnosťou výpočtu minimálnej kostry. Preto celková náročnosť je $O(n^2)$. Na výpočet najlacnejšej kostry používam pri realizácii programu Kruskalov algoritmus.



Obrázok 7

Na obrázku č.7 sme ilustrovali činnosť algoritmu zdvojenia kostry. Opäť ide o euklidovskú úlohu a najlacnejšiu kostru môžeme vidieť na obrázku b). Na obrázku c) je zobrazené zdvojenie kostry a na d) je možný výsledok v prípade, že procedúru ZÍSKANIE_CYKLU, by sme iniciovali z vrcholu J. Táto metóda má konštantu ohraničujúcu najhorší možný výsledok $c \leq 2$. Ľahko sa o tom presvedčíme ak uvážime, že vypustením jednej hrany z hamiltonovského cyklu môžeme v ideálnom prípade získať najlacnejšiu kostru.

3.3.2. Christofides

(Christofides)

Autorom tejto metódy je Christofides [7], podľa ktorého je aj pomenovaná. V porovnaní s predchádzajúcim algoritmom, sa jedná o omnoho sofistikovanejší prístup k problematike, aj keď obidva vychádzajú z najlacnejšej kostry. Christofides totiž namiesto zdvojenia všetkých

hrán kostry, pridá hrany len nevyhnutne potrebné k zaručeniu eulerovského ťahu. A aj tieto hrany sú podrobené minimalizačným požiadavkám.

C H R I S T O F I D E S

(1.) Vypočítajte najlacnejšiu kostru grafu.

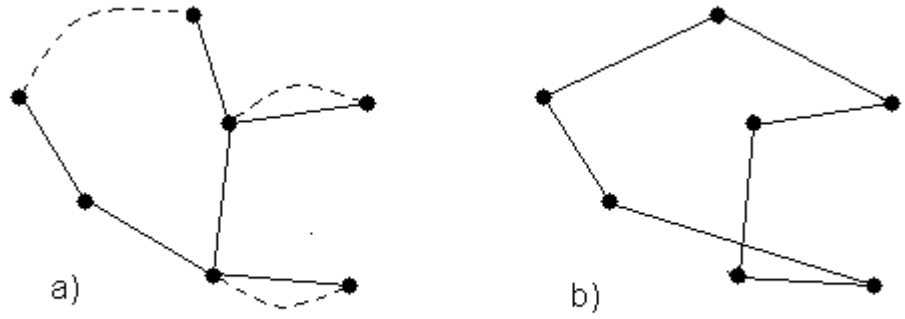
(2.) Vypočítajte systém najlacnejších hrán pre páry vrcholov nepárneho stupňa a získané hrany pridajte ku kostre.

(3.) Nájdite eulerovský ťah.

(4.) Použite ZÍSKANIE_CYKLU na vytvorenie hamiltonovského cyklu.

Táto metóda samozrejme zaberie viac času ak predchádzajúca. Jej časová náročnosť je O(n^3).

Na obrázku č.8 ilustrujem činnosť metódy Christofides. Keďže prvý krok je totožný s prvým krokom predchádzajúceho algoritmu nie je nutné ho opakovať. Na obrázku a) vidíme spôsob doplnenia kostry a obrázok d) ukazuje finálny výsledok.



Obrázok 8

Pre úlohy spĺňajúce trojuholníkovú nerovnosť metóda Christofides nedáva hroší výsledok ako 1,5 násobok optimálneho riešenia (Christofides [7]).

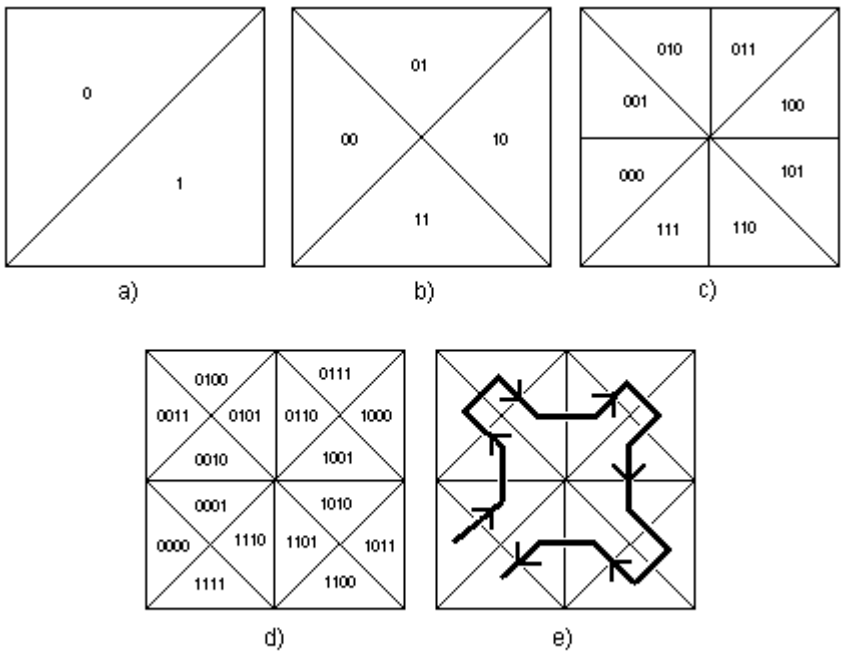
3.4. Heuristiky geometrického prístupu

Problémy, ktoré rieši úloha obchodného cestujúceho sú veľmi často geometrickej povahy, kde vrcholy sú definované ako body rozložené v priestore. Dĺžka hrán medzi jednotlivými vrcholmi sa zhoduje so vzdialenosťou podľa nejakej metriky (euklidovskej). Z tohto dôvodu sú snahy vyriešiť úlohu obchodného cestujúceho s určitou geometrickou implementáciou.

3.4.1. Heuristika vyplňovacej krivky

(Space Filling Curve heuristic)

Túto metódu, na euklidovskej rovine rozpracovali ako prvý Bartoldi a Platzman [8]. Heuristika je postavená na tzv. vyplňovacej krivke, ktorá je bijektívnym priradením definovaným nasledovne $\psi : [0, 1] \rightarrow [0, 1] \times [0, 1]$. Názov pochádza z faktu, že ak argument ψ prechádza všetkými bodmi z intervalu [0, 1], potom funkčné hodnoty vyplnia celý jednotkový štvorec. Nespornou výhodou tejto metódy je veľmi nízka časová náročnosť. Pre daný bod $y \in [0, 1] \times [0, 1]$ sa výpočet hodnoty $x \in [0, 1]$, pre ktorú platí $\psi(x) = y$ dá previesť veľmi efektívne. Autori používajú funkciu, ktorá modeluje rekurzívne delenie štvorca na rovnaké trojuholníky. Delenie prebieha do takej hĺbky, aby sa v každom trojuholníku nachádzal najviac jeden bod. Môžeme si to ilustrovať na nasledujúcom obrázku:



Obrázok 9

Na obrázkoch a) až d) je znázornené ako prebieha rekurzívne delenie. Trojuholníky sú číslované v dvojkovej sústave, spôsobom akým ich algoritmus rozdeľuje. Toto číslovanie určí poradie, v ktorom budú body zapojené. Na obrázku d) je vidieť akým spôsobom by boli trojuholníky zapojené do cyklu.

S P A C E F I L L

(1.) Prepočítajte súradnice bodov na súradnice v jednotkovom štvorci.

(2.) Pre každý bod *i* so súradnicami x_i a y_i vypočítajte z_i, pre koré platí $\psi(z_i) = (x_i, y_i)$.

(4.) Vzostupne zoraďte čísla z_i .

(5.) Spájajte body v poradí určenom z_i a na vytvorenie cyklu spojte posledný bod s prvým.

Napriek tomu, že výpočet hodnôt z_i je možný vo veľmi krátkom čase, čas na ich usporiadanie je limitujúcim faktorom pre časovú náročnosť. Odtiaľ je časová náročnosť heuristiky O(n log n).

