Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra matematiky

Diplomová práce

Optimalizace v přepravních úlohách

Plzeň 2019 Tina Bočkayová

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracovala samostatně a výhradně s použitím uvedených zdrojů.

V Plzni dne 30. července 2019 .....
Tina Bočkayová

Poděkování

Mé obrovské poděkování patří vedoucímu diplomové práce panu Doc. Ing. Romanu Čadovi, Ph.D. za odborné a cenné rady a připomínky a za pomoc s implementační částí.

Abstrakt

Diplomová práce je věnována základním úlohám vyskytujícím se v přepravní logistice. Jako první je zpracována úloha obchodního cestujícího a její rozšíření na úlohu vícera obchodních cestujících. Dále se podrobněji zabýváme problémem okružních jízd s kapacitami a úlohou plnění zásobníků, respektive úlohou o batohu. Jsou zpracovány matematické modely těchto úloh a metody jejich řešení. Je zaveden dynamický MILP model pro úlohu okružních jízd s kapacitami a pojem (λ, α)-optimality řešení. Také je popsán vztah řešení jednotlivých modelů pro úlohy, kde ocenění hran grafu splňuje trojúhelníkovou nerovnost.

Jsou otestovány standardně dostupné řešiče pro nekomerční využití na středně velkých modelových úlohách. Cílem je porovnat kvalitu řešení získaných pomocí těchto řešičů a případné doporučení některého z nich pro využití v praxi.

Abstract

The diploma thesis is dedicated to basic tasks occurring in transport logistics. The first is the traveling salesman problem and its extension to the multiple traveling salesman problem. Further we deal in more detail with vehicle routing problem with capacities and bin packing problem, respectively knapsack problem. Mathematical models of these problems and methods of their solution are worked out. We introduce MILP model for vehicle routing problem with capacities and a concept of (λ, α)-optimality of soluton. There is also described a relationship between solutions of models, where costs of edges meet the triangular inequality.

Standardly available solvers for non-commercial use on medium-sized model tasks are tested. The aim is to compare the quality of solutions obtained using these solvers and possible recommendations of any of them for use in practice.

Obsah

Prohlášení i

Poděkování ii

Abstrakt iii

Abstract iii

Obsah vi

1 Úvod 1

2 Historie 1

3 Zadefinování grafových pojmů 2

4 Lineární programování 4
4.1 Simplexový algoritmus 5
4.2 Dualita lineárního programování 9
4.3 Duální simplexová metoda 12
4.4 Metody vnitřního bodu 12
4.4.1 Primární centrální cesta 14
4.4.2 Duální centrální cesta 14
4.4.3 Primárně-duální centrální cesta 15
4.5 Výpočetní složitost LP 15

5 Celočíselné a smíšené lineární programování 16
5.1 Metoda větví a mezí (Branch and Bound) 17
5.2 Metoda větví a cen (Branch and Price) 18
5.2.1 Metoda generování sloupců 18
5.3 Metoda větví a řezů (Branch and Cut) 19
5.3.1 Gomoryho řezy 19
5.4 Výpočetní složitost ILP 21

6 Základní přepravní/dopravní problémy 21
6.1 Problém obchodního cestujícího 22
6.1.1 Dantzig-Fulkerson-Johnsonův model (1954) 23
6.1.2 Polynomiální Miller-Tucker-Zemlinův model (1960) . . 23
6.1.3 Gavish-Gravesův model (1978) 24
6.1.4 Symetrický problém obchodního cestujícího 25
6.2 Eukleidovský problém obchodního cestujícího 26

6.3 Problém vícero obchodních cestujících 26
6.4 Okružní dopravní problém 27

7 Převodové techniky mezi VRP a TSP **33**

8 Vztahy mezi úlohami v metrickém případě **34**
8.1 Vztah mezi TSP a m-TSP 34
8.2 Vztahy pro CVRP 35

9 Aproximační algoritmy **36**
9.1 Aproximační schéma 36

10 Řešení problémů pomocí heuristik **37**
10.1 Metoda nejbližšího souseda 37
10.2 Metoda Clarke Wrighta 37
10.3 Lin-Kernighanův algoritmus 38
10.3.1 (λ, α)-postoptimalizace 40
10.4 Evoluční metody s operátorem EAX 41

11 Uložení nákladu **42**
11.1 Úloha o batohu 42
11.1.1 Dynamické programování pro úlohu o batohu 43
11.2 FPTAS pro úlohu o batohu 44
11.3 Plnění zásobníků 45
11.3.1 1D problém 46
11.3.2 2D problém 46
11.3.3 3D problém 47

12 Dynamické modely přepravních úloh **47**
12.1 Odložená omezení 47
12.2 Dynamický MILP model pro úlohu CVRP 48

13 Software pro řešení dopravních optimalizačních úloh **49**
13.1 Gurobi Optimization 50
13.2 CPLEX Optimizer 51
13.3 MOSEK 51
13.4 LKH Solver 51
13.5 Concorde TSP Solver 52

14 Praktické výsledky **52**
14.1 TSP 53
14.1.1 Velikost 37 54

14.1.2 Velikost 74 54
14.1.3 Velikost 111 55
14.1.4 Velikost 148 55
14.1.5 Velikost 185 56
14.1.6 Velikost 222 56
14.1.7 Velikost 362 57
14.1.8 Velikost 532 57
14.1.9 Komentář k výsledkům 57
14.2 m-TSP 58
14.2.1 Velikost 37 59
14.2.2 Velikost 74 59
14.2.3 Velikost 111 60
14.2.4 Velikost 148 60
14.2.5 Velikost 185 60
14.2.6 Velikost 222 61
14.2.7 Velikost 362 61
14.2.8 Komentář k výsledkům 61
14.3 CVRP 62
14.3.1 Velikost 37 63
14.3.2 Velikost 74 63
14.3.3 Velikost 111 63
14.3.4 Velikost 148 63
14.3.5 Velikost 185 64
14.3.6 Velikost 222 64
14.3.7 Velikost 362 65
14.3.8 Komentář k výsledkům 65
14.4 Poznámky k použitým řešičům 66
14.5 Interpretace proměnných z modelu 67
14.6 2D BPP 68

15 Závěr **69**

1 Úvod

Zaměříme se na úlohy, které jsou spojeny s přepravní logistikou, a to hlavně na okružní dopravní problém s kapacitami (CVRP, Capacity Vehicle Routing Problem), s kterým souvisí úloha plnění zásobníků (BPP, Bin Packing Problem). K úloze CVRP dojdeme hierarchicky v návaznosti na úlohy obchodního cestujícího (TSP, travelling salesman problem) a vícera obchodních cestujících (m-TSP, multiple travelling salesman problem). Stejně tak úloha BPP bude navazovat na úlohu o batohu (KP, Knapsack Problem).

V prvních několika sekcích se budeme zabývat teorií potřebnou pro řešení vybraných problémů. Jedná se o pojmy z teorie grafů, popis lineárního, celočíselného a smíšeného lineárního programování a popis metod, kterými lze dané úlohy řešit.

Poté přímo popíšeme modely pro základní přepravní úlohy (TSP, m-TSP a CVRP). Ukážeme možný převod CVRP, m-TSP na TSP a rovněž vztah optimálních řešení těchto problémů. Vzhledem ke složitosti přímých metod budou zmíněny i některé z nejznámějších aproximačních, heuistických a evolučních algoritmů. Také je určen vztah mezi optimálním řešením TSP, m-TSP a CVRP, pokud jsou řešeny na stejném grafu, jehož ohodnocení splňuje trojúhelníkovou nerovnost. K pojmu λ-optimality, která je určující pro TSP, zavedeme pojem (λ, α)-optimality pro CVRP. Jako poslední bude zpracována úloha KP a BPP. Pro KP popíšeme princip dynamického programování a úlohu BPP rozdělíme na 1D, 2D a 3D problémy.

Vybrané modely jsou implementovány a jejich řešení je otestováno pomocí dostupného softwaru. Otestujeme je na menších a středně velkých úlohách, které se podobají problémům řešeným ve středně velkých firmách. Vzhledem k využívání převážně softwaru pro nekomerční využití, bude hlavním cílem implementační části posouzení vhodnosti jednotlivých přístupů a metod pro velikosti úloh (počet zákazníků, vozidel, kapacity, požadavky) vyskytujících se při reálném denním provozu a nikoli přímé řešení reálné úlohy.

2 Historie

Lidé už kdysi přišli na to, že ne každá cesta k dosažení cíle je ta správná nebo nejrychlejší. Popisem optimalizace se více začali zajímat v druhé polovině 18. století. Mezi první slavná jména spojená s hledáním optimálního řešení patří např. J. B. J. Fourier nebo C. F. Gauss, jehož eliminační algoritmus je všem v oboru dobře znám. Jako další lze zmínit H. Minkowského, který se zabýval studiem konvexních množin, které mají v lineárním programování velkou roli.

Velkému zájmu se lineární programování začalo těšit po druhé světové

válce, kde bylo zapotřebí co nejrychleji vzpamatovat firmy a podniky, aby se obnovily dodávky všeho potřebného. Při vývojích optimalizačních postupů pro americké letectvo vyvinul -otec lineárního programování- G. B. Dantzig v roce 1947 známý simplexový algoritmus. Podrobněji je historie zpracována v článku [25].

Byť se tento algoritmus ukázal efektivní, tak v roce 1972 Klee a Minty na několika příkladech ukázali, že v nejhorším případě může k nalezení optimálního řešení vést exponenciální počet iterací, tj. nejedná se o polynomiální algoritmus. Ostatní matematici nezaháleli a snažili se přijít s lepšími algoritmy. V roce 1979 byl představen Chačianův elipsoidový algoritmus, který je polynomiální, ale v praxi je stále rychlejší simplexový algoritmus. Další vylepšení přišlo v roce 1984, kdy Karmarkar uvedl polynomiální algoritmus pod názvem metoda vnitřního bodu, viz [33].

Optimalizovat se dá v řadě odvětví, simplexový algoritmus byl např. testován na tzv. dietním problému, který řeší, z jakých surovin poskládat stravu, aby obsahovala potřebné množství živin za co nejmenší cenu [25]. Myšlenka maximalizace zisku či minimalizace nákladů za určitých podmínek se dá aplikovat také v dopravních problémech, na které se v této práci zaměříme.

Nejvíce prozkoumanou a studovanou úlohou je problém obchodního cestujícího, který byl už v 18. století studován irským matematikem Sirem W. R. Hamiltonem a britským matematikem T. P. Kirkmanem. Od té doby byla sepsána celá řada článků a odborných textů, viz [29].

Z problému obchodního cestujícího byla časem odvozena řada úloh, známá pod názvem okružní dopravní problémy, na kterých je i dnes postavena optimalizace v logistickém plánování. První článek o "truck dispatchingu"byl publikován v roce 1959 Dantzigem a Ramserem [4].

Spolu s přesnými algoritmy, které jsou povětšinou pomalé, obzvlášť při řešení rozsáhlých problémů, byly vyvíjeny heuristické algoritmy. Ty sice nejsou zárukou nalezení optimálního řešení, ale jsou rychlé, a pokud naleznou nějaké řešení, nebývá daleko od optima. Nejvíce využívanými jsou metoda Clarke-Wrighta z roku 1964 a Lin-Kernighanův algoritmus, který vychází z λ-optimalizace z roku 1965 [4]. Z trochu jiné oblasti lze zmínit evoluční algoritmy, které jsou inspirovány biologickými principy. Jejich vznik můžeme datovat do 60. let, kdy se jimi začal zabývat J. Holland (1962) [1].

