UNIVERZITA PALACKÉHO V OLOMOUCI
PŘÍRODOVĚDECKÁ FAKULTA

BAKALÁŘSKÁ PRÁCE

Exaktní algoritmy pro řešení úlohy obchodního cestujícího

Katedra matematické analýzy a aplikací matematiky
Vedoucí práce: **RNDr. Pavel Ženčák, Ph.D.**
Vypracoval(a): **Matyáš Vávra**
Studijní program: B1103 Aplikovaná matematika
Studijní obor: Matematika–ekonomie se zaměřením na bankovnictví/pojišťovnictví
Forma studia: prezenční
Rok odevzdání: 2024

BIBLIOGRAFICKÁ IDENTIFIKACE

Autor: Matyáš Vávra

Název práce: Exaktní algoritmy pro řešení úlohy obchodního cestujícího

Typ práce: Bakalářská práce

Pracoviště: Katedra matematické analýzy a aplikací matematiky

Vedoucí práce: RNDr. Pavel Ženčák, Ph.D.

Rok obhajoby práce: 2024

Abstrakt: Problém obchodního cestujícího je úloha z oblasti teorie grafů, kdy je cílem najít nejkratší cestu přes soubor bodů tak, že každý z těchto bodů je navštíven právě jednou. V této práci jsem popsal exaktní metody, kterými úloha může být spočtena. Následně jsem metody zpracoval v softwaru MATLAB a porovnal je.

Klíčová slova: Problém obchodního cestujícího, exaktní metody, matlab

Počet stran: 56

Počet příloh: 1

Jazyk: český

BIBLIOGRAPHICAL IDENTIFICATION

Author: Matyáš Vávra

Title: Exact algorithms for solving the Travelling salesman problem

Type of thesis: Bachelor's

Department: Department of Mathematical Analysis and Application of Mathematics

Supervisor: RNDr. Pavel Ženčák, Ph.D.

The year of presentation: 2024

Abstract: The traveling salesman problem is a graph theory problem where the goal is to find the shortest path through a set of points such that each of these points is visited exactly once. I implemented the described methods into MATLAB software and compared them.

Key words: Travelling salesman problem, exact methods, matlab

Number of pages: 56

Number of appendices: 1

Language: Czech

Prohlášení

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně pod vedením pana RNDr. Pavla Ženčáka, Ph.D. a všechny použité zdroje jsem uvedl v seznamu literatury.

V Olomouci dne ....

podpis

Obsah

Poděkování

Rád bych poděkoval především panu RNDr. Pavlu Ženčákovi, PhD., za nekonečnou vstřícnost a ochotu po celou dobu práce na tomto textu. Dále bych chtěl zde chtěl poděkovat rodině za podporu během studia, především svému bratrovi za neutuchající motivaci vedoucí k dokončení této práce.

Úvod

Tématem bakalářské práce je nastudování exaktních metod pro řešení problému obchodního cestujícího. Podstata tohoto problému je víceméně snadno popsatelná i laikovi, nicméně problém je náročným hlavně z hlediska výpočetní náročnosti a právě té je v této práci věnována hlavní pozornost.

V práci se čtenář nejprve seznámí s historickým kontextem a vývojem řešení úlohy TSP, následně bude seznámen s formulacemi tohoto problému a jejich využtím při konstrukci exaktních metod řešení. Nakonec je čtenář seznámen s programovou realizací, kdy hlavní pozornost je věnována metodě CLP *(zkratka pro celočíselné lineární programování)* relaxace, a nakonec jsou porovnány různé přístupy řešení.

Pro programové zpracování jednotlivých metod je využito softwaru MATLAB, v jehož prostředí fungují všechny skripty a funkce, které jsou přílohami této práce.

Primárním cílem je porovnání různých metod z hlediska výpočetní náročnosti a prozkoumání limitů jednotlivých metod. Dalším cílem je porovnání těchto metod i z hlediska paměťové náročnsoti. Zpracování jednotlivých metod a vyhodnocení metod může být využito při volbě způsobu řešení pro různé aplikace řešení TSP.

1. Seznámení s problémem obchodního cestujícího

V této kapitole bude představena matematická formulace problému obchodního cestujícího (anglicky *Traveling Salesman Problem*, zkráceně TSP) a budou definovány klíčové pojmy, které budou používány po celou dobu práce s problémem obchodního cestujícího . Též bude doplněna historie problému, krátce nastíněn vývoj jeho řešení a popsáno využití této problematiky v praxi.

1.1. Původ zkoumání problému obchodního cestujícího

Díky ilustraci okolností kolem problému bude lépe možno chápat problém, jeho důležitost a důležitost aplikace řešení.

1.1.1. Původ názvu TSP

Původ názvu problému není bohužel zcela jasný a není možno dohledat žádné autorské články ani práce, kde by se oficiálně poprvé vyskytlo dnešní označení problému jako problém obchodního cestujícího. Za první referenci použití názvu se dá považovat zmínka Julie Robinson z roku 1949 v práci „On the Hamiltonian game (a traveling salesman problem)“, nicméně samotná autorka toto pojmenování nevymyslela (viz [1]).

Jisté zmínky jsou vedeny od pánů Flood a Whitney z Princetonovy univerzity, kdy prvně jmenovaný řešil problém okruhu autobusu v Západní Virginii a druhý se podílel na řešení problému 48 měst. Jeden z těchto autorů údajně poprvé pro takový problém použil označení Travelling Salesman Problem, jenž se od té doby stalo využívano pro tento druh problematiky, jejíž specifický popis a definice je popsána níže v textu. Oba pánové své problémy

řešili někdy v 30. letech 20. století (viz [5])

Později se ukázalo, že existují i dřívější snahy a popisy něčeho, co se nyní nazývá problémem obchodního cestujícího. Příkladem může být německá příručka „Obchodní cestující - jaký má být a co má dělat, aby získal zakázky a měl jistotu úspěchu ve svém podnikání“ (německy *Der Handlungsreisende—wie er sein soll und was er zu thun hat, um Auftrage zu erhalten und eines glu cklichen Erfolgs in seinen Gescha ften gewiss zu sein—Von einem alten Commis-Voyageur* z roku 1832 obsahující popis problému, jak je znám dnes. Jako autor se podepsal alten Commis-Voyageur (v překladu starý obchodní cestovatel), jehož jméno je neznámé. Pro kontext je jistě vhodné doplnit, že v dřívějších dobách bylo pro obchodníky běžnou praxí a možná i jedinou možností osobní navštěvování zákazníků. Pro obchodníky tak mělo velký význam hledání „optimálních“ tras, kdy dojde k navštívení všech požadovaných míst při co nejnižších nákladech (viz [1])

1.1.2. Historické způsoby řešení

Jak bude později vysvětleno, problém obchodního cestujícího je i na dnešní poměry poměrně náročným problémem k řešení, zejména při zvyšujícím se počtu bodů, pro které je řešen. I před rychlým rozvojem výpočetní techniky ovšem existovaly metody jak „optimalizovat“ cestu mezi několika body.

Možným řešením je při řešení využívat geometrii. Například při plánování cesty Abraham Lincolna americkými městy se za pomocí odstranění ostrých uhlů, které jsou svírány dvojící hran, z mapy řešení dosáhlo optimální cesty. Dalším, kdo využil geometrii k řešení problému hledání nejkratší cesty byl Sir William Rowan Hamilton, který pro hledání nejkratšího okruhu mezi 20 body využíval dvanáctistěnu. Přes všemožné další možné způsoby (pro vý-

běr nejkratší kružnice byla například využívana i estetika) se nakonec projevil rozmach výpočetní techniky a hledání exaktních řešení tak probíhá algoritmicky (viz [1]).

1.2. Aplikace TSP

Pro aplikaci v reálném světě není třeba dlouze hledat. Hledaní nejkratší cesty mezi několika body se jistě řadí mezi časté problémy každodenního života. Jistě není běžnou praxí, že by k výpočtu nejkratší cesty běžně člověk kalkuloval matice a vymýšlel algoritmy pro optimalizaci své cesty, ale i s aplikací těchto postupů se člověk běžně nevědomky setkává, například při využívání služeb logistických firem (viz [6]). To jen dokazuje důležitost problému obchodního cestujícího a jeho praktickou využitelnost.

Využití nalezá problém obchodního cestujícího i v genetice, konkrétně při sestavování pořadí markerů v genomu. Na markery v genomu lze pohlížet stejně jako na města a na uspořádání genomu jako na kružnici procházející každým markerem (viz [1]).

Dalším odvětvím, kde se setkáváme s aplikací problému obchodního cestujícího jsou integrované obvody na deskách s plošnými spoji. V deskách se vyskytuje poměrně velké množství děr, do kterých jsou usazovány čipy. Pro zeefektivěnní výroby je využívano TSP, jehož pomocí jsou optimalizovány přesuny vrtačky mezi pozicemi otvorů.

Toto je krátký výčet možných využití řešení TSP dokazující opravdu velkou šíři spketra oborů, kde je možné se hledáním nejkratší cesty setkat. Je tak zcela zřejmá důležitost zkoumání tohoto problému a rozvoj výpočetních technik vedoucích k jeho řešení.

1.3. Základní pojmy z teorie grafů

Před zavedením samotného problému budou definovány základní pojmy z oblasti teorie grafů. Jejich zavedení usnadní chápání a orientaci v následném textu.

Definice 1 ***Grafem*** *rozumíme dvojici množin $G = (U, H)$, kde U je množina uzlů a H množina hran. Hranou rozumíme neuspořádanou dvojici $[u, v]$ kde $u \in U$ a $v \in U$ jsou uzly grafu.*

Definice 2 ***Sledem*** *(viz [14]) v grafu G nazýváme posloupnost hran a uzlů $u_{i0}, u_{i1}, \ldots, u_{ik}$ takovou, že $[u_{ij-1}, u_{ij}] \in H$, pro $j = 1, 2, \ldots, k$.*

Definice 3 ***Cestou*** *v G nazýváme sled, v němž se ani jeden uzel nevyskytuje vícekrát.*

Definice 4 ***Kružnicí*** *(viz [14]) nazveme takový sled, pro který platí, že se v něm všechny uzly krom prvního a posledního objevují nejvýše jednou.*

Definice 5 *Cesta, která obsahuje právě všechny uzly grafu G se nazývá* ***hamiltonovská cesta*** *(viz [14]).*

Definice 6 *Kružnice, která obsahuje všechny uzly grafu G právě jednou, se nazývá* ***hamiltonovská kružnice*** *(viz [14]).*

Definice 7 *Nechť h je reálná funkce definovaná na na množině hran H grafu $G = (U, H)$. Potom trojici $G_h = (U, H, h)$ nazveme* ***hranově ohodnoceným grafem*** *(viz [14]).*

Definice 8 ***Délkou cesty*** *(viz [14]) v ohodnoceném grafu rozumíme součet ohodnocení všech hran v cestě.*

Definice 9 ***Úplným grafem*** *(viz [14]) je graf, jehož každé dva uzly $i, j \in U$ jsou spojeny hranou $h_{i,j} \in H$, $i \neq j$.*

1.4. Zavedení problému v teorii grafů

V této části bude vysvětlena podstata problému obchodního cestujícího a bude vysvětleno o čem pojednává.

