UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A
INFORMATIKY

Problém dvoch obchodných cestujúcich

2013

Vladimír Boža

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A
INFORMATIKY

Problém dvoch obchodných cestujúcich

(diplomová práca)

Študijný program:	Informatika
Študijný odbor:	9.2.1 Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	RNDr. Michal Forišek PhD.

Bratislava, 2013

Vladimír Boža

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Vladimír Boža
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský

Názov: Problém dvoch obchodných cestujúcich

Cieľ: Cieľom práce je skúmať problém dvoch obchodných cestujúcich (v angličtine "2-peripatetic traveling salesman problem"). Inštanciou pre tento optimalizačný problém je neorientovaný úplný ohodnotený graf výstupom sú dve vrcholovo disjunktné kružnice pokrývajúce všetky vrcholy grafu optimalizuje sa súčet dĺžok kružníc. Keďže súvisiace rozhodovacie problémy sú zjavne NP-ťažké, je cieľom práce preskúmať iné možné prístupy k riešeniu problému: skúmanie jeho ľahších podproblémov (napr. verzie s Euklidovskou metrikou), hľadanie zaručených aproximačných algoritmov vrátane prípadnej PTAS, ako aj použitie heuristických metód. Súčasťou práce by mala byť ako teoretická analýza, tak aj praktická implementácia navrhnutých algoritmov.

Vedúci: RNDr. Michal Forišek, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: doc. RNDr. Daniel Olejár, PhD.

Dátum zadania: 26.02.2013

Dátum schválenia: 18.03.2013 prof. RNDr. Branislav Rovan, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Abstrakt

V práci sa zaoberáme modifikáciou problému obchodného cestujúceho – problémom dvoch obchodných cestujúcich, t.j. hľadáme dve hranovo disjuktné hamiltonovské kružnice, ktorých súčet dĺžok je čo najmenší. Pre riešenie tohoto problému sme pomocou dynamického programovania zostrojili exaktný algoritmus, ktorého čas behu je výrazne lepší ako čas behu triviálneho algoritmu. Ďalej sme modifikovali PTAS pre problém obchodného cestujúceho v euklidovskej rovine, aby sme pomocou neho zostrojili PTAS pre problém dvoch obchodných cestujúcich. Nazáver ešte porovnávame praktickú výkonnosť heuristík využívajúcich celočíselné lineárne programovanie.

Kľúčové slová: obchodný cestujúci, PTAS, dynamické programovanie, celočíselné lineárne programovanie

Abstract

In this work we focus on modification of travelling salesman problem – 2-peripatetic salesman problem, where the aim is to find two disjoint hamiltonian circles of minimum total cost. We use dynamic programming to develop exact algorithm, which running time is much smaller than running time of trivial algorithm. We modified PTAS for travelling salesman problem in euclidean plane to develop PTAS for 2-peripatetic salesman problem. We finish our work with practical comparison of several heuristics using integer linear programming.

Key words: travelling salesman, PTAS, dynamic programming, integer linear programming

Obsah

Úvod

Problém obchodného cestujúceho je pomerne známy a dobre preskúmaný problém v teoretickej informatike. Navyše má veľa aplikácii v plánovaní, logistike, atď.

V tejto práci sa zaoberáme modifikáciou problému obchodného cestujúceho. Namiesto jednej hamiltonovskej kružnice budeme hľadať dve hranovo disjuktné hamiltonovské kružnice a budeme minimalizovať ich celkovú dĺžku. Tento problém nazveme problémom dvoch obchodných cestujúcich.

Ako prvý tento problém zaviedol [Kra75]. V literatúre sa spomína niekoľko aplikácii tohoto problému. Napríklad [WCC03] skúma návrh ciest pre strážnikov, kde je cieľom nájsť niekoľko rôznych kružníc, aby sa cesty neopakovali. Ďalšie aplikácie spomína [DLS07]. Napríklad pri transporte nebezpečných materiálov chceme distribuovať riziko a trasy kamiónov navrhnúť tak, aby zdieľali čo najmenej ciest.

V tejto práci navrhneme niekoľko nových algoritmov pre riešenie problému dvoch obchodných cestujúcich. Práca je členená nasledovne: V kapitole 1 formálne zadefinujeme problém a jeho variácie, ktoré budeme riešiť. V kapitole 2 zhrnieme doterajšie výsledky pri skúmaní problému dvoch obchodných cestujúcich. V kapitolách 3 a 4 sme navrhli algoritmy, ktorých význam je hlavne teoretický. V kapitole 3 sme navrhli exaktný algoritmus pre riešenie problému dvoch obchodných cestujúcich. V kapitole 4 popisujeme PTAS pre inštancie, kde vrcholy predstavujú body v euklidovskej rovine. Napokon v kapitole 5 navrhneme niekoľko nových heuristík porovnávame ich výsledky s heuristikami známymi doteraz.

Kapitola 1

Definícia problému

V tejto kapitole formálne zadefinujeme problém dvoch obchodných cestujúcich a jeho variácie, ktoré budeme riešiť.

Definícia 1.0.1 (Problém dvoch obchodných cestujúcich) *Majme daný úplný ohodnotený graf $G = (V, E)$ s n vrcholmi, pričom každá hrana $e \in E$ má dĺžku $c(e)$. Cieľom je nájsť dve hranovo disjuktné hamiltonovské kružnice $K_1 \subset E$, $K_2 \subset E$, tak aby súčet ich dĺžok bol minimálny. Kružnice K_1, K_2 budeme volať aj TSP-cesty.*

Poznámka 1.0.1 Cenu hranu spájajúcej vrcholy u, v budeme označovať ako $c(u, v)$.

V niektorých prípadoch kladieme na dĺžky hrán ďalšie obmedzenia. Jedným z obmedzení je trojuholníková nerovnosť:

Definícia 1.0.2 (Trouholníková nerovnosť) *Hrany uhodnoteného grafu $G = (V, E)$ spĺňajú trouholníkovú nerovnosť, ak platí:*

$$\forall u, v, w \in V : c(u, v) + c(v, w) \geq c(u, w)$$

Ešte silnejším obmedzením je požadovať, aby vrcholy grafu zodpovedali bodom v rovine, t.j. každému vrcholu priradíme reálne čísla x_v, y_v a vzdialenosť vrcholov u, v bude: $c(u, v) = \sqrt{(x_u - x_v)^2 + (y_u - y_v)^2}$.

Kapitola 2

Doterajšie výsledky

Problém dvoch obchodných cestujúcich je zjavne NP-úplný, ako to dokázal už [Kra75], keď tento problém zadefinoval.

V nasledujúcich riadkoch uvedieme niekoľko doterajších výsledkov, ktoré sa vyskytli pri riešení tohoto problému.

2.1 Aproximačné algoritmy

Dá sa pomerne ľahko ukázať, že pre problém dvoch obchodných cestujúcich vo všeobecných grafoch neexistuje aproximačný algoritmus s konštantným aproximačným pomerom. Dôkaz je v prakticky rovnaký ako pre problém obchodného cestujúceho.

V prípade, že graf spĺňa trojuholníkovu nerovnosť sa už dajú nájsť approximačné algoritmy. Dá sa napríklad ukázať, že ak má graf dostatočne veľa vrcholov, tak ku každej TSP ceste existuje TSP cesta, ktorá je s ňou disjuktná a má dĺžku najviac rovnú $(2+\epsilon)$-násobku dĺžky pôvodnej cesty. Použítím 1.5-aproximácie pre problém obchodného cestujúceho ([Chr76]) vieme dostať $(9/4+\epsilon)$-aproximačný algoritmus.

Dá dosiahnúť aj 2-aproximačný algoritmus, ako to je ukázané v [AP08]. Hlavnou ideou je zostrojiť najprv dve disjuktné kostry s minimálnou cenou pomocou algoritmu uvedeného v [RT85] a následne pomerne technickou konštrukciou zo-

strojiť s použitím týchto kostier dve disjunktné hamiltonovské kružnice.

V prípade, že ceny hrán môžu byť len z množiny $\{1, 2\}$ existuje 11/9-aproximačný algoritmus ([Gim]). V tomto článku sa nachádza navyše niekoľko výsledkov pre variáciu problému, kde cenu ciest maximalizujeme.

2.2 Riešenia pomocou celočíselného lineárneho programovania

2.2.1 Riešenie problému obchodného cestujúceho pomocou celočíselného lineárneho programovania

Najprv zhrnieme niekoľko výsledkov o riešení problému obchodného cestujúceho pomocou celočíselného lineárneho programovania, ktoré sa nachádzajú v [BN68].

Problém obchodného cestujúceho sa dá priamočiaro popísať nasledujúcim celočíselným lineárnym programom:

$$\min \sum_{e \in E} c(e) x_e$$

Za predpokladov:

$$\forall e \in E : x_e \in \{0, 1\} \tag{2.1a}$$

$$\forall v \in V : \sum_{e \in \delta(v)} x_e = 2 \tag{2.1b}$$

$$\forall S \subset V, S \neq \emptyset : \sum_{e \in \delta(S)} x_e \geq 2 \tag{2.1c}$$

Kde $\delta(v)$ je množina všetkých hrán, ktoré majú koniec vo vrchole v a $\delta(S)$ je množina všetkých hrán, ktorá majú jeden koniec v S a druhý v $V \setminus S$.

Podmienky (2.1b) zabezpečujú, aby z každého vrchola vychádzali práve dve hrany. Podmienky (2.1c) zabezpečujú, aby sme mali iba jeden cyklus. Táto formulácia má ale exponenciálne veľa podmienok a na prvý pohľad je neužitočná.

Existuje formulácia, ktorá má iba polynomiálne veľa podmienok. V tomto prípade ale použijeme orientované hrany:

$$\min \sum_{i \neq j} c(i,j) x_{ij}$$

Za predpokladov:

$$\forall j \neq i_0 : \sum_{i=1}^{n} x_{ij} = 1 \tag{2.2a}$$

$$\forall i \neq i_0 : \sum_{j=1}^{n} x_{ij} = 1 \tag{2.2b}$$

$$\forall i, j, i \neq i_0 \vee j \neq j_0 : u_i + u_j + n x_{ij} \leq n - 1 \tag{2.2c}$$

Cieľom premenných u_i je zaviesť očíslovanie vrcholov na kružnici. Tento program má síce len polynomiálne veľa podmienok, ale ukazuje sa, že nie je ľahko riešiteľný pomocou bežných nástrojov na riešenie celočíselných lineárnych programov.

Ukazuje sa, že pomerne účinný algoritmus je nasledovný:

1. Vytvor lineárny program, ktorá obsahuje iba podmienky (2.1a), (2.1b).
2. Najdi riešenie daného lineárneho programu.
3. Ak je riešenie lineárneho programu iba jedna kružnica, tak máme dobré riešenie a skonči.
4. Ináč nájdi porušené podmienky (2.1c) a pridaj ich do lineárneho programu. A pokračuj krokom 2.