3 Zadefinování grafových pojmů

Definice 1. Necht' V je konečná množina. Dvojice $G = (V(G); H(G))$, kde $H(G) \subseteq (V(G) \times V(G)) \cup \binom{V(G)}{2}$ se nazývá smíšený graf. Je-li speciálně $H(G) \subseteq V(G) \times V(G)$, potom G nazveme orientovaným grafem, pokud

$H(G) \subseteq \binom{V(G)}{2}$, pak se G nazývá neorientovaný (prostý) graf. Prvky množiny $V(G)$ budeme nazývat vrcholy (uzly), prvkům množiny $H(G)$ budeme říkat hrany grafu G.

Poznámka 1. Výraz $\binom{V(G)}{2}$ v předchozí definici značí množinu všech možných různých dvojic vrcholů z množiny V.

Definice 2. Symetrizací orientovaného grafu $\vec{G}$ nazveme neorientovaný graf G, kde $V(G) = V(\vec{G})$ a $H(G) = \{\{x, y\}; (x, y) \in H(\vec{G}\}$.

Definice 3. Nechť G je neorientovaný graf, $v \in V(G)$. Potom číslo

$$d_G(v) = |\{u \in V(G); \{v, u\} \in H(G)\}|$$

se nazývá stupeň vrcholu v v grafu G.

Definice 4. Nechť $\vec{G}$ je orientovaný graf, $x \in V(\vec{G})$. Vstupním stupněm vrcholu x v $\vec{G}$ nazveme číslo $d^+_{\vec{G}}(x) = |\{v \in V(\vec{G}); (v, x) \in H(\vec{G})\}|$, výstupním stupněm vrcholu x v $\vec{G}$ nazveme číslo $d^-_{\vec{G}}(x) = |\{v \in V(\vec{G}); (x, v) \in H(\vec{G})\}|$.

Definice 5. Pravidelným grafem stupně k nazveme graf, ve kterém jsou všechny vrcholy stupně k.

Definice 6. (Speciální grafy)
Úplný graf: $G = \left(\{1, 2, \ldots, n\}, \binom{\{1,2,\ldots,n\}}{2}\right)$.
Kružnice délky $n \geq 3$: $C = (\{1, 2, \ldots, n\}, \{\{i, i+1\}, i = 1, \ldots, n-1\} \cup \{\{n, 1\}\})$.
Cesta délky $n \geq 0$: $P = (\{1, 2, \ldots, n+1\}, \{\{i, i+1\}, i = 1, \ldots, n\})$.

Definice 7. Nechť G_1, G_2 jsou grafy. Řekneme, že G_1 je podgrafem grafu G_2, značíme $G_1 \subseteq G_2$, jestliže $V(G_1) \subseteq V(G_2)$ a současně $H(G_1) \subseteq H(G_2)$. Řekneme, že G_1 je faktorem grafu G_2, jestliže $V(G_1) = V(G_2)$ a $H(G_1) \subseteq H(G_2)$.

Definice 8. Graf G je souvislý právě tehdy, když v G existuje cesta mezi každými dvěma vrcholy $x, y \in G$.

Definice 9. Souvislý graf, který neobsahuje jako podgraf žádnou kružnici, se nazývá strom.

Definice 10. Faktor grafu G, který je stromem, se nazývá kostra grafu G.

Definice 11. Nechť $\vec{G}$ je orientovaný graf. Funkce $c : H(\vec{G}) \to (0, \infty)$ se nazývá hranové ohodnocení grafu $\vec{G}$. Graf se zadaným ohodnocením se nazývá ohodnocený graf.

Definice 12. Nechť $\vec{G}$ je orientovaný graf, c jeho hranové ohodnocení. Potom matici $C(\vec{G})$ definovanou předpisem

$$c_{ij} = \begin{cases} c(i,j) & \text{pokud } (i,j) \in H(\vec{G}), \\ 0 & \text{jinak,} \end{cases}$$

nazveme váženou maticí sousednosti nebo také cenovou maticí grafu $\vec{G}$.

Poznámka 2. Pro neorientovaný graf definujeme ohodnocení hran i cenovou matici stejným způsobem.

Definice 13. (Δ-nerovnost)
Mějme úplný ohodnocený neorientovaný graf G s ohodnocením c. Pokud pro každou trojici vrcholů i, j, k platí

$$c(i,j) + c(j,k) \geq c(i,k),$$

pak ohodnocení c splňuje Δ-nerovnost.

Definice 14. (Hamiltonovská kružnice)
Podgraf K grafu G s n uzly je hamiltonovskou kružnicí v G, právě tehdy, když platí, že K má n uzlů, n hran, je souvislý a je pravidelným grafem stupně 2.

4 Lineární programování

Důležitým a užitečným nástrojem pro řešení optimalizačních úloh, jako je optimalizace výrobních plánů, míšení surovin a optimalizace dopravních problémů, kterými se budeme zabývat, je lineární programování, anglicky Linear Programming, dále LP. V takových úlohách řešíme maximalizaci (zisku) nebo minimalizaci (nákladů) lineární účelové funkce za podmínek, které jsou dané soustavou lineárních nerovnic. Můžeme se setkat s několika různými tvary úloh LP, viz tabulka 1. Kde $\mathbf{c} \in \mathbb{R}^n$ je vektor cen jednotlivých proměnných

Kanonický tvar		Standardní tvar	
min $\mathbf{c}^T\mathbf{x}$	max $\mathbf{c}^T\mathbf{x}$	min $\mathbf{c}^T\mathbf{x}$	max $\mathbf{c}^T\mathbf{x}$
$\mathbf{Ax} \geq \mathbf{b}$	$\mathbf{Ax} \leq \mathbf{b}$	$\mathbf{Ax} = \mathbf{b}$	$\mathbf{Ax} = \mathbf{b}$
$\mathbf{x} \geq \mathbf{0}$	$\mathbf{x} \geq \mathbf{0}$	$\mathbf{x} \geq \mathbf{0}$	$\mathbf{x} \geq \mathbf{0}$

Tabulka 1: Různé tvary úloh LP.

z vektoru $\mathbf{x}$, matice $\mathbf{A} \in \mathbb{R}^{m,n}$ je matice podmínek typu $m \times n$ a $\mathbf{b} \in \mathbb{R}^m$ je

vektor pravých stran. Vektor proměnných $\mathbf{x}$ požadujeme nezáporný, protože vyjadřuje množství popřípadě vzdálenost, a ta logicky nemůže být záporná. V praxi se samozřejmě můžeme setkat s úlohami, kde jsou podmínky tvořeny rovnostmi i nerovnostmi. Dá se ukázat, že uvedené formulace jsou navzájem ekvivalentní, viz např. [24].

Úlohu LP lze řešit graficky, kde jednotlivé podmínky, řádky soustavy $\mathbf{Ax} \lesseqgtr \mathbf{b}$, ohraničují přípustnou množinu řešení. V praxi je tato metoda použitelná pouze pro úlohy se dvěma proměnnými, se třemi proměnnými se dá zobrazit ve 3D grafu, ale je to nepřehledné. Ostatní úlohy s více proměnnými graficky řešit nelze. Proto se úlohy LP nejčastěji řeší známým simplexovým algoritmem, nebo metodami vnitřního bodu. Pro pokročilejší techniky se v LP využívá následující věta.

Věta 1. (Minkowského věta)
Pro každou soustavu $\mathbf{Ax} \leq \mathbf{b}$ existují vektory $\mathbf{v}^1, \mathbf{v}^2, \ldots, \mathbf{v}^M$ a $\mathbf{w}^1, \mathbf{w}^2, \ldots, \mathbf{w}^N$ s následující vlastností: vektor $\mathbf{x}^*$ vyhovuje $\mathbf{Ax}^* \leq \mathbf{b}$ právě tehdy, když

$$\mathbf{x}^* = \sum_{r=1}^{M} p_r \mathbf{v}^r + \sum_{s=1}^{N} q_s \mathbf{w}^s,$$

kde $p_1, p_2, \ldots, p_M$ a $q_1, q_2, \ldots, q_N$ jsou nezáporná čísla a platí $\sum p_r = 1$.

4.1 Simplexový algoritmus

Popis algoritmu je založený na textu přednášek [28] a publikaci [38], kde se lze dočíst více informací. Uvažujme tedy úlohu minimalizace LP ve standardním tvaru, viz tabulka 1,

$$\begin{aligned} \min \mathbf{c}^T\mathbf{x} \\ \mathbf{Ax} &= \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0}, \end{aligned}$$

kde $\mathbf{A}$ je typu $m \times n$ a platí $m < n$. Dále budeme vždy předpokládat, že hodnost matice $\mathbf{A}$ je rovna m (tento předpoklad prakticky nesnižuje obecnost).

Definice 15. Je-li $\mathcal{B} = \{A_{j1}, A_{j2}, \ldots, A_{jm}\}$ množina m lineárně nezávislých sloupců matice $\mathbf{A}$, pak vektor $\mathbf{x} \in \mathbb{R}^n$, splňující podmínky

- $x_j = 0$ pro $A_j \notin \mathcal{B}$,
- x_k je k-tá složka vektoru $\mathbf{B}^{-1}\mathbf{b}$, kde $\mathbf{B} = [A_{j_i}]_{i=1}^m$ pro $k = 1, \ldots, m$,

se nazývá bazické řešení příslušné množině $\mathcal{B}$.

Poznámka 3. Bazické řešení dostaneme takto:

1. zvolíme množinu $\mathcal{B}$ lineárně nezávislých sloupců matice $\mathbf{A}$, (tj. některou bázi sloupcového prostoru matice $\mathbf{A}$),
2. položíme rovny nule všechny složky vektoru $\mathbf{x}$ odpovídající sloupcům, které nejsou v množině $\mathcal{B}$,
3. řešíme vzniklou soustavu rovnic s regulární maticí (jež má právě jedno řešení).

Definice 16. Bazické řešení, pro které platí $x_j \geq 0$, $j = 1, \ldots, n$, se nazývá přípustné bazické řešení.

Dá se dokázat následující věta, viz [38].

Věta 2. Optimální řešení úlohy LP je některé z jejích přípustných bazických řešení.

Soustavu můžeme do simplexové tabulky zapsat tak, aby prvních m sloupců tvořilo bázi sloupcového prostoru matice $\mathbf{A}$, tj. ve tvaru:

$$\left[\begin{array}{cccccccc} \tilde{c}_1 & \tilde{c}_2 & \ldots & \tilde{c}_m & \tilde{c}_{m+1} & \ldots & \tilde{c}_n & \tilde{z} \\ \hline a_{11} & a_{12} & \ldots & a_{1m} & a_{1m+1} & \ldots & a_{1n} & b_1 \\ a_{21} & a_{22} & \ldots & a_{2m} & a_{2m+1} & \ldots & a_{2n} & b_2 \\ \vdots & \vdots & \ldots & \vdots & \vdots & \ldots & \ldots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mm} & a_{mm+1} & \ldots & a_{mn} & b_m \end{array}\right],$$

kde první řádek je tzv. nultý řádek, ve kterém $\tilde{c}_1, \ldots, \tilde{c}_n$ označují relativní ceny a $\tilde{z}$ aktuální hodnotu účelové funkce. Více si o nich řekneme později.