Nechť $U = \{u_1, u_2, \ldots, u_n\}, n \in \mathbb{N}$ je množina uzlů. Nechť dále H je množina hran, kde $h_i = [u_i, u_{i+1}]$ pro všechna $u_i \in U$. Cílem a řešením problému obchodního cestujícího je najít nejkratší hamiltonovskou kružnici z úplného grafu $G = (U, H)$ (viz [1]).

V praxi může být problém prezentován tak, že pro obdrženou množinu bodů (například měst) je hledána nejkratší cesta, po níž budou navštíveny všechny body a která po navštívení všech bodů povede zpět do jakéhokoliv zvoleného výchozího bodu.

Komplikací u problému obchodního cestujícího není nalézt teoretický postup řešení. Například zcela intuitivním řešení je prohlédání všech možných hamiltonovských kružnic, které přes uzly mohou vést, kalkulace ceny kružnice (v grafu počítáme s hranovým ohodnocením, které může být reprezentováno náklady na hranu, časem či délkou) a porovnání cen všech kružnic. Pro takové řešení bude jistě platit, že je přesné. Onou komplikací bude efektita hledání takového řešení.

Pokud bude uvažována malá instance problému, například 4 bodů, nebude složité najít řešení. Bude existovat pouze 6 hran a bude existovat pouze 12 možných kružnic, které budeme porovnávat. Pokud ale bude problém řešen pro například 50 bodů bude třeba kalkulovat s celkovým počtem $\frac{50(50-1)}{2}$ hran a tudíž celkovým počtem $\frac{50!}{2}$ možných řešení. Pro celkový počet hran bude platit $\frac{n(n-1)}{2}$ a pro celkový počet možných řešení $\frac{n!}{2}$. Složitost řešení problému tedy poroste opravdu rychle (exponencionálně), a je tudíž potřebné hledat efektivnější metody pro její řešení.

Variantou řešení, které neobnáší tak velkou výpočetní náročnost jsou heu-

ristiky. Jedná se o postupy řešení, které mohou sloužit jako aproximace správného řešení. Tyto postupy ovšem nejsou exaktními, jejich výsledky nemusí být optimální a i v případě, že optimální jsou, nepoznáme to (viz [7]). Tato práce se nebude heuristikami zabývat a bude zaměřena na exaktní metody, které jsou navrženy tak, aby hledaly optimální řešení a snahou je tyto exaktní metody tvořit co nejefektivnější, kdy efektivita může být reprezentována například časovou náročností výpočtu.

2. Formulace TSP v podobě úlohy CLP

V této kapitole se seznámíme s formulací úlohy obchodního cestujícího v podobě úloh celočíselného lineárního programování. Představíme si 2 různé přístupy k zamezení rozpadu na podokruhy.

Nejprve bude představena matematická formulace úlohy CLP

$$\min \quad \boldsymbol{c}^T \boldsymbol{x} \tag{1a}$$

$$\text{za podmínek} \quad \mathbf{A} \cdot \boldsymbol{x} \leq \boldsymbol{b} \tag{1b}$$

$$\mathbf{Aeq} \cdot \boldsymbol{x} = \boldsymbol{beq} \tag{1c}$$

$$\boldsymbol{lb} \leq \boldsymbol{x} \leq \boldsymbol{ub} \tag{1d}$$

$$x_1, x_2, \ldots, x_n \in \mathbf{Z} \tag{1e}$$

tedy je hledán vektor $\boldsymbol{x}$, tak aby byly minimalizovány náklady za dodržení podmínek. Prvky $x_1, x_2, \ldots, x_n$ vektoru $\boldsymbol{x}$ mohou nabývat pouze celočíselných hondot a to v rozmezí daném vektory $\boldsymbol{lb}$ a $\boldsymbol{ub}$.

Vektorem délek hran tak bude vektor $\boldsymbol{c} = (c_1, \ldots, c_k)$, kde k odpovídá počtu hran h.

2.0.1. Zavedení metriky délek

Pro měření délek v rovině existuje více možností. Pro tuto práci je k určení délek hran určena euklidovská vzdálenost, dána následující formulací.

Vektor délek hran bude odpoovídat euklidovské vzdálenosti mezi uzly $u \in U$. Pro výpočet vzdálenosti je použit vzorec

$$c_{i,j} = \sqrt{(u_{i1} - u_{j1})^2 + (u_{i2} - u_{j2})^2}, \tag{2}$$

kde $[u_{i1}, u_{i2}]$ budou souřadnice uzlu $u_i \in U$ a $[u_{j1}, u_{j2}]$ budou souřadnice uzlu u_j v rovině.

2.1. Popis úlohy TSP v CLP

V této podkapitole bude úloha CLP specifikována a důkladně vysvětlena v kontextu problému obchodního cestujícího. K vysvětlení bude použito obecného vyjádření úlohy CLP.

Jak bylo uvedeno, v úloze CLP bude probíhat minimalizace nákladové funkce, vektor $\boldsymbol{c}$ bude vektorem délek hran h z množiny hran H, vektor $\boldsymbol{x} = (x_1, \ldots, x_h, \ldots, x_n)$ bude rozhdovacím vektorem, tedy bude udávat, zda hrana h z množiny hran H je součástí kružnice ($x_h = 1$), či nikoliv ($x_h = 0$). Dále matice $\mathbf{A}$ s vektorem $\boldsymbol{b}$ poslouží k uložení nerovnostních podmínek a matice $\mathbf{Aeq}$ s vektorem $\boldsymbol{beq}$ k vygenerování rovnostních podmínek. Jak tyto podmínky vypadají bude předvedeno v následujícím textu.

Je dobré zmínit, že vektor $\boldsymbol{x}$ obsahuje každou hranu pouze jednou, pokud tedy obsahuje hranu $[i, j]$ už nebude obsahovat hranu $[j, i]$, pro $i < j$. Toto je možné díky předpokladu neorientovanosti a souvislosti grafu. Tato skutečnost má vliv i na indexování hran v rámci vektoru $\boldsymbol{x}$, kdy nejsou definovány indexy pro hrany uzlů $[j, i]$, $i < j$.

2.2. Degree constraints

V následující části textu budou popsány rovnostní podmínky úlohy, které se týkají omezení počtu hran procházejících jedním uzlem využitých v kruž-

nici.

Z definice je zřejmé, že je zapotřebí do uzlů přijít právě jednou cestou a jinou právě jednou zase odejít - to znamená celkem dvě hrany pro každý uzel. Tyto podmínky můžeme označit za tvz. *degree constraints* (tj. podmínky pro stupeň uzlu) a mohou být formálně vyjádřeny takto:

$$\sum_{\forall u \in U} (x_h : u \text{ je koncem } h) = 2, \tag{3}$$

kde x_h je prvek rozhodovacího vektoru x odpovídající h-té hraně v tomto vektoru (h je index hrany v množině hran H) a u je uzel z množinu uzlů U. Tedy pro každou hranu h musí existovat přesně dva konce $u \in U$. [1]

V literatuře je taky možno setkat s následující formulací *degree constraints*:

$$\begin{aligned} \sum_{i=1}^{n} x_{i,j} &= 1, \quad j = 1, \ldots, n \\ \sum_{j=1}^{n} x_{i,j} &= 1, \quad i = 1, \ldots, n \end{aligned} \tag{4}$$

kde $x_{i,j}$ bude označovat zda hrana jdoucí z i-tého do j-tého uzlu je součástí řešení ($x_{i,j} = 1$) či nikoliv ($x_{i,j} = 0$). Tj. i v této formulaci proměnné odpovídají hranám, pouze se číslují dvojicí indexů odpovídající příslušným uzlům a nikoliv přímo číslem hrany. Tyto podmínky říkají, že do libovolného uzlu j vstupuje právě jedna hrana a z libovolného uzlu i jedna hrana vychází. Proto jsou pro neorientovaný graf ekvivalentní k 3.

V praxi stanovení stupňových podmínek bude znamenat, že pro každý uzel $u \in U$ bude v matici **Aeq** existovat jeden řádek a počet sloupců této

matice bude odpovídat počtu hran množiny H (počet hran tedy bude roven $n(n-1)/2$, kde n označuje počet uzlů). Bude se jednat o matici jedniček a nul. Jedničky budou na i-tém řádku na pozicích indexů hran, které mohou vstupovat/vystupovat do i-tého uzlu. V teorii grafů se tato matice označuje jako incidenční matice grafu. Například pro úlohu o 5 uzlech bude matice **Aeq** vypadat následovně:

$$\mathbf{Aeq} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Pro 5 uzlů má tedy úloha 5 řádků. Množina hran H bude obsahovat celkem $5 \cdot (5-1)/2 = 10$ hran, proto 10 sloupců (ve všech úlohách platí předpoklad neorintovaných hran). Například jedničku na pozici $[3, 2]$ lze interpretovat jako hranu z/do 3. uzlu. Tato informace společně s informací, že na pozici $[1, 2]$ je taky jednička, lze interpretovat, že druhý sloupec se týká hrany z prvního do třetího uzlu.

Vektor $\boldsymbol{beq}$ bude vektorem dvojek, jak plyne z rovnice (3). Počet řádků vektoru $\boldsymbol{beq}$ bude odpovídat počtu uzlů v úloze. V tomto konkrétním případě tedy následovně:

$$\boldsymbol{beq} = \begin{pmatrix} 2 \\ 2 \\ 2 \\ 2 \\ 2 \end{pmatrix}$$

Pokud by tedy řešením byl vektor $\boldsymbol{x}e = (0, 0, 1, 1, 1, 1, 0, 0, 1, 0)$, lze snadno

ověřit, zda splňuje uvedené podmínky:

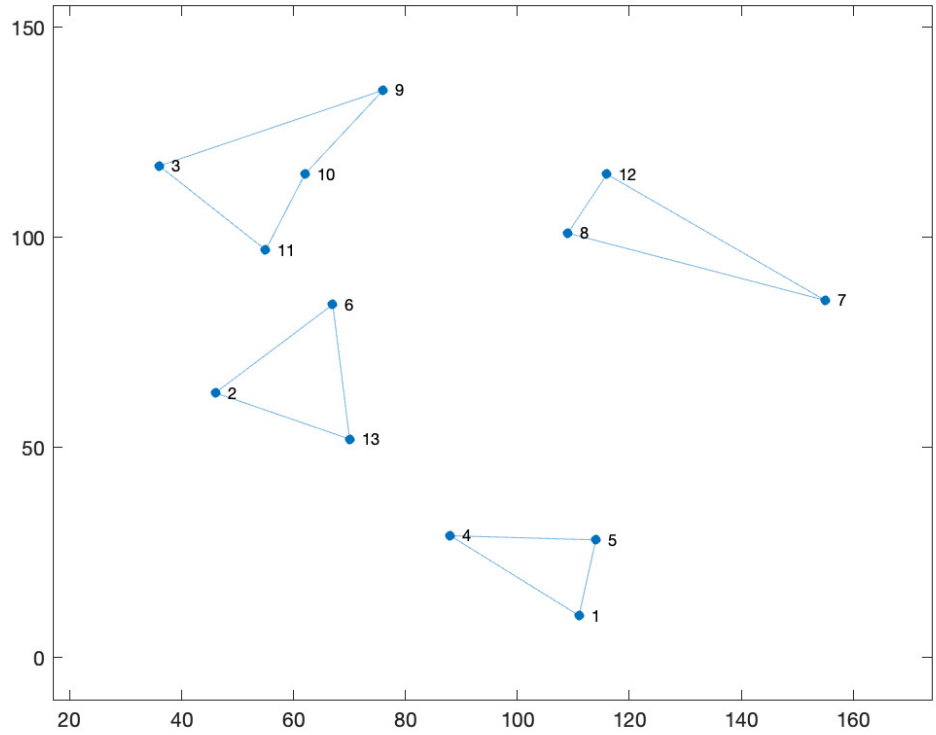
$$\begin{pmatrix} 1&1&1&1&0&0&0&0&0&0 \\ 1&0&0&0&1&1&1&0&0&0 \\ 0&1&0&0&1&0&0&1&1&0 \\ 0&0&1&0&0&1&0&1&0&1 \\ 0&0&0&1&0&0&1&0&1&1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \\ 2 \\ 2 \\ 2 \end{pmatrix},$$

kde rovnost platí, čímž je tedy prokázáno, že v tomto případě daný vektor x splňuje podmínku uvedenou v rovnici (1c).