Tomuto postupu sa zvykne hovoriť aj „subtour elimination“.

2.2.2 Riešenie problému dvoch obchodných cestujúcich pomocou celočíselného lineárneho programovania

V nasledujúcej časti zhrnieme výsledky z [DLS07]. Priamym rozšírením postupu pre obchodného cestujúceho na dvoch cestujúcich dostaneme nasledovný program:

$$\min \sum_{e \in E} c(e) x_e + \sum_{e \in E} c(e) y_e$$

Za predpokladov:

$$\forall e \in E : x_e \in \{0, 1\} \tag{2.3a}$$

$$\forall e \in E : y_e \in \{0, 1\} \tag{2.3b}$$

$$\forall e \in E : x_e + y_e \leq 1 \tag{2.3c}$$

$$\forall v \in V : \sum_{e \in \delta(v)} x_e = 2 \tag{2.3d}$$

$$\forall v \in V : \sum_{e \in \delta(v)} y_e = 2 \tag{2.3e}$$

$$\forall S \subset V, S \neq \emptyset : \sum_{e \in \delta(S)} x_e \geq 2 \tag{2.3f}$$

$$\forall S \subset V, S \neq \emptyset : \sum_{e \in \delta(S)} y_e \geq 2 \tag{2.3g}$$

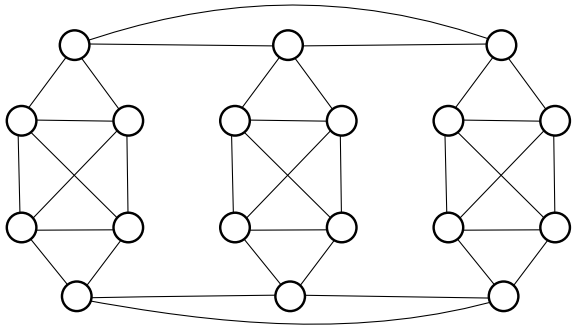
Spôsob riešenie je podobný ako pri riešení problému obchodného cestujúceho – najprv nepoužijeme žiadne z podmienok (2.3f), (2.3g) a postupne pridáme len porušené podmienky. V [DLS07] sa tento algoritmus nazýva „3-index“.

Ukazuje sa, že tento algoritmus je pomerne pomalý a potrebuje veľa iterácií riešenia lineárneho programu. Pri praktickom testovaní na náhodných euklidovských grafoch je tento algoritmus úspešný maximálne pre grafy s 50 vrcholmi. Hlavným problémom tohoto algoritmu je, že pomerne často nastáva taký prípad, keď len prehadzujeme hrany z x do y a naopak.

V [DLS07] tento postup ešte vylepšili. Použili algoritmus s nasledujúcou myšlienkou: Namiesto hľadania dvoch disjuktných hamiltonovských kružníc budeme hľadať najlacnejší 4-regulárny 4-súvislý [1] podgraf. Následne ak sa tento podgraf dá rozložiť na dve hamiltonovské kružnice, tak máme riešenie. Toto sa nemusí vždy podariť, na 2.1 je príklad grafu, ktorý je 4-súvislý a 4-regulárny, ale nemá hamiltonovskú kružnicu.

Ak sa nám nepodarí 4-faktor rozložiť na dve hamiltonovské kružnice, tak tento 4-faktor zakážeme a budeme hľadať najlacnejší nezakázaný podgraf. Iný pohľad na tento algoritmus je taký, že v lineárnom programe zlúčíne premenné x_e a y_e do jednej, čiže aktuálny program, ktorý budeme nazývať 2-index vyzerá takto:

[1] tu a v ďalšom texte 4-súvislý znamená hranovo 4-súvislý



Obr. 2.1: Príklad grafu, ktorý je 4-súvislý a 4-regulárny, ale nie je hamiltonovský.

$$\min \sum_{e \in E} c(e) x_e$$

Za predpokladov:

$$\forall e \in E : x_e \in \{0, 1, 2\} \tag{2.4a}$$

$$\forall v \in V : \sum_{e \in \delta(v)} x_e = 4 \tag{2.4b}$$

$$\forall S \subset V, S \neq \emptyset : \sum_{e \in \delta(S)} x_e \geq 4 \tag{2.4c}$$

Aktuálne už hľadanie porušených podmienok (2.4c) nie je také jednoduché. Nestačí už len hľadať komponenty vzniknutého grafu potrebujeme hľadať rezy, ktoré majú veľkosť menšiu ako 4. Na toto môžeme napríklad použiť algoritmus z [SW97].

Väčším problémom je zistenie, či sa dá 4-regulárný 4-súvislý podgraf rozdeliť na dve disjuktné hamiltonovské kružnice. Tento problém je NP-úplný ako ukázal [Pik96]. V [DLS07] na riešenie tohoto problému používajú upravený 3-index algoritmus. Takýto 2-index algoritmus funguje pre náhodné euklidovské grafy, ktoré majú menej ako 200 vrcholov.

Kapitola 3

Pomalé metódy riešenia

V tejto kapitole sa budeme venovať pomalým metódam riešenia problému dvoch obchodných cestujúcich. Najskôr prezentujeme niekoľko pomerne triviálnych výsledkov na základne známych algoritmov a potom prezentujeme náš algoritmus, ktorý je asympoticky rýchlejší.

Navyše v tento kapitole budeme používať označenie $f(n) = \tilde{O}(g(n))$, ktoré má rovnaký význam ako $f(n) = O(g(n)n^{O(1)})$, t.j. okrem konštatných faktorov, ignorujeme aj všetky faktory, ktoré sú menšie alebo rovné ako polynomiálne.

3.1 Skúšanie všetkých možností

V grafe s n vrcholmi sa nachádza $n!$ rôznych kružníc. Pokiaľ hľadáme dve disjuktné kružnice, tak rôznych dvojíc kružníc je $\tilde{O}((n!)^2)$. Každú z týchto možností vieme overiť v čase $O(n)$. Celkový čas algoritmu je teda $\tilde{O}((n!)^2)$.

3.2 Dynamické programovanie pre problém obchodného cestujúceho

Problém obchodného cestujúceho sa dá riešiť pomocou dynamického programovania v čase $O(2^n n^2)$ ([HK61]). Hlavnou myšlienkou tohoto algoritmu je riešiť podproblémy tvaru: Nájdite najkratšiu cestu, ktorá začína vo vrchole 1, prechádza cez

množinu vrcholov $X \subseteq V$ a končí vo vrchole $x \in X$.

Tento algoritmus vieme využiť pomerne priamočiaro. Pre každú z $n!$ možných kružníc v čase $\tilde{O}(2^n)$ nájdeme najkratšiu druhú kružnicu. Takto dostaneme algoritmus, ktorý beží v čase $\tilde{O}(2^n n!)$.

3.3 Lepšie dynamické programovanie

V tejto časti prezentujeme vlastný exaktný algoritmus na riešenie problému dvoch obchodných cestujúcich.

Budeme robiť dynamické programovanie priamo pre problém dvoch obchodných cestujúcich. Náš prístup bude podobný ako pri probléme obchodného cestujúceho. Budeme hľadať jednu najkratšiu cestu cez danú podmnožinu vrcholov X, ktorá začína vo vrchole 1 a končí v danom vrchole x. Potrebujeme ešte vyriešiť druhú kružnicu. V tomto prípade použijeme pomerne hrubú silu. Množinu X rozložíme na tri disjuktné množiny A, B, C. Vrcholy v množine A zatiaľ do druhej kružnice zapojené nebudú. Vrcholy v množine C budú spojené v druhej kružnici s inými dvoma vrcholmi. A vrcholy v množine B budú spojené iba s jedným vrcholom. Čiže druhá kružnica je rozložená na niekoľko ciest, ktoré začínajú vo vrcholoch množiny B, idú cez niekoľko (alebo nula) vrcholov z množiny C a následne končia vo vrchole z množiny B. Ešte ale potrebujeme jednu informáciu navyše – ktoré koncové body ciest v množine B sú spojené (čiže máme rozdelenie množiny B do dvojíc). Túto informáciu potrebujeme nato, aby sme zabránili zacykleniu druhej cesty.

V konečnom dôsledku pre každú kombináciu množín A, B, C vrchola x a rozdelení vrcholov v množine B hľadáme cestu p cez vrcholy v množine $A \cup B \cup C$, ktorá začína vo vrchole 1 a končí vo vrchole x a množinu ciest R, ktoré spĺňajú podmienky množín A, B, C, kde cesty z R sú disjuktné s p a navyše cesty z R a p sú v súčte čo najkratšie.

Tento podproblém vieme jednoducho riešiť pomocou riešení menších podproblémov. Vyskúšame všetky predchádzajúce vrcholy pre vrchol x v prvej ceste a zároveň podľa jeho príslušnosti do množiny A, B, C si vyberieme vhodné zapoje-

nie do druhej cesty (kontrolujeme, či cesty sú disjuktné a či sa druhá cesta nezacyklila). Keďže nové hrany pridávame len poslednému vrcholu, vieme priamočiaro kontrolovať podmienku disjuknosti ciest.

Tento algoritmus vieme ešte urýchliť jedným orezávaním. Veľkosť množiny B môže byť maximálne $\frac{2}{3}n$. V opačnom prípade by sme nevedeli tieto cesty spojiť do kružnice (každý zo zvýšných vrcholov vie spojiť dva voľné konce v množine B).

Teraz odhadneme zložitosť vyššie spomínaného algoritmu. Hlavným faktorom bude počet rôznych podproblémov (vyriešenie podproblému nám trvá polynomiálny čas od n).

Nech $P(2k)$ je počet rôznych rozdelení $2k$ prvkov do dvojíc:

$$P(2k) = \frac{(2k)!}{k!2^k}$$

Pre jednoduchosť dodefinujeme $P(2k+1) = P(2k)$.

Celkový počet podproblémov odhadneme zhora. Máme najviac 4^n možností pre rozdelnie vrcholov do množín A, B, C a zvyšok. Jeden vrchol navyše vyberieme ako špeciálny. Ešte potrebujeme spárovať množinu B a na to máme najviac $P\left(\left\lfloor\frac{2}{3}n\right\rfloor\right)$ možností. A teda počet rôznych podproblémov môžeme odhadnúť ako:

$$S(n) \leq n4^n P\left(\left\lfloor\frac{2}{3}n\right\rfloor\right)$$

Keď na $P(2k)$ použijeme Stirlingovu approximáciu dostávame:

$$P(2k) = \frac{\sqrt{4\pi k}\frac{(2k)^{2k}}{e^{2k}}(1+o(1))}{\sqrt{2\pi k}\frac{k^k}{e^k}2^k(1+o(1))} = \frac{\sqrt{2}\cdot 2^k n^k}{e^k}(1+o(1)) = \frac{2^k k!}{\sqrt{\pi n}}(1+o(1))$$

Po dosadení máme:

$$S(n) = \tilde{O}\left(2^{2.334n}\left(\frac{n}{3}\right)!\right)$$

Rovnaká (až na polynomiálny faktor) je aj časová zložitosť vyššie popisaného algoritmu.