Pomocí Gauss-Jordanovy eliminace získáme konkrétní hodnoty bazického řešení, $\mathbf{x}_B = [x_1^1, x_2^1, \ldots, x_m^1, 0, \ldots, 0]^T$. Soustavu tak upravíme do následujícího tvaru:

$$\left[\begin{array}{cccccccc} \tilde{c}_1^1 & \tilde{c}_2^1 & \ldots & \tilde{c}_m^1 & \tilde{c}_{m+1}^1 & \ldots & \tilde{c}_n^1 & \tilde{z}^1 \\ \hline 1 & 0 & \ldots & 0 & a_{1m+1}^1 & \ldots & a_{1n}^1 & x_1^1 \\ 0 & 1 & \ldots & 0 & a_{2m+1}^1 & \ldots & a_{2n}^1 & x_2^1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ldots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & a_{mm+1}^1 & \ldots & a_{mn}^1 & x_m^1 \end{array}\right].$$

Přechodu k jinému přípustnému bazickému řešení dosáhneme výměnnou některé bazické proměnné za jinou, nebazickou. Otázkou zůstává, kterou proměnnou z báze vyjmout, a kterou tam naopak vložit.

Začneme otázkou, jakou proměnnou se vyplatí vložit do báze. Aby byl algoritmus efektivní, chceme v každém kroku dosáhnout snížení hodnoty účelové funkce, označme ji $z = c_1x_1 + c_2x_2 + \cdots + c_nx_n$. Pro zvolené bazické řešení $\mathbf{x}_B$ má účelová funkce hodnotu $z_0 = c_1x_1 + c_2x_2 + \cdots + c_mx_m$. Pokud bychom přiřadili proměnným $x_{m+1}, x_{m+2}, \ldots, x_n$ libovolné hodnoty, můžeme z tvaru soustavy po Gauss-Jordanově eliminaci vyjádřit bazické proměnné jako:

$$\begin{aligned} x_1 &= x_1^1 - \sum_{j=m+1}^{n} a_{1j}^1 x_j, \\ x_2 &= x_2^1 - \sum_{j=m+1}^{n} a_{2j}^1 x_j, \\ &\vdots \\ x_m &= x_m^1 - \sum_{j=m+1}^{n} a_{mj}^1 x_j. \end{aligned} \tag{1}$$

Dosazením (1) do rovnice pro cílovou funkci získáme výraz

$$z = z_0 - (z_{m+1} - c_{m+1})x_{m+1} - \ldots - (z_n - c_n)x_n, \tag{2}$$

kde

$$z_j = a_{1j}^1 c_1 + a_{2j}^1 c_2 + \cdots + a_{mj}^1 c_m, \quad m+1 \leq j \leq n. \tag{3}$$

Je-li některý z výrazů $(z_j - c_j)$, $j \in \{m+1, \ldots, n\}$, kladný, pak se s rostoucí hodnotou $x_j > 0$ snižuje hodnota účelové funkce.

Pravidlo 1. Jestliže pro q-tý sloupec platí nerovnost $z_q - c_q > 0$, pak q-tou proměnnou zahrneme do nové báze.

Hodnota $z_q - c_q$ se nazývá relativní cena q-té nebazické proměnné vzhledem k dané bázi.

Už víme, kterou proměnnou je vhodné přidat do báze, a ptáme se, za jakou z bazických proměnných ji vyměníme. Označíme si sloupce matice $\mathbf{A}$ jako $\mathbf{a}_s$, $s = 1, 2, \ldots, n$. Pro bazické řešení $\mathbf{x}_B$ platí

$$x_1\mathbf{a}_1 + x_2\mathbf{a}_2 + \cdots + x_m\mathbf{a}_m = \mathbf{b}, \tag{4}$$

kde $\mathbf{b} = [x_1^1, x_2^1, \ldots, x_m^1]^T$. Řekněme, že jsme si za vstupní proměnnou vybrali x_q, $q > m$. Příslušný sloupec $\mathbf{a}_q$ lze vyjádřit jako lineární kombinaci bazických vektorů:

$$\mathbf{a}_q = a_{1q}^1 \mathbf{a}_1 + a_{2q}^1 \mathbf{a}_2 + \cdots + a_{mq}^1 \mathbf{a}_m. \tag{5}$$

Přenásobením vztahu (5) proměnnou $\varepsilon \geq 0$ a odečtením od (4) dostaneme

$$(x_1 - \varepsilon a_{1q}^1)\mathbf{a}_1 + (x_2 - \varepsilon a_{2q}^1)\mathbf{a}_2 + \cdots + (x_m - \varepsilon a_{mq}^1)\mathbf{a}_m + \varepsilon \mathbf{a}_q = \mathbf{b}. \tag{6}$$

Vektor $\mathbf{b}$ je pro libovolné $\varepsilon \geq 0$ lineární kombinací nanejvýš $m + 1$ vektorů. Pro $\varepsilon = 0$ obdržíme původní bazické řešení. Pro dostatečně malé kladné ε dává (6) přípustné, ale nikoli bazické řešení. Hodnota koeficientů $(x_j - \varepsilon a_{jq})$ roste nebo klesá v závislosti na hodnotě ε. Pokud některé hodnoty koeficientů klesají, můžeme ε nastavit na první hodnotu, při které se jeden nebo více koeficientů vynulují, tj.

$$\varepsilon = \min_i \left\{ \frac{x_i}{a_{iq}^1}; a_{iq}^1 > 0 \right\}. \tag{7}$$

Takto získáme nové bazické řešení s vektorem $\mathbf{a}_q$, kterým jsme nahradili vektor $\mathbf{a}_p$, kde p je hodnota indexu minima (7). Tím dostáváme, že proměnná x_q se dostane do báze a nahradí tak proměnnou x_p.

Pravidlo 2. Zahrnujeme-li do nové báze proměnnou x_q, tak jí nahradíme tu bazickou proměnnou x_p, pro kterou je výraz $\frac{x_p}{a_{pq}^1}$ minimální nezáporný.

Prvku a_{pq}^1 říkáme pivot a upravíme podle něj simplexovou tabulku tak, že koeficient x_q je v p-tém řádku roven 1 a ostatní prvky v q-tém sloupci jsou nulové:

$$\begin{cases} \tilde{c}_j^2 = \tilde{c}_j^1 - \dfrac{\tilde{c}_j^1}{a_{pq}^1} \tilde{c}_q^1, \\ a_{ij}^2 = a_{ij}^1 - \dfrac{a_{pj}^1}{a_{pq}^1} a_{iq}^1 \quad i \neq p, \\ a_{pj}^2 = \dfrac{a_{pj}^1}{a_{pq}^1}. \end{cases}$$

Pomocí těchto pravidel upravujeme simplexovou tabulku, dokud nedosáhneme optima, nebo nezjistíme, že úloha nemá řešení.

Věta 3. Jestliže pro všechny sloupce platí $z_q - c_q \leq 0$, pak je toto přípustné bazické řešení optimální.

Mohou nastat případy, kdy zjistíme, že řešení není jediné, že úloha nemá řešení, nebo že se pohybujeme v kruhu. Všechny tyto případy se dají rozpoznat ze simplexové tabulky. Pokud existuje více optimálních řešení, tak mají stejnou hodnotu účelové funkce.

Tvrzení 1. Je-li v konečné simplexové tabulce relativní cena u některé nebazické proměnné nulová, pak má úloha LP víc než jedno optimální řešení. Pro všechna optimální řešení pak platí, že mají stejnou hodnotu účelové funkce.

Úloha je neřešitelná, pokud je množina přípustných řešení neomezená ve směru optimalizace, tj. není omezená zdola v případě minimalizace a naopak u maximalizace.

Tvrzení 2. Jsou-li všechny výrazy $\frac{x_i}{a_{iq}}$ záporné, pak funkce $\mathbf{c}^T\mathbf{x}$ není zdola omezená a úloha LP nemá řešení.

V případě, že se simplexový algoritmus zacyklí a nastává problém s jeho konečností, tak říkáme, že úloha je degenerovaná. Degenerovanost úlohy se dá rozpoznat z tvaru přípustného bazického řešení, neboli podle pravé strany v simplexové tabulce.

Tvrzení 3. Přípustné bazické řešení $\mathbf{x}_B$ se nazývá nedegenerované, pokud je každá bazická složka tohoto řešení nenulová. V opačném případě říkáme, že přípustné bazické řešení je degenerované. Úloha LP se nazývá degenerovaná, pokud má alespoň jedno degenerované přípustné bazické řešení, jinak se tato úloha nazývá nedegenerovaná.

Věta 4. Pro nedegenerované úlohy je základní simplexový algoritmus konečný.

Že je úloha degenerovaná, neznamená, že je neřešitelná. K řešení těchto úloh slouží anticyklické metody, mezi něž patří např.:

- Perturbační metoda, kde provedeme malou změnu parametrů, která má za následek, že úloha přestane být degenerovaná.
- Metoda nejmenších indexů, kde vybíráme sloupce a řádky ne podle hodnoty ale podle indexu v tabulce.

4.2 Dualita lineárního programování

Zajímavou otázkou při řešení úloh lineárního programování je, zda můžeme stanovit horní, respektive dolní, odhad optimální hodnoty účelové funkce pro úlohu maximalizace, respektive minimalizace.

Uvažujme úlohu minimalizace v kanonickém tvaru z tabulky 1.

$$\min c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &\geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &\geq b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &\geq b_m \end{aligned}$$

Horní hranicí hodnoty účelové funkce lze nazvat každé uhádnuté přípustné řešení, tj. vektor $\mathbf{x} = [x_1, \ldots, x_n]^T$, jehož složky jsou nezáporná reálná čísla a vyhovují všem zadaným podmínkám. Jak ale určit dobrou dolní hranici?

Tím, že jsou podmínky nerovnosti typu $\geq$, a pokud by pro některý řádek $i \in \{1, 2, \ldots, m\}$ platilo $a_{ij} \leq c_j$, $j = 1, 2, \ldots, n$, dala by se za dolní hranici považovat pravá strana dané nerovnosti. Nejedná se ale o dobrý odhad. Mnohem lepší odhad získáme lineární kombinací všech nerovností.

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &\geq b_1 \quad / \cdot y_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &\geq b_2 \quad / \cdot y_2 \\ &\vdots \\ \underline{a_{m1}x_1 + +a_{m2}x_2 + \cdots a_{mn}x_n} &\underline{\geq b_m \quad / \cdot y_m} \end{aligned}$$

$$x_1 \underbrace{(a_{11}y_1 + \cdots + a_{m1}y_m)}_{\leq c_1} + \cdots + x_n \underbrace{(a_{1n}y_1 + \cdots + a_{mn}y_m)}_{\leq c_n} \geq \underbrace{b_1y_1 + \cdots + b_my_m}_{\to \max}$$

Myšlenka zůstává stejná, porovnáváme koeficienty u jednotlivých x_j, $j = 1, \ldots, n$, a hledáme násobky jednotlivých řádků soustavy $y_i \geq 0$, $i = 1, \ldots, m$, tak, že $(a_{1j}y_1 + \cdots + a_{mj}y_m) \leq c_j$. Lineární kombinace pravé strany pak dává dolní odhad hodnoty účelové funkce. Aby byl nejlepší, chceme určit největší z dolních odhadů, tj. dostáváme úlohu maximalizace lineární kombinace pravé strany za podmínek daných porovnáním koeficientů proměnných x_j.

$$\max b_1y_1 + b_2y_2 + \cdots + b_my_m$$

$$
\begin{aligned}
a_{11}y_1 + a_{21}y_2 + \cdots + a_{m1}y_m &\leq c_1 \\
a_{12}y_1 + a_{22}y_2 + \cdots + a_{m2}y_m &\leq c_2 \\
&\vdots \\
a_{1n}y_1 + a_{2n}y_2 + \cdots + a_{mn}y_m &\leq c_n
\end{aligned}
$$

Úlohu minimalizace nazveme primární úlohou a získanou úlohu maximalizace úlohou k ní duální. Pokud bychom hledali úlohu duální k duální úloze, obdržíme původní primární úlohu. V případě, že primární úloha je ve standardním tvaru, podmínky jsou tvaru rovností, je možné uvažovat $y_i \in \mathbb{R}$. Tvary těchto úloh jsou v tabulce 2.