2.3. Subtour constraints

V této podkapitole bude text věnován podmínkám ve tvaru nerovností uvedených v 1b. Ty jsou v popisu úlohy TSP řešené celočíselným programováním využity pro zamezení vzniku tzv. *subtours*, neboli zabrání rozpadu na více nepropojených kružnic. Takové řešení je nepřípustné, neboť je nutné, aby řešením byla jedna propojená kružnice. Samotné *degree conditions* tomu nezabrání, příklad *subtours* vyhovující všem *degree conditions* je na následujícím obrázku 1:

Jednotlivé subtoury můžeme popsat množinami uzlů $S_i \subset U$, $\left|S_i\right| > 2$.



Obrázek 1: Příklad iterace řešení TSP s výskytem subtours

2.3.1. DFJ formulace subtour constraints

Nejprve bude popsána formulace podmínek pro zamezení vzniku subtours podle pánů Dantzig, Fulkerson, Johnson (jak je uvedeno například zde [1]), zkráceně DFJ formulace.

Vzniku subtouru lze zamezit sestavením následující nerovnice:

$$\sum_{i=1}^{s}(x_h\text{: } h \text{ má jeden konec v } S_i \text{ a jeden mimo } S_i) \geq 2, \tag{5}$$

kde s bude označovat počet subtourů, x_h budou prvky vektoru $\boldsymbol{x}$ a h je indexem hrany v tomto vektoru.

Tyto podmínky tedy známe jako tvz. *subtour constraints*.

Alternativně lze pro zamezení vzniku subtours využít i následující myšlenku formulace subtour constraints. Pokud bude platit, že počet hran mezi uzly, které se vyskytují v daném *subtour* S_i, bude o jednu menší, než je počet uzlů v tomto *subtour* S_i, bude efektivně zamezeno vzniku tohoto *subtour*. Toto může být formulováno takto:

$$\sum_{h \in S_i} x_h = \left|S_i\right| - 1, \tag{6}$$

kde $\left|S_i\right|$ označuje početnost množiny S_i, tj. počet jejích prvků (zde uzlů).

Pro každou množinu S_i bude v matici A a ve vektoru $\boldsymbol{b}$ existovat jeden řádek. Počet sloupců matice $\mathbf{A}$ bude opět odpovídat počtu hran z množiny hran H. Matice $\mathbf{A}$ bude opět maticí jedniček a nul. Bude platit, že $A_{i,h} = 1$, pokud hrana $h \in H$ náleží $u \in S_i$. Na ostatních pozicích matice budou nuly.

V úloze pro n uzlů bude existovat celkem 2^n podmnožin uzlů. Protože degree constraints zamezují vzniku jednoprvkových (celkem n) a dvou prvkových množin (celkem $\frac{n(n-1)}{2}$), je třeba je od celkového počtu podmnožin odečíst. Zároveň v úloze nebude vznikat prázdná množina a úplná množina bude řešením. Obě tyto množiny tak je tedy opět nutné od celkového počtu podmnožin odečíst. V úloze tak bude existovat celkem $2^n - (2 + n + \frac{n(n-1)}{2})$ podmnožin množiny U (pro velká n bude dominovat člen 2^n). Tedy s rostoucím n poroste počet podmnožin exponencionálně. Pro zamezení vzniku jednoho subtouru bude potřeba pro každou jednu podmnožinu jeden řádek matice A, jeden řádek omezení. To bude znamenat pro rostoucí n velmi rychle rostoucí náročnost řešení. Pro úlohu o 10 uzlech tak bude existovat celkem 968 řádků omezení, pro úlohu o 25 uzlech přes milion řádků omezení, jen pro omezení zabraňující vzniku subtour. To znamená, že již pro poměrně malá n

bude úloha velmi rozsáhlá a bude potřeba příliš velké množství paměti pro její reprezentaci v počítači.

2.3.2. MTZ formulace subtour constraints

Pro práci se *subtours* v rámci řešení CLP je možno ještě využít dalšího přístupu zavedeného pány Miller, Tucker a Zemlin (viz [12]). Opět se bude jednat o zavedení podmínek ve tvaru nerovností, stejně jako v případě DFJ formulace.

Matematická formulace vypadá následovně

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j}x_{i,j} \\
\text{za podmínek} \quad & \sum_{i=1}^{n} x_{i,j} = 1, \quad j = 1,\ldots,n \\
& \sum_{j=1}^{n} x_{i,j} = 1, \quad i = 1,\ldots,n \\
& u_i - u_j + (n-1)x_{i,j} \leq n-2, \quad i,j = 2,\ldots,n,\ i \neq j \\
& 1 \leq u_i \leq n-1, \quad i = 2,\ldots,n \\
& x_{i,j} \in \{0,1\}, \quad \forall i,j
\end{aligned} \tag{7}$$

kde u_i označují pořadí ve kterém je uzel i navštíven. Pro u_1 je pevně stanoveno $u_1 = 1$. Dále označení $x_{i,j}$ nese stejný význam jako již bylo uvedeno v případě *degree constraints* (viz [12]).

Výhodou této formulace je výrazně menší počet omezení (uvést počet) za cenu o n většího počtu neznámých (jsou zde navíc u_i). Pro velké úlohy je ovšem i v tomto případě podmínek příliš mnoho. Úskalím tohoto přístupu ke konstrukci podmínek pro zamezení vzniku *subtours* je jejich efektivita. Re-

laxace (princip relaxace bude popsán v následujícím textu) problému těmito podmínkami není tak efektivní jako u přístupu DFJ. Pro MTZ se postupem času podařilo vyvinout další formulace tohoto typu. Nebylo sice dokázáno, že tyto modifikované alternativy jsou méně efektivní, než přístup DFJ. Nicméně jsou příklady, kdy i tyto modifikvoané (vylepšené) varianty jsou méně efektivní než DFJ a proto se v následujícím textu budeme věnovat zpracování DFJ formulace (viz [12]).

3. Exaktní metody pro TSP

Pro exaktní řešení úlohy TSP je možno volit více přístupů. Tyto přístupy lze rozdělit do 3 skupin:

1. jednoduchá metoda hrubé síly,
2. dynamické programování
3. CLP relaxace, případně LP relaxace

V následujícím textu budou představeny zástupci z těcto skupin a budou popsány jejich principy společně s limitacemi a úskalími, kterým tyto jednotlivé metody čelí.

3.1. Metoda hrubé síly

První exaktní metodou, kterou lze použít pro řešení úlohy obchodního cestujícího je tzv. „metoda hrubé síly“ (anglicky *brute force*).

Princip této metody je velmi jednoduchý. Uživatel určí všechny kombinace uzlů, tak aby byla splněna podmínka pro okruh všech uzlů z množiny U. Následně pro všechny tyto kombinace jsou napočítány náklady (ceny) cesty přes tyto okruhy a z těchto hodnot je nakonec vybrána ta nejmenší.

Tento způsob řešení je poměrně intuitivní a jednoduchý nicméně je velmi neefektivní z hlediska výpočtu, kdy je potřeba napočítat $n!$ délek okruhu, a náročnost tak roste rychleji než exponencionálně. Stejně tak je tento přístup náročný na výpočetní paměť a hodí se tak pro řešení jen velmi malých úloh.

3.2. Dynamické programování

Dalším popsaným přístupem bude dynamické programování, konkrétně Held-Karpův algoritmus, který uplatňuje níže popsané principy dynamického

programování v úloze obchodního cestujícího.

Principem dynamického programování je rozkládání složitých problémů na menší (tedy jednodušší) podproblémy, které jsou následně řešeny. Při výpočtu je využíváno rekurzivních vztahů a ukládání dílčích výsledků. [10]

Mějme podmnožinu S množiny uzlů U. Do této podmnožiny jsou postupně přidávány uzly. Pro přidání uzlu je kalkulován vždy stav pro tuto podmnožinu (viz následující definice). Vždy je kalkulován mininální náklad pro dosažení dalšího uzlu (uzlu j) jako minimální náklad pro cestu mezi všemi uzly 1 až i a náklad pro cestu mezi uzlem i uzlem j.

Definice 10 ***Stav*** *je reprezentován uspořádanou dvojicí* (S, j)*, kde* S *je podmnožina vrcholů včetně výchozího a vrchholu* j*, kde cesta končí.*

Definice 11 ***Minimální náklad cesty*** *je označen* $C(S, j)$ *a je dán vztahem*

$$C(S, j) = min_{i \in S, i \neq j}[C(S \backslash \{j\}, i) + d(i, j)],$$

kde $d(i, j)$ *je náklad na cestu z vrcholu* i *do vrcholu* j*.*

V průběhu výpočtu se tedy iteruje přes všechny podmnožiny množiny uzlů U. Těchto podmnožin je v každé úloze celkem 2^n, kde n je počet uzlů. Nemá smysl počítat s prázdnou podmnožinou, celkový počet iterací tak bude roven $2^n - 1$.

Pomocí těchto podmnožin jsou následně kalkulovány náklady potřebné pro dosažení i-tého z předcházející podmnožiny uzlů. Celkové náklady pro dosažení i-tého uzlu jsou součtem pro průchod předcházející podmnožiny uzlů a nákladem na cestu do i-tého uzlu.

Hlavním úskalím této metody je její exponencionálně rostoucí asymptotická náročnost a velká paměťová náročnost. Oba tyto nedostatky se projeví

při programové realizaci a numerickém srovnání exaktních metod, kterému je věnována další kapitola následující po popisu všech exaktních metod.

3.3. Metoda CLP relaxace

Zjednodušení výpočetní náročnosti úlohy obchodního cestujícího je provedono procesem relaxace problému. Proces začne s nějakou základní množinou omezení a podmínky budou postupně přidávány podle potřeby, tj. pouze ty, které budou porušeny. Postupné přidávání podmínek bude provedeno u nerovnostních podmínek pro subtours, jejichž počet se, jak bylo výše uvedeno, exponenicionálně zvyšuje s rostoucím počtem uzlů. Počet degree constraints roste lineráně a tak v tomto smyslu nepředstavují komplikaci.