Kapitola 4

Aproximačné algoritmy

Zamerajme sa teraz na inštancie, kde vrcholy grafu reprezentujú body v rovine a hrany vzdialenosti medzi nimi. V prípade obyčajného problému obchodného cestujúceho existuje PTAS (polynomial time approximation scheme) ([Aro98]), t.j. algoritmus, ktorý pre ľubovoľný aproximačný pomer beží v polynomiálnom čase od veľkosti vstupu.

My v nasledujúcej časti použijeme myšlienky z tohoto algoritmu a ukážeme náš PTAS algoritmus pre problém dvoch obchodných cestujúcich.

Hlavnou myšlienkou algoritmu je pomocou techniky rozdeľuj a panuj vybudovať nad vstupom randomizovaný quad-tree. Následne ukážeme, že pre $(1 + 1/c)$-aproximačný algoritmus stačí, aby TSP-cesty pretínali hrany delenia iba $O(c)$ krát. Navyše tieto prieniky sa môžu udiať iba v niektorých špeciálnych bodoch.

4.1 Rekurzívne delenie

Pre začiatok predpokladajme, že všetky body majú celočíselné súradnice a vzdialenosť medzi nimi je aspoň 8. Neskôr ukážeme, že každý vstup vieme takto upraviť a ak budeme mať PTAS pre upravený vstup, tak budeme mať PTAS aj pre pôvodný vstup.

Nech všetky body ležia vo vnútri štvorca S so stranou L, ktorého ľavý dolný roh má súradnice $[0, 0]$. Navyše bez ujmy na všeobecnosti nech je L mocnina dvojky.

Rekurzívnym delením štvorca S nazývame jeho rekurzívne delenie na menšie štvorce tak, že každy štvorec rozdelíme **deliacimi čiarami** na štyri rovnaké štvorce. Deliť prestaneme, keď je strana štvorca ≤ 1. Toto delenie predstavuje 4-árny strom, kde synovia každé ho štvorca sú štyri menšie štvorce. Hĺbka takéhoto stromu je najviac $O(\log_2 L)$. Štvorcom delenia intuitívne priradíme úrovne. Štvorec S má úroveň 0, jeho synovia úroveň 1, atď. Deliaca čiara má úroveň i vtedy, keď obsahuje hranu nejakého štvorca úrovne i, ale žiadneho úrovne väčšej ako i.

Quadtree definujeme podobne až na to, že rekurzívne delenie skončí vtedy, keď v danom štvorci je najviac jeden bod. Keďže každý list quadtree obsahuje bod zo vstupu, alebo je súrodenec takého listu, tak quadtree má maximálne $O(n)$ listov a teda dokopy najviac $O(n \log_2 L)$ štvorcov.

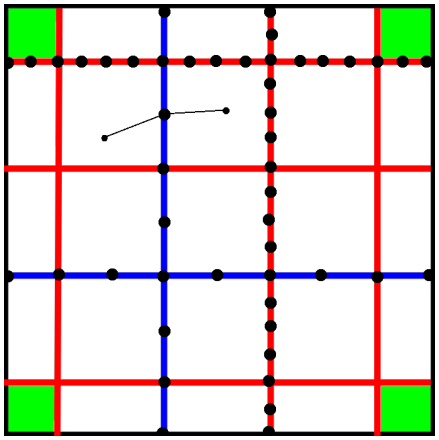
Teraz popíšeme posunuté delenie. Nech a, b sú celé čísla a platí $0 \leq a, b < L$. (a, b)-posunutie rekurzívneho delenia definujeme tak, že x-, $y-$ súradnice každej deliacej čiary zvýšime o a resp. b a následne spočítame ich zvyšok po delení číslom L (vstupnými bodmi nehýbeme). Takisto posunieme aj okrajové čiary štvorca (začnú tvoriť deliace čiary). Pôvodný štvorec pokladáme za zacyklený, teda okrajové oblasti pri posunutom delení tvoria súvislú oblasť ako na obrázku 4.1.

Z posunutého rekurzívneho delenia vieme dodefinovať quadtree s posunom tak, že nebudeme deliť štvorce, ktoré obsahujú najviac jeden bod.

4.2 Portály a (m, r)-ľahké cesty

Nás algoritmus dovolí TSP-cestám pretínať čiary delenia iba v niekoľkých predpísaných bodoch, tzv. **portáloch**. Toto je na prvý pohľad trochu kontraintuitívne. My ale dovolíme cestujúcim cestovať medzi bodmi po ľubovoľnej čiare, teda nie nutne najkratšej ceste medzi dvoma bodmi (viď. obr. 4.1). Stále ale zachovávame disjuktnosť ciest, t.j. keď jeden cestujúci ide z vrchola x do vrcholu y, tak druhý nemôže ísť z x do y ani z y do x (nech ide ľubovoľne zakrivenou cestou).

Definícia 4.2.1 *Nech m je párne kladné celé číslo. m-regulárna množina portálov pre posunuté rekurzívne delenie je množina bodov taká, že každá deliacia čiara úrovne i obsahuje portály vo vzdialenostiach $\frac{L}{m2^i}$.*



Obr. 4.1: Posunuté rekurzívne delenie a portály

Definícia 4.2.2 *TSP-cesta sa nazýva (m, r)-ľahká ak pretína čiary rekurzívneho delenia iba v m-regulárnej množine portálov a navyše každú hranu štvorca pretína najviac r-krát.*

Teraz sformulujeme hlavnú vetu, ktorá dokazuje koreknosť nášho algoritmu.

Veta 4.2.1 *Nech $c > 0$ je ľubovoľná konštanta. Nech je každá nenulová vzdialenosť vo vstupe aspoň 8 a nech všetky body ležia vo vnútri štvorca so stranou L. Nech $0 \leq a, b < L$ sú vybrané náhodne. Potom s pravdepodobnosťou aspoň $1/2$ existujú dve disjuktné (m, r)-ľahké TSP-cesty, ktorých dĺžka je maximálne $(1+1/c)$-násobok optimálnej dĺžky, pričom $m = O(c \log_2 L)$, $r = O(c)$.*

Dôkaz tejto vety uvedieme neskôr. Najprv sa pozrime na hľadanie najkratších (m, r)-ľahkých ciest.

4.3 Popis algoritmu

V tejto časti ukážeme ako sme algoritmus od [Aro98] upravili tak, aby riešil problém dvoch obchodných cestujúcich.

Ako sme už spomínali, algoritmus predpokladá, že vstupná inštancia je „slušná“, t.j.:

- Body majú celočíselné súradnice.
- Nenulová vzdialenosť bodov je aspoň 8.
- Najväčšia vzdialenosť bodov je $O(n)$.

Vstupnú inštanciu upravíme na inštanciu, ktorá spĺňa tieto podmienky nasledovne: Nech L_0 je dĺžka strany štvorca v ktorom leží vstupná inštancia a d je dĺžka optimálneho riešenia. Navyše platí $L_0 \leq d$. V štvorci vyrobíme mriežku s veľkosťou bunky $\frac{d}{8nc}$ a každy bod presunieme do najbližšieho mrežového bodu. Dĺžka jednej kružnice sa zmení maximálne o $\frac{2nL_0}{8nc} < \frac{d}{4c}$. Dĺžka dvoch kružníc sa zmení maximálne o $\frac{d}{2c}$. Teraz zmenšíme všetky vzdialenosť $\frac{L_0}{64nc}$ krát. Teda všetky súradnice budú celočíselné a minimálna nenulová vzdialenosť bodov bude aspoň 8. Navyše dĺžka strany štvorca, v ktorom ležia body, bude $O(nc)$, čo je $O(n)$, keďže c je konštanta. Po takejto transformácii, ale namiesto $1+1/c$ aproximácie musíme hľadať $1+1/2c$ aproximáciu, ale to nám nevadí, keďže c môže byť ľubovoľná kladná konštanta.

V nasledujúcom kroku zvolíme hodnoty a, b – posuny quad-tree. Môžeme ich zvoliť náhodne a uspokojiť sa s tým, že máme iba pravdepodnostný algoritmus, alebo môžeme postupne vyskúšať všetky hodnoty za cenu vyššej časovej zložitosti o faktor $O(n^2)$. Pre zvyšok tejto kapitoly budeme predpokladať, že a, b volíme náhodne. Teraz môžeme skonštruovať posunutý quad-tree. Keďže po predchádzajúcom kroku máme $L = O(n)$ a hĺbka quad-tree je teda najviac $O(\log_2 n)$, potom počet štvorcov v quad-tree je $O(n \log_2 n)$ (lebo máme maximálne n listov). Tento quadtree vieme pomerne jednoducho skonštruovať v čase $O(n \log_2 n)$.

Teraz ideme nájsť v skonštruovanom posunutom quad-tree nájsť najlacnejšie (m, r)-ľahké disjuktné TSP cesty. Využijeme techniku dynamického programovania.

Zoberme si štvorec S z quad-tree a optimálne (m, r)-ľahké TSP cesty. Prvá z nich pretína hranicu štvorca S postupne v bodoch $a_1, a_2, \ldots, a_{2p}$, kde $2p \leq 4r$ a druhá v bodoch $b_1, b_2, \ldots, b_{2q}$, kde $2q \leq 4r$.

Časti TSP ciest vo vnútri S je postupnosť ciest, ktoré majú konce v bodoch a_{2i-1}, a_{2i}, resp. b_{2j-1}, b_{2j}, pre $i = 1, 2, \ldots, p$, resp. $j = 1, 2, \ldots q$. Tieto cesty navyše

prechádzajú všetkými bodmi vo vnútri S a ich zjednotenie je (m, r)-ľahké. Navyše sú tieto cesty disjuktné, t.j. neexistujú dva vrcholy v_i, v_j vo vnútri S také, že v obidvoch postupnostiach sú v_i a v_j spojené. Ešte si všimnime vrcholy, ktoré nasledujú na cestách po jednotlivých prienikoch (keď pokračujeme po ceste od prieniku smerom dovnútra štvorca). Postupne ich označme pre prvú cestu ako $v_1^a, v_2^a, \ldots, v_{2p}^a$ a pre druhú cestu ako $v_1^b, v_2^b, \ldots, v_{2q}^b$. Tieto vrcholy nemusia byť nutne vrcholy vo vnútri S, keďže časť cesty môže prechádzať cez štvorec S, ale nemusí prechádzať cez žiadny vrchol.

Keďže máme optimálnu (m, r)-ľahkú cestu, tak aj postupnosť ciest uvedená vyššie je optimálna, t.j. najlacnejšia taká, že jednotlivé cesty majú konce v zadaných bodoch $a_1, a_2, \ldots, a_{2p}$, resp. $b_1, b_2, \ldots, b_{2q}$, najbližšie vrcholy ku koncom sú $v_1^a, v_2^a, \ldots, v_{2p}^a$, resp. $v_1^b, v_2^b, \ldots, v_{2q}^b$ a tieto cesty prechádzajú všetkými bodmi vo vnútri S a zároveň sú disjuktné.