Platí tedy, že každé přípustné řešení maximalizační úlohy je dolní odhad optima minimalizační úlohy, naopak každé přípustné řešení minimalizační úlohy je horní odhad optima maximalizační úlohy. Z toho lze odvodit, že pokud najdeme přípustné řešení primární úlohy $\mathbf{x}$ a přípustné řešení duální úlohy $\mathbf{y}$ a hodnoty účelových funkcí se budou shodovat, jedná se o řešení optimální.

Kanonický tvar		Standardní tvar	
Primární	Duální	Primární	Duální
$\min\ \mathbf{c}^T\mathbf{x}$ $\mathbf{Ax} \geq \mathbf{b}$ $\mathbf{x} \geq \mathbf{0}$	$\max\ \mathbf{b}^T\mathbf{y}$ $\mathbf{A}^T\mathbf{y} \leq \mathbf{c}$ $\mathbf{y} \geq \mathbf{0}$	$\min\ \mathbf{c}^T\mathbf{x}$ $\mathbf{Ax} = \mathbf{b}$ $\mathbf{x} \geq \mathbf{0}$	$\max\ \mathbf{b}^T\mathbf{y}$ $\mathbf{A}^T\mathbf{y} \leq \mathbf{c}$ $\mathbf{y} \in \mathbb{R}^m$
$\max\ \mathbf{c}^T\mathbf{x}$ $\mathbf{Ax} \leq \mathbf{b}$ $\mathbf{x} \geq \mathbf{0}$	$\min\ \mathbf{b}^T\mathbf{y}$ $\mathbf{A}^T\mathbf{y} \geq \mathbf{c}$ $\mathbf{y} \geq \mathbf{0}$	$\max\ \mathbf{c}^T\mathbf{x}$ $\mathbf{Ax} = \mathbf{b}$ $\mathbf{x} \geq \mathbf{0}$	$\min\ \mathbf{b}^T\mathbf{y}$ $\mathbf{A}^T\mathbf{y} \geq \mathbf{c}$ $\mathbf{y} \in \mathbb{R}^m$

Tabulka 2: Dualita LP.

Věta 5. (Slabá věta o dualitě)
Jestliže $\mathbf{x} = [x_1, \ldots, x_n]^T$ a $\mathbf{y} = [y_1, \ldots, y_m]^T$ jsou příslušná přípustná řešení primární a duální úlohy, pak platí $\mathbf{c}^T\mathbf{x} \geq \mathbf{b}^T\mathbf{y}$.

Věta 6. (Silná věta o dualitě)
Primární úloha má konečné optimální řešení $\mathbf{x}^* = [x_1^*, \ldots, x_n^*]^T$ právě tehdy, když duální úloha má konečné optimální řešení $\mathbf{y}^* = [y_1^*, \ldots, y_m^*]^T$. Pro optimální řešení platí $\mathbf{c}^T\mathbf{x}^* = \mathbf{b}^T\mathbf{y}^*$.

Více se lze dočíst v [36] nebo[38].

4.3 Duální simplexová metoda

Při sestavování nebo výpočtu simplexové tabulky může nastat situace, že nultý řádek neobsahuje žádné kladné číslo (nelze vybrat pivot), ale získané řešení není přípustné, tj. obsahuje záporný prvek. Řešení není přípustné pro primární úlohu, ale je duálně přípustné. Než sestavovat novou tabulku pro duální úlohu, je lepší použít duální simplexovou metodu,viz [38], která pracuje s tabulkou pro primární úlohu. Sestává se z několika kroků:

1. Mějme duálně přípustné bazické řešení $\mathbf{x}_B$. Jestliže $\mathbf{x}_B \geq 0$, jedná se o řešení optimální. Pokud ne, vybereme řádek i tak, že i-tá složka vektoru $\mathbf{x}_B$ je záporná, $\mathbf{x}_{Bi} < 0$.

2. Pokud jsou všechny prvky v i-tém řádku simplexové tabulky nezáporné, pak duální úloha nemá maximum. V opačném případě vybereme sloupec podle

$$\varepsilon = \frac{\tilde{c}_k}{|a_{ik}|} = \min_j \frac{\tilde{c}_j}{|a_{ij}|},$$

   kde ε má podobný význam jako ε u simplexové metody (7).

3. Vytvoříme novou bázi B nahrazením i-tého sloupce k-tým. S touto novou bází určíme příslušné bazické přípustné řešení $\mathbf{x}_B$ a vrátíme se ke kroku 1.

4.4 Metody vnitřního bodu

Na rozdíl od simplexové metody, kde se řešení hledá průchodem přes krajní body přípustné množiny, začíná metoda vnitřního bodu uvnitř přípustné množiny a postupnými iteracemi se blíží k optimálnímu bodu. Body, po kterých se pohybujeme, tvoří tzv. centrální cestu. V primární úloze mluvíme o primární centrální cestě, v duální úloze o duální centrální cestě. Centrální cesta je tvořena za pomoci technik pro řešení nelineárních úloh, které jsou založeny na derivaci funkcí. Metody vnitřního bodu jsou založeny konkrétně na Karush-Kuhn-Tuckerových podmínkách optimality [38].

Věta 7 (KKT podmínky – obecné)**.** Buďte f, g_i, h_j spojitě diferencovatelné na otevřené množině obsahující množinu přípustných řešení S. Funkce g_i jsou funkce nerovnostních omezení a funkce h_j rovnostních. Buď $\mathbf{x}^*$ bod lokálního minima takový, že vektory $\nabla g_i(\mathbf{x}^*)$, $i = 1, \ldots, p$, a $\nabla h_j(\mathbf{x}^*)$, $j = 1, \ldots, t$, jsou lineárně nezávislé. Pak platí KKT podmínky:

1. $g_i(\mathbf{x}^*) \geq 0$, $\forall i = 1, \ldots, p$, $h_j(\mathbf{x}^*) = 0$, $j = 1, \ldots, t$,

2. existují $\lambda_i \geq 0$ a $\nu_j \in \mathbb{R}$, $j = 1, \ldots, t$, tak, že $\lambda_i g_i(\mathbf{x}^*) = 0$ pro všechna $i = 1, \ldots, p$ a

3. $\nabla f(\mathbf{x}^*) + \sum_{i=1}^{p} \lambda_i \nabla g_i(\mathbf{x}^*) + \sum_{j=1}^{t} \nu_j \nabla h_j(\mathbf{x}^*) = 0.$

Definice 17. (Konvexní funkce)
Konvexní funkcí rozumíme takovou funkci, pro jejíž libovolné dva body platí, že jejich spojnice leží nad grafem této funkce mezi zmíněnými body.

Definice 18. (Afinní funkce)
Funkce $f : \mathbb{R}^n \to \mathbb{R}$ je afinní, pokud $f(\mathbf{x}) = \mathbf{a}^T\mathbf{x} + b$, pro nějaké $\mathbf{a} \in \mathbb{R}^n$ a $b \in \mathbb{R}$.

Věta 8. Jsou-li funkce f, g_i konvexní a h_j afinní, pak jsou KKT podmínky i postačující.

Pro úlohu $\min_{\mathbf{x} \in S} f(\mathbf{x})$, kde $S = \{\mathbf{x} \mid g_i(\mathbf{x}) \leq 0,\ i = 1, \ldots, p,\ h_j(\mathbf{x}) = 0,\ j = 1, \ldots, t\}$ definujeme Lagrangeovu funkci předpisem

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{g}(\mathbf{x}) + \boldsymbol{\nu}^T \mathbf{h}(\mathbf{x}), \tag{8}$$

kde $\mathbf{g}(\mathbf{x}) = \begin{bmatrix} g_1(\mathbf{x}) \\ g_2(\mathbf{x}) \\ \vdots \\ g_p(\mathbf{x}) \end{bmatrix}$ a $\mathbf{h}(\mathbf{x}) = \begin{bmatrix} h_1(\mathbf{x}) \\ h_2(\mathbf{x}) \\ \vdots \\ h_t(\mathbf{x}) \end{bmatrix}$.
KKT podmínky pak můžeme napsat (za předpokladu diferencovatelnosti) jako:

1. $\nabla_{\boldsymbol{\lambda}} L(\mathbf{x}^*, \boldsymbol{\lambda}, \boldsymbol{\nu}) \leq 0,$

2. $\nabla_{\boldsymbol{\nu}} L(\mathbf{x}^*, \boldsymbol{\lambda}, \boldsymbol{\nu}) = 0,$

3. $\nabla_{\mathbf{x}} L(\mathbf{x}^*, \boldsymbol{\lambda}, \boldsymbol{\nu}) = 0,$

4. $\boldsymbol{\lambda}^T \mathbf{g}(\mathbf{x}^*) = 0.$

Využíváme zde potenciálovou funkci $\Psi(\mathbf{x}) = -\sum_{j=1}^{m} \log g_j(\mathbf{x})$, která je definovaná na vnitřku přípustné množiny S. Pro primární a duální úlohu vytvoříme pomocí této funkce bariérové úlohy s parametrem $\mu \geq 0$, viz tabulka 3.

Původní úloha	
Primární	Duální
$\min\ \mathbf{c}^T\mathbf{x}$ $\mathbf{Ax} = \mathbf{b}$ $\mathbf{x} \geq \mathbf{0}$	$\max\ \mathbf{b}^T\mathbf{y}$ $\mathbf{A}^T\mathbf{y} + \mathbf{s} = \mathbf{c}$ $\mathbf{s} \geq \mathbf{0}$
Bariérová úloha	
Primární	Duální
$\min\ \mathbf{c}^T\mathbf{x} - \mu \sum_{j=1}^{m} \log x_j$ $\mathbf{Ax} = \mathbf{b}$ $\mathbf{x} > \mathbf{0}$	$\max\ \mathbf{b}^T\mathbf{y} + \mu \sum_{j=1}^{m} \log s_j$ $\mathbf{A}^T\mathbf{y} + \mathbf{s} = \mathbf{c}$ $\mathbf{s} > \mathbf{0}$

Tabulka 3: Úprava na bariérovou úlohu pro primární a duální problém.