Algoritmus pro výpočet optimálního řešení TSP metodou CLP

1. Stanovení délek hran a sestavení úlohy pouze s degree constraints
2. Výpočet úlohy pomocí celočíselného lineárního programování
3. Prohledání řešení na výskyt subtours
    a) Pokud kontrola neodhalí subtours je řešení optimální a výpočet končí
    b) Pokud kontrola odhalí subtours, následuje další krok
4. Přidání podmínek na zamezení vzniku odhalených subtours
5. Zpět k 2

3.3.1. Prohledávání řešení na výskyt subtours

Z výše uvedeného algoritmu je zřejmá nutnost algoritmu, sloužícího pro vyhledávání subtours v řešení. Zvolil jsem algoritmus prohledávání grafu do

šířky neboli anglicky *Breadth-first search method* (zkráceně BFS).

Po volbě libovolného uzlu jako prvního, jsou postupně projity všechny sousední uzly v řešení.

Algoritmus nejprve najde první nenavštívený uzel, navštíví všechny sousedící uzly a opět navštěvuje sousední uzly, dokud žádné sousední uzly nezbudou. Všechny tyto uzly jsou následně označeny jako navštívené a indexy těchto uzlů jsou uloženy jako jeden subtour. Následně je nalezen první neprozkoumaný uzel a tento proces je opakován do doby, než jsou všechny uzly označeny jako navštívené (viz [4]).

Tímto způsobem jsou tedy postupně odhaleny všechny subtoury. Jednotlivé subtours jsou označeny jako množiny $S_i, i = 1, 2, \ldots, m$, kdy S_i označuje i-tý *subtour*.

Podmínky pro zamezení vzniku subtours jsou následně přidány pouze pro nalezené subtoury S_i a to ve stejné podobě popsané v rovnici 6, tedy že počet hran mezi uzly v rámci jednoho *subtour* S_i je menší nebo roven $\left|S_i\right| - 1$.

Díky tomuto postupu lze pomocí celočíselného programování řešit i úlohy s větším počtem uzlů, oproti řešení úlohy se všemi podmínkami, neboť počet omezení je obvykle významně snížen, byť v nejhorším případě můžeme přidat všechna omezení. Na druhou stranu ovšem musíme řešit posloupnost postupně se zvětšujících se úloh CLP.

3.4. Metoda LP relaxace

V dnešní době není problémem používat celočíselné řešiče úlohy celočíselného programování. V dřívějších dobách ovšem takové řešiče chyběly a tak úloha obchodního cestujícího byla řešena jako úloha lineárního programování. S tím se pojí problém výskytu neceločíselných hodnot ve vektoru $\boldsymbol{x}$. Dosavadní postup pro řešení celočíselným programováním je nutné rozšířit

o podmínky zakazující vznik tzv.*combs*, které budou popsány níže. Combs indikují nesplnění podmínek celočíselnosti.

Pokud budou odstraněny všechny *subtours* a to v jakékoli výše popsané podobě může se objevit problém neceločíselného řešení. I tento problém je popsán pány Dantzigem, Fulkersonem a Johnsonem (popsáno viz [1]).

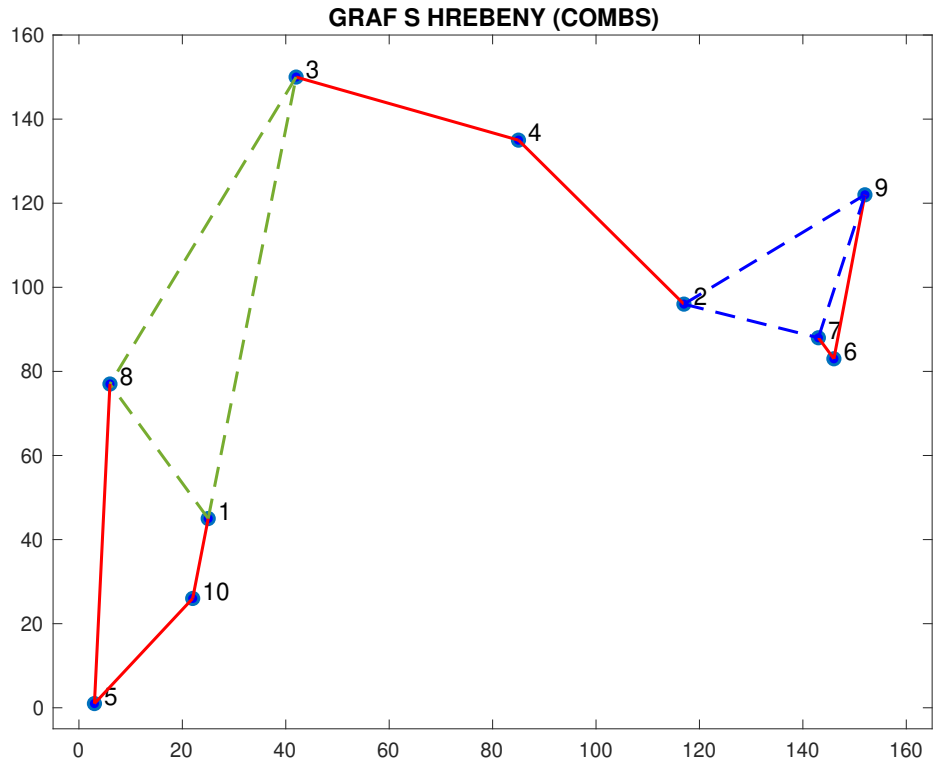
Příklad výskytu neceločíselného řešení může být pozorován na obrázku 2 například v trojúhelníku uzlů 1,3,8, společně i s přilehlými hranami $h = \{1, 10\}, h = \{5, 8\}, h = \{3, 4\}$. Zmiňovaný trojúhelník je v grafu zvýrazněn zelenou barvou. Přerušovaná čára znamená, že pro danou hranu platí, že je jí ve vektoru $\boldsymbol{x}$ přiřazena hodnota $x_h = 1/2$, nepřerušovaná čára znamená $x_h = 1$.

Útvar zvaný combs je tvořen výše zmíněným trojúhelníkem uzlů 1, 3, 8 společně s přilehlými vrcholy 10, 5 a 4, které jsou s ním spojené hranami (1,10) , (8, 5) a (3,4). Pro jednotlivé hrany bude platit:

$$x_{1,3} = 1/2, x_{1,8} = 1/2, x_{3,8} = 1/2,$$

$$x_{1,10} = 1, x_{3,4} = 1, x_{5,8} = 1, x_{5,10} = 1, x_{2,4} = 1$$

Jednotlivé uzly v *combs* se dají rozdělit do dvou typů množin. První skupinou je takzvaná rukojeť (množina R) a druhou skupinou jsou takzvané zuby (množiny Z_i). Množiny Z_i jsou vzájemně disjunktní a množina R obsahuje po jednom bodu z každé z množin Z_i. Body jsou vybrány tak, aby přes takovou množinu R šlo propojit ostatní části grafu.



Obrázek 2: Příklad neceločíselného řešení

V konkrétním případě obrázku 2 by rozdělení do jednotlivých množin tedy vypadalo následovně:

$$R = \{1, 3, 8\}, Z_1 = \{1, 10\}, Z_2 = \{5, 8\}, Z_3 = \{3, 4\}$$

Problém je opět možné v lineárním programování řešit přidáním nerovnic.

Dantzig, Fulkerson a Johnson (popsáno viz [1]) navrhli řešit problém přidáním lineární nerovnice pro každou combs.

$$\sum_{i=1}^{3}(\sum(x_h : h\text{má jeden konec v } Z_i \text{ a jeden ne v } Z_i))+$$
$$+\sum(x_h : h\text{má jeden konec v } R \text{ a jeden v } Z) \geq 10$$

Pro combs z obrázku 2 je to následující nerovnice:

$$(x_{1,10} + x_{5,8} + x_{3,4}) + ((x_{1,8} + x_{1,3} + x_{5,10}) + \dots$$
$$(x_{3,8} + x_{1,8} + x_{5,10}) + (x_{3,8} + x_{1,3} + x_{2,4})) = 9.$$

Podmínka vychází z toho, že pro libovolnou podmnožinu Z množiny všech uzlů musí platit, že do ní vede minimálně jedna cesta a minimálně jedna cesta vede z ní, a počet hran vedoucích z/do množiny Z musí být sudý. Jinak by nebylo možné propojit podmnožinu Z se zbylými uzly grafu do jednoho okruhu. Jelikož to platí pro každou podmnožinu, tak i pro množiny R, Z_1, Z_2, Z_3, což vyjadřují jednotlivé členy v závorkách na levé straně předchozí nerovnosti. Pokud nějaká podmnožina (zde R) má průnik se 3 jinými disjunktními množinami (zde Z_1, Z_2, Z_3), tak hodnota prvního členu nerovnosti musí být aspoň 3 resp. 4 neboť počet musí být sudý. Hodnota ostatních členů (tj. pro Z_1, Z_2, Z_3) musí být nejméně 2. Minimální součet je tedy opravdu 10.

Obecně lze tyto podmínky zformulovat pomocí definice takto:

Definice 12 *Mějme množiny R a $Z_1, \dots, Z_z$ pro liché z, takové že $Z_i \cap Z_j = \emptyset$, $i \neq j$, a $Z_i \cap R \neq \emptyset$, potom nerovnice pro odstranění combs (hřebenů) je dána vztahem*

$$\sum_{i\in R, j\notin R} x_{i,j} + \sum_{k=1}^{z} \sum_{i\in Z_k, j\notin Z_k} x_{i,j} \geq 3z + 1. \tag{8}$$

Obecně bude platit, že proměnná $x_{i,j}$ je opět prvkem vektoru $\boldsymbol{x}$, indexy i, j určují odkud kam hrana vede a určují tak o kterou hranu $h \in H$ se jedná (dříve v textu byly hrany označeny pouze indexem pořadí v rámci vektoru $\boldsymbol{x}$) (viz [1])

První suma tedy bude sčítat hodnoty hran z vektoru $\boldsymbol{x}$, pro které platí, že mají jeden konec v množině R a druhý konec mimo tuto množinu R.

Dvojná suma potom sčítá hodnoty z vektoru $\boldsymbol{x}$ pro hrany, jež mají jeden konec v množině Z_k a druhý v kterémkoliv uzlu mimo Z_k. Toto se sčítá přes všechny množiny $Z_1, \ldots, Z_z$, tedy přes všechny zuby.

Algoritmus pro řešení je v porovnání s algoritmem pro řešení úlohy celočíselného programování upraven do této podoby:

Algoritmus pro výpočet optimálního řešení lineárním programováním

1. Výpočet řešení zvolenou metodou lineárního programování
2. Kontrola výskytu subtours
    a) Pokud se v řešení $\boldsymbol{x}$ nevyskytují žádné subtours, následuje krok 3
    b) Pokud řešení $\boldsymbol{x}$ obsahuje subtours - stanovení podmínek pro zamezení jejich vzniku a zpět 1.
3. Kontrola výskytu combs
    a) Pokud se v řešení $\boldsymbol{x}$ nevyskytují žádné combs, $\boldsymbol{x}$ je řešení
    b) Pokud se v řešení $\boldsymbol{x}$ vyskytují combs - stanovení podmínek pro jejich zamezení, zpět k 1 (výpočet proběhne s nově nastavenými podmínkami)

V případě, že je tedy řešena úloha celočíselným programováním, tento problém nenastane a není potřeba se jím zabývat.