Teraz môžeme definovať vhodný podproblém pre dynamické programovanie. Jeho vstupmi budú:

- Štvorec S z posunutého quad-tree.
- Postupnosti portálov, ktoré majú cesty pretínať: $a_1, a_2, \ldots, a_{2p}, b_1, b_2, \ldots, b_{2q}$.
- Postupnosti vrcholov, do ktorých pôjdu cesty za portálmi: $v_1^a, v_2^a, \ldots, v_{2p}^a$,. $v_1^b, v_2^b, \ldots, v_{2q}^b$.

Jeho výstupom je postupnosť ciest definovaná vyššie.

Predtým než ukážeme ako tento podproblém riešiť spočítame počet rôznych podproblémov. Ako sme spomínali máme $O(n \log_2 n)$ štvorcov. Rôznych postupnosti portálov bude $O(m^{O(r)})$. Rôznych postupnosti vrcholov bude $O(n^{O(r)})$. Keďže $r = O(c)$, tak stále máme PTAS. Neskôr ukážeme ako zmenšiť člen $O(n^{O(r)})$ na člen $O(n)$.

Podproblémy vieme riešiť nasledovne. Pokiaľ obsahuje štvorec iba jeden vrchol vieme v čase $O(r)$. vyskúšať všetky možné priradenia. Zobereme si teraz štvorec S, ktorý má synov S_1, S_2, S_3, S_4. Pre jeho synov už máme vyriešené všetky podproblémy. Chceme teraz nájsť najlacnejšie (m, r)-ľahké cesty pre štvorec S. Preskúšame všetky možnosti akoby cesty mohli prechádzať hranami štvorcov S_1, S_2, S_3, S_4. To

obnáša vybrať poradie prechodu portálmi a vrcholy, ktorými cesta bude prechádzať po prechode portálom. Ešte potrebujeme skontrolovať disjukntosť ciest. Medzi vrcholmi vrámci štvorcov S_1, S_2, S_3, S_3 máme disjuknosť zaručenú. Treba ešte zaručiť aj disjuktnosť ak cesta prechádza medzi vrcholmi z rôznych štvorcov. To vieme zaručiť pomocou toho, že sa pozrieme aké vrcholy nasledujú za portálmi a nepovolíme tie možnosti, v ktorých sú spojené rovnaké dvojice vrcholov v oboch TSP cestách. Časová zložitosť tohoto kroku je najviac $O(n^{O(r)}m^{O(r)})$.

Celkový čas tohoto algoritmu je $O(n^{O(r)}m^{O(r)})$, čo je $O(n^{O(c)})$.

4.4 Dôkaz hlavnej vety

Pri dôkaze hlavnej vety využijeme dve pomocné lemy. Prvou z nich je tzv. „Patching lema“. Pre problém obchodného cestujúceho je jej znenie nasledovné:

Lemátko 4.4.1 (Patching lema) *Existuje konštanta $g > 0$ taká, že platí nasledovné: Nech S je úsečka dĺžky s a π je uzavretá cesta, ktorá pretína S aspoň 3-krát. Potom existuje množina podúsečiek S s dĺžkou najviac gs, ktorých pridaním do π dostaneme cestu π', ktorá pretína S najviac 2-krát.*

Dôkaz tejto lemy sa dá nájsť v [Aro98]. My ho tu stručne zhrnieme.

Dôkaz. Predpokladajeme, že π pretína S práve t krát v bodoch $M_1, M_2, \ldots, M_t$ – body označíme v poradí v akom sa vyskytujú na úsečke S. Cestu π v týchto bodoch rozsekneme na t častí $P_1, P_2, \ldots, P_t$. Z každého bodu M_i si pripravíme dve kópie – M_i' a M_i'', každú kópiu umiestnime na jednu stranu S. Nech J je multimnožina úsečiek, ktorá obsahuje nasledovné:

- Najlacnejšiu kružnicu, ktorá prechádza cez body $M_1, M_2, \ldots, M_t$.
- Najlacnejšie perfektné párenie medzi bodmi $M_1, M_2, \ldots, M_{2k}$, kde $2k$ je najväčšie párne číslo menšie ako t.

Množina J má dĺžku najviac $3s$. Množinu J opäť zoberieme v dvoch kópiách – J', J'' a pridáme ju k ceste π. Ešte navyše ak je t nepárne, tak spojíme body M'_{2k+1}

a M''_{2k+1} a ak je t párne, tak spojíme body M'_{2k+1} a M''_{2k+1} a body M'_{2k+2} a M''_{2k+2}. Spolu cesty $P_1, P_2, \ldots, P_t$ a pridané úseky tvoria 4-regulárny graf na vrcholoch $\{M'_1, \ldots, M'_t\} \cup \{M''_1, \ldots, M''_t\}$. Eulerovský ťah na tomto grafe prechádza všetkými cestami $P_1, \ldots, P_t$ a navyše pretína S najviac dvakrát. Navyše súčet všetkých pridaných úsečiek je najviac $6s$. A teda sme dokázali túto vetu pre $g = 6$.

My sme túto lemu upravili pre dvoch obchodných cestujúcich.

Lemátko 4.4.2 *Existuje konštanta $g > 0$ taká, že platí nasledovné: Nech S je úsečka dĺžky s a π je uzavretá cesta, ktorá pretína S aspoň 20-krát. Nech π_2 je cesta disjuktná s π. Potom existuje množina podúsečiek S s dĺžkou najviac gs, ktorých pridaním do π dostaneme cestu π', ktorá pretína S najviac 20-krát a zároveň nepretína cestu π_2.*

Dôkaz. Vďaka patching leme vieme cestu π previesť na cestu π', ktorá pretína S najviac dvakrát. Predpokladajme, že π' bola zostrojená pomocou predchádzajúceho dôkazu a teda vedie cez body $M'_1, \ldots, M'_t, M''_1, \ldots, M''_t$. Táto cesta môže ale zdieľať niektoré hrany s cestou π_2. Teda môže existovať niekoľko dvojíc bodov (x, y) takých, ktoré sú spojené aj v π aj v π_2. Týchto dvojíc je najviac $2t$, keďže najviac $2t$ bodom sme zmenili suseda.

Zakázanou dvojicou budume nazývať dvojicu bodov (x, y) takú, že x a y sú susedné na ceste π_2. Každý vrchol sa teda vyskytuje práve v dvoch zakázaných dvojiciach.

Našim cieľom je upraviť cestu π' tak, aby neobsahovala ani jednu zakázanú dvojicu. Zakázané dvojice budeme odstránovať postupne.

Nech (x, y) je zakázaná dvojica, ktorá sa vyskytuje v π'. Rozlišujeme niekoľko prípadov (všetky prípady majú obdobu pre druhú stranu úsečky S).

1. Cesta π' od x smerom k y najprv prechádza bodmi M'_i a M'_{i+1} (resp. M'_{i-1}.
2. Cesta π' od x smerom k y najprv prechádza bodmi M'_t a M'_1 (alebo opačne).
3. Cesta π' od x smerom k y najprv prechádza bodmi M'_i a M''_i (alebo opačne), kde $i = t$, alebo $i = t - 1$.

Myšlienka odstránenia zakázanej dvojice bude nasledovná. V blízkosti $M_i^{'}$ (resp. $M_t^{'}$) nájdeme úsečku medzi bodmi $M_j^{'}$, $M_j^{'}$, cez ktorú cesta π' spája vrcholy u, v a dvojice $(x,u),(x,v),(y,u),(y,v)$ nie sú zakázané. Potom prehodíme úsečky tak, že spojíme vrcholy x a v a vrcholy y a u (resp. opačne), tak aby sme opäť dostali uzavretú cestu. Teraz už len potrebujeme dokázať, že úsečka s požadovanou vlastnosťou existuje.

Ukážeme to pre prvý prípad, ostatné sú obdobné. Nech (x,a) a (y,b) sú zakázané dvojice pre body x, y iné ako (x,y). Zoberme si postupne najbližších susedov $M_i^{'}$. Pre suseda $M_j^{'}$ sa pozrime na najbližší vrchol na ceste P_j (resp. P_{j+1}), ktorá tam končí (predpokladáme, že táto cesta má aspoň jeden vrchol, ináč by sme počet pretnutí úsečky vedeli znížiť iným spôsobom). Tento vrchol označme z. Bod $M_j^{'}$ leží teda medzi vrcholom z a nejakým iným vrcholom $z^{'}$.

Našim cieľom je nájsť $M_j^{'}$ také, že z a $z^{'}$ sú iné ako x, y, a, b. Každý vrchol sa môže vyskytnúť ako najbližší vrchol dvakrát. Navyše sa každý z nich môže dva krát vyskytnúť ako sused najbližšieho vrchola. To znamená, že potrebujeme prešetriť aspoň 17 bodov $M_j^{'}$. Aby sme sa vyhli zavádzaniu ďalších pretnutí úsečky S nebudeme sa pozerať na body $M_t^{'}$ a $M_{t-1}^{'}$. To znamená, že ak máme 20 rôznych bodov pretnutia, tak určite nájdeme nejaký vhodný bod.

Úplne rovnakým spôsobom vieme vyriešiť druhý prípad. V treťom prípade je jediným problémom to, že nám vznikne navyše niekoľko pretnutí úsečky S, ale pre jedno odstránenie zakázanej dvojice vznikne maximálne jedno nové pretnutie. A takého zakázane dvojice búdu najviac 4.

Takýmto spôsobom odstránime všetky zakázané dvojice. Ešte treba ukázať, že nezvýšime dĺžku cesty o viac ako konštatný násobok s. Keďže každý bod pripojíme v najhoršom prípade k 17-tému najbližšiemu bodu, tak celkové predĺženie bude najviac $34s$ (lebo môžeme prepájať body na oboch stranách).

Zvyšok dôkazu je rovnaký ako v [Aro98] až na zmenu niekoľkých konštánt a to, že rozdiel medzi optimálnym riešením a (m,r)-ľahkou cestou počítame pre dve cesty miesto jednej.

4.5 Rýchlejší algoritmus

Počet stavov v doteraz popísanom algoritme najviac dvíhala nutnosť pamätať si, ktorý vrchol nasleduje za ktorým prechodom. Túto informáciu sme si pamätali preto, aby sme zabezpečili disjuktnosť ciest, keď spájame informácie z jednotlivých synov štvorca v quad-tree. Teraz ukážeme ako algoritmus upraviť, aby jeho časová zložitosť bola menšia.

Disjuktnosť vieme zabezpečiť aj s menším počtom stavov. Stačí nám vedieť, že či za dvoma prechodmi portálmi leží rovnaký vrchol. Keď cesta pretne štvorec v portáli a_i, tak máme nasledovné možnosti:

- Cesta prechádza štvorcom bez toho, aby prešla cez nejaký vrchol.
- Cesta vojde do štvorca a prvý vrchol cez ktorý prejde je vrchol v_i. V tomto prípade môže existovať žiaden, jeden alebo dva portály – b_{i_1}, b_{i_2} – pre druhú cestu také, že druhá cesta po prejdení týmito portálmi vojde do vrchola v_i.