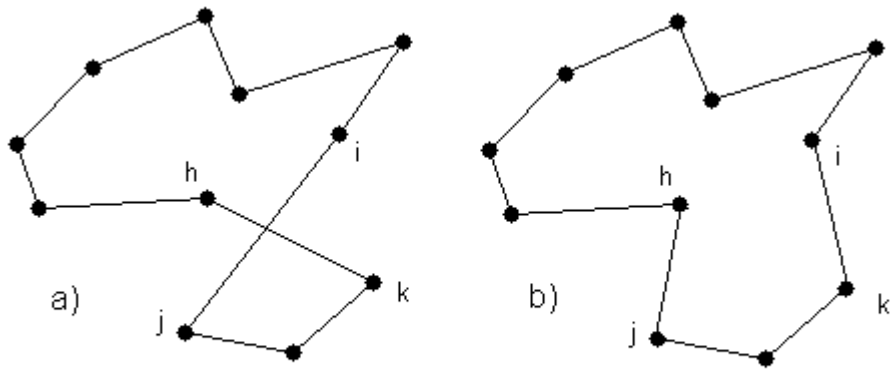
3.5. Vylepšovacie heuristiky

Doteraz sme sa zaoberali len algoritmami, ktoré nejakým spôsobom konštruovali hamiltonovský cyklus zo zadaného grafu. Výsledky však boli len priemernej kvality, a hoci sú použiteľné v niektorých aplikáciách, vo všeobecnosti stále nie sú uspokojivé. V tejto časti sa budeme zaoberať otázkou ako vylepšiť tieto známe riešenia. Teda algoritmami, ktoré vylepšujú už známe riešenie, získané heuristikami z predchádzajúcich častí.

3.5.1. Dvoj-optimalizácia

(2-opt)

Tento jednoduchý spôsob vylepšenia, vyplynul z pozorovaní euklidovských problémov. Ak hamiltonovký cyklus v určitej časti križuje sám seba, výmenou zapojenia sa dá získať cyklus kratšej dĺžky. Ilustrujme si to na obrázku č.10 :



Obrázok 10

Dvoj-optimalizačný krok pozostáva z dvoch krokov: eliminácie dvoch hrán a zapojenia takto vzniknutých voľných vrcholov, tak aby bola zachovaná súvislosť cyklu.

Celý algoritmus vyzerá nasledovne:

2 – O P T I M A L I Z Á C I A

(1) Nech *T* je ľubovoľný hamiltonovský cyklus.

(2) Kým sa neohlási chyba pre každý vrchol *i* opakujte.

(2.1) Vyberte vrchol *i* .

(2.2) Preskúmajte všetky možné 2-optimalizačné kroky pre hranu medzi vrcholom *i* a jeho pravým susedom v cykle. Zo všetkých možných skrátení vyberte to najlepšie a zapojte ho. Ak neexistuje žiadne skrátenie pre vrchol *i*, deklarujte chybu pre *i* .

(3) Ako výstup vráťte zmenené *T*

Uvedená procedúra je síce konečná, ale vo všeobecnosti nemožno potrebný čas ohraničiť polynomiálnou funkciou. Záleží na rýchlosti konvergencie vylepšení k lokálnemu minimu. Zistenie najlepšieho vylepšenia v jednom kroku si žiada $O(n^2)$ operácií, lebo je nutné preskúmať všetky páry hrán v cykle. Ale počet týchto krokov je dopredu neznámy.

3.5.2. Troj-optimalizácia

(3-opt)

Pre získanie flexibilnejších zmien môžeme použiť 3-optimalizáciu, ktorá podobne ako 2-optimalizácia pracuje so zmenami cyklu pomocou kombinovaného prepojenia voľných vrcholov, ktoré získame eliminovaním troch nesusedných hrán. Nesusednosť indikuje čistú 3-optimalizáciu, lebo vylučuje 2-optimalizačné kroky. Na rozdiel od 2-optimalizácie, kde je možné len jediné znovu zapojenie, v čistej 3-optimalizácii (bez zdegenerovaných 2-optimalizačných krokov) môžeme zapojiť vrcholy štyrmi rôznymi spôsobmi (obrázok č. 11, kde sú znázornené varianty 3-optimalizačného kroku pre vrcholy *i, j, k* a ich pravých susedov *h(i), h(j), h(k)*).

3 – O P T I M A L I Z Á C I A

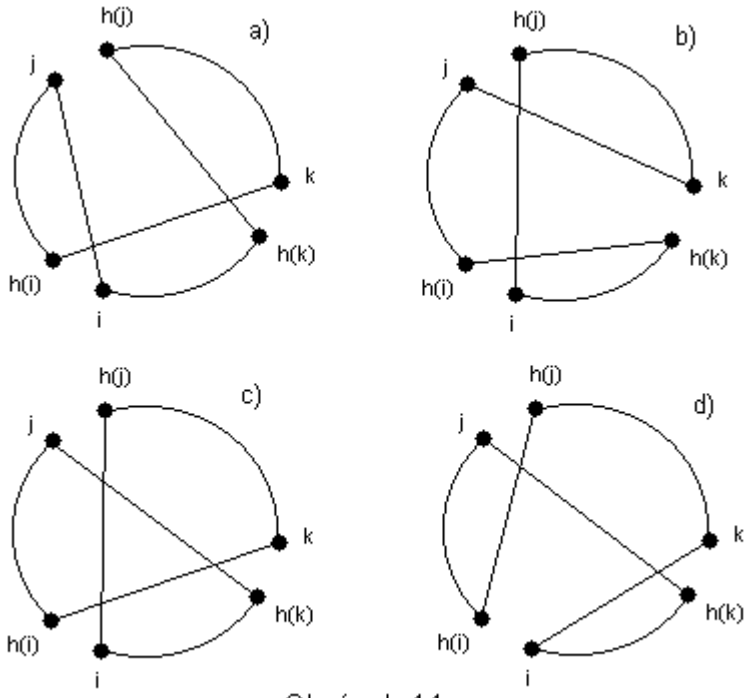
(1) Nech *T* je ľubovoľný hamiltonovský cyklus.

(2) Kým sa neohlási chyba pre každý vrchol *i* opakujte.

(2.1) Vyberte vrchol *i* .

(2.2) Preskúmajte všetky možné 3-optimalizačné kroky pre vrchol *i* a ostatné vrcholy v cykle. Zo všetkých možných skrátení (získame ho eliminovaním troch hrán a vzájomným spojením voľných vrcholov) vyberte to najlepšie a zapojte ho. Ak neexistuje žiadne skrátenie pre vrchol *i,* deklarujte chybu pre *i* .

(3) Ako výstup vráťte zmenené *T*



Obrázok 11

Rovnako ako v predchádzajúcej heuristike, tiež nevieme presne určiť počet vylepšení, ktoré sa v kroku (2) vykonajú. Vieme ale, že na nájdenie najlepšieho 3-optimalizačného kroku je nutné vykonať $O(n^3)$ operácií. A tiež úprava cyklu po 3-optimalizačnej výmene je komplikovanejšia a u 2-optimalizácii.

3.5.3. Simulované žíhanie

(Simulated Annealing)

Metóda simulovaného žíhania je založená na vzťahu medzi procesom hľadania optimálneho riešenia pre kombinatorickú optimalizáciu a niektorými fyzikálnymi fenoménmi. Podrobne sa tomu venuje Černý [9], ktorý nachádza analógiu v pomalých zmenách tepla pri minimálnej možnej zmene energii (energia je v tomto účelová funkcia podobne ako dĺžka v úlohe obchodného cestujúceho).

Hlavným prínosom tejto metódy je fakt, že všetky doteraz spomenuté metódy si nevedeli poradiť s pascou lokálneho minima. Nepripustili totiž žiadne zväčšenie dĺžky cyklu a to ani v prípade, keď by v nasledujúcich krokoch mohlo dojsť k výraznému zlepšeniu. Metóda simulovaného žíhania však kroky predlžujúce celkovú dĺžku cyklu s istou pravdepodobnosťou pripúšťa. Hlavný algoritmus simulovaného žíhania vyzerá približne takto:

S I M U L O V A N É _ Ž Í H A N I E

(1) Nech T je ľubovoľný hamiltonovský cyklus, vyberte počiatočnú teplotu τ a faktor opakovaní r.