4. Programová realizace s numerické srovnání metod

Tato kapitola se věnuje zpracování vybraných metod v softwaru MATLAB (viz [8]). Největší pozornost jsem věnoval sestavení kódu pro výpočet řešení TSP metodou CLP relaxace, další kódy byly využity hlavně za účelem srovnání náročnosti výpočtu a byly čerpány z různých zdrojů.

4.1. Popis funkcí

V této části popíšu fungování funkcí pro jejich uživatelské použití.

4.1.1. Řešiče dle jednotlivých přístupů

Tyto funkce jsou samotnými řešiči pro jednotlivé přístupy řešení úlohy TSP. Jsou volány uživatelem a jejich výsledky jsou řešeními úloh TSP.

Metoda	Funkce
CLP relaxace (náhodné body)	`clp_relaxace`
CLP relaxace (ze souboru)	`clp_relaxace_vlsi`
Dynamické programování	`dynamicke_programovani`
Metoda hrubé síly	`bruteforce`

Tabulka 1: Řešice dle jednotlivých přístupů

Vstupy a výstupy jednotlivých funkcí

1. **CLP relaxace (náhodné body)**

   [*opt_ cena,cas_ vypoctu*]=`clp_relaxace`(*pocet_ bodu,malovani,kresleni_ bodu,seminko)*

   a) *opt_ cena*

   Prvním výstupem funkce jsou celkové náklady na výslednou kružnici.

   b) *cas_ vypoctu*

   Druhým výstupem funkce je čas potřebný pro výpočet optimálního řešení.

   c) *pocet_ bodu*

   Uživatel jako vstup volí, pro kolik bodů se bude náhodná úloha generovat, vstupem je tak libovolné celé číslo.

   d) *malovani*

   Uživatel si může zvolit, zda chce aby se v průběhu výpočtu vykreslovala grafická řešení pro jednotlivé iterace. Pokud bude argument *malovani*= 1, grafy se budou vykreslovat. Pokud bude *malovani*= 0, grafy se vykreslovat nebudou.

   e) *kresleni_ bodu*

   Stejně tak má uživatel možnost vybrat, zda chce, aby jednotlivé uzly (body) byly v grafickém řešení očíslované. Například pro lepší přehlednost při řešení větších úloh doporučuji číslování bodů nepoužívat. Pokud tedy uživatel nechce body vykreslovat, bude argument nastaven na hodnotu *vykreslovani_ bodu*= 0. V opačném případě *vykreslovani_ bodu*= 1.

   f) *seminko*

   Pro náhodné úlohy je možnost volit různé hodnoty *seedu*. Díky

tomu je pak generována vždy posloupnost stejných čísel. Vstupem by tak mělo být celé číslo.

2. **CLP relaxace (ze souboru)**

   [*opt_ cena,cas_ vypoctu*]=`clp_relaxace_vlsi`*(xqf131,malovani,kresleni_ bodu)*

   a) Výstupy funkce jsou stejné jako v případě funkce `clp_relaxace`

   b) *xqf131*

      Uživatel tímto vstupem volá nahraný soubor, který je součástí přílohy. Nahrání soboru je popsáno ve skriptu `skript_clp_relaxace_vlsi`

   c) *malovani* a *kresleni_ bodu*

      Tyto vstupy mají stejnou funkci jako `clp_relaxace`

3. **Dynamické programování**

   [*minCost,optimalTour*=`dynamicke_programovani`*(distMatrix)*

   K sestrojení této funkce bylo využito nástroje ChatGPT (viz [11]).

   a) *minCost*

      Prvním výstupem funkce jsou náklady na výslednou kružnici.

   b) *optimalTour*

      Druhým výstupem je pořadí návštěvy uzlů ve výsledné kružnici.

   c) *distMatrix*

      Jediným potřebným vstupem je matice vdzáleností pro dannou úlohu

4. **Metoda hrubé síly**

   [*bestDistance,bestRoute*=`bruteforce`*(distanceMatrix)*

Tato funkce mnou nebyla sestrojena, ale použita z přiloženého zdroje (viz [13]).

a) *bestDistance*

Prvním výstupem funkce jsou náklady na optimální kružnici.

b) *bestRoute*

Druhým výstupem je pořadí návštěv uzlů v optimální kružnici.

c) *distanceMatrix*

Vstupem je opět matice vzdáleností pro danou úlohu.

4.1.2. Pomocné funkce

Tyto funkce jsou využívány v předpisech řešičů hlavní funkce, některé z nich jsou volány v přiložených skriptech za účelem správného nastavení vstupních argumentů.

Účel	Funkce
Tvorba matice sousednosti	`najdi_sousednost`
Grafické znázornění řešení	`nakresli_graf`
Hledání subtours	`najdi_subtours`
Vytvoření vektoru indexů	`najdi_vektor_indexu`
Tvorba matice vzdáleností	`matice_vzdelensoti`

Tabulka 2: Pomocné funkce

Využití pomocných funkcí

1. `najdi_sousednost`

   Funkce slouží pro tvorbu matice sousednosti z vektoru x a je použita ve funkcích `clp_relaxace_vlsi` a `clp_relaxace`

2. `nakresli_graf`

Tato funkce slouží pro vykreslování grafických řešení úloh. Opět je využita ve funkcích `clp_relaxace_vlsi` a `clp_relaxace`.

3. `najdi_subtours`

   Tato funkce slouží pro prohledání řešení na výskyt subtours. Její použití je ve funkcích `clp_relaxace_vlsi` a `clp_relaxace`.

4. `najdi_vektor_indexu`

   Tato funkce je použita ve funkcích pro CLP metodu. Je využita při tvorbě matice, kde jsou uloženy nerovnice pro zabránění vzniku *subtours*.

5. `matice_vzdalenosti`

   Tato funkce převádí vektor vzdáleností na matici vzdáleností. Je využita ve skriptech `brute_force_skript` a `dynamicke_programovani_skript`

4.2. Skripty s programovou realizací

Součástí přílohy jsou i skripty s programovou realizací jednotlivých metod a využitím výše uvedených funkcí. V této podkapitole krátce popíšu fungování jednotlivých skriptů.

4.2.1. Skript pro CLP relaxaci s náhodnými úlohami

Skript pro účel použití CLP relaxace na náhodných úlohách se nachazí v příloze `skript_ clp_relaxace.m`. Zde je příklad jeho použití:

```
1 pocet_bodu=150;
2 malovani=0;
3 kresleni_bodu=0;
4 seminko=123;
```

```
5
6 [opt_cena,cas_vypoctu]=clp_relaxace(pocet_bodu,malovani,kresleni_bodu,seminko);
```

V 1. řádku je nastaven počtu uzlů na 150, grafické znázorňování během výpočtu je vypnuté na řádcích 2 a 3, a *seed* pro náhodné generování byl nastaven na hodnotu 123 na řádku 4.

Na řádku 6 je samotné volání funkce, která bude řešit úlohu s danými parametry.

4.2.2. Skript pro CLP relaxaci s úlohou VLSI

Tento skript je v příloze `skript_ clp_relaxace_vlsi.m`. Slouží pro zadání úlohy VLSI, která je v příloze `xqf131.txt`. Zde je příklad použití:

```
1 xqf131=importfile("/Users/matyasvavra/Downloads/xqf131.txt",[9,139]);
2 [opt_cena,cas_vypoctu]=clp_relaxace_vlsi(xqf131,1,0);
```

Na řádku 1 načtena úloha ze souboru. Pro načtení je potřeba nastavit cestu ke zdrojovému souboru. Na řádku 2 je již použití funkce pro řešení dané úlohy.

4.2.3. Skript pro metodu dynamického programování

Tento skript se nachází v příloze `dynamicke_programovani_skript.m`. Opět přikládám příklad použití zde:

```
1  rng(123)
2  pocet_bodu=12;
3
4  %nastaveni argumentu pro pouziti funkce dynamicke_programovani
5  souradnice=randi(158,pocet_bodu,2);
```

```
6  hrany=nchoosek(1:pocet_bodu,2);
7  p1 = souradnice(hrany(:,1),:); %pocatecni uzly
8  p2 = souradnice(hrany(:,2),:); %koncove uzly
9  distance = sqrt(sum((p1-p2).^2,2)); %delky hran
10 distance=round(distance);
11
12 %prevedeni vzdalenosti do matice
13 matice_vzdalenosti=matice_vzdalenosti(distance,pocet_bodu);
14
15 tic
16 naklad=dynamicke_programovani(matice_vzdalenosti);
17 cas=toc;
```

Na prvních dvou řádcích volím pro kolik uzlů chci počítat náhodnou úlohu a nastavuji *seed*. Na řádcích 5-10 probíhá vygenerování úlohy o požadovaném rozsahu a spočtení vzdáleností mezi uzly. Na řádku 13 je vypočtený vektor vzdáleností převeden do matice. Na řádku 16 je použita funkce pro řešení samotné úlohy. Příkazy na řádcích 15 a 17 slouží pro měření času výpočtu úlohy.

4.2.4. Skript pro metodu hrubé síly

Posledním přiloženým skriptem je `brute_force_skript`, kde je pro řešení využita metoda hrubé síly. Ukázka skriptu zde:

```
1  rng(123)
2  pocet_bodu=12;
3
4  %nastaveni parametru pro pouziti funkce bruteforce
```

```
5  souradnice=randi(158,pocet_bodu,2);
6  hrany=nchoosek(1:pocet_bodu,2);
7  p1 = souradnice(hrany(:,1),:); %pocatecni uzly
8  p2 = souradnice(hrany(:,2),:); %koncove uzly
9  distance = sqrt(sum((p1-p2).^2,2)); %delky hran
10 distance=round(distance);
11
12 %prevedeni vzdalenosti do matice
13 distance=matice_vzdalenosti(distance,pocet_bodu);
14
15 [bestDistance,bestRoute]=bruteforce(distance);
```

V ukázce je opět na řádcích 1 a 2 vidět nastavení *seedu* a počtu uzlů. Na řádcích 4-10 znovu probíhá vygenerování náhodné úlohy a spočtení vzdálenostní mezi uzly. Na řádku 10 je vektor vzdáleností převeden na matici a na řádku 15 je použita funkce pro řešení úlohy metodou hrubé síly.

4.3. CLP relaxace v MATLAB

V této části tak budou uvedeny ukázky výpočtu včetně grafických ukázek.

4.3.1. Popis fungování řešiče pro úlohy CLP

K výpočtu je použita výchozí funkce pro řešení úloh celočíselného programování `intlinprog` z optimalizačního balíčku OPTIMALIZATION TOOLBOX. Tento celočíselný řešič je použit ve funkcích `clp_relaxace` a `clp_relaxace_vlsi`.

Pro snadnější pochopení bude nastíněno základní fungování této funkce, budou její vstupní argumenty a jejich význam.

V případě `intlinprog` se jedná o funkci tzv. smíšeného lineárního programování, tedy kombinaci lineárního a celočíselného lineárního programování.

Tato funkce vychází z předpisu

$$\begin{aligned} \min \quad & c^T \boldsymbol{x} \\ \text{za podmínek} \quad & \mathbf{A} \cdot x \leq \boldsymbol{b} \\ & \mathbf{Aeq} \cdot x = \boldsymbol{beq} \\ & \text{lb} \leq x \leq \text{ub} \\ & x(intcon) \in \mathbb{Z}, \end{aligned}$$

kde c^T bude zastupovat v tomto případě vektor délek hran, výsledný vektor $\boldsymbol{x}$ bude rozhodovacím vektorem (viz [2]).