Pri spájaní podproblémov potom vieme overiť disjuktnosť práve na základe tejto informácie.

Vstupy podproblému teda budú:

- Štvorec S z posunutého quad-tree.
- Postupnosti portálov, ktoré majú cesty pretínať: $a_1, a_2, \ldots, a_{2p}, b_1, b_2, \ldots, b_{2q}$.
- Pre každé pretnutie portálu informácia o rovnakom vrchole za portálom, tak ako je uvedené vyššie.

Keď spočítame počet podproblémov, tak dostaneme. $O(n \log_2 n)$ štvorcov, $O(m^{O(r)})$ postupnosti portálov a $O(r^{O(r)})$ možností ako môžu mať pretnutia spoločné vrcholy. Celkovo máme teda aj časovú zložitosť $O(n(\log_2 n)^{O(c)} c^{O(c)})$.

Kapitola 5

Heuristiky

V tejto časti popíšeme niekoľko algoritmov, ktoré sú pomerne úspešné na praktických inštanciách problému, ale väčšinou je ich teoretická analýza pomerne ťažká. Popíšeme niekoľko heuristík, ktoré v pomerne krátkom čase dávajú riešenie, ktoré je väčšinou blížko optimálneho. Aby sme vedeli odhadnúť kvalitu týchto heuristík, najprv ukážeme niekoľko algoritmov na hľadanie dolného odhadu veľkosti riešenia. Následne ešte vylepšíme algoritmus používajúci celočíselné lineárne programovanie od [DLS07], aby sme dokázali optimálne riešiť inštancie, ktoré majú najviac 400 vrcholov.

5.1 Dolné odhady veľkosti riešenia

5.1.1 Odhad pomocou dvoch disjuktných kostier

Pomerne priamočiarym dolným odhadom veľkosti riešenia je veľkosť dvoch nakratších disjuktných kostier. My to jemne vylepšíme budeme používať tzv. 1-kostry.

Definícia 5.1.1 *1-kostra v grafe $G = (V, E)$ je zjednotenie kostry na vrcholoch $V \setminus \{1\}$ a dvoch hrán susedných s vrcholom 1.*

Lemátko 5.1.1 *Veľkosť dvoch najlacnejších disjuktných 1-kostier je najviac tak veľká ako dĺžka riešenia problému dvoch obchodných cestujúcich.*

Dôkaz. Každá hamiltonovská kružnica je 1-kostra. Riešenie problému dvoch obchodných cestujúcich sú teda dve disjuktné 1-kostry.

Dve najkratšie disjuktné 1-kostry môžeme hľadať v čase $O(m \log_2 m + n^2)$ pomocou algoritmu popísaného v [RT85] – nájdeme najkratšie disjuktné kostry na $V \setminus \{1\}$ a k vrcholu 1 ešte pridáme štyri najkratšie hrany.

Tento odhad sa dá vylepšiť pomocou techniky opísanej v [HK70] a [CH80]. Jej princíp je nasledovný. Každému vrcholu prirádíme potenciál $p_i \in \mathbb{R}$. A zavedenieme nové dĺžky hrán:

$$d(u, v) = c(u, v) + p_u + p_v$$

Lemátko 5.1.2 *Majme potenciály $p_1, p_2, \ldots, p_n$. Riešenie problému dvoch obchodných cestujúcich pri dĺžkach $d(u, v)$ definovaných vyššie je rovnaké ako riešenie problému dvoch obchodných cestujúcich pri dĺžkach $c(u, v)$.*

Dôkaz. Majme hamiltonovskú kružnicu $v_1, v_2, \ldots, v_n$ s dĺžkou $C = c(v_1, v_2) + c(v_2, v_3) + \cdots + c(v_n, v_1)$. Pokiaľ zavedieme dĺžky hrán $d(u, v)$ dostaneme cenu:

$$C' = c(v_1, v_2) + p_{v_1} + p_{v_2} + c(v_2, v_3) + p_{v_2} + p_{v_3} + \cdots + c(v_n, v_1) + p_{v_n} + p_{v_1}$$

$$C' = C + 2 \sum_{v \in V} p_v$$

To znamená, že ku dĺžke každej hamiltonovskej kružnice prirátame rovnaké číslo a teda najlepšie riešenie pri upravených dĺžkach je rovnaké ako najlepšie riešenie pri pôvodných dĺžkach.

Podstatné je, že zmenou potenciálov vieme zmeniť najlacnejšie disjuktné 1-kostry. Našim cieľom bude nájsť také potenciály, aby rozdiel medzi dĺžkou najlacnejších disjuktných 1-kostier a dĺžkou riešenia problému dvoch obchodných cestujúcich bol čo najmenší. Nech C je dĺžka riešenia problému dvoch obchodných cestujúcich pri nulových potenciáloch. Nech D je dĺžka dvoch najlacnejších disjuktných 1-kostier pri potenciáloch $p_1, \ldots, p_n$. Potom spomínaný rozdiel vieme vyjadriť ako:

$$C + 2 \sum_{v \in V} p_v - D$$

V našom prípade je C nám neznáma konštanta a teda stačí minimalizovať výraz:

$$2\sum_{v\in V} p_v - D$$

Vhodné potenciály vieme nájsť napríklad pomocou subgradientovej optimalizácie. Dá sa totiž ukázať, že ak d_v je súčet stupňov vrchola v v obidvoch kostrách, tak vektor so zložkami $g_v = 4 - d_v$ je subgradient pre vyššie spomínaný optimalizačný problém.

Algoritmus optimalizácie sa dá popísať nasledovne. Na začiatku položíme všetky potenciály ako $p_v^{(0)} = 0$. V i-tej iterácii si najprv zvolíme veľkosť kroku $t^{(i)}$. Následne spočítame subgradienty $g_v^{(i)} = 4 - d_v^{(i-1)}$ a následne vyrátame nové potenciály ako: $p_v^{(i)} = p_v^{(i-1)} + t^{(i)} g_v^{(i)}$. V našej implementácii sme $t^{(i)}$ volili ako konštantu a iterácie opakovali dovtedy, kým sa výsledok zlepšoval.

5.1.2 Dolný odhad pomocou štyroch najbližších susedov

Ešte ukážeme jeden dolný odhad, ktorý je o trochu rýchlejší ako odhad pomocou kostier a navyše má niektoré príjemné vlastnosti, ktoré sa ukážu neskôr.

Definícia 5.1.2 *Daný je ohodnotený graf $G = (V, E)$. Graf k najbližších susedov $G_s = (V, E)$ je orientovaný graf taký, že z každého vrchola $v \in V$ vychádza k najkratších hrán, ktoré s vrcholom v susedia v G.*

Lemátko 5.1.3 *Nech s je celková dĺžka hrán v grafe 4 najbližších susedov, nech f je dĺžka najlacnejšieho 4-faktora a t je dĺžka riešenia problému dvoch obchodných cestujúcich. Potom:*

$$\frac{1}{2}s \leq f \leq t$$

Dôkaz. Druhá nerovnosť vyplýva z toho, že dve hamiltonovské kružnice tvoria dokopy 4-faktor. Prvá nerovnosť vyplýva z toho, že keď z každej hrany 4-faktoru spravíme dve orientované hrany, tak dostaneme graf v ktorom z každého vrchola vychádzajú práve 4 hrany.

My budeme hľadať graf 4 najbližších susedov. Vieme ho nájsť pomerne priamočiaro v čase lineárnom od počtu hrán. Opäť môžeme použiť aj transformáciu pomocou potenciálou. Cieľom je tento krát dostať hodnotu $s/2$ čo najbližšie k hodnote f (keďže najlacnejší 4-faktor sa pri zmene potenciálov nezmení). Celý postup subgradientovej optimalizácie je rovnaký ako pri disjuktných 1-kostrách.

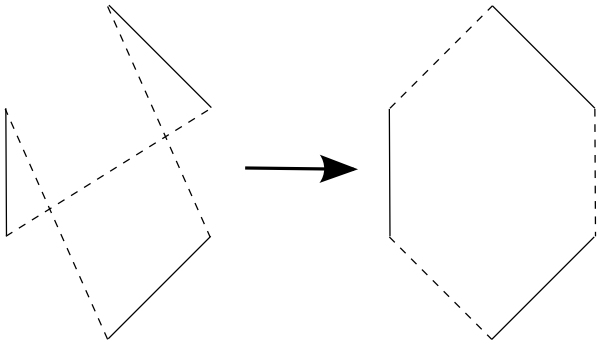
Poznámka 5.1.1 Vedeli by sme aj priamo hľadať najlacnejší 4-faktor prevedením na hľadanie najlacnejšieho 1-faktora v správne pozmenenom grafe. Výhoda nášho prístupu sa ukáže neskôr, keď budeme vyhľadávať hrany, ktoré zaručene nepoužijeme v optimálnom riešení.

5.2 Heuristiky na hľadanie horného odhadu veľkosti riešenia

Na hľadanie horného odhadu veľkosti riešenia použijeme upravenú Lin-Kernighanovu heuristiku pre problém obchodného cestujúceho ([LK73]). Táto heuristika postupne vylepšuje nájdené riešenie pomocou lokálnych zmien. V každom kroku vyberie množinu k hrán X, ktoré z aktuálneho riešenia odstráni a množinu k hrán Y, ktorú do riešenia pridá. Tejto operácii sa zvykne hovoriť aj k-opt krok, v našej implementácii budeme používať k maximálne 5. Lin-Kernighanova heuristika kladie na tieto množiny ešte niekoľko obmedzení:

- Hrany z $X \cup Y$ musia tvoriť cyklus. Kroku, ktorých spĺňa túto podmienku hovoríme aj sekvenčný k-opt krok. Na tomto cykle sa striedavo budú vyskytovať hrany z množín X a Y.

- Hrany z Y musia byť podmnožinou tzv. množiny kandidátov. Množina kandidátov je množina hrán, o ktorej predpokladáme, že väčšina z hrán riešenia bude práve z nej. Obvykle je množina kandidátov tvorená tak, že pre každý vrchol zoberieme m najkratších hrán, ktoré s ním susedia. V našej implementácii sme položili $m = 8$.



Obr. 5.1: 3-opt krok, čiarkovane sú vyznačené hrany, ktoré odstraníme, resp. pridáme

Sekvenčné k-opt kroky vieme hľadať priamočiaro pomocou rekurzie – v každom kroku si vyberieme, ktorú hranu pridáme do množiny X alebo Y (postupujeme striedavo), začíname najprv množinou X. Po pridaní hrany do množiny X skúsime uzavrieť cyklus (ignorujeme, či hrana, ktorá uzavrie cyklus je z množiny kandidátov). A skontrolujeme, či po takomto ťahu dostaneme stále hamiltonovskú kružnicu a či jej dĺžka bude kratšia. V momente, keď nájdeme vhodný ťah aplikujeme ho – t.j. nehľadáme najlepší k-opt ťah.