4.4.1 Primární centrální cesta

Jelikož máme v úloze pouze omezení typu rovností, vytvoříme Lagrangeovu funkci pro bariérovou úlohu ve zjednodušeném tvaru:

$$L_P(\mathbf{x}, \mathbf{y}) = \mathbf{c}^T\mathbf{x} - \mu \sum_{j=1}^{m} \log x_j - \mathbf{y}^T(\mathbf{Ax} - \mathbf{b}), \tag{9}$$

kde $\mathbf{y}$ má význam $\boldsymbol{\nu}$ z (8). Použitím KKT podmínek (v tomto případě pouze 2. a 3.) a využitím substituce $s_j = \dfrac{\mu}{x_j}$ obdržíme soustavu

$$\begin{aligned} \mathbf{x} \circ \mathbf{s} &= \mu\mathbf{1} \\ \mathbf{Ax} &= \mathbf{b} \\ \mathbf{A}^T\mathbf{y} + \mathbf{s} &= \mathbf{c}, \end{aligned}$$

kde $\mathbf{x} \circ \mathbf{s} = [x_1 s_1, x_2 s_2, \ldots, x_n s_n]^T$ a $\mathbf{1}$ je vektor samých jedniček velikosti n. Řešením této soustavy je vektor $(\mathbf{x}(\mu))$, jehož hodnoty pro jednotlivá μ tvoří body primární centrální cesty.

4.4.2 Duální centrální cesta

Obdobně jako pro primární úlohu vytvoříme Lagrangeovu funkci a sestavíme KKT podmínky:

$$L_D(\mathbf{y}, \mathbf{s}) = \mathbf{y}^T\mathbf{b} + \mu \sum_{j=1}^{m} \log s_j - (\mathbf{y}^T\mathbf{A} + \mathbf{s}^T - \mathbf{c}^T)\mathbf{x}, \tag{10}$$

zde má $\mathbf{x}$ význam $\boldsymbol{\nu}$. Obdržíme totožnou soustavu rovnic

$$\begin{aligned} \mathbf{x} \circ \mathbf{s} &= \mu \mathbf{1} \\ \mathbf{A}\mathbf{x} &= \mathbf{b} \\ \mathbf{A}^T\mathbf{y} + \mathbf{s} &= \mathbf{c}, \end{aligned}$$

jejímž řešením je vektor $(\mathbf{y}(\mu), \mathbf{s}(\mu))$.

4.4.3 Primárně-duální centrální cesta

Primárně-duální centrální cesta je dána vektorem $(\mathbf{x}(\mu), \mathbf{y}(\mu), \mathbf{s}(\mu))$, který vyhovuje soustavě rovnic

$$\begin{aligned} \mathbf{x} \circ \mathbf{s} &= \mu \mathbf{1} \\ \mathbf{A}\mathbf{x} &= \mathbf{b} \\ \mathbf{A}^T\mathbf{y} + \mathbf{s} &= \mathbf{c} \\ \mathbf{x} \geq 0, \mathbf{s} &\geq 0. \end{aligned}$$

4.5 Výpočetní složitost LP

U každého algoritmu nás zajímá jeho efektivita a hlavně rychlost (udána počtem iterací). V průběhu zkoumání simplexového algoritmu se podařilo ukázat, že v nejhorším případě roste počet iterací exponenciálně s rozměrem úlohy. Avšak v průměru je algoritmus vcelku rychlý, při některých numerických experimentech se ukázalo, že úloha LP s m omezujícími podmínkami se dá vyřešit $2 - 3m$ iteracemi simplexového algoritmu. Dantzig obdržel, že pro problémy s $m \leq 50$ a $n \leq 20$ je počet iterací běžně méně než $1.5m$ [38]. Zda existuje polynomiální verze simplexového algoritmu není dosud známo.

Na druhou stranu se v roce 1979 podařilo L. G. Chačijanovi publikovat elipsoidový algoritmus, který teoreticky dokazuje polynomialitu úlohy LP. Pro praktické využití je ale stále lepší simplexový algoritmus. V roce 1984 se podařilo Karmarkovi vyvinout základ metod vnitřního bodu, o kterých se ukázalo, že jsou rychlé jak z praktického hlediska, tak se povedlo dokázat polynomiálnost těchto metod.

Výpočetní složitost zde nebudeme formalizovat, uvedeme pouze praktické přiblížení. Pokud pro nějakou úlohu (např. TSP) uvažujeme určitý vstup (vstupní graf), mluvíme o instanci I dané úlohy. Velikostí $|I|$ této instance pak míníme délku zápisu (počet cifer, bitů) všech nutných vstupních dat (např. seznam vrcholů, hran, ohodnocení hran a další parametry) zakódovaných v binární soustavě.

V práci uvažujeme deterministické a nedeterministické algoritmy. U deterministického je v každém kroku jasně daný postup, u nedeterministického se může výpočet větvit (a uvažují se pak všechny větve).

Algoritmus je pak polynomiální, pokud počet jeho kroků je shora omezen hodnotou nějakého (pevného) polynomu p v bodě $|I|$, tj. hodnotou $p(|I|)$, pro všechny možné instance I.

Úloha je polynomiální, pokud pro ni existuje polynomiální algoritmus. Množina všech těchto úloh, u kterých navíc může jako výsledek být pouze „ano“ nebo „ne“ (rozhodovací úloha), pak tvoří třídu P.

Třída NP se definuje jako třída rozhodovacích úloh, u kterých je potenciálně nejvýše exponenciální množství kandidátů na řešení (exponenciálně v $|I|$) a ověření správnosti každého z kandidátů zabere polynomiální množství kroků v $|I|$. V NP jsou tedy úlohy, které lze řešit nedeterministicky a polynomiálně lze ověřit řešení.

Platí P $\subseteq$ NP a velkou otevřenou otázkou je, zda je tato inkluze vlastní, což je tzv. problém P = NP.

Úlohu A nazveme NP-těžkou, pokud je na ni možné převést každou úlohu v NP. Nemusí být $A \in$ NP.

5 Celočíselné a smíšené lineární programování

Speciálním případem LP je celočíselné lineární programování, anglicky Integer Linear Programming, tedy dále ILP, případně smíšené lineární programování, MILP - Mixed Integer Linear Programming. Na proměnné x_i, u smíšeného programování jen na některé, je zde kromě nezápornosti ještě kladena podmínka celočíselnosti.

Definice 19. Úlohou ILP rozumíme

$$\begin{aligned} \min\ & \mathbf{c}^T\mathbf{x} \\ & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & x_i \geq 0 \\ & x_i \text{ celočíselné,} \end{aligned}$$

kde $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{A}$ je reálná matice typu $m \times n$ a $\mathbf{b} \in \mathbb{R}^m$.

Definice 20. Úlohou MILP rozumíme

$$\begin{aligned} &\min \mathbf{c}^T\mathbf{x} \\ &\mathbf{Ax} = \mathbf{b} \\ &x_i \geq 0 \\ &x_i \text{ celočíselné}, i \in N \subset \{1, \ldots, n\} \\ &x_i \text{ reálné}, i \notin N, \end{aligned}$$

kde $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{A}$ je reálná matice typu $m \times n$ a $\mathbf{b} \in \mathbb{R}^m$.

Specifickým případem ILP, který budeme dále využívat, je tzv. binární programování, kde proměnné x_i nabývají pouze hodnot 0 a 1, tj. $x_i \in \{0, 1\}$.

Po krátkém zamyšlení je zřejmé, že celočíselná přípustná množina řešení úlohy ILP je podmnožinou přípustné množiny stejné úlohy bez podmínky celočíselnosti, toho se využívá při řešení úloh ILP. Pokud řešíme úlohu, kde jsme upustili od celočíselnosti, říkáme, že řešíme její LP relaxaci. Jednou z nejznámějších a nejpoužívanějších metod pro řešení ILP je metoda větví a mezí, která v kombinaci s metodou generování sloupců tvoří metodu větví a cen, a kombinace s Gomoryho řezy se nazývá metoda větví a řezů. Pro ILP neplatí věty o dualitě.

5.1 Metoda větví a mezí (Branch and Bound)

Z názvu metody lze usuzovat, že v průběhu algoritmu budeme úlohu větvit a hledat její horní a dolní omezení pro hodnotu účelové funkce. Využíváme zde výše zmíněného vztahu mezi úlohou ILP a její LP relaxací. V každém kroku tedy vyřešíme úlohu LP relaxace a provedeme následující dva kroky:

1. Pokud optimální řešení obsahuje neceločíselnou složku $\hat{x_i}$, rozvětvíme podle ní úlohu na dvě další, kde předchozí úlohu doplníme v jednom případě podmínkou $x_i \leq \lfloor \hat{x_i} \rfloor$ a v druhém $x_i \geq \lceil \hat{x_i} \rceil$. (Část větvení)

2. Při větvení nacházíme i celočíselná řešení a pamatujeme si to s nejnižší (u minimalizace) hodnotou účelové funkce, jakékoliv jiné nalezené řešení pak porovnáváme s tímto dočasně nejlepším řešením. Řešení, která mají hodnotu horší než je dočasná nejlepší, již dále nevětvíme. (Část určování mezí)

Jestliže neexistuje žádná větev s lepší hodnotou cílové funkce, která by se dala dále větvit, algoritmus ukončíme. Více v [37].

5.2 Metoda větví a cen (Branch and Price)

Metoda větví a cen je kombinací principu větvení z metody větví a mezí a metody generování sloupců.

5.2.1 Metoda generování sloupců

Metoda generování sloupců neboli Dantzig-Wolfeho dekompozice je založena na faktu, že každá úloha LP

$$\begin{aligned} \max\ & \mathbf{c}^T\mathbf{x} \\ & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \end{aligned}$$

se dá převést na tzv. hlavní problém

$$\begin{aligned} \max\ & \tilde{\mathbf{c}}^T\tilde{\mathbf{x}} \\ & \tilde{\mathbf{A}}\tilde{\mathbf{x}} = \tilde{\mathbf{b}} \\ & \tilde{\mathbf{x}} \geq \mathbf{0}, \end{aligned}$$

kde $\tilde{\mathbf{A}}$ má méně řádků ale většinou více sloupců než původní matice $\mathbf{A}$.

Rozdělíme matici $\mathbf{A}_{m\times n}$ libovolně na matice $\mathbf{A}'_{m'\times n}$ a $\mathbf{A}''_{m''\times n}$ a problém přepíšeme do tvaru

$$\begin{aligned} \max\ & \mathbf{c}^T\mathbf{x} \\ & \mathbf{A}'\mathbf{x} = \mathbf{b}' \\ & \mathbf{A}''\mathbf{x} = \mathbf{b}'' \\ & \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}. \end{aligned}$$

Podle Minkowského věty 1 dále určíme bazická přípustná řešení vyhovující $\mathbf{A}''\mathbf{x} = \mathbf{b}''$, $\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}$ jako $\mathbf{v}^1, \mathbf{v}^2, \ldots, \mathbf{v}^M$ a bazické přípustné směry jako $\mathbf{w}^1, \mathbf{w}^2, \ldots, \mathbf{w}^N$ ve tvaru

$$\mathbf{x} = \sum_{k=1}^{M} r_k\mathbf{v}^k + \sum_{k=1}^{N} s_k\mathbf{w}^k,$$

kde $r_1, r_2, \ldots, r_M$, $s_1, s_2, \ldots, s_N$ jsou nezáporná čísla a $\sum r_k = 1$. Dosazením

do zbylých vztahů dostaneme problém ve tvaru

$$\begin{aligned} \max\ \mathbf{c}^T \left(\sum_{k=1}^{M} r_k \mathbf{v}^k + \sum_{k=1}^{N} s_k \mathbf{w}^k \right) \\ \mathbf{A}' \left(\sum_{k=1}^{M} r_k \mathbf{v}^k + \sum_{k=1}^{N} s_k \mathbf{w}^k \right) = \mathbf{b}' \\ \sum_{k=1}^{M} r_k = 1 \\ r_k \geq 0 \quad (1 \leq k \leq M) \\ s_k \geq 0 \quad (1 \leq k \leq N), \end{aligned}$$

což je popis hlavního problému s proměnnými r_k, s_k. Vektory $\mathbf{v}^k$ a $\mathbf{w}^k$ nejsou v této formulaci známy a hledání bazických řešení hlavního problému se provádí řešením pomocných úloh LP, viz [17].