(2) Kým nie sú naplnené podmienky pre ukončenie opakujte:

(2.1) Opakujte r-krát:

(2.1.1) Urobte náhodnú zmenu cyklu T na cyklus T_1 a vypočítajte rozdiel ich celkových dĺžok $\Delta = c(T) - c(T_1)$.

(2.1.2) Ak $\Delta < 0$, potom $T = T_1$. Inak vyberte náhodné číslo x, $0 \leq x \leq 1$, ak $x < e^{-(\Delta / \tau)}$, potom $T = T_1$.

(2.2) Upravte τ a r.

(3) Ako výstup vráťte najlepšie dosiahnuté T.

Takáto formulácia algoritmu dáva značnú voľnosť pri implementácii. Preto uvedieme niektoré konkrétne nastavenia: v kroku (1) faktor r sme inicializovale ako počet vrcholov grafu a teplotu na hodnotu 1000. K týmto hodnotám sme dospeli po sérii pokusov. Modifikáciu cyklu T na cyklus T_1 v kroku (2.1.1) sme realizovali pomocou náhodných 2, alebo 3-optimalizačných krokov. Zníženie teploty v kroku (2.2) bolo dosiahnuté položením $\tau = \lambda * \tau$, kde $\lambda < 1$ je reálne číslo blízko 1, faktor opakovania upravujeme obdobne $r = \beta * r$, β je ľubovoľné reálne číslo z intervalu (1, 2). Po dosiahnutí určitého počtu opakovaní faktor r už ďalej nezvyšujeme. Ešte zostáva uviesť kritériá na zastavenie procedúry, ktoré sme nastavili tak, aby sa proces zastavil ak nedôjde k zmene T po viacerých úpravách teploty (experimentálne sme dospeli k hodnote 10).

Počas celého procesu simulovaného žíhania sa môžeme dostať do rôznych lokálnych miním, ale môžeme ich aj opustiť. Preto je dobré ak si pamätáme najlepšie riešenie, ktoré sme počas simulácie dosiahli a za riešenie, ku ktorému algoritmus dospel položíme práve toto najlepšie riešenie, ktoré počas procesu dosiahol.

KAPITOLA 4

Experimentálne porovnanie heuristík

V kapitole 3 sme prezentovali pomerne veľký počet heuristík, a tak sa môže zdať, že niektoré z nich sú zbytočné, prípadne, že by stačilo uviesť len tú najlepšiu. Problémom je však jednoznačné rozhodnutie o víťazovi. Porovnávanie heuristík sa odohráva na dvoch úrovniach: teoretickej a experimentálnej. Keďže teoretické porovnanie už realizovali iní autori, zamerali sme sa na experimentálne porovnanie heuristík.

Porovnanie heuristík sme realizovali pomocou vlastných naprogramovaných algoritmov, na niekoľkých súboroch príkladov, ktoré pozostávali z náhodne generovaných grafov ako aj z príkladov, ktoré sme stiahli z internetu. Výsledky získané takýmto spôsobom síce môžu byť ovplyvnené kvalitou programovania, hlavne čo sa času realizácie jednotlivých príkladov týka, ale keďže všetky použité programy sú naprogramované jedným programátorom a navzájom porovnávané verzie programov sú naprogramované v tom istom programovacom jazyku, získané časy môžeme považovať za smerodajné. Štýl programovania a programovací jazyk kvalitu výsledkov (dĺžku cyklu) nijakým spôsobom neovplyvňujú.

Samotný priebeh porovnávania heuristík pozostával z vytvorenia niekoľkých verzií programov pre každú heuristiku. Typové obmedzenia pre štruktúry nám pre niektoré algoritmy nedovoľovali testovať úlohy väčších rozmerov. Tieto sú preto z testov niektorých vybraných príkladov vynechané.

Každý naprogramovaný algoritmus testuje popri kvalite riešenia aj čas behu heuristiky. Z takto získaných výsledkov sme vypracovali niektoré štatistické ukazovatele ako napríklad strednú hodnotu, najhorší chybový pomer, absolútnu odchýlku, priemernú relatívnu odchýlku a pod. Keďže pre generované úlohy sme nemali k dispozícii žiadne známe riešenia, odchýlky sa vzťahujú na výsledok tej najlepšej heuristiky pre daný príklad.

Kvôli jednoznačnosti teraz uvedieme zoznam naprogramovaných heuristík používaných v tejto kapitole. Všetky boli vysvetlené v 3. kapitole:

NN – Heuristika najbližšieho suseda

INS – Heuristika vkladania

SAVE – Metóda úspor

2TREE – Heuristika zdvojenia kostry

CHR – Christofides

SPF – Heuristika vyplňovacej krivky

2OPT – 2-optimalizačné zlepšenie

3OPT – 3-optimalizačné zlepšenie

SAN – Simulované žíhanie

Porovnávať ich budem po skupinách:

{NN, INS, SAVE, SPF, 2TREE, CHR},

{2OPT-SUS, 2OPT-RND, 3OPT-SUS, 3OPT-RND, SAN-SUS, SAN-RND}.

V prvej skupine sú združené heuristiky, ktoré konštruujú hamiltonovské cykly. Druhá skupina pozostáva z heuristík, ktorých vstupom je cyklus vytvorený niektorým z predchádzajúcich algoritmov, teda len vylepšujú už známe riešenie. Keďže sme chceli preskúmať aj citlivosť vylepšovacích algoritmov na vstupný cyklus v druhej skupine sa porovnávajú dve verzie vylepšovacích algoritmov. Príponu SUS majú algoritmy, ktorých vstupným cyklom je cyklus získaný heuristikou najbližšieho suseda a príponu RND majú algoritmy, ktoré začínajú vylepšovať náhodne vyrobený hamiltonovský cyklus.

Skôr ako sa pustíme do samotného porovnávania, uvedieme niektoré dôležité informácie o implementácii heuristík.

Všetky heuristiky z prvej skupiny boli zrealizované v základnej verzii bez dodatočných krokov, ktoré slúžia ako nadstavba pre základnú verziu. V druhej skupine ako sme už uviedli vystupujú dve verzie z každej heuristiky. Pri realizácii heuristiky 3OPT sú vylúčené 2-optimalizačné kroky, ktoré by mohli nastať ako degenerovaná verzia 3-optimlizačného kroku. Z dôvodu preukázania kvality čistého 3-optimalizačného kroku je táto podmienka nutná, a vďaka nej nedôjde k ku skresleniu výsledkov 2-optimalizáciou.

Všetky údaje použité v tejto kapitole pramenia z konkrétnych numerických experimentov, ktorých vybranú časť uvádzame v prílohe.

4.1. Porovnanie 1. skupiny algoritmov na generovaných príkladoch

Všetky algoritmy boli porovnávané na súbore 480 náhodne nagenerovaných príkladov. Generované príklady mali rozmerové rozpätie 25 až 200 bodov v euklidovskej rovine, ktoré predstavovali vrcholy kompletného grafu a dĺžky hrán boli definované ako euklidovské vzdialenosti medzi vrcholmi. Euklidovské grafy boli zvolené zámerne , aby aj algoritmus SPF mohol byť porovnávaný so všetkými ostatnými heuristikami. Teraz uvediem priemerné hodnoty dĺžky cyklov pre každú heuristiku:

SPF	NN	INS	2TREE	CHR	SAVES
7597,8	7707,2	6785,91	8034,44	7290,52	7769,23

Ako je na prvý pohľad viditeľné, rozdiely priemerných dĺžok nie sú veľmi veľké, ale poradie kvality heuristík je jednoznačné. Od najlepšieho v klesajúcom poradí kvality: INS, CHR, SPF, SAVE, NN, 2TREE. Zaujímavý je pohľad na tabuľku percentuálnych odchýliek od najlepšieho nami vypočítaného riešenia.

.