Matice **Aeq** a vektor $\boldsymbol{beq}$ jsou využity pro zadání degree constraints. Jeden řádek matice **Aeq** bude odpovídat jednomu uzlu $u \in U$ (viz [2]).

Vektor *intcon* je vektor indexů určující, které prvky $\boldsymbol{x}$ musí být celočíselné. V případě řešení úlohy obchodního cestujícího tedy bude obsahovat indexy všech hran $h \in H$.

Matice **A** a vektor $\boldsymbol{b}$ jsou podmínky ve tvaru nerovností. V CLP relaxaci jsou využity při generování *subtour constraints*. Pro subtour S_i bude existovat i-tý řádek matice **A**. [2]

Hodnoty *lb* a *ub* slouží k určení intervalu, ve kterém se mohou pohybovat hodnoty x_i. V případě řešení úlohy TSP tedy $ub = 1$ a $lb = 0$. [2]

Pro potřeby testování relaxační metody, budou generovány náhodné úlohy (jsou náhodně generovány body v rovině) a ty následně budou řešeny vypracovaným algoritmem.

4.3.2. Aplikace algoritmu na data VLSI

Nejprve si ukážeme fungování metody CLP relaxace na úloze s daty o vrtání otvorů do VLSI desky o velikosti 131, tedy optimalizace dráhy pohybu vrtačky nad deskou.

Data jsou čerpána z knihovny TSP benchmarků University of Waterloo (viz [3]). Součástí této knihovny jsou i správné výsledky a bude tak ověřena i správnost výpočtu sestaveným algoritmem.

Vzdálenosti mezi jednotlivými uzly jsou v úlohohách o VLSI datech zaokrouhlovány na nejbližší celé číslo (viz. [3]), toto zaokrouhlední je zahrnuto i do mého algoritmu.

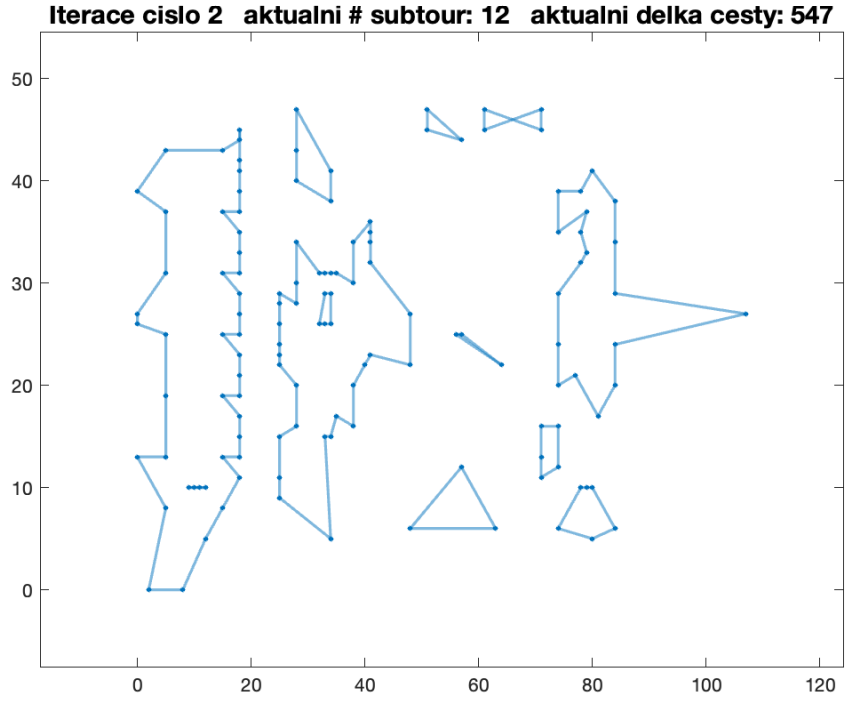
Z obrázků 3-12 je tedy zřejmé, že algoritmus nalezl správné řešení v průběhu celkem 10 iterací. Výpočet trval celkem 12.5 s. Pro tuto úlohu je v práci uveden vývoj všech iterací v jednotlivých obrázcích.

V grafech křížek "#"označuje počet iterací, a jednou iterací je myšlena jedna iterace relaxace, tj. tedy přidání podmínek pro zamezení vzniku subtours z předcházející iterace a výpočet s nově stanovenými podmínkami.

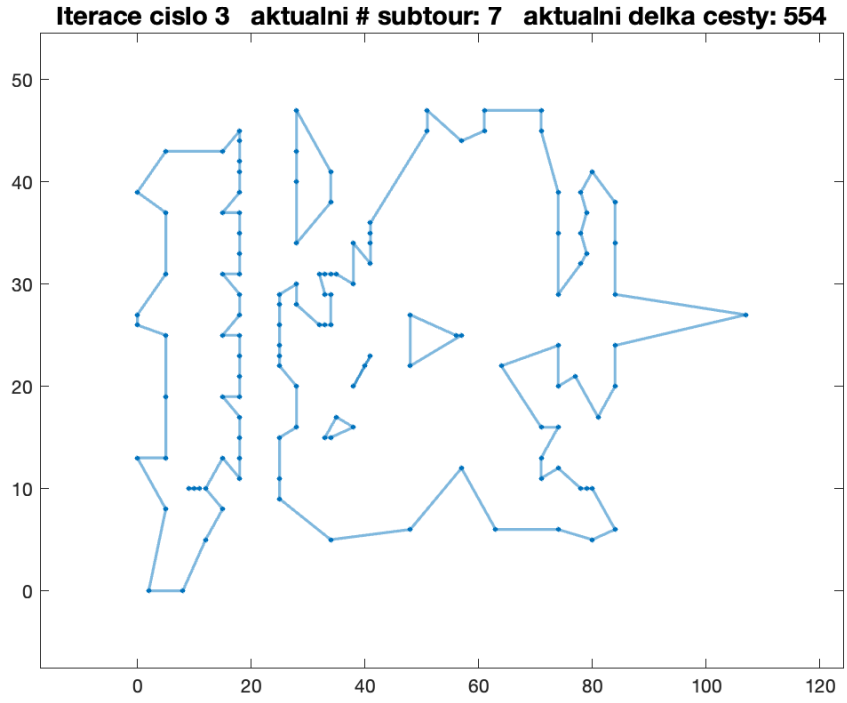
Lze pozorovat, že v průběhu iteračního procesu délka cesty postupně roste. Po první iteraci je délka cesty 543, v řešení se ovšem vyskytuje 13 subtourů (viz 3).

Dále lze pozorovat, že není nijak v algoritmu zaručeno, že se počet subtourů bude během iteračního procesu snižovat. Tento jev lze sledovat například mezi iteracemi číslo tři a pět (viz 5 a 7), kdy počet subtourů mezi jednotlivými iteracemi vzrostl o 1. Tento jev není nežádoucí a v principu relaxace nelze vyžadovat, aby se počet subtourů s každou iterací snižoval. Smyslem relaxační metody je problematické subtoury postupně nalézat a následně je odstraňovat. Zamezením vzniku jednoho může dojít v procesu výpočtu k vytvoření dalších několika nových, které je nutné taktéž nalézt a