5.3 Metoda větví a řezů (Branch and Cut)

Metoda větví řezů je kombinace větvení z metody větví a mezí a technikou řezů.

5.3.1 Gomoryho řezy

V principu ořezávají přípustnou množinu LP relaxace problému, až se dostaneme k celočíselnému řešení.

- Gomoryho řezy I. typu

1. Vyřešíme problém bez celočíselných omezení, tedy jeho LP relaxaci. Obdržíme řešení $\mathbf{x}^*$, pokud existuje. Nastavíme index $p := 1$.
2. Pokud jsou všechny složky řešení $\mathbf{x}^*$ celočíselné kladné, pak je toto řešení optimální. Pokud ne, pokračujeme na krok 3.
3. Určíme $\tilde{r}_{q_0} = \min_i \{r_{i_0} | r_{i_0} := b_i - \lfloor b_i \rfloor; r_{i_0} > 0\}$, řádek q se nazývá zdrojový řádek.
4. Vytvoříme nové omezení (řez) z řádku q s přídavnou proměnnou y_p^*:

$$y_p^* - \sum_{j:nbv} r_{q_j} x_j = \tilde{r}_{q_0},$$

kde $r_{q_j} := a_{q_j} - \lfloor a_{q_j} \rfloor$ a nbv je zkratka pro nebazické proměnné. Přidáme řez k aktuální simplexové tabulce a určíme pivot podle duální simplexové metody a provedeme potřebný počet operací.

5. Pokud existuje přípustné řešení rozšířeného problému, nastavíme $p := p + 1$ a pokračujeme krokem 2. V opačném případě končíme, jelikož neexistuje přípustné celočíselné řešení.

- Gomoryho řezy II. typu
  Tato metoda je založena na opakovaném použití duální simplexové metody na celočíselnou tabulku. Pokud by se měla tabulka stát neceločíselnou, je přidáno nové omezení (řez). Opět popíšeme metodu v krocích. Řešíme daný problém $\tilde{P}$:

$$\begin{aligned} \min\ & \mathbf{c}^T\mathbf{x} \\ \tilde{\mathbf{A}}\mathbf{x} &\leq \tilde{\mathbf{b}} \\ \mathbf{x} &\in \mathbb{N}_0^n \\ \tilde{\mathbf{b}} &\in \mathbb{R}^m. \end{aligned}$$

1. Jsou-li všechny prvky simplexové tabulky celočíselné, definujeme $a_{ij} = \tilde{a}_{ij}$, $b_i = \tilde{b}_i$ pro všechna i, j a pokračujeme krokem 3. Pokud ne, jdeme na krok 2.
2. Uvažujeme koeficienty i-tého omezení a určíme jejich společný dělitel q_i. Vypočítáme $a_{ij} = \tilde{a}_{ij}q_i$ a $b_i = \tilde{b}_i q_i$ pro všechna i, j.
3. Připravíme simplexovou tabulku pro problém P:

$$\begin{aligned} \min\ & \mathbf{c}^T\mathbf{x} + \mathbf{0}\mathbf{y} \\ \tilde{\mathbf{A}}\mathbf{x} + y &\leq \tilde{\mathbf{b}} \\ \mathbf{x} &\in \mathbb{N}_0^n \\ \mathbf{y} &\in \mathbb{R}_+^m \\ \tilde{\mathbf{b}} &\in \mathbb{R}^m \end{aligned}$$

   a nastavíme index $p := 1$.
4. Je-li $b_i \geq 0$ pro všechna i, končíme, jelikož jsme nalezli optimální celočíselné řešení. Pokud ne, pokračujeme krokem 5.
5. Určíme provizorní pivot $a_{r's}$ podle duální simplexové metody.
6. Pokud existuje $a_{r's} < 0$, pokračujeme krokem 7. V opačném případě neexistuje přípustné řešení.
7. Platí-li $a_{r's} = -1$, nastavíme $a_{rs} = a_{r's}$ a pokračujeme krokem 9. Jinak jdeme na krok 8.

8. Vytvoříme nové omezení (řez) z r'-tého řádku s přídavnou proměnnou y_p^*:

$$\left\lfloor \frac{a_{r's}}{|\lambda|} \right\rfloor x_j + y_p^* = \left\lfloor \frac{b_{r'}}{|\lambda|} \right\rfloor,$$

kde $\lambda := \min_{j:nbv}\{a_{r'j}\}$ a přidáme toto omezení k aktuální simplexové tabulce. Buď tohle r-tý řádek. Zapomeneme na provizorní pivot $a_{r's}$ a určíme nový podle duální simplexové metody. Nastavíme $p := p + 1$.

9. Provedeme jeden krok duální simplexové metody a jdeme na krok 4.

Existuje celá řada jiných řezů, viz [17].

5.4 Výpočetní složitost ILP

Zatímco u úlohy LP bylo prokázáno, že ji lze řešit v polynomiálním čase, tedy rychle, podmínka celočíselnosti v úloze ILP situaci značně komplikuje.

Dá se ukázat, že rozhodovací verze úlohy ILP (tj. otázka na existenci přípustného řešení s omezenou hodnotou cílové funkce) patří do třídy NP, jde tedy o těžší úlohu než je úloha LP. Viz např. [24].

6 Základní přepravní/dopravní problémy

Klíčovou otázkou každého distributora je, jak obsloužit všechny své zákazníky efektivně a s co nejmenšími náklady. Přirozeným požadavkem je nejlevnější cesta, což většinou znamená ta nejkratší. Problém lze interpretovat na grafu (def. 1), který je základním objektem teorie grafů. Je tvořen množinou vrcholů a množinou hran. Vrcholy zastupují zákazníky a depo a hrany, které je spojují, jsou cesty mezi nimi. Do takového grafu můžeme zanést všechny potřebné informace a požadavky. Cenu trasy mezi dvěma body určíme přiřazením ohodnocení příslušné hrany (def. 11). Každému vrcholu lze přiřadit poptávka daného zákazníka.

V této práci začneme od nejjednoduššího požadavku, a tím je nalézt nejlevnější okružní trasu mezi depem a zákazníky. Toho se týká problém obchodního cestujícího. Ten rozšíříme na problém vícero obchodních cestujících. Přidáním dalších požadavků na poptávku zákazníků a kapacitu vozidel dojdeme k optimalizaci různých variant okružního dopravního problému.

Příklad 1. Následující úlohy budeme ilustrovat na jednodušším umělém příkladu [26]. Mějme tedy úplný neorientovaný graf na 10 vrcholech označených i, $i = 1, 2, \ldots, 10$, se symetrickou maticí vzdáleností:

$$D = \begin{pmatrix} 0 & 12 & 11 & 7 & 10 & 10 & 9 & 8 & 6 & 12 \\ 12 & 0 & 8 & 5 & 9 & 12 & 14 & 16 & 17 & 22 \\ 11 & 8 & 0 & 9 & 15 & 17 & 8 & 18 & 14 & 22 \\ 7 & 5 & 9 & 0 & 7 & 9 & 11 & 12 & 12 & 17 \\ 10 & 9 & 15 & 7 & 0 & 3 & 17 & 7 & 15 & 18 \\ 10 & 12 & 17 & 9 & 3 & 0 & 18 & 6 & 15 & 15 \\ 9 & 14 & 8 & 11 & 17 & 18 & 0 & 16 & 8 & 16 \\ 8 & 16 & 18 & 12 & 7 & 6 & 16 & 0 & 11 & 11 \\ 6 & 17 & 14 & 12 & 15 & 15 & 8 & 11 & 0 & 10 \\ 12 & 22 & 22 & 17 & 18 & 15 & 16 & 11 & 10 & 0 \end{pmatrix}.$$

V případě potřeby je za depo považován vrchol č. 1, pro ostatní vrcholy jsou známy hodnoty poptávky d_i:

i	1	2	3	4	5	6	7	8	9	10
d_i	0	10	15	18	17	3	5	9	4	6

Kapacita vozidla $Q = 40$.
Řešení jednotlivých modelů získáme pomocí softwaru MATLAB a Gurobi. Výpočty provedeme na notebooku HP 250 G3 s procesorem INTEL Pentium (N3530 s frekvencí 2 160 MHz) a pamětí RAM 4GB. U vykreslených grafů hrany neodpovídají vzdálenostem.

6.1 Problém obchodního cestujícího

Problém obchodního cestujícího, anglicky Traveling Salesman Problem, budeme používat zkratku TSP, se zabývá tím, jak objet požadovaná místa s co nejmenšími náklady. Náklady lze interpretovat vzdáleností, kterou musíme urazit, abychom navštívili všechna místa a zároveň se do každého podívali pouze jednou. V terminologii teorie grafů hledáme hamiltonovskou kružnici (def. 14) na všech n vrcholech úplného ohodnoceného neorientovaného nebo orientovaného grafu G, která má nejmenší součet ohodnocení hran c_{ij} (def. 1, 6, 11). V případě neorientovaného grafu a rovnosti $c_{ij} = c_{ji}$ mluvíme o symetrickém problému v opačném případě o nesymetrickém. Úvodem popíšeme několik modelů pro popis nesymetrické verze problému a poté se zaměříme na symetrické modely, více k těmto modelům v [29].

6.1.1 Dantzig-Fulkerson-Johnsonův model (1954)

Model DFJ je formulován pomocí celočíselného lineárního programování, který počítá s $n(n-1)$ binárními proměnnými x_{ij}. Úkolem je minimalizovat účelovou funkci (11) za podmínek (12) - (15).

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}, \qquad i \neq j, \tag{11}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1, \ldots, n, \;\; i \neq j, \tag{12}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1, \ldots, n, \;\; i \neq j, \tag{13}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \qquad S \subset V, \;\; |S| \neq \emptyset, \;\; i \neq j, \tag{14}$$

$$x_{ij} \in \{0, 1\}, \qquad i, j = 1, \ldots, n, \;\; i \neq j, \tag{15}$$

kde $x_{ij} = 1$ pouze, pokud je hrana (i, j) součástí optimální okružní trasy (anglicky tour). Podmínky (12) a (13) zajišťují, že každý vrchol navštívíme a opustíme pouze jednou. Podmínky (14) jsou tzv. podmínky eliminující (okružní) podtrasy (anglicky Subtour Elimination Constraints, zkráceně SECs). Množina S je postupně každá vlastní podmnožina vrcholů množiny V. Podmínky SECs kontrolují, že mezi $|S|$ vrcholy, které patří do množiny S, je maximálně $|S| - 1$ hran, tj. není možné, aby na nich ležela kružnice, (kružnice na k vrcholech má k hran).

Nepříjemnou vlastností této formulace je, že zatímco podmínek (12) a (13) je v závislosti na počtu vrcholů $2n$, počet podmínek (14) roste exponenciálně v závislosti na n, tj. $2^n - 2$.