Zároveň si pri rekurzii počítame doterajší zisk, t.j. rozdiel medzi dĺžkou odobraných a pridaných hrán. Pokiaľ chceme, aby náš k-opt krok bol dobrý, tak jeho zisk musí byť kladný. My použijeme ešte agresívnejšie orezanie – požadujeme, aby po každom kroku rekurzie sme malý kladný zisk. Toto vôbec neobmedzí prehľadané kroky, lebo ak máme celkovo kladný zisk, tak vieme cyklus prejsť v takom poradí, aby bol zisk po každom kroku kladný.

V našom prípade máme hamiltonovské kružnice dve. Priamočiara implementácia by mohla pre každý k-opt ťah kontrolovať, či neporuší podmienku disjuktnosti ciest a povoliť len tie, ktoré ju neporušujú.

Naše kroky budú mierne komplikovanejšie. Algoritmus na ich nájdenie by sa dal popísať nasledovne:

1. Nájdi k-opt krok v jeden z kružníc, ktorý túto kružnicu zlepší. Nech tento krok vymaže hrany z množiny X a pridá hrany z množiny Y.

2. Ak sa žiadna hrana z množiny Y nenachádza v druhej kružnici, vykonaj tento krok.

3. Ak má prienik druhej kružnice a Y veľkosť 1, tak nájdi v druhej kružnici najlacnejší 2-opt alebo 3-opt krok, ktorý obsahuje túto hranu. Ak sa po vykonaní týchto krokov zlepší súčet dĺžok kružníc, vykonaj tieto kroky.

4. V prípade, že má tento prienik veľkosť viac ako 1, označme hrany v prieniku ako $e_1, \ldots, e_k$. Tieto hrany nám druhú kružnicu rozdelia na niekoľko ciest. Vyskúšame všetky možnosti ako tieto cesty pospájať bez toho, aby sme použili hrany $e_1, \ldots, e_k$ a vyberieme najlacnejšiu. Ak sa po vykonaní týchto krokov zlepší súčet dĺžkov kružníc vykonaj tieto krokov.

5. Ináč nájdi iný k-opt krok v prvej kružnici.

Tieto kroky opakujeme kým sa riešenie nedá zlepšiť. Pokiaľ sa zasekneme v lokálnom optime, tak sa pokúsime dostať tak, že spravíme niekoľko krokov, ktoré jednu cestu zlepšia a súcet nezhoršia o viac ako 1%. A následne sa opäť snazíme znižovať celkovú dĺžku. Algoritmus skončí vtedy, keď sa už ani po takýchto krokoch nepodarí riešenie zlepšiť.

5.3 Riešenia pomocou celočíselného lineárneho programovania

Naše riešenie vychádza z riešenia od [DLS07], ktoré sme stručne opísali v druhej kapitole. Pripomeňme, že hlavnou myšlienkou tohoto riešenia je hľadať 4-súvislý 4-faktor, ktorý sa následne pokúšeme rozdeliť na dve hamiltonovské kružnice. Pokiaľ sa toto rozdelenie nepodarí, daný 4-faktor zakážeme a hľadáme ďalší.

My toto riešenie vylepšíme pomocou niekoľkých vecí.

5.3.1 Rozdeľovanie 4-súvislého 4-faktora pomocou SAT solvera

Ako sme už spomínali zistiť, či sa dá rozdeliť 4-súvislý 4-faktor na dve hamiltonovské kružnice je NP-úplný problém. Pôvodné riešenie tento problém rieši pomocou celočíselného lineárneho programu. Keďže ale tento problém je rozhodovací a nie optimalizačný, nám sa zdá rozumnejšie tento problém riešiť pomocou SAT solvera. Technika riešenia bude takmer podobná ostatným. Na začiatku budeme požadovať iba rozklad na dva 2-faktory (t.j. rozložíme hrany do dvoch množín A, B tak, aby každý vrchol mal dve susedné hrany z A a dve susedné hrany z B). Pokiaľ sa vyskytne nejaký cyklus C, ktorý neprechádza celým grafom, tak tento cyklus zakážeme (t.j. budeme požadovať výskyt aspoň jednej hrany medzi množinami C a $V \setminus C$).

5.3.2 Zrýchlenie rozdeľovania pomocou hľadania vhodných podgrafov

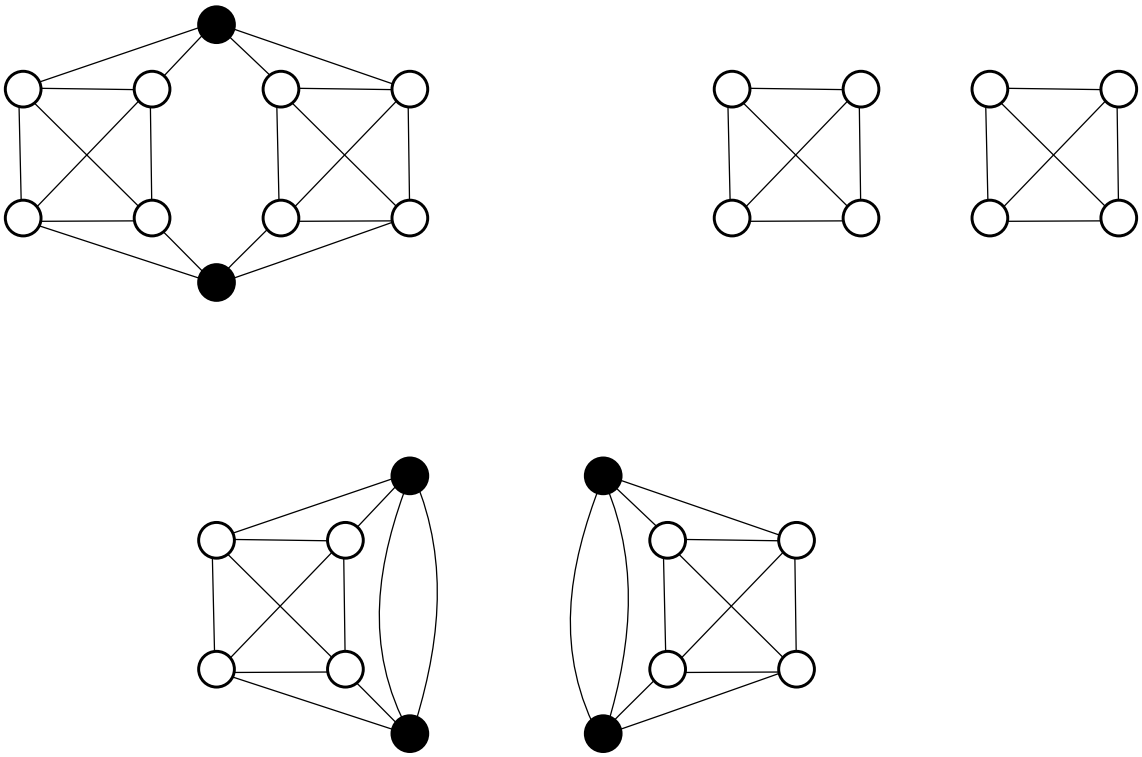
Už [DLS07] ukázal jeden spôsob ako zrýchliť rozhodovanie, či sa dá 4-súvislý 4-faktor rozdeliť na dve hamiltonovské kružnice.

Definícia 5.3.1 *Artikulačný pár v 4-súvislom 4-regulárnom grafe $G = (V, E)$ je dvojica vrcholov v_1, v_2 také, že keď z grafu odstránime tieto vrcholy a ich susediace hrany, tak sa graf rozpadne na dva grafy G_1, G_2.*

Artikulačné páry vieme triviálne hľadať v čase $O(n^3)$.

Pokiaľ nájdeme v grafe G artikulačný pár v_1, v_2 môžeme ho rozdeliť pomocou nasledovnej dekompozície: Nech sa po odstránení vrcholov v_1, v_2 graf rozpadne na dva grafy G_1, G_2. Graf G_1' vytvoríme z G_1 tak, že mu pridáme vrcholy v_1, v_2 a všetky hrany, ktoré spájali vrcholy v_1, v_2 s grafom G_1 v grafe G. Následne ešte spojíme vrcholy v_1, v_2 medzi sebou pomocou dvoch hrán. Obdobne vyrobíme z grafu G_2 graf G_2'.

Grafy G_1', G_2' budú 4-súvislé a 4-regulárne. Navyše o nich platí nasledovné.



Obr. 5.2: Dekompozícia grafu. Hore vľavo: vstupný graf G s vyznačeným artikulačným párom. Hore vpravo: Graf po odstranení artikulačného páru. Dole: Vzniknuté grafy, po pridaní artikulačným párov a hrán medzi nimi.

Veta 5.3.1 *Nech G je 4-súvislý 4-regulárny graf, ktorý obsahuje artikulačný pár v_1, v_2. Nech $G_1^{'}, G_1^{'}$ sú grafy, ktoré vznikli dekompozíciou popísanou vyššie. Graf G sa dá rozložiť na dve hamiltonovské kružnice práve vtedy a len vtedy, keď sa grafy $G_1^{'}$ a $G_2^{'}$ dajú rozložiť na dve hamiltonovské kružnice.*

Dôkaz. Pozri [DLS07].

Túto techniku môžeme aplikovať rekurzívne. Navyše pokiaľ sme spravili iba jednu dekompozíciu, tak môžeme zakázať menšiu časť grafu ako celý. T.j. ak sa nepodarí daný graf G rozložiť na dve hamiltonovské kružnice, preto, lebo graf $G_1^{'}$ sa nepodarilo rozložiť, tak môžeme miesto podmienky:

$$\sum_{e \in G} x_e \leq 2n - 1$$

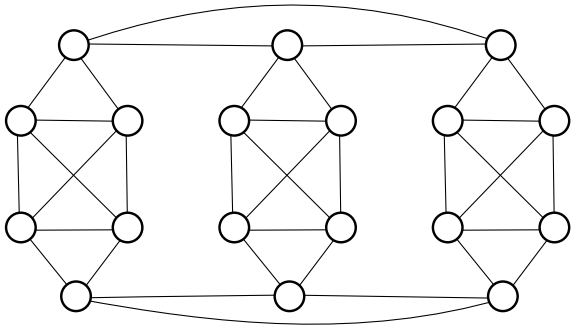
Použiť podmienku:

$$\sum_{e \in G_1^{'}} x_e \leq |E_1^{'}| - 1$$

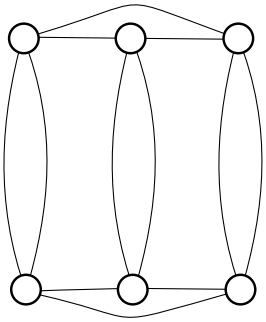
Táto podmienka hovorí, že daný graf potrebujeme zmeniť. Táto technika ale nefunguje rekurzívne, ale iba pri prvej dekompozícii. Protipríklad môžeme vidieť na obrázku 5.3. Po troch krokoch dekompozície by sme pôvodný graf rozložili na grafy G_1, G_2, G_3, G_4 a následne by sme zistili, že problematický je graf G_1. Avšak riešenie sa dá dosiahnúť aj tak, že graf G_1 nezmeníme, ale prepojíme medzi sebou napríklad grafy G_2 a G_3.