Odchýlka%	SPF	NN	INS	2TREE	CHR	SAVES
Priemerná	33,46	35,59	19,22	39,84	27,71	37,06
Maximálna	58,15	67,72	48,82	79,62	60,84	76,19

Poradie zostáva nezmenené, pričom ako jednoznačne najlepšia heuristika sa ukazuje metóda vkladania

Pre vytvorenie dokonalého obrazu o úspešnosti jednotlivých heuristík uvediem ešte aj priemerné a maximálne časy, za ktoré boli výpočty zrealizované:

Čas [sec]	SPF	NN	INS	2TREE	CHR	SAVES
Priemerný	0,000050	0,000010	0,000023	0,000380	0,000400	0,001778
Maximálny	0,050000	0,000231	0,001000	0,011000	0,011000	0,060000

Údaje sú v sekundách a vyjadrujú priemerný čas na jeden príklad a najdlhší čas pre jeden príklad. Všetky algoritmy na súbore generovaných príkladov zbehli veľmi rýchlo, a napriek tomu, že sa dá určiť najrýchlejší algoritmus tieto údaje nie sú smerodajné a časové porovnanie algoritmov uvedieme až na príklade väčších úloh.

Jednoznačným víťazom tohto porovnávania je heuristika vkladania, ktorá dosiahla ďaleko najviac víťazstiev a rovnako mala aj najlepší najhorší chybový pomer.

4.2. Porovnanie 2. skupiny algoritmov na generovaných príkladoch

Na rovnakom súbore príkladov ako predchádzajúcu skupinu sme otestovali aj vylepšovacie algoritmy. Tabuľka priemerných dĺžok vyzerá nasledovne:

2OPT-SUS	2OPT-RND	3OPT-SUS	3OPT-RND	SAN-RND	SAN-SUS
6350,02	6669,45	7072,8016	7508,29	5737,917	6311,61

V tomto porovnaní je jednoznačne najlepšou heuristikou SAN-RND. Tento poznatok by mohol naznačovať, že simulované žíhanie potrebuje pre optimálnu činnosť značnú voľnosť pri inicializácii a uzavretie inicializácie do lokálneho minima môže nepriaznivo ovplyvniť kvalitu zlepšenia . Na druhom mieste skončilo SAN-SUS, ďalej v rastúcom poradí 2OPT-SUS, 2OPT-RND, 3OPT-RND a 3OPT-SUS.

Príklady boli pravdepodobne malých rozmerov a tak sa nepreukázala predpokladaná sila 3-optimalizačného kroku. Uvedieme ešte tabuľku percentuálnych odchýliek:

Odchýlka %	2OPT-SUS	2OPT-RND	3OPT-SUS	3OPT-RND	SAN-RND	SAN-SUS
Maximálna	38,35	42,59	60,068653	82,48	7,075467	38,35
Priemerná	14,879633	17,290298	25,192658	33,37	0,0707547	11,24

Nízke hodnoty v stĺpci SAN-RND naznačujú, že najčastejším víťazom bola práve táto heuristika a teda ostatné sú porovnávané väčšinou s jej výsledkom.

Úspešnosť heuristík je charakterizovaná nielen kvalitou výsledku, ale aj prijateľným časom realizácie. Preto nahliadnime do tabuľky priemerných časov.

Čas [sec]	2OPT-SUS	2OPT-RND	3OPT-SUS	3OPT-RND	SAN-RND	SAN-SUS
Priemerný	0,003000	0,012720	0,770910	3,715350	7,955460	7,414860
Maximálny	0,020000	0,050000	4,367000	22,142000	9,744000	8,803000

Z tabuľky je zrejmé, že najlepšie kvalitatívne výsledky algoritmu SAN-RND sú vykúpené najdlhšími časmi realizácie.

Z výsledkov druhého porovnania sa dá predpovedať, že ak na príkladoch väčších rozmerov chceme pre algoritmus SAN-RND zachovať vysokú kvalitu výsledkov, pravdepodobne budeme musieť povoliť väčší počet opakovaní na jednotlivých teplotách, čo výrazne zvýši časovú

náročnosť. Taktiež by mohol nastať problém s časovou náročnosťou pre algoritmy 3OPT-SUS a 3OPT-RND, keďže pre niektoré príklady sa ukázali ako veľmi pomaly konvergujúce.

4.3. Porovnanie všetkých algoritmov internetových úlohách

Výsledky z predchádzajúcich porovnaní nám poskytli bližší pohľad na kvalitu použitých heuristík, ale zároveň vyvstala otázka, či si zachovajú svoje kvality aj na úlohách veľkých rozmerov. Z tohto dôvodu sme sa pokúsili otestovať inmlementované heuristiky na niekoľkých príkladoch väčších rozmerov. Aby sme mohli lepšie posúdiť ich fungovanie a kvalitu výsledkov, zvolili sme si príklady, ktoré už boli riešené inými riešiteľmi.

Keďže sme boli obmedzení štruktúrami programovacieho jazyka, algoritmy CHR a 2TREE sme nemohli otestovať na veľkých úlohách. Vyžadujú totiž usporiadanie hrán kompletného grafu, ktoré je pamäťovo náročné. Všetky ostatné algoritmy sa nám podarilo zrealizovať.

Uviesť priemerné dĺžky, alebo odchýlku by z dôvodov veľkej rozmanitosti príkladov nebolo práve najzreteľnejším ukazovateľom kvality výsledku, a preto uvedieme pomer našou hodnotou a uvedenou hodnotou dĺžky hamiltonovského cyklu.

úloha	spacefil	sused	vkladanie	uspory
d1655.tsp	1,49	1,21	1,27	1,08
d2103.tsp	1,51	1,08	1,10	1,05
dsj1000.tsp	1,40	1,32	1,11	1,21
fl1400.tsp	1,33	1,35	1,09	1,33
fl1577.tsp	1,70	1,23	1,24	1,10
gr666.tsp	4,37	1,40	1,31	1,32
nrw1379.tsp	1,38	1,24	1,30	1,17
pcb1173.tsp	1,44	1,24	1,29	1,21
pr1002.tsp	1,38	1,22	1,26	1,14
pr2392.tsp	1,43	1,22	1,31	1,18
rl1323.tsp	1,76	1,31	1,32	1,21
rl1889.tsp	1,65	1,27	1,17	1,13
u1060.tsp	1,45	1,36	1,26	1,27
u1817.tsp	1,50	1,25	1,29	1,12
u2152.tsp	1,51	1,23	1,27	1,12
u2319.tsp	1,14	1,19	1,13	1,13
vm1084.tsp	1,51	1,26	1,19	1,16

Z tabuľky jasne vyplýva, že až na jedinú výnimku všetky výsledky sú maximálne dvakrát horšie ako známe najlepšie riešenie. V prvej skupine sa ale na miesto víťaza dostal algoritmus SAVES. Hoci na menších úlohách nebol nijak výnimočný, na veľkých úlohách sa javí byť omnoho lepší ako ostatné algoritmy v skupine. Kvalita riešenia je však zaplatená veľmi dlhými

časmi realizácie. Jej výsledky sú však najhoršie. Oproti ostatným algoritmom, ktoré boli zrealizované takmer okamžite, je metóda SAVES veľmi pomalá. Tabuľku časov, ako aj dĺžok uvádzam v prílohe.

Rovnako ako pre algoritmy prvej skupiny aj pre druhú skupinu si uvedieme tabuľku pomerov dĺžok.