6.1.2 Polynomiální Miller-Tucker-Zemlinův model (1960)

Model MTZ je dán podobně jako DFJ výrazy (11) - (13) a (15), podmínky (14) jsou nahrazeny podmínkami

$$u_i - u_j + (n-1)x_{ij} \leq n - 2, \qquad i, j = 2, \ldots, n, \;\; i \neq j, \tag{16}$$

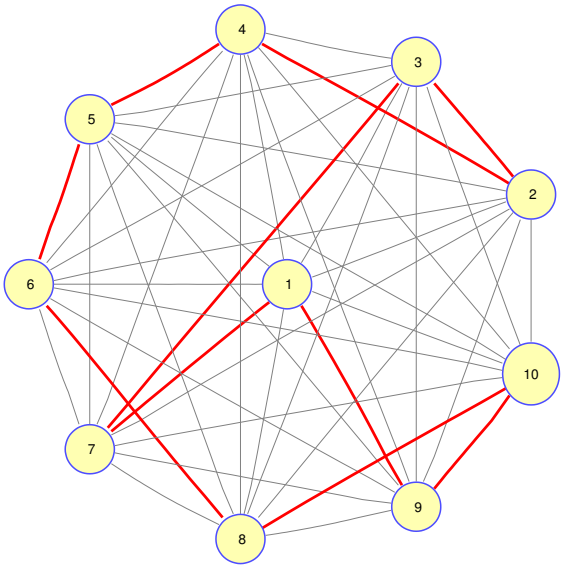
kde u_i, $i = 2, \ldots, n$, je libovolné reálné číslo označující pořadí uzlu i, ve kterém byl navštíven na optimální trase. Model byl prvotně představen bez omezení na proměnné u_i, ale později bylo přidáno omezení $1 \leq u_i \leq n-1$. Výhodou podmínek (16) je, že oproti SECs je jejich počet polynomiální, a to

$2\binom{n-1}{2} = (n-1)(n-2)$.

Příklad
Model jsme aplikovali na příklad 1. Doba výpočtu a výsledky jsou shrnuty v následující tabulce 4, graf řešení je na obrázku 1.

Počet x_{ij}	Sestavení matice	Doba výpočtu	Celkový čas	Řešení
99	0,003s	0,49s	0,493s	73

Tabulka 4: Výsledky k příkladu 1 k modelu MTZ.



Obrázek 1: Řešení TSP z příkladu 1.

6.1.3 Gavish-Gravesův model (1978)

Model GG je formulace problému obchodního cestujícího, ve které jsou okružní podtrasy eliminovány přidáním $n(n-1)$ nezáporných proměnných g_{ij}, $i = 2, \ldots, n$, $j = 1, \ldots, n$. Model GG je tvořen výrazy (11) - (13) a (15) a následujícími podmínkami

$$\sum_{j=1}^{n} g_{ij} - \sum_{j=2}^{n} g_{ji} = 1, \qquad i = 2, \ldots, n, \;\; i \neq j, \tag{17}$$

$$0 \leq g_{ij} \leq (n-1)x_{ij}, \qquad i = 2, \ldots, n, j = 1, \ldots, n, \;\; i \neq j, \tag{18}$$

kde g_{ij} značí počet hran na cestě mezi vrcholem 1 a hranou (i, j) na optimální trase.

Příklad
Model jsme aplikovali na příklad 1. Doba výpočtu a výsledky jsou shrnuty v následující tabulce 5, graf řešení je na obrázku 1.

Počet x_{ij}	Sestavení matice	Doba výpočtu	Celkový čas	Řešení
180	0,007s	0,65s	0,657s	73

Tabulka 5: Výsledky k příkladu 1 k modelu GG.

6.1.4 Symetrický problém obchodního cestujícího

V symetrické verzi TSP můžeme v modelu DFJ snížit počet proměnných x_{ij} na polovinu a model upravit:

$$\min \sum_{i=1}^{n-1} \sum_{j=2}^{n} c_{ij} x_{ij}, \qquad i < j, \tag{19}$$

$$\sum_{i=1, i<k}^{n} x_{ik} + \sum_{j=2, j>k}^{n} x_{kj} = 2, \qquad k = 1, \ldots, n, \tag{20}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \qquad S \subset V,\ 3 \leq |S| \leq n-3,\ i \neq j, \tag{21}$$

$$x_{ij} \in \{0, 1\}, \qquad i, j = 1, \ldots, n,\ i \neq j. \tag{22}$$

U ostatních modelů počet proměnných x_{ij} pro symetrický problém zredukovat nelze.
Příklad
Model jsme aplikovali na příklad 1. Doba výpočtu a výsledky jsou shrnuty v následující tabulce 6, graf řešení je na obrázku 1.

Počet x_{ij}	Sestavení matice	Doba výpočtu	Celkový čas	Řešení
45	2,77s	3,13s	5,90s	73

Tabulka 6: Výsledky k příkladu 1 k modelu symetrického DFJ.

6.2 Eukleidovský problém obchodního cestujícího

Tento problém je specifický v tom, že jeho vrcholy jsou umístěné v d-dimenzionálním Eukleidovském prostoru $\mathbb{R}^d$. Vzdálenost mezi libovolnými dvěma vrcholy i a j je definovaná jako Eukleidovská vzdálenost, tj. $\left(\sum_{k=1}^{d}(i_k - j_k)^2\right)^{1/2}$.

Definice 21. (Konvexní obal)
Nechť $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, $\mathbf{x}_i \in \mathbb{R}^d$. Konvexním obalem množiny X rozumíme množinu $conv(X) = \left\{\sum_{i=1}^{n} \beta_i \mathbf{x}_i | \sum_{i=1}^{n} \beta_i = 1, \beta_i \in \mathbb{R}\right\}$.

Optimální řešení Eukleidovského TSP má následující vlastnosti:

- Hrany optimální trasy se neprotínají.

  Pokud by se libovolné dvě hrany protínaly, lze je díky trojúhelníkové nerovnosti nahradit dvěma nekřižujícími se hranami tak, že výsledná trasa je kratší.

- Část z n vrcholů tvoří konvexní obal celé množiny. Pořadí těchto m vrcholů v optimálním řešení je shodné s pořadím, v jakém se vyskytují na konvexním obalu.

Druhá vlastnost je přímým důsledkem té první. Obě vlastnosti zároveň redukují počet přípustných tras, více v [36], [35].

6.3 Problém vícero obchodních cestujících

Zde uvažujeme, že máme jedno depo, ale m obchodních cestujících, $m > 1$, kteří mohou obsloužit zákazníky. Odtud m-TSP (Multiple Traveling Salesman Problem) [21]. Nehledáme tedy jednu trasu, ale m tras, které začínají a končí v depu.

Pokud označíme depo jako 1, lze úlohu popsat modelem celočíselného lineárního programování:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \qquad i \neq j, \tag{23}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j = 2, \ldots, n, \;\; i \neq j, \tag{24}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 2, \ldots, n, \;\; i \neq j, \tag{25}$$

$$\sum_{i=2}^{n} x_{i1} = m, \tag{26}$$

$$\sum_{j=2}^{n} x_{1j} = m, \tag{27}$$

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1, \qquad S \subseteq V \backslash \{1\}, \;\; S \neq \emptyset, \;\; i \neq j, \tag{28}$$

$$x_{ij} \in \{0, 1\}, \qquad i, j = 1, \ldots, n, \;\; i \neq j. \tag{29}$$

Z tvaru SECs (28) je patrné, že jejich počet roste exponenciálně, stejně jako u modelu DFJ 6.1.1 popisující TSP. Polynomiálního počtu podmínek dosáhneme použitím MTZ podmínek, které jsou pro m-TSP dány vztahem:

$$u_i - u_j + p x_{ij} \leq p - 1, \qquad 2 \leq i \neq j \leq n, \tag{30}$$

$$1 \leq u_i \leq p - 1, \qquad i = 2, \ldots, n, \tag{31}$$

kde p označuje maximální počet zákazníků, které může jeden obchodní cestující obsloužit. Proměnné $u_{i,j}$ mají stejný význam jako u modelu MTZ 6.1.2. Model označíme jako m-MTZ.

Příklad

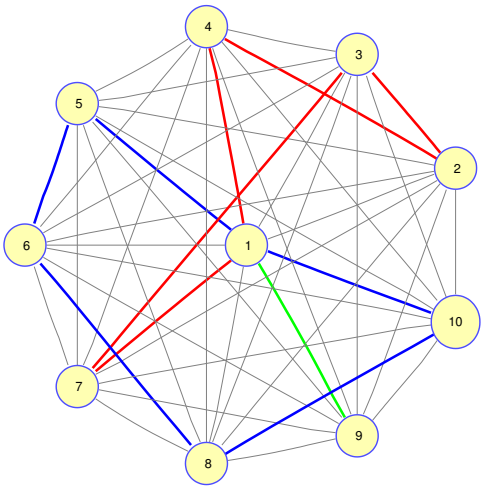
Model jsme aplikovali na příklad 1, počet vozidel jsme volili $m = 3$. Doba výpočtu a výsledky jsou shrnuty v následující tabulce 7, graf řešení je na obrázku 2, kde je zeleně vyznačena trasa 1-9-1.

Počet x_{ij}	Sestavení matice	Doba výpočtu	Celkový čas	Řešení
99	0,001s	0,32s	0,321s	91

Tabulka 7: Výsledky k příkladu 1 k modelu m-MTZ.

6.4 Okružní dopravní problém

Okružní dopravní problém, anglicky Vehicle Routing Problem, budeme používat zkratku VRP, je rozšířením a zobecněním problému obchodního cestujícího, jelikož kromě vzdálenosti (ceny trasy) se berou v úvahu další aspekty



Obrázek 2: Řešení m-TSP z příkladu 1.

týkající se distribuce zboží a služeb od dodavatele k zákazníkovi. Základním rozšířením je okružní dopravní problém s kapacitami, kde jsou známy kapacity Q všech m vozidel a poptávané množství d_i od zákazníků. Poté je úkolem najít m tras, které začínají a končí v depu tak, že součet poptávaného množství na každé trase nepřekračuje kapacitu Q příslušného vozidla, a všechna vozidla najedou minimální potřebnou vzdálenost. Dále se dají např. přidat požadavky na časová okna, tj. že obsluha zákazníka probíhá v daném časovém intervalu a vozidlo se u něj zastaví na předem daný čas, tím se ale zabývat nebudeme. Podrobnější popis problému a jeho rozšíření je k dispozici v [33].