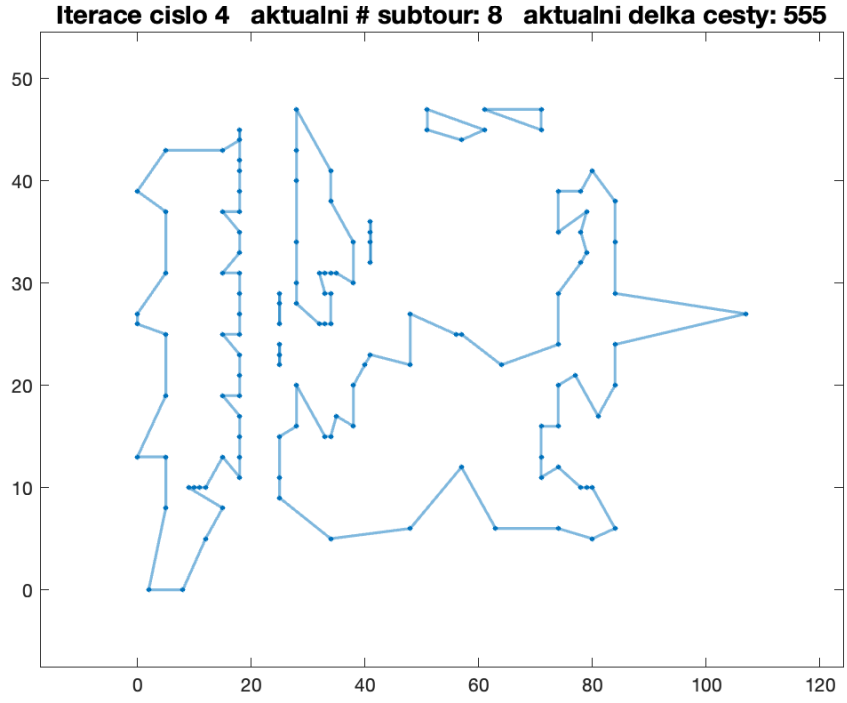
Obrázek 3: I. iterace - 131 uzlů VLSI



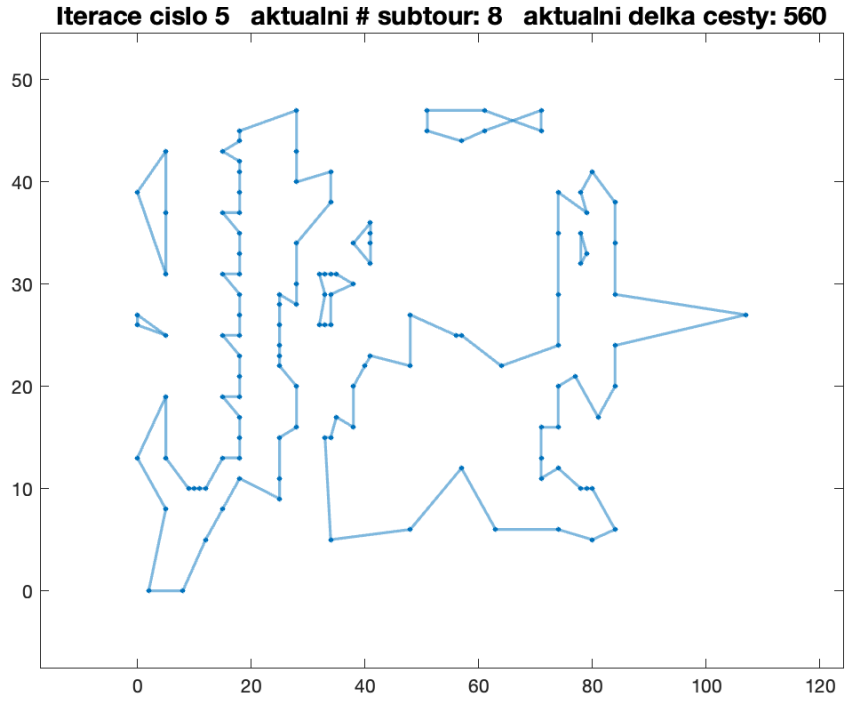
Obrázek 4: II. iterace - 131 uzlů VLSI



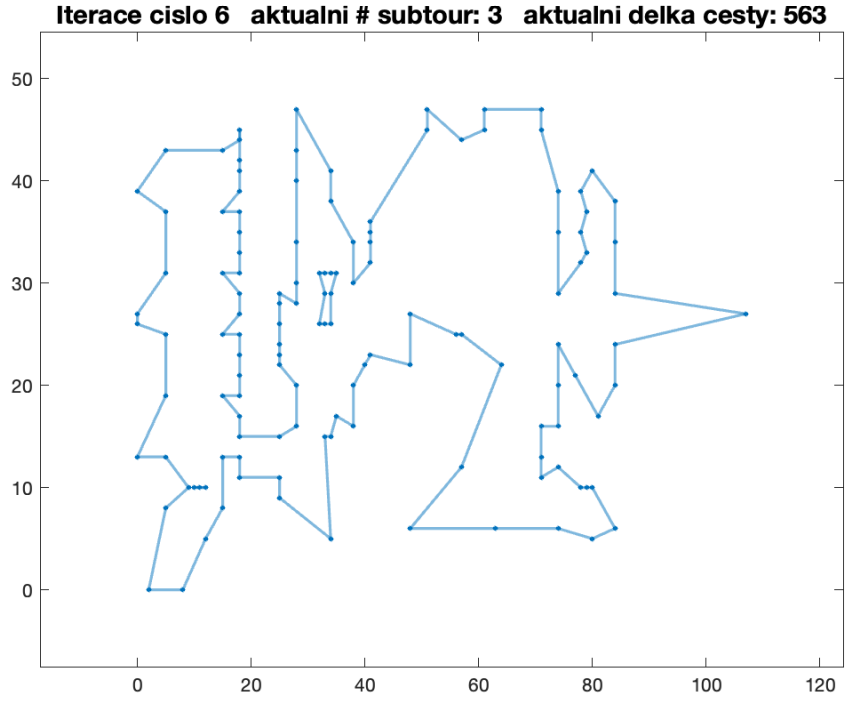
Obrázek 5: III. iterace - 131 uzlů VLSI



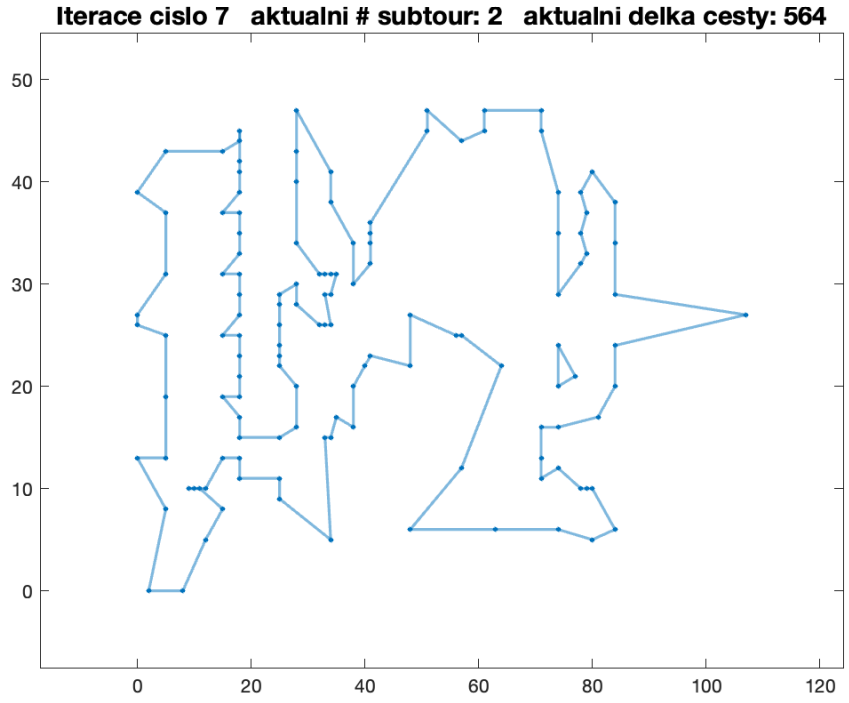
Obrázek 6: IV. iterace - 131 uzlů VLSI



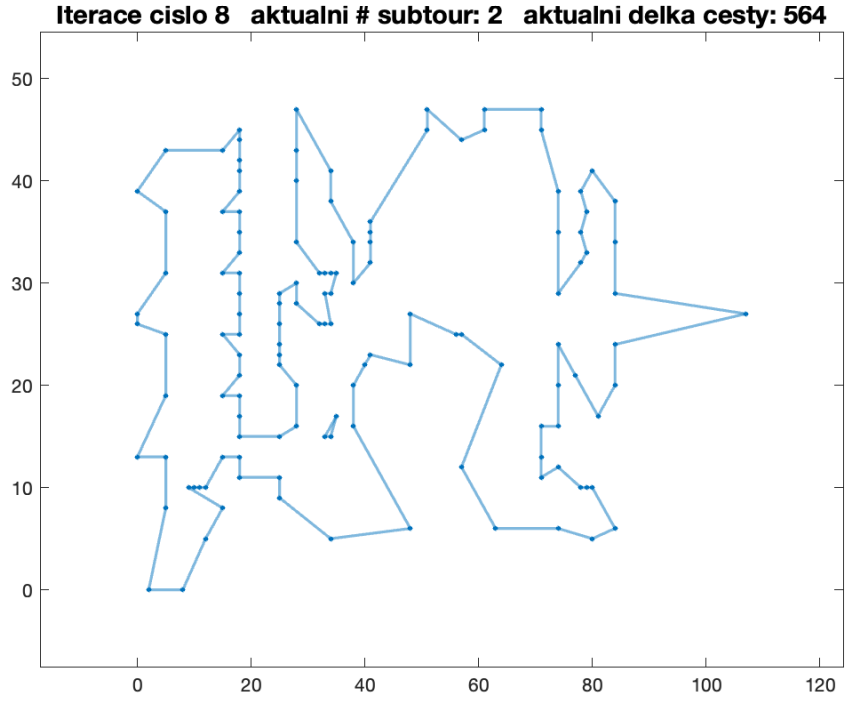
Obrázek 7: V. iterace - 131 uzlů VLSI



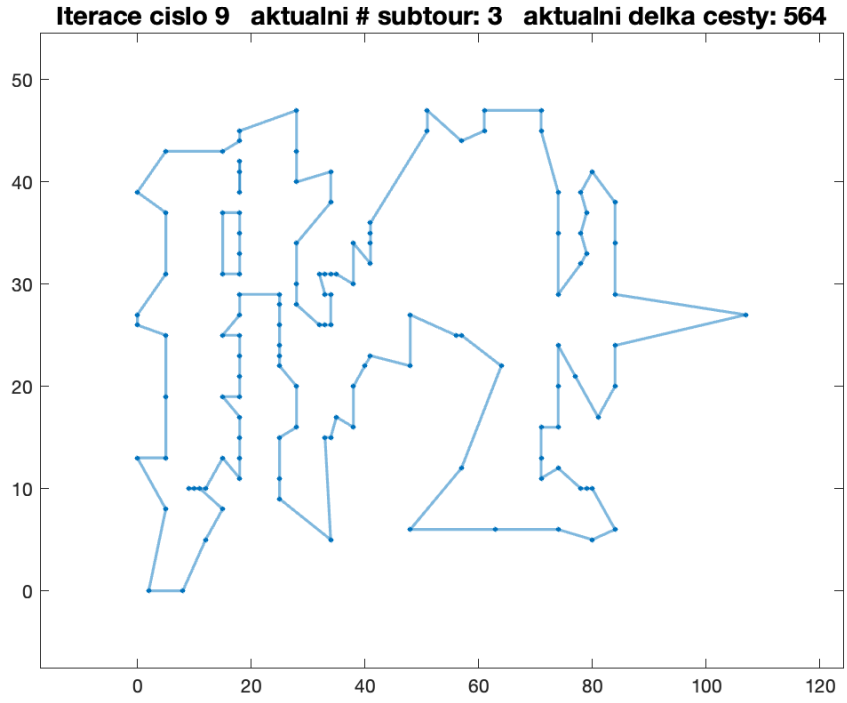
Obrázek 8: VI. iterace - 131 uzlů VLSI



Obrázek 9: VII. iterace - 131 uzlů VLSI



Obrázek 10: VIII. iterace - 131 uzlů VLSI



Obrázek 11: IX. iterace - 131 uzlů VLSI

Obrázek 12: X. iterace - 131 uzlů VLSI

odstranit.

Správnost algoritmu je ověřena porovnáním výsledku délky cesty 543, se správným výsledkem knihovny University of Waterloo (viz [3]).

Lze demostrovat klíčový rozdíl v náročnosti výpočtu kompletní formulace v podobě CLP a metody CLP relaxace. Pokud by algoritmus princip relaxace neobsahoval, bylo by nutné (jak je výše uvedeno) před výpočtem definovat celkem téměř 2^{131} řádků nerovností, které by zamezovaly vzniku subtours. Principem relaxace, tedy postupem, kdy jsou do úlohy omezení přídávána postupně a pouze ta, která jsou objevena jako strukturální porušení, bylo přidáno pouze 58 nerovnic a limitní hodnotě jsme se tak zdaleka ani nepřiblížili.

4.3.3. Analýza výpočetní náročnosti relaxační metody CLP

Lze testovat pro jak velké instance problému je možné používat daný algoritmus při zachování rozumné délky trvání výpočtu. Lze generovat náhodné množiny uzlů o různých počtech. Test algoritmu bude proveden na zařízení s procesorem Apple M1 (3.2 GHz), 8GB RAM paměti.

V následující tabulce je zachycen vývoj délky řešení v sekundách pro různě velké varianty problému TSP od 150 po 500 uzlů. Uzly byly generovány náhodně při zasasení různých seedů. Vzdálenosti stejně jako v případě řešení VLSI dat byly zaokrouhlovány na celé čísla.

Z dat v tabulce 3 lze pozorovat trend vývoje časové náročnosti výpočtu TSP relaxační metodou. Je evidentní, že časová náročnost má tendenci stoupat s rostoucím počtem uzů, pro které je daná úloha řešena.