	2opt-sus	2opt-rnd	3opt-sus	3opt-rnd	zihanie-rnd	zihanie-sus
D1655.tsp	1,06	1,14	1,03	1,05	1,38	1,13
D2103.tsp	1,02	1,04	1,02	1,11	1,65	1,05
dsj1000.tsp	1,09	1,08	1,11	1,15	1,10	1,12
Fl1400.tsp	1,08	1,05	1,06	1,17	1,10	1,18
Fl1577.tsp	1,04	1,08	1,03	1,08	1,43	1,14
gr666.tsp	1,12	1,14	1,17	1,33	1,20	1,23
nrw1379.tsp	1,05	1,12	1,04	1,12	1,17	1,13
pcb1173.tsp	1,06	1,08	1,04	1,10	1,15	1,11
pr1002.tsp	1,04	1,08	1,04	1,13	1,08	1,14
pr2392.tsp	1,06	1,00	1,01	1,04	1,02	1,05
rl1304.tsp	1,08	1,11	1,09	1,16	1,22	1,12
rl1323.tsp	1,12	1,18	1,19	1,37	1,30	1,23
rl1889.tsp	1,04	1,10	1,09	1,12	1,45	1,15
U1060.tsp	1,05	1,10	1,02	1,05	1,14	1,18
U1817.tsp	1,07	1,12	1,04	1,21	1,11	1,22
U2152.tsp	1,06	1,12	1,04	1,12	1,19	1,13
U2319.tsp	1,03	1,09	1,01	1,15	1,13	1,10
vm1084.tsp	1,06	1,09	1,21	1,09	1,21	1,11

Výsledky sú veľmi vyrovnané a preto nám opäť pomôže tabuľka časov (v prílohe). Z nej jednoznačne vyplýva neefektívnosť algoritmov 3OPT-RND, 3OPT-SUS. Jej o niečo kvalitnejšie výsledky nijako nevyvážia jej časovú náročnosť. Na veľkých úlohách sa 3-optimlizácia ukázala ako časovo veľmi náročná procedúra. 3-optimalizačný krok síce môže byť použitý ako nástroj iných algoritmov, ale vylepšenia dosiahnuté výhradne pomocou 3-joptimalizačných krokov sú veľmi draho zaplatené.

4.4. Zhrnutie experimentálneho porovnania

Výsledky, ku ktorým sme dospeli, nepreukázali ani jednu heuristiku jednoznačne lepšou ako ostatné, ale dostatočné množstvo testov nám dovoľuje dojsť k určitým záverom. Záleží od našich preferencií, či obetujeme dlhý čas výpočtu pre kvalitnejšie riešenie, alebo za veľmi krátky čas získame riešenie, aj keď nie až tak dobré.

Z prvej skupiny algoritmov však ako najlepšia sa javí heuristika vkladania, ktorá dáva pomerne dobré výsledky za približne rovnaký čas ako heuristika najbližšieho suseda, ktorá sa ukázala ako najrýchlejšia.

V druhej skupine sa víťaz hľadá ťažšie, lebo na veľkých úlohách boli všetky pomerne vyrovnané. Nesmieme zabudnúť na časovú náročnosť, ktorá „diskvalifikovala“ 3-optimalizáciu. Z tohto dôvodu za víťaza druhej skupiny môžeme vyhlásiť metódu simulovaného žíhania.

Čo sa kvality riešenia týka, ani jedna z vylepšovacích heuristík nepreukázala výraznú citlivosť na inicializačný cyklus. Časová náročnosť však pri vhodnej voľbe inicializačného cyklu môže pozorovateľne klesnúť.

Z našich pozorovaní môžeme usúdiť, že vhodnou kombináciou viacerých heuristík môžeme dospieť ku kvalitnému riešeniu za veľmi prijateľný čas.

KAPITOLA 5

Záver

Cieľom tejto diplomovej práce bolo preskúmať heuristiky na riešenie úlohy obchodného cestujúceho, podať prehľad heuristík, vrátane ilustračných príkladov, experimentálne ich porovnať a vyhodnotiť.

V prehľade sme uviedli dostatočné množstvo heuristík rôznych druhov. Podarilo sa nám všetky heuristiky implementovať a otestovať na dostatočnom množstve úloh.

Vo výsledku experimentálnej časti sme dospeli k jednoznačnému riešeniu pre konštrukčné ako aj pre vylepšovacie algoritmy.

Vopred vytýčené ciele sme teda splnili. Naviac, výsledky získané pri experimentálnom porovnávaní umožňujú lepšie využitie heuristík, a teda aj dosiahnutie lepších výsledkov.

Zoznam literatúry

[1] M.Jünger,G. Reinelt, G.Rinaldi, The Traveling Salesman Problem ,IWR, 1994, 1-45

[2] J.Plesník, Grafové algoritmy, Veda, 1983

[3] J. Bosák, Grafy a ich aplikácie, Alfa, 1980

[4] H. Crowder, M.W. Padberg, Solving Large-Scale Symmetric Traveling Salesman Problems to Optimality, Management Science, 1980, 495-509

[5] D.J.Rosenkratz, R.E.Stearns, P.M.Lewis, An analysis of several heuristics for the traveling salesman problem, SIAM J. Computing 6, 1977, 563-581

[6] G.Clark, J.W.Wright, Scheduling of vehicles from a central depot to a number of delivery points, Operation Research 12, 1964, 568-581

[7] N.Christofides, Worst Case Analysis of a New Heuristic for the Travelling Salesman Problem, Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, 1979

[8] J.J. Bartholdi, L.K. Platzman, Spacefilling Curves and the Planar Travelling Salesman Problem, Journal of the Association for Computing Machinery, 1989

[9] V.Černý, A Thermodynamical Approach to the Travelling Salesman Problem: An Efficient Simulation Algoritm, J.Optimization Theory and Applications 45, 1985, 41-51

PRÍLOHA A

Výsledky experimentálneho testovania

Tabuľky dĺžok internetových úloh:

úloha	Internet	spacefil	sused	vkladanie	uspory
d1655,tsp	62128	92573	75474	78756	67295
d2103,tsp	80450	121436	87279	88713	84423
dsj1000,tsp	18659688	26216164	24630960	20764022	22552880
fl1400,tsp	20127	26712	27179	21865	26713
fl1577,tsp	22249	37747	27429	27631	24502
gr666,tsp	294358	12871	4111	3860	3877
nrw1379,tsp	56638	78050	70015	73646	66227
pcb1173,tsp	56892	82133	70278	73296	68869
pcb3038,tsp	137694	181351	175573	175207	163906
pr1002,tsp	259045	356585	315597	326518	296488
pr2392,tsp	378032	541567	461207	493661	445380
rl1304,tsp	252948	424923	339797	307207	289901
rl1323,tsp	252948	445052	332095	334003	305763
rl1889,tsp	316536	520966	400685	369266	356588
u1060,tsp	224094	325721	305781	283256	284240
u1817,tsp	57201	85835	71406	73645	63970
u2152,tsp	64253	97273	79340	81363	71726
u2319,tsp	234256	266136	278783	265736	265822
vm1084,tsp	239297	362393	301469	284172	276909

úloha	internet	2opt-sus	2opt-rnd	3opt-sus	3opt-rnd	zihanie-rnd	zihanie-sus
d1655,tsp	62128	65855,68	70825,92	63991,84	65234,4	85736,64	70204,64
d2103,tsp	80450	82059	83668	82059	89299,5	132742,5	84472,5
Dsj1000,tsp	18659688	20339060	20152463	20712254	21458641	20525657	20898851
fl1400,tsp	20127	21737,16	21133,35	21334,62	23548,59	22139,7	23749,86
fl1577,tsp	22249	23138,96	24028,92	22916,47	24028,92	31816,07	25363,86
gr666,tsp	294358	329681	335568,1	344398,9	391496,1	353229,6	362060,3
Nrw1379,tsp	56638	59469,9	63434,56	58903,52	63434,56	66266,46	64000,94
pcb1173,tsp	56892	60305,52	61443,36	59167,68	62581,2	65425,8	63150,12
pr1002,tsp	259045	269406,8	279768,6	269406,8	292720,9	279768,6	295311,3
pr2392,tsp	378032	400713,9	378032	381812,3	393153,3	385592,6	396933,6
rl1304,tsp	252948	273183,8	280772,3	275713,3	293419,7	308596,6	283301,8
rl1323,tsp	252948	283301,8	298478,6	301008,1	346538,8	328832,4	311126
rl1889,tsp	316536	329197,4	348189,6	345024,2	354520,3	458977,2	364016,4
u1060,tsp	224094	235298,7	246503,4	228575,9	235298,7	255467,2	264430,9
u1817,tsp	57201	61205,07	64065,12	59489,04	69213,21	63493,11	69785,22
u2152,tsp	64253	68108,18	71963,36	66823,12	71963,36	76461,07	72605,89
u2319,tsp	234256	241283,7	255339	236598,6	269394,4	264709,3	257681,6
Vm1084,tsp	239297	253654,8	260833,7	289549,4	260833,7	289549,4	265619,7