My se zaměříme na okružní dopravní problém s kapacitami, CVRP (Capacity Vehicle Routing Problem). Jednotlivé okružní trasy se tvoří tak, aby vozidlo, které tuto trasu objede, uvezlo všechno zboží, které zákazníci poptávají. Opět můžeme úlohu rozdělit na nesymetrickou a symetrickou. Model pro nesymetrický CVRP má následující tvar:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}, \qquad i \neq j, \tag{32}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j = 2, \ldots, n, \;\; i \neq j, \tag{33}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 2, \ldots, n, \;\; i \neq j, \tag{34}$$

$$\sum_{i=2}^{n} x_{i1} = M, \tag{35}$$

$$\sum_{j=2}^{n} x_{1j} = M, \tag{36}$$

$$\sum_{i \notin S} \sum_{j \in S} x_{ij} \geq r(S), \qquad S \subseteq V \backslash \{1\}, \;\; S \neq \emptyset, \;\; i \neq j, \tag{37}$$

$$x_{ij} \in \{0, 1\}, \qquad i, j = 1, \ldots, n, \;\; i \neq j. \tag{38}$$

Podmínky (33) a (34) zaručují, že přes každý vrchol vede pouze jedna trasa. Podmínky (35) a (36) zase vyžadují, že z a do výchozího bodu vede právě m tras. Podmínky (37) jsou takzvané Capacity Cut Constraints (CCCs), kde $r(S)$ představuje minimální počet vozidel potřebný k obsloužení vrcholů z množiny S. Podmínky CCCs hlídají spojitost trasy jako SECs a navíc i požadavky na kapacitu. Bohužel jejich počet roste exponenciálně s počtem proměnných. Polynomiální počet podmínek, které mají stejnou vlastnost jako CCCs, získáme úpravou MTZ podmínek pro klasický TSP:

$$u_i - u_j + Q x_{ij} \leq Q - d_j, \qquad i, j = 2, \ldots, n, \;\; i \neq j, \tag{39}$$

$$d_i \leq u_i \leq Q, \qquad i = 2, \ldots, n. \tag{40}$$

Pomocná spojitá proměnná u_i představuje zatížení vozidla po návštěvě vrcholu i.

V symetrické verzi CVRP stačí pracovat s proměnnými x_{ij}, kde $i < j$. Problém nastane, pokud bude některá z poptávek tak velká, že vozidlo bude moci obsloužit jen tohoto jediného zákazníka. Mezi proměnnými máme jen ty ve tvaru x_{1j}, chybí nám hrana x_{j1}. Napravíme to přidáním podmínky (46), kde proměnná x_{1j} nabývá hodnoty 2 v případě, že je součástí optimálního řešení na trase $(1, j, 1)$.

$$\min \sum_{i=1}^{n-1} \sum_{j=2}^{n} c_{ij} x_{ij}, \qquad i < j, \tag{41}$$

$$\sum_{k=1,k<i}^{n-1} x_{ki} + \sum_{j=2,j>i}^{n} x_{ij} = 2, \qquad i = 2, \ldots, n, \tag{42}$$

$$\sum_{j=2}^{n} x_{1j} = 2M, \tag{43}$$

$$\sum_{i \in S} \sum_{k<i, k \notin S} x_{ki} + \sum_{i \in S} \sum_{j>i, j \notin S} x_{ij} \geq 2r(S), \qquad S \subseteq V \backslash \{1\}, \;\; S \neq \emptyset, \;\; i \neq j, \tag{44}$$

$$x_{ij} \in \{0, 1\}, \qquad i, j = 2, \ldots, n, \;\; i < j, \tag{45}$$

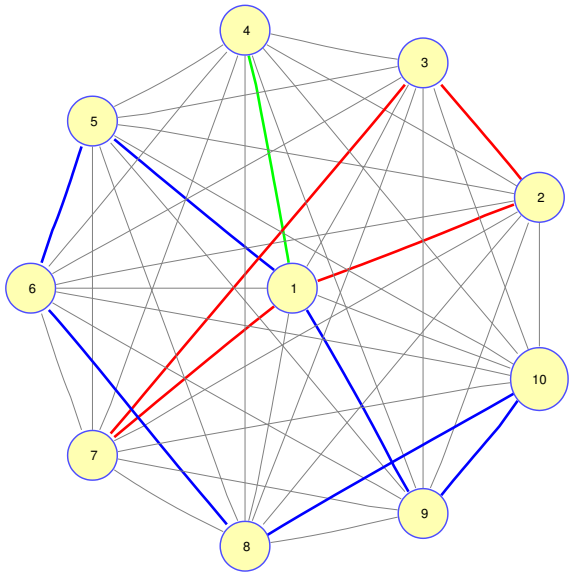
$$x_{1j} \in \{0, 1, 2\}, \qquad j = 2, \ldots, n. \tag{46}$$

Příklad

Model jsme aplikovali na příklad 1. Doba výpočtu a výsledky jsou shrnuty v následující tabulce 8, graf řešení je na obrázku 3, kde je zeleně vyznačena trasa 1-4-1.

Počet x_{ij}	Sestavení matice	Doba výpočtu	Celkový čas	Řešení
45	5,64s	6,12s	11,76s	97

Tabulka 8: Výsledky k příkladu 1 z modelu symetrického CVRP.



Obrázek 3: Řešení symetrického CVRP z příkladu 1.

Dalším možným modelem pro zápis CVRP je dvouindexový tokový model [23], kde máme množinu zákazníků, které označíme $1, \ldots, n$. Depo reprezentují dva vrcholy označené jako 0 a $n+1$ a požadujeme, aby všechny trasy vycházely z vrcholu 0 a končili ve vrcholu $n+1$. Model zapíšeme jako:

$$\min \sum_{i=0}^{n+1} \sum_{j=0}^{n+1} c_{ij} x_{ij}, \qquad i \neq j, \tag{47}$$

$$\sum_{j=1}^{n+1} x_{ij} = 1, \qquad i = 1, \ldots, n, \;\; i \neq j, \tag{48}$$

$$\sum_{i=0, i\neq h}^{n} x_{ih} - \sum_{j=1, j\neq h} x_{hj} = 0, \qquad h = 1, \ldots, n, \tag{49}$$

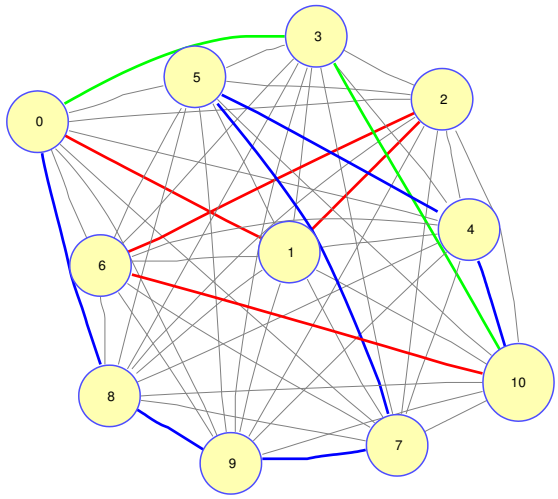
$$\sum_{j=1}^{n} x_{0j} \leq M, \tag{50}$$

$$y_j \geq y_i + d_j x_{ij} - Q(1 - x_{ij}), \qquad i, j = 0, \ldots, n+1, \tag{51}$$

$$d_i \leq y_i \leq Q, \qquad i = 0, \ldots, n+1, \tag{52}$$

$$x_{ij} \in \{0, 1\}, \qquad i, j = 0, \ldots, n+1, \tag{53}$$

kde y_i je spojitá proměnná, která vyjadřuje součet poptávek na trase z vrcholu 0 do j.



Obrázek 4: Řešení tokového CVRP z příkladu 1.

Příklad
Model jsme aplikovali na příklad 1. Doba výpočtu a výsledky jsou shrnuty v následující tabulce 9, graf řešení je na obrázku 4.

Počet x_{ij}	Sestavení matice	Doba výpočtu	Celkový čas	Řešení
110	0,002s	1,031s	1,033s	97

Tabulka 9: Výsledky k příkladu 1 z tokového modelu CVRP.

V předchozích modelech CVRP obdržíme řešení, z kterého vidíme, přes které vrcholy povedou jednotlivé okružní trasy. Ale není zřejmé, které vozidlo je přiřazeno k jaké trase. Příslušnost vozidla k trase určíme přidáním indexu m a binární proměnné y_{im} do modelu. Index m, $m = 1, \ldots, M$, označuje vozidlo a binární proměnná y_{im} je rovna jedné, pokud je zákazník i obsloužen vozidlem m. Získáme tak model ILP pro úlohu nesymetrického CVRP:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \sum_{m=1}^{M} x_{ijm}, \qquad i \neq j, \tag{54}$$

$$\sum_{m=1}^{M} y_{im} = 1, \qquad i = 2, \ldots, n, \tag{55}$$

$$\sum_{m=1}^{M} y_{0m} = M, \tag{56}$$

$$\sum_{j=1}^{n} x_{ijm} = \sum_{j=1}^{n} x_{jim} = y_{im}, \qquad i = 1, \ldots, n, \;\; m = 1, \ldots, M, \tag{57}$$

$$\sum_{i=1}^{n} d_i y_{im} \leq Q, \qquad m = 1, \ldots, M, \tag{58}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ijm} \leq |S| - 1, \qquad S \subseteq V \backslash \{1\}, \;\; S \geq 2, \;\; m = 1, \ldots, M, \tag{59}$$

$$y_{im} \in \{0, 1\}, \qquad i = 1, \ldots, n, \;\; m = 1, \ldots, M, \tag{60}$$

$$x_{ijm} \in \{0, 1\}, \qquad i, j = 1, \ldots, n, \;\; m = 1, \ldots, M. \tag{61}$$

Podmínky (55)-(57) zajišťují, že každý vrchol je navštíven pouze jednou, že z výchozího bodu vyjelo právě M vozidel, a že stejné vozidlo, které přijelo k danému vrcholu, ho taky opustí. Podmínky (58) jsou pro hlídání nepřekročení kapacity vozidla a podmínky (59) jsou známé SECs, které lze nahradit

zobecněnými MTZ podmínkami:

$$u_{im} - u_{jm} + Qx_{ijm} \leq Q - d_j, \qquad i, j = 2, \ldots, n, \;\; i \neq j, \qquad (62)$$
$$d_i + d_j \leq Q, \;\; m = 1, \ldots, M,$$

$$d_i \leq u_{im} \leq Q, \qquad i = 2, \ldots, n, \;\; m = 1, \ldots, M, \qquad (63)$$

které zároveň nahrazují podmínky kapacity (58).

Z jiného úhlu pohledu se VRP problémy řeší metodami využívajícími rozklad množin, v angličtině známé jako set partitioning, viz [23].

7 Převodové techniky mezi VRP a TSP

Pokud bychom u problému VRP nebo m-TSP nahradili výchozí bod x_0 vrcholy $x_0^1, x_0^2, \ldots, x_0^m$, které leží na souřadnicích bodu x_0, pak problém m tras převedeme na problém jedné trasy, která začíná v x_0^1, po několika zákaznících se dostane do x_0^2 atd., až se nakonec zpět vrátí do x_0^1. Obdržíme tak úlohu s $N + m$ vrcholy a omezeními pro každou z cest mezi novými sklady, které vychází z podmínek v původní úloze VRP. Převodu se využívá hlavně proto, že efektivní aproximační algoritmy jsou předně vyvíjeny pro TSP. Zvláště užitečné jsou λ-optimální algoritmy jako Lin-Kernighanův algoritmus 10.3.

Transformace dosáhneme úpravou matice vzdáleností D (popřípadě cenové matice). Výsledná matice je tvořena čtyřmi bloky, viz obrázek 5. Do levého horního rohu umístíme čtvercovou matici velikosti m, která je složená z prvků γ, v pravém dolním rohu bude původní matice D, zbylé řádky v horní části matice jsou shodné s prvním řádkem původní matice a stejně tak sloupce v levém dolním bloku jsou shodné s prvním sloupcem původní matice D.

$\begin{matrix}\gamma & \cdots & \gamma \\ \vdots & & \vdots \\ \gamma & \cdots & \gamma\end{matrix}$	d_{0j}
d_{i0}	D

Obrázek 5: Upravená matice transformace.

Podle zvolené hodnoty γ bude řešení TSP odpovídat různým zadáním původního problému VRP. Rozlišujeme případy