V našom algoritme používame ešte jednu operáciu, ktorá vstup pre SAT solver ešte zjednoduší. Myšlienka vyzerá nasledovne: Ak nájdeme nejakú štvorcu bodov, ktoré sú všetky navzájom spojené (kliku veľkosti 4), tak ich môžeme zlučiť do jedného bodu.

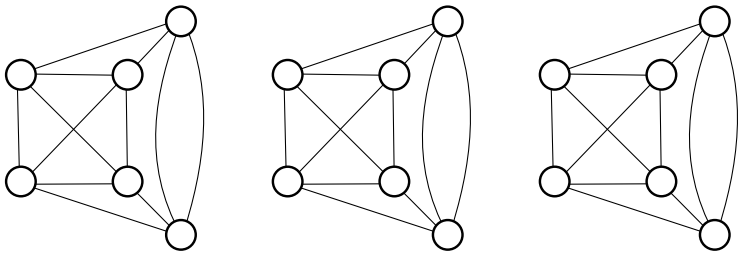
Veta 5.3.2 *Nech G je 4-súvislý 4-regulárny graf a nech v_1, v_2, v_3, v_4 sú jeho vrcholy také, že medzi každými dvoma vedie hrana. Nech graf $G^{'}$ vznikne z grafu G tak, že vrcholy v_1, v_2, v_3, v_4 zlúčime do jedného. Potom graf G je rozložiteľný na dve hamiltonovské kružnice práve vtedy a len vtedy, keď je graf $G^{'}$ rozložiteľný na dve hamiltonovské kružnice.*



(a) Pôvodný graf



(b) G_1

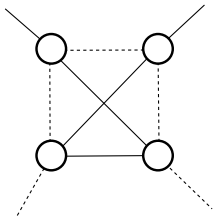


(c) G_2, G_3, G_4

Obr. 5.3: Protipríklad pri rekurzívnej dekompozícii

Dôkaz. Zoberme hrany, ktoré vedú medzi množinami $V_x = \{v_1, v_2, v_3, v_4\}$ a $V \setminus V_x$. Takéto hrany sú práve 4. Ak by ich bolo menej, tak G nie je 4-súvislý, ak by ich bolo viac, tak G nemôže byť 4-regulárny.

Ak sa graf G dá rozložiť na dve hamiltonovské kružnice, tak triviálne vidno, že aj graf G' sa dá rozložiť (kružnice budú presne rovnaké akuráť vynecháme hrany medzi vrcholmi $v_1, \ldots, v_4$).



Obr. 5.4: Prechod kružníc cez kliku veľkosti 4

Ak sa graf G' dá rozložiť na dve hamiltonovské kružnice, tak pomocou vhod-

ného prepojenia (viď. 5.4) medzi vrcholmi $v_1, \ldots, v_4$ vieme graf G rozložiť na dve hamiltonovské kružnice (ostatné hrany rozložíme medzi kružnice rovnako ako v grafe G').

Poznamenajme, že takéto štvorice vrcholov vieme hľadať v čase $O(n)$ – vyberieme si jeden vrchol v_1 a v konštantom čase prejdeme všetky jeho kombinácie susedov.

5.3.3 Odstráňovanie nepoužiteľných hran

Predstavme si graf, ktorého vrcholy zodpovedajú bodom v rovine. Pomerne intuitívne vieme označiť veľa hrán, o ktorých si sme istý, že v optimálnom riešení určite nebudú – sú to hlavne tie hrany, ktoré vedú cez celú rovinu. My by sme v tejto časti chceli ukázať ako tieto hrany hľadať algoritmicky. Pokiaľ by sme ich vedeli hľadať, môžeme obmedziť počet premenných, ktoré sú v lineárnom programe a tak urýchliť jeho riešenie.

Lemátko 5.3.3 *Nech $l(G, e)$ je dolný odhad veľkosti riešenia, ktoré obsahuje hranu e. Nech $u(G)$ je nejaký horný odhad riešenia. Ak $l(G, e) > u(G)$, tak sa hrana e určite nepoužije v optimálnom riešení.*

Dôkaz. Triviálny.

Ako získať horný odhad veľkosti riešenia sme už popísali. Ešte potrebujeme popísať ako získať dolný odhad, ktorý obsahuje hranu e.

Pokiaľ používame dolný odhad pomocou štyroch najbližších susedov, tak sa dolný odhad, ktorý obsahuje hranu e dá získať pomerne jednoducho. Nech e vedie medzi vrcholmi x, y. Potom do veľkosti dolného odhadu miesto 4 najbližších susedov započítame iba 3 najbližších susedov (okrem hrany e) a navyše hranu e započítame $2 - krt$. Pokiaľ máme už dopredu spočítaný dolný odhad bez nutnej hrany, tak pre danú hranu e vieme tento odhad prerátať v konštantnom čase. To znamená, že v čase $O(n^2)$ vieme nájsť nepoužiteľné hrany.

Pokiaľ používame dolný odhad pomocou dvoch disjuktných kostier, tak je situácia trochu zložitejšia. Pokiaľ máme spočítané najlacnejšie dve disjuktné kostry a chceme, aby sa nutne použila hrana e, tak potrebujeme z týchto kostier jednu hranu vyhodiť. Nestačí sa ale len pozrieť na cykly v ktorých leží hrana e, lebo môžeme si hrany medzi kostrami presúvať. To znamená, že vložime hranu e, inú hranu e_1 presunieme z kostry M_1 do kostry M_2 (e a e_1 ležali po pridaní do M_1 na jednom cykle), ďalšiu hranu e_2 presunieme z M_2 do M_1, atď., až nakoniec odstraníme hranu e_k. Teda hľadáme cestu medzi hranami kostier a chceme vyhodiť najlacnejšiu hranu, ku ktorej sa vieme dostať. Tento problém efektívne rieši značkovacia procedúra z [RT85], akurát si potrebujeme pamätať akú najlacnejšiu hranu vieme dosiahnúť. Táto značkovacia procedúra trvá pre jednu hranu čas $O(n)$. Následne keď zistíme, ktorú hranu musíme z kostier vyhodiť, tak veľkosti dolného odhadu vieme prerátať triválne. Celkovo na to, aby sme o každej hrane zistili či je zbytočná potrebujeme čas $O(n^3)$.

Poznámka 5.3.1 Dobre spravený algoritmus by toto vedel riešiť aj v čase $O(n^2)$. Keďže ale vo všetkých naších vstupoch platí $n \leq 500$, tak je $O(n^3)$ riešenie postačujúce a nepotrebujeme sa zaoberať komplikovanejším riešením.

5.4 Experimenty

Vyššie popísané algoritmy sme implementovali s cieľom empiricky overiť ich účinnosť na niektorých druhoch dát. Testovacie dáta budú tvoriť náhodné grafy v euklidovskej rovine (kde súradnice bodov sú vyberané rovnomerne náhodne) a niektoré inštancie z TSPLIB ([Rei91]).

Hľadanie dolných a horných odhadov sme implementovali v C++. Na riešenie celočíselných lineárnych programov sme použili Numberjack, ktorý používa SCIP a ako SAT solver sme použili knižnicu STP, ktorá používa MINISAT. Réžiu okolo celočíselných lineárnych programov a SAT solvera sme implementovali v jazyku Python. Algoritmy sme testovali na počítači s procesorom Intel i7 Q720.

Testovali sme tri rôzne implementácie. Jedna nepoužívala žiadne vyhadzovanie nepotrebných hrán, druhá používala vyhadzovanie nepotrebných hrán cez štyroch

najbližších susedov a tretia používala vyhadzovanie nepotrebných hrán pomocou dvoch disjuktných kostier.

V prípade náhodných grafov v euklidovskej rovine sme generovali grafy veľkosti 100, 150, 200, 250, 300. Pre každú veľkosť sme vygenerovali 10 grafov. Následne sme každú našu implementáciu pustili a sledovali niekoľko údajov:

- Či úspešne skončí do jednej hodiny. Ak neskončí, tak rátame riešenie za neúspešné.
- Ako rýchlo skončí.
- Koľko krát sa vykoná rozklad 4-súvislého 4-faktora na dve hamiltonovské kružnice.
- Koľko hrán bolo označených ako nepoužiteľné.

5.4.1 Výsledky pre náhodné grafy v euklidovskej rovine

V tabuľke 5.1 sú zhrnuté priemerné výsledky pre každú veľkosť grafu. Označenia:

- n – veľkosť grafu
- f_a – priermerný počet rozkladov na dve hamiltonovské kružnice pre všetky inštancie (v prípade, že sme do hodiny nedostali výsledok, tak rátame počet rozkladov, ktoré program stihol vykonať za hodinu)
- f_u – priemerný počet rozkladov pre inštancie, kde sme úspešne skončili
- f_1 – počet inštancií, kde prvý rozklad bol úspešný
- s_n – uspešnosť algoritmu, keď sme žiadne nepotrebné hrany nehľadali
- t_n – priemerný čas behu algoritmu, keď sme žiadne nepotrebné hrany nehľadali (pokiaľ sme neskončili v priebehu hodiny, do priemeru zarátame hodinu) v sekundách
- u_n - to isté, čo t_u ale rátame priemer iba na inštanciách, kde sme úspešne skončili
- s_4, t_4, u_4 – úspešnosť a časy behu v prípade, že nepoužiteľné hrany, hľadáme pomocou štyroch najbližších susedov

- d_4 – priemerný podiel nájdených nepoužíteľných hrán
- s_k, t_k, u_k – úspešnosť a časy behu v prípade, že nepoužiteľné hrany, hľadáme pomocou dvoch disjuktných kostier
- d_k – priemerný podiel nájdených nepoužíteľných hrán

n	f_a	f_u	f_1	s_n	t_n	u_n	s_4	t_4	u_4	d_4	s_k	t_k	u_k	d_k
100	57	29	5	0.9	736	378	0.9	687	322	0.64	0.9	585	249	0.70
150	83	28	4	0.7	1380	428	0.7	1327	353	0.61	0.7	1314	334	0.58
200	68	8	3	0.5	2000	400	0.5	1903	206	0.56	0.5	1913	226	0.60
250	46	22	2	0.3	2769	819	0.4	2662	1255	0.50	0.4	2615	1136	0.57
300	34	6	0	0.2	3087	1035	0.2	3029	745	0.52	0.2	3021	705	0.52

Tabuľka 5.1: Výsledky pre náhodné grafy.