Tabuľky časov realizácie výpočtov internetových úloh:

(údaje sú v sekundách)

úloha	spacefil	sused	vkladanie	uspory
d1655.tsp	0,000	0,060	0,170	204,684
d2103.tsp	0,000	0,050	0,390	395,429
dsj1000.tsp	0,000	0,000	0,060	46,827
fl1400.tsp	0,000	0,060	0,110	118,791
fl1577.tsp	0,000	0,000	0,170	174,892
gr666.tsp	0,000	0,000	0,060	13,009
nrw1379.tsp	0,001	0,050	0,110	123,027
pa561.tsp	0,000	0,000	0,000	8,623
pcb1173.tsp	0,000	0,000	0,110	76,380
pr1002.tsp	0,000	0,050	0,050	46,697
pr2392.tsp	0,002	0,110	0,600	640,951
rl1304.tsp	0,000	0,050	0,110	105,161
rl1323.tsp	0,000	0,060	0,110	109,187
rl1889.tsp	0,002	0,110	0,280	319,269
u1060.tsp	0,000	0,050	0,050	54,478
u1817.tsp	0,000	0,050	0,220	293,792
u2152.tsp	0,002	0,050	0,380	484,507
u2319.tsp	0,002	0,110	0,550	501,662
vm1084.tsp	0,000	0,060	0,110	57,543

úloha	2opt-sus	2opt-rnd	zih-rnd	zih-sus	3opt-sus	3opt-rnd
d1655.tsp	37,81	176,75	35,16	42,31	5116,49	5643,16
d2103.tsp	38,43	77,80	41,54	39,64	9164,11	10036,30
dsj1000.tsp	12,88	89,16	16,30	12,16	1694,31	2641,64
fl1400.tsp	48,79	196,07	34,79	33,63	2894,31	3142,64
fl1577.tsp	41,43	87,97	36,14	26,69	1946,32	2616,31
gr666.tsp	2,62	4,24	26,00	16,30	1261,10	891,61
nrw1379.tsp	33,94	207,34	34,35	31,20	6941,11	7631,94
pa561.tsp	1,54	2,32	19,30	9,66	931,16	650,31
pcb1173.tsp	18,57	35,14	27,29	27,29	1012,03	1643,09
pr1002.tsp	10,46	6,96	18,02	20,65	813,62	900,61
pr2392.tsp	207,42	0,65	41,86	38,69	8654,36	9846,91
rl1304.tsp	20,83	140,27	27,27	62,39	5001,32	6435,91
rl1323.tsp	15,70	158,86	27,57	29,61	3619,31	4036,61
rl1889.tsp	72,48	682,85	35,82	39,65	6435,19	7694,96
u1060.tsp	16,03	8,12	22,89	31,20	1369,89	4619,61
u1817.tsp	68,13	83,85	38,75	40,97	2983,65	3649,91
u2152.tsp	93,91	164,76	43,82	39,84	8695,60	10369,16
u2319.tsp	148,50	188,25	41,91	40,62	12369,13	19496,16
vm1084.tsp	10,94	99,82	20,67	21,36	7316,10	3451,61

.

PRÍLOHA B

Naprogramovaných bolo 12 heuristík, všetky v jazyku Delphi 6. Podrobný návod na použitie prikladám v súbore README.TXT spolu so zdrojovými programami na priloženej diskete.