V některých případech lze pozorovat velké výkyvy času řešení v rámci stejně velkých úloh. Ty lze pozorovat například u úlohy o 150 uzlech, kdy řešení úlohy při seedu rng(602) trvalo 10x déle než řešení stejně „velké“ úlohy

Počet uzlů	rng(123) (s)	rng(1234) (s)	rng(305) (s)	rng(602) (s)	Průměr (s)
150	24.97	5.20	5.99	51.26	21.86
185	25.26	7.14	14.30	10.96	14.42
235	14.24	37.56	37.18	28.57	29.39
280	220.41	46.30	22.76	56.32	86.45
327	332.11	103.26	122.65	291.90	212.48
367	201.18	339.37	661.50	106.50	327.14
380	232.51	345.15	241.15	182.07	250.22
390	331.30	301.51	123.43	116.45	218.17
405	251.82	261.66	285.34	173.99	243.20
413	530.98	464.99	91.09	428.81	378.97
435	235.36	359.51	213.37	270.80	269.76
440	826.02	1151.00	454.37	532.36	740.94
445	233.12	351.19	513.18	1110.80	552.07
455	315.92	223.87	420.21	1301.10	565.28
465	474.28	315.85	1240.40	235.98	566.63
470	200.68	832.09	544.24	712.15	572.29
485	565.35	832.00	796.35	305.50	624.80
495	414.62	842.03	1389.50	748.93	848.77
500	1230.97	743.26	2010.90	417.31	1100.37

Tabulka 3: Tabulka časů řešení náhodně generovaných úloh

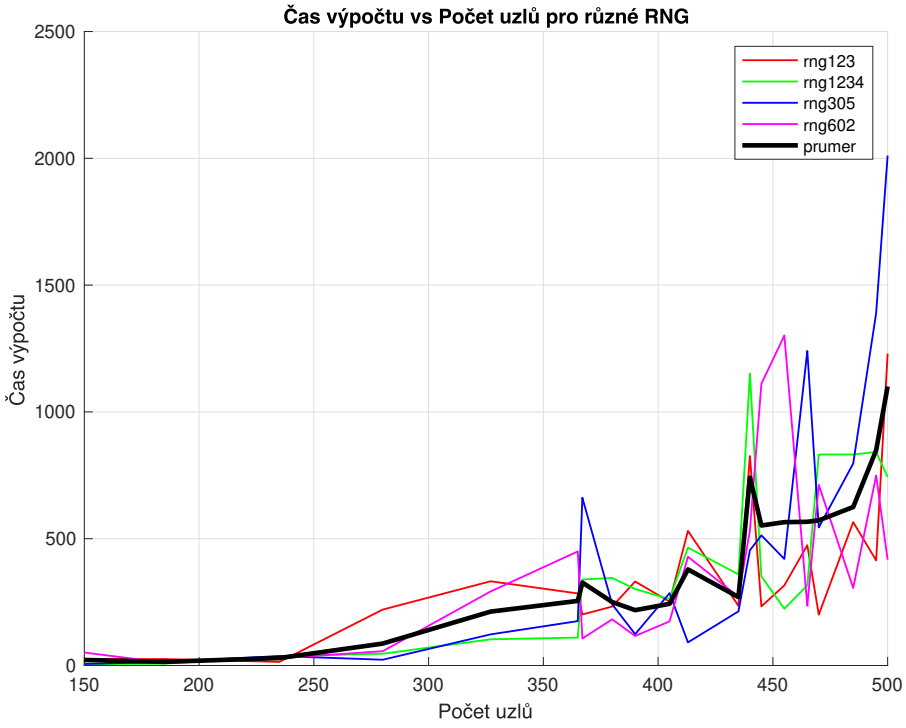
při seedu rng(1234). Na tomto lze demonstrovat, že časová náročnost výpočtu závisí i na samotné geometrii úlohy, tedy na vzájemné poloze uzlů, kdy změna pozic uzlů v rovině má vliv na náročnost výpočtu.

Při použití relaxační metody není vývoj časové náročnosti příliš překvapující. Jak bylo zmíněno výše, není možné přesně dopředu určit časovou náročnost, která je závislá na složitosti a počtu přidaných podmínek (podmínek pro zamezení vzniku suboutours) a poečet těchto podmínek se dá považovat za individuální pro každou instanci problému TSP. Například v uvedených instancích 150 a 185 uzlů klesl počet iterací z 15 na 9.

Je viditelné, že s rostoucím počtem uzlů mají úlohy tendenci být náročnější. Je zřejmé, že šance, že úloha o 150 uzlech bude řešena kratší dobu než úloha o 500 uzlech je veliká, nicméně nelze vyloučit, že by se nenašla taková instance problému, kdy by toto neplatilo a výpočet menší úlohy by byl skutečně náročnější než výpočet zdánlivě složitější - větší úlohy.

I v grafickém znázornění dat z tabulky 3 na obrázku 13 lze pozorovat jakýsi kolísavý vývoj časové náročnosti výpočtů a to při všech testovaných seedech. Nejsložitějším případem byl dle tabulky případ pro 500 uzlů při seedu 305, který trvalo vyřešit 33.52 minut. Vzhledem k omezeným možnostem procesoru nebyly provedeny testy na větších úlohách.

Černou linií v grafu je vyznačen vývoj zprůměrovaných časů pro úlohy danných rozsahů. I přes to, že ani při zprůměrování hodnot se nevyhneme velkým výkyvům ve vývoji časové náročnosti, lze pozorovat tendenci k exponencionálnímu růstu náročnosti.



Obrázek 13: Vývoj časové náročnosti výpočtu pro různé náhodné instance problému TSP

4.4. Srovnání CLP relaxace, dynamického programování a hrubé síly

Dynamické programování je z hlediska „nejhoršího, co může nastat“ tou nejlepší variantou, tj. má nejlepší známý horní odhad výpočetní složitosti ze všech metod (viz [1]) a to asymptosickou náročnost $O(n^2 \cdot 2^n)$. Ovšem jedná se jen o srovnání odhadů, skutečné chování metod pro konkrétní data mohou být jiná. V případě relaxačního přístupu k úloze CLP se například ani u jedné z testovaných instancí nestalo, že by musely být zavedeny podmínky pro zamezení vzniku všech subtours a řešení bylo nalezeno již při zavedení poměrně nízkého počtu těchto podmínek. Ovšem náročnost úlohy dynamického

Počet uzlů	DP (s)	BF (s)	CLP - relaxace (s)
5	0.018	0.003	0.180
8	0.022	0.009	0.113
11	0.120	2.734	0.115
12	0.24	33.97	0.25
14	1.245	-	0.098
17	14.101	-	0.112
20	155.998	-	0.108

Tabulka 4: Tabulka časů řešení přístupy DP, BF a relaxační metodou CLP

programování je pevně daná a její zmenšení není možné.

V tabulce 4 jsou srovnávány časy řešení v sekundách pro stejné úlohy o rozsazích uvedených v prvním sloupci, ve druhém sloupci jsou časy řešení úlohy dynamického programování, ve třetím časy pro metodu hrubé síly (*brute force* a ve čtvrtém sloupci jsou časy řešení úlohy řešenou relaxační metodou CLP.

Z časů lze potvrdit předchozí tvrzení. V málo rozsáhlých instancích problému není problém řešit TSP jako úlohy dynamického programování či hrubou silou, DP je v tomto případě dokonce i efektivnější než CLP, zpočátku i přístup hrubou silou.

Ovšem už při stále poměrně malých instancích problému se projevuje exponenicionálně rostoucí náročnost obou metod a ve srovnání s relaxační metodou CLP dynamické programování a hlavně hrubá síla hluboce zaostávají. Za nejefektivnější exaktní metodu řešení z testovaných metod tak lze považovat CLP relaxaci.

Zároveň lze pozorovat jak rychle hlavně hrubá síla naráží na své limity. Již při 13 uzlech paměťová náročnost výpočtu přesahuje 8GB a tak je 12 uzlů maximem, které je možné při dané konfiguraci počítače řešit. Dynamické programování na své maximum naráží při úloze o 28 uzlech a tak i z tohoto hlediska je opět relaxace celočíselného programování nejefektivnějším

způsobem řešení úlohy TSP, ta během testování na žádný paměťový limit nenarazila.

Závěr

Hlavním cílem práce bylo se seznámit s úlohou TSP, exaktními metodami prořešení této úlohy a následně vybrat z těchto metod nejefektivnější způsob řešení tohoto problému.

První kapitola se věnovala obecnému seznámení s problémem obchodního cestujícího, ve druhé kapitole byl čtenář seznámen s matematickými formulacemi problému v podobě celočíselného programování. Třetí kapitola se detailně věnovala využití přístupu CLP relaxace a byly popsány principy dynamického programování a metoda hrubé síly, které mohou být taktéž využity při řešení úlohy TSP. Poslední kapitola se věnovala programovému zpracování a demonstraci výpočetní náročnosti, jak z hlediska časové náročnosti výpočtu tak i paměťové náročnosti. Na konci této kapitoly byla z uvedených metod vybrána i ta nejefektivnější.

Téma bakalářské práce pro mě bylo přínosné primárně pro možnost využití práce v softwaru MATLAB a prohloubení znalostí v oblasti operačního výzkumu, kam řešení úlohy TSP spadá.

Literatura

[1] Applegate, D. L., Bixby, R. E., Chvátal, V., Cook, W. J.: *The Traveling Salesman Problem: A Computational Study* (Princeton Series in Applied Mathematics, 17), Princeton University Press, 2006.

[2] MathWorks: *intlinprog - Mixed-Integer Linear Programming*. [Online] Dostupné z: `https://uk.mathworks.com/help/optim/ug/intlinprog.html?s_tid=doc_ta`. [Citováno: 2. května 2024].

[3] Waterloo, University of: *VLSI Benchmarks - The Traveling Salesman Problem*. [Online] Dostupné z: `https://www.math.uwaterloo.ca/tsp/vlsi/index.html#XQF131`. [Citováno: 5. května 2024].

[4] GeeksforGeeks: *Breadth First Search (BFS) for a Graph*. [Online] Dostupné z: `https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/`. [Citováno: 5. května 2024].

[5] Flood, M. M.: *Merrill Flood (with Albert Tucker), Interview of Merrill Flood in San Francisco on 14 May 1984. The Princeton Mathematics Community in the 1930s, Transcript Number 11 (PMC11)*, Princeton University, 1984.

[6] Tobin, B., Hassett, C.: *The Travelling Salesman Problem [And How To Solve It]*. [Online] Dostupné z: `https://smartroutes.io/blogs/the-travelling-salesman-problem/`. [Citováno: 17. června 2024].

[7] Patel, A.: *Heuristics*. [Online] Dostupné z: `https://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html`. [Citováno: 17. června 2024].

[8] MathWorks, Inc.: *MATLAB. Version 9.11 (R2024a)*. [Online] Dostupné z: `https://www.mathworks.com/products/matlab.html`. [Citováno: 17. června 2024].

[9] Carnegie Mellon University: *15-451: Design and Analysis of Algorithms - Lecture 10: Dynamic Programming II*. [Online] Dostupné z: `https://www.cs.cmu.edu/~15451-s23/lectures/lec10-dp2.pdf`. [Citováno: 17. června 2024].

[10] Held, M., Karp, R. M.: *A dynamic programming approach to sequencing problems*. Journal of the Society for Industrial and Applied Mathematics, 10(1):196–210, 1962.

[11] OpenAI: *ChatGPT*. [Online] Dostupné z: https://www.openai.com/research/chatgpt. [Citováno: 17. června 2024].

[12] Laporte, G.: *The Traveling Salesman Problem: An overview of exact and approximate algorithms*. European Journal of Operational Research, 59:231–247, 1992. Elsevier Science Publishers B.V.

[13] OneToughMonkey: *TravellingSalesmanProblem: bruteforce.m*. [Online] Dostupné z: https://github.com/OneToughMonkey/TravellingSalesmanProblem/blob/master/bruteforce.m. [Citováno: 10. července 2024].

[14] Jiří Demel: *Grafy a jejich aplikace*. Vydal Jiří Demel, Libčice nad Vltavou. Sazba a obálka Jiří Demel. Vydání třetí, elektronické (vlastním nákladem druhé), 2019. Toto dílo podléhá licenci Creative Commons CC BY NC ND 4.0 — http://creativecommons.org/licenses/by-nc-nd/4.0/.