Z tabuľky sa dá usúdiť niekoľko záverov.

Vidíme, že odhad pomocou dvoch kostier odstráni o niečo viac hrán ako odhad pomocou štyroch najbližších susedov Z hľadiska celkového času sú tieto algoritmy porovnateľné. Ďalej vidíme, že odstráňovanie týchto hrán sa opláca (pri veľkosťi 250 je menší čas pri neorezávaní spôsobený tým, že zarátavame iba 3 inštancie namiesto 4).

V porovnaní s [DLS07] máme o niečo vyššiu úspešnosť, oni pri inštanciách veľkosti 190 dosahujú úspešnosť 0.2 a pre väčšie inštancie vyzerá, že sú neúspešní. My túto hranicu posúvame na 300.

5.4.2 Výsledky pre niektoré inštancie z TSPLIB

Výsledky pre inštancie z TSPLIB zhŕňa nasledujúca tabuľka. Označenia (uvádzame iba iné ako v predchádzajúcej tabuľke a miesto priermerných časov uvádzame čas behu jednej inštancie):

- Inst – názov inštancie, číslo v názve vyjadruje počet vrcholov
- t_d – čas riešenia uvádzaný v [DLS07], ak bol uvádzaný

Inst	f_a	t_n	t_4	d_4	t_k	d_k	t_d
kroA100	26	200	178	0.66	187	0.73	–
kroB100	3	25	23	0.70	24	0.61	209
kroC100	5	45	41	0.65	43	0.67	137
kroD100	1	14	16	0.70	18	0.90	10
kroE100	1	8	10	0.73	16	0.68	23
bier127	24	220	225	0.08	219	0.63	–
kroA150	1	18	26	0.55	20	0.85	749
kroA200	1	65	67	0.50	50	0.73	311
a280	1	201	203	0.70	317	0.09	3085
lin318	1	573	276	0.34	276	0.29	–
rd400	1	813	857	0.48	741	0.50	–
pcb442	1	–	1503	0.20	1775	0.23	–

Výsledky ukazujú, že náš algoritmus je rádovo rýchlejši ako algoritmus od [DLS07]. Naša najväčšia zriešená inštancia má veľkosť 442, v ich prípade je to 280. Navyše sa ukazuje, že vymazávanie nepotrebných hrán pomáha, ale nie je to hlavný faktor nášho zrýchlenia. Autori prechádzajúceho algoritmu hovoria, že drvivú väčšinu času ich algoritmus strávi zistovaním, či je príslušný 4-súvislý 4-faktor rozložiteľný na dve hamiltonovské kružnice. Preto sme zobrali niektoré inštancie z TSPLIB, ktoré boli rozložiteľné na prvý krat a pustili ich na našom algoritme bez vyhadzovania hrán. Tentokrát sme nemerali celkový čas, ale aké percento času algoritmus strávil hľadaním rozkladu na kružnice (p_r). Toto sme porovnali s predchádzajúcimi výsledkami p_d. Pre prehľadnosť uvádzame aj časy riešenia.

V tabuľky vidno, že náš algoritmus trávi oveľa menej času pri zistovaní, či je daný 4-súvislý 4-faktor rozložiteľný na dve hamiltonovské kružnice. To mu umožňuje vyriešiť aj niektoré inštancie, ktoré toto potrebujú zistovať aj viac krát (napr. bier127).

Z týchto meraní teda vyplýva, že hlavnou výhodou nášho algoritmu je lepší algoritmus na rozkladanie grafu na dve kružnice. V niektorých prípadoch vieme

Inst	p_r	p_d	t_n	t_d
kroD100	0.43	0.90	14	10
kroE100	0.71	1.00	8	23
kroA150	0.35	0.95	18	749
kroA200	0.44	0.93	65	311
a280	0.93	0.73	201	3085
lin318	0.22	–	573	–

navyše algoritmus zrýchliť aj tým, že nájdeme nepoužiteľné hrany, ale toto zrýchlenie je oveľa slabšie. Náš algoritmus vie riešiť náhodné inštancie do veľkosti 300 a aj niektoré väčšie inštancie, pokiaľ sa mu ich podarí rozložiť na prvý pokus.

Zaujímavou teoretickou otázkou aký očakávaný počet takýchto rozkladov pre rôzne n. Táto otázka je pomerne príbuzná otázke: koľko percent 4-súvislých 4-regulárny grafov s n vrcholmi sa dá rozložiť na dve hamiltonovské kružnice. Bohužial nepodarilo sa nám nájsť žiadny odhad na počet 4-súvislých 4-regulárnych grafov a teda na túto otázku odpovedať nevieme.

5.5 Heuristiky pre väčšie inštancie

Na webovej stránke Kaggle.com sa konala súťaž Travelling santa problem, v ktorej bola zadaná inštancia problému dvoch obchodných cestujúcich s drobnými obmenami, ktorá mala 150000 vrcholov, ktoré predstavovali body v rovine. Spomínané obmeny boli:

- Cieľom nebolo nájsť disjuktné kružnice, ale disjuktné cesty, ktoré prechádzali všetkými vrcholmi.

- Cieľom nebolo minimimalizovať súčet dĺžok ciest, ale minimalizovať dĺžku dlhšej cesty.

Do tejto súťaže sme sa zapoli spolu s Petrom Perešínim. Skončili sme na prvom mieste, kde naše riešenie malo dĺžku 6526972, pričom nami vyratány dolný odhad

mal veľkosť: 6525773. To znamená, že sme boli od opmima vzdialený najviac 1199, čo je asi 0.02%.

Okrem techník spomenutých vyššie sme v tejto súťaži využili niekoľko ďalších heuristík:

Neriešili sme problém zo zadania, t.j. minimalizovať dĺžku dlhšej cesty, ale našim cieľom bolo minimalizovať súčet dĺžok ciest. Zistili sme, že nie je problém zabezpečiť, aby cesty mali takmer podobnú dĺžku (stačí niekoľko prehodení hrán medzi cestami) a preto nám táto zmena nerobí veľký problém.

Robili sme dolný odhad priamo pomocou najmenšieho 4-faktoru. Tento sme rátali pomocou celočíselného lineárneho programu. Navyše sme ho nemohli rátať priamo pre inštanciu veľkosti 150000. Preto sme danú inštanciu nasekali na menšie obdĺžnikové kúsky a hľadali najmenší 4-faktor v nich. Nepríjemnou technickou záležisťou bolo urobiť tento dolný odhad pre 2 cesty miesto 2 cyklov.

Pri riešení sme využívali aj vyššie spomínané celočíselné lineárne programovanie spolu so SAT solverom. Opäť sme nechceli nájsť optimálne riešenie pre celú inštanciu, ale optimálne riešenie pre nejakú danú časť.

Ešte sme používali jednu heuristiku. Jej kroky sa dajú zhrnúť nasledovne:

1. Spoj obidve cesty do jedného grafu (dostaneme takmer 4-regulárny graf)
2. V tomto grafe nájde alternujúcu kružnicu, ktorá má zápornú cenu (t.j. cena vložených hrán je menšie ako cena odstránených hrán) a následne pomocou nej zmenší veľkosť daného grafu.
3. Pokús sa graf spätne rozložiť na dve cesty.

V kroku 2 sme hľadali pomerne dlhé alternujúce kružnice. Na toto sme opäť použili heuristiku. Tak isto sme použili aj heuristiku na rozkladanie grafu na dve cesty. Viac o týchto heuristikách sa dá nájsť v [KAG].

Záver

V tejto práci sme zhrnuli a navrhli niekoľko nových prístupov pre riešenie problému dvoch obchodných cestujúcich. Naše hlavné prínosy sú nasledovné:

- Navrhli sme exaktný algoritmus, ktorého časová zložitosť je oveľa lepšia ako zložitosť triviálneho algoritmu.
- Ukázali sme ako urobiť PTAS pre problém dvoch obchodných cestújich. Navyše jeho zložitosť nie je horšia ako zložitosť PTASu pre problém obchodného cestujúceho.
- Navrhli sme niekoľko heuristík a porovnali ich s doterajšími. Kým doterajšie heuristiky zvládali inštancie s maximálnou veľkosťou 280, naše heuristiky zvládajú inštancie s maximálnou veľkosťou 442.

V spomínaných výsledkoch je stále niekoľko miest na zlepšenie. Pri exaktnom algoritme je zaujímavé zistiť, či obmedzenie na vstupný graf (napr. trojuholníková nerovnosť) neumožní návrh ešte rýchlejšieho algoritmu. Pri heuristikách by bolo veľmi nápomocné, ak by sa podarilo nájsť postup ako znížiť počet zistovaní toho, či je 4-súvislý 4-regulárny graf rozložiteľný na dve hamiltonovské kružnice.

Literatúra

[AP08] Alexander A Ageev and Artem V Pyatkin. A 2-approximation algorithm for the metric 2-peripatetic salesman problem. In *Approximation and Online Algorithms*, pages 103–115. Springer, 2008.

[Aro98] Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, September 1998.

[BN68] Mandel Bellmore and George L Nemhauser. The traveling salesman problem: a survey. *Operations Research*, 16(3):538–558, 1968.

[CH80] Jens Clausen and Lone Aalekjær Hansen. *Finding k edge-disjoint spanning trees of minimum total weight in a network: An application of matroid theory*. Springer, 1980.

[Chr76] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, DTIC Document, 1976.

[DLS07] Éric Duchenne, Gilbert Laporte, and Frédéric Semet. The undirected m-peripatetic salesman problem: Polyhedral results and new algorithms. *Operations research*, 55(5):949–965, 2007.

[Gim] E Kh Gimadi. Approximation efficient algorithms with performance guarantees for some hard routing problems.

[HK61] Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. In *Proceedings of the 1961 16th ACM national*

meeting, ACM '61, pages 71.201–71.204, New York, NY, USA, 1961. ACM.

[HK70] Michael Held and Richard M Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

[KAG] `http://icanpeefurther.blogspot.sk/2013/01/our-solution-part-v-going-last-mile.html`.

[Kra75] Jakob Krarup. The peripatetic salesman and some related unsolved problems. In B. Roy, editor, *Combinatorial Programming: Methods and Applications*, volume 19 of *NATO Advanced Study Institutes Series*, pages 173–178. Springer Netherlands, 1975.

[LK73] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[Pik96] D.A Pike. *Hamilton Decompositions of Graphs.* PhD thesis, Auburn University, 1996.

[Rei91] Gerhard Reinelt. Tsplib—a traveling salesman problem library. *ORSA journal on computing*, 3(4):376–384, 1991.

[RT85] James Roskind and Robert E Tarjan. A note on finding minimum-cost edge-disjoint spanning trees. *Mathematics of Operations Research*, 10(4):701–708, 1985.

[SW97] Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *Journal of the ACM (JACM)*, 44(4):585–591, 1997.

[WCC03] R Wolfler Calvo and R Cordone. A heuristic approach to the overnight security service problem. *Computers & Operations Research*, 30(9):1269–1287, 2003.