Výpis použitých procedúr:

```
procedure Sused;
var m,i,j,l,pom,pom1:integer;
      koniec:integer;
      min:real;
      W:array[1..max] of integer;
 {mnoz. volnych vrcholov}
begin
  min:=10000000; koniec:=1;
  for j:=1 to N do W[j]:=1;
  dlzka:=0; W[1]:=0;  i:=1;
  repeat
    for j:=1 to N do begin
 {vyhladame najblizsieho suseda z volnych}
      if W[j]=1 then begin
        if G[i,j] < min then begin
          min:=G[i,j]; koniec:=j; l:=i;
        end;
      end;
    end;
    W[koniec]:=0;
    dlzka:=dlzka + min;
    min:=10000000;
    H[l]:=koniec;
    i:=koniec;
    pom:=0;
    for pom1:=1 to N dopom:=pom+W[pom1];
  until pom=0;
  dlzka:=dlzka + G[1,koniec];
  H[koniec]:=1;
end;

 procedure Zmena_cesty(odkial,kam:integer);
          var a,b,poz:integer;
      Cesta:array[1..max] of integer;
 begin
    poz:=2;  a:=odkial;
    Cesta[1]:=odkial;
    while H[a] <> kam do begin
  {nacita povodnu cestu}
      Cesta[poz]:=H[a];   a:=H[a];   poz:=poz+1;
    end;
    Cesta[poz]:=kam;
    for b:=poz downto 2 do H[Cesta[b]]:=Cesta[b-1];
  {vymeni cestu za opacnu}
 end;


 procedure DvaOptKrok;
  var i,j,k,pomi,pomj,
     minbod1,minbod2:integer;
      min:double;
      {minim. lokalnych zmien}
      lokuspora,pom:real;
      minor:double;
  {dlzka cyklu bez zamienanych hran}
 begin
  repeat
    lokuspora:=0; pom:=0;
    for i:=1 to N do begin
      for j:=i+1 to N do begin
        if (H[i]<>j) and (H[j]<>i) then begin
          pom:=G[i,H[i]]+G[j,H[j]]-G[i,j]-G[H[i],H[j]];
        end;
        if lokuspora<pom then begin
          lokuspora:=pom;
          minbod1:=i; minbod2:=j;
pomi:=H[minbod1];            pomj:=H[minbod2];
        end;
      end;
    end;
    dlzka:=dlzka-lokuspora;
    if lokuspora<>0 then begin
      Zmena_cesty(pomj,minbod1);
      H[pomj]:=pomi;
      H[minbod2]:=minbod1;
    end;
  until lokuspora=0;
       end;

procedure pgraf;
{očísluje hrany}
var i,j:integer;
      dlzka:real;
 begin
  index:=1;
  for i:=1 to N do begin
    for j:=i+1 to N do begin
       D[index].i:=i; D[index].j:=j;
       D[index].dlzka:=G[i,j];
       index:=index+1;
  end; end; end;



procedure Mintree;
{nájde najlacnejšiu kostru}
  var znak:array[1..max] of integer;
      pridal:boolean;
      pocethran,pom1,pom2,kk,jj:integer;
 begin
```

```
  index:=index-1;
  QuickSort(D, 1, index);
  pocethran:=0;
  for jj:=1 to N do znak[jj]:=jj;
  for kk:=1 to max do Kostra[kk]:=0;
  jj:=1;
  while pocethran<>N-1 do begin
    pridal:=False;
    while not pridal do begin
      if znak[D[jj].i]<>znak[D[jj].j] then begin
        Kostra[pocethran+1]:=jj;  pridal:=True;
        if znak[D[jj].i] < znak[D[jj].j] then begin
          pom1:=znak[D[jj].i]; pom2:=znak[D[jj].j];
        end else begin
          pom2:=znak[D[jj].i]; pom1:=znak[D[jj].j];
        end; end;
      for kk:=1 to N do begin
        if znak[kk]=pom2 then znak[kk]:=pom1;
      end;  jj:=jj+1;
    end; pocethran:=pocethran+1;
  end; end;
procedure euler;
 var OBJ:array[1..max,1..max] of byte;
    znak,navsteva:array[1..max] of byte;
    E:array[1..maxhran] of integer;
    som,posun1,posun2,posun3,i,j,counter:integer;
    {OBJ: neprejdene  0  neobjavitelske  1
    objavitelske  2    protismerne  3
    neexistujuce  5}

 begin    {procedury euled}
   {inicializacia pomocnej struktury pre cestovanie
grafom}
   for i:=1 to N do begin
     for j:=1 to N do  OBJ[i,j]:=5; end;
   for i:=1 to (N-1)do begin
      OBJ[D[Kostra[i]].i,D[Kostra[i]].j]:=0;
      OBJ[D[Kostra[i]].j,D[Kostra[i]].i]:=0;
   end;
   {vytvorenie eulerovskeho ťahu}
   {znak: neznamy vrchol  0    znamy vrchol  1}
   for i:=1 to maxhran do E[i]:=0; posun1:=1;
   for i:=1 to N do znak[i]:=0;    som:=1;
   znak[som]:=1;  counter:=1;  E[1]:=som;
   while (posun1<>0) or (posun2<>0) or
(posun3<>0) do begin
     posun1:=0; posun2:=0; posun3:=0;
     for i:=1 to N do begin
       if (OBJ[som,i]=0) and (OBJ[i,som]=0) then
posun1:=i;
       if (OBJ[som,i]=0) and (OBJ[i,som]=1) then
posun2:=i;
       If (OBJ[som,i]=0) and (OBJ[i,som]=2) then
posun3:=i; end;
     if posun1<>0 then begin
       if znak[posun1]=0 then begin
         OBJ[som,posun1]:=2; som:=posun1;
         znak[posun1]:=1; counter:=counter+1;
         E[counter]:=posun1;
       endelse begin
         OBJ[som,posun1]:=1; som:=posun1;
         counter:=counter+1; E[counter]:=posun1;
       end; end else begin
       if posun2<>0 then begin
         OBJ[som,posun2]:=3; som:=posun2;
         counter:=counter+1;
         E[counter]:=posun2;
       end else begin
         if posun3<>0 then begin
           OBJ[som,posun3]:=3; som:=posun3;
           counter:=counter+1; E[counter]:=posun3;
       end; end; end; end;
   {ziskanie cyklu z euklidovskeho sledu}
   som:=E[1]; dlzka:=0;
   for i:=1 to N do navsteva[i]:=0;
   navsteva[E[1]]:=1; i:=2; j:=1;
   while j<N do begin
     if navsteva[E[i]]=0 then begin
       dlzka:=dlzka+G[som,E[i]];
       navsteva[E[i]]:=1;
       H[som]:=E[i]; som:=E[i]; j:=j+1;
     end;  i:=i+1;
   end;  H[E[i-1]]:=E[1]; end;

procedure stupen(var stup:pole);
{urci stupen vrchola}
 var i:integer;
 begin
   for i:=1 to N do stup[i]:=0;
   for i:=1 to N-1 do begin
     inc(stup[D[Kostra[i]].i]);
     inc(stup[D[Kostra[i]].j]);
   end;  end;

 procedure perfectmatch;
{vypočíta pridané }
 var i,j,k:integer;
    nebola:boolean;
 begin
   for i:=1 to N do neparnost[i]:=True;
   pocetneparnych:=0;
   for i:=1 to N do odskok[i]:=0;
   for i:=1 to N do begin
     if (stup[i] mod 2) = 0 then begin
       neparnost[i]:=False
     end else begin
       pocetneparnych:=pocetneparnych+1;
     end; end;
   j:=1;i:=0;
   while pocetneparnych<>0 do begin
     if neparnost[D[j].i] and neparnost[D[j].j] then
begin
       nebola:=true; k:=1;
       while nebola and (k <= N-1) do begin
         if Kostra[k] = j then nebola := false;
         k:=k+1;
       end;
       if nebola then begin          {nie je to odbocka}
         Kostra[N+i]:=j;
         pocetneparnych:=pocetneparnych-2;
```

```
        neparnost[D[j].j]:=False;
        neparnost[D[j].i]:=False;
        i:=i+1;
       end else begin              {je to odbocka}
        if (stup[D[kostra[k-1]].i]<3) or
(stup[D[kostra[k-1]].j]<3) then begin
          if stup[D[kostra[k-1]].i] = 1 then begin
            odskok[D[kostra[k-1]].i]:=D[kostra[k-1]].j;
          end else  begin
            odskok[D[kostra[k-1]].j]:=D[kostra[k-1]].i;
          end;
          Kostra[k-1]:=0;
          pocetneparnych:=pocetneparnych-2;
          neparnost[D[j].j]:=False;
          neparnost[D[j].i]:=False;
        end; end; end;
    j:=j+1; end;
   X:=i; end;



procedure saving(var priebdlzka:real);
    const uhlopriecka=1.0e10;
    var i,j,sur1,sur2,minbod1,minbod2,bbbbb:integer;
{bbbbb-indikuje  neexistenciu dalsieho zlepsenia }
        Dvojnik:array[1..max] of integer;{indikator
dvojiteho zapojenia}
        Opakovac:array[1..max,1..2] of integer;
{kontroluje neopakovanie neustale toho isteho
vylepsenia}
  lokalmin,min,opakovanie:real;
  begin
    min:=1.0e10; bbbbb:=2; opakovanie:=0;
    for i:=2 to N do Dvojnik[i]:=2;
    for i:=1 to N do for j:=1 to 2 do Opakovac[i,j]:=0;
    while bbbbb<>0 do begin
      min:=uhlopriecka;
      for i:=2 to N do begin
        for j:=i+1 to N do begin
          if (Dvojnik[i]>0) and (Dvojnik[j]>0) and
(Opakovac[i,1]<>j)
          and(Opakovac[i,2]<>j)  then begin
            if (G[1,i]+G[1,j])>G[i,j] then begin
        lokalmin:=G[i,j];sur1:=i;sur2:=j;opakovanie:=0;
            end;     end;
          if min>lokalmin then begin
 min:=lokalmin;  minbod1:=sur1;    minbod2:=sur2;
end; end;  end;
      if opakovanie=1 then bbbbb:=0;
      opakovanie:=1;
      if min<1.0e10 then begin
        priebdlzka:=priebdlzka+G[minbod1,minbod2];
        if Opakovac[minbod1,1]=0 then
Opakovac[minbod1,1]:=minbod2
        else Opakovac[minbod1,2]:=minbod2;
        Dvojnik[minbod1]:=Dvojnik[minbod1]-1;
        Dvojnik[minbod2]:=Dvojnik[minbod2]-1;
      end;   end; for i:=2 to N do begin
      if Dvojnik[i]<0 then Dvojnik[i]:=0;
      priebdlzka:=priebdlzka+(Dvojnik[i]*G[1,i]);
    end;  end;
function locate(xs,ys:real):real;
{priradí hodnotu pre každý bod štvorca podľa
rekurzívneho delenia SPACEFILL}
const kmax=20;
  var x,y,k,z:real;
      i:real;
  begin
   x:=xs;  y:=ys;
   i:=0;k:=1;
   if x>y then begin
     i:=1; x:=m-x;  y:=m-y;
   end;
   while k<kmax do begin
     i:=i+i;  k:=k+1;
     if x+y>m then begin
       i:=i+1;  z:=m-y;
       y:=x;  x:=z;
     end;
     if k<kmax then begin
       i:=i+i;  k:=k+1;
       x:=x+x;  y:=y+y;
       if y>m then begin
         i:=i+1;  z:=y-m;
         y:=m-x;   x:=z;
       end;  end;  end;
   locate:=i;
       end;

function Optimalizuj(i,ii,j,jj,k,kk:integer; kod:byte;
minor:double):double;
  var pom:double;
  begin
    case kod of
     1: pom:=minor+G[i,j]+G[H[i],k]+G[H[j],H[k]];
     2: pom:=minor+G[i,k]+G[H[i],H[j]]+G[j,H[k]];
     3: pom:=minor+G[i,H[j]]+G[H[i],H[k]]+G[j,k];
     4: pom:=minor+G[i,H[j]]+G[j,H[k]]+G[H[i],k]
    end;
    Optimalizuj:=pom;
  end;

 procedure Upravit_cyklus(i,ii,j,jj,k,kk:integer;
kod:byte);
    procedure Zmena_cesty(odkial,kam:integer);
      var a,b,poz:integer;
          Cesta:array[1..max] of integer;
     begin
       poz:=2;  a:=odkial;
       Cesta[1]:=odkial;
       while H[a] <> kam do begin     {nacita povodnu
cestu}
         Cesta[poz]:=H[a];   a:=H[a];   poz:=poz+1;
       end;
       Cesta[poz]:=kam;
       for b:=poz downto 2 do H[Cesta[b]]:=Cesta[b-1];
                             {vymeni cestu za opacnu}
     end;
   begin
    case kod of
```

```
    1: begin
       Zmena_cesty(ii,j);
       Zmena_cesty(jj,k);
       H[ii]:=k; H[i]:=j; H[jj]:=kk;
      end;
    2: begin
       Zmena_cesty(jj,k);
       H[jj]:=ii;  H[i]:=k;  H[j]:=kk;
      end;
    3: begin
       Zmena_cesty(ii,j);
       H[ii]:=kk;  H[i]:=jj;  H[k]:=j;
      end;
    4: begin
       H[i]:=jj;  H[j]:=kk;  H[k]:=ii;
      end;
   end;
  end;

 procedure TriOpt;
   var i,j,k:integer;   {lave vrcholy vymennych hran}
       ii,jj,kk:integer;  {prave vrcholy vymennych
hran}
       LMIN:array[1..4] of double;   {lokalne zmeny}
       min:double;          {minim. lokalnych zmien}
       kkk,ikoniec,jkoniec,kkoniec:integer;
{maximalna pozicia i,j,k-tej prem.}
       kod:byte;      {kod pouzitej alternativy zameny
hran}
       a,imin:byte;
       skok:boolean;    {indikacia zlepsenia}
       minor:double;    {dlzka cyklu bez zamienanych
hran}
  begin
    repeat { for  kkk:=1 to 10 do begin}
      ikoniec:=1;
      for i:=1 to N-6 do begin
        ikoniec:=H[ikoniec];
      end;
      jkoniec:=H[H[ikoniec]];
      kkoniec:=H[H[jkoniec]];
      i:=1; j:=H[H[i]]; k:=H[H[j]];
      ii:=H[i]; jj:=H[j]; kk:=H[k];
      skok:=false;
      while (i <> ikoniec) and not skok do begin
        while (j <> jkoniec) and not skok do begin
          while (k <> kkoniec) and not skok do begin
            minor:=dlzka-(G[i,ii]+G[j,jj]+G[k,kk]);
            for kod:=1 to 4 do
LMIN[kod]:=Optimalizuj(i,ii,j,jj,k,kk,kod,minor);
            min:=LMIN[1];  imin:=1;
            for a:=2 to 4 do begin
              if LMIN[imin] > LMIN[a] then begin
                imin:=a;  min:=LMIN[a];
              end;
            end;
            if dlzka > min then begin
              skok:=true;
              Upravit_cyklus(i,ii,j,jj,k,kk,imin);
              dlzka:=min;
              end;
              k:=kk;  kk:=H[k];
            end;
            j:=jj; jj:=H[j];  k:=H[jj]; kk:=H[k];
          end;
          i:=H[i]; ii:=H[i]; j:=H[ii];  jj:=H[j]; k:=H[jj];
kk:=H[k];
        end;
        if skok then begin
{         Upravit_cyklus(i,ii,j,jj,k,kk,kod);
        dlzka:=min;}
        end else begin
          minor:=dlzka-(G[i,ii]+G[j,jj]+G[k,kk]);
          for kod:=1 to 4 do
LMIN[kod]:=Optimalizuj(i,ii,j,jj,k,kk,kod,minor);
          min:=LMIN[1];  imin:=1;
          for a:=2 to 4 do begin
            if LMIN[imin] < LMIN[a] then begin
              imin:=a;  min:=LMIN[a];
            end;
          end;
          if dlzka > min then begin
            skok:=true;
            Upravit_cyklus(i,ii,j,jj,k,kk,imin);
            dlzka:=min;
          end;
        end;
    {   write('Dlzka = ');
        writeln(dlzka:0:4);  }
     until not skok; (*  }  end;*)
  end;

procedure Vkladanie;
   var i,j,jmin:integer;
       vzd,min:double;
  begin
    for i:=4 to N do begin
      min:=maxint;
      for j:=1 to i-1 do begin
        vzd:=dlzka+G[i,j]+G[H[j],i]-G[j,H[j]];
        if vzd<min then begin
          min:=vzd;  jmin:=j;
        end;
      end;
{dlzka:=dlzka-G[jmin,H[jmin]]+min;}  dlzka:=min;
      H[i]:=H[jmin];  H[jmin]:=i;
    end;
  end;
```

```
procedure Zihanie;
    var
teplota,zmena,Delta,Lambda,Beta,XX,pom2:real;
        {R je faktor opakovania}
        i,x,y,z,R,bezzmeny,a,counter:integer;
   begin
```

```
   teplota:=1000; R:=N;
   Bezzmeny:=0;
   Lambda:=0.99;
   Beta:=1.2;
   min:=0;
   mincyklus:=dlzka;
   while bezzmeny < 10 do begin
    for i:=1 to R do begin
      zmena:=random;
      x:=0; y:=0; z:=0;
      if zmena < 0.5 then begin
        while (x=y) or (H[x]=y) or (H[y]=x) do begin
          x:=random(N)+1;
          y:=random(N)+1;
        end;
        Delta:=DvaOpt(x,y);pomx:=H[x];
pomy:=H[y];
      end
      else begin
        while (x=y) or (x=z) or (y=z) or (x=H[z]) or
(h[x]=z)
        or (H[x]=y) or (H[y]=z) or (H[z]=y) or
(H[y]=x) do begin
          x:=random(N)+1;
          y:=random(N)+1;
          z:=random(N)+1;
        end;
        a:=x;
        while (H[a] <> y) and (H[a] <> z) do a:=H[a];
        if H[a]=z then begin
          a:=z; z:=y; y:=a;
        end;
        Delta:=TriOpt(x,y,z); pomy:=H[y];
pomx:=H[x]; pomz:=H[z];
      end;
      if Delta < 0 then begin
        if zmena < 0.5 then begin
          Zmena_cesty_2opt(H[y],x);
          H[pomy]:=pomx; H[y]:=x;
          Bezzmeny:=0; dlzka:=dlzka+delta;
          lokmin:=dlzka;
          if lokmin < mincyklus then begin
          min:=lokmin;
       for counter:=1 to N
doLH[counter]:=H[counter];
          end;
        end else   begin
         Upravit_cyklus(x,pomx,y,pomy,z,pomz);
          Bezzmeny:=0;
          dlzka:=dlzka+delta;
          lokmin:=dlzka;
          if lokmin < mincyklus then begin
            min:=lokmin;
            for counter:=1 to N do
LH[counter]:=H[counter];
          end;    end;
      end else  begin
        XX:=exp(-Delta/teplota); pom2:=random;
        if pom2 < XX then begin
          if zmena < 0.5 then begin
          Zmena_cesty_2opt(H[y],x);
H[pomy]:=pomx;  H[y]:=x;
          Bezzmeny:=0;    dlzka:=dlzka+Delta;
          lokmin:=dlzka;
          if lokmin < mincyklus then begin
            min:=lokmin;
            for counter:=1 to N do
LH[counter]:=H[counter];
          end;
        end else  begin
Upravit_cyklus(x,pomx,y,pomy,z,pomz);
Bezzmeny:=0;
          dlzka:=dlzka+Delta;
          lokmin:=dlzka;
          if lokmin < mincyklus then begin
            min:=lokmin;
            for counter:=1 to N do
LH[counter]:=H[counter];
          end;  end;  end;  end;  end;
      Bezzmeny:=Bezzmeny+1;
      if teplota < 1 then Bezzmeny:=20;
      Teplota:=Teplota*Lambda;
      R:=trunc(R*Beta);
      if R> 10000 then R:=10000;
    end;
  end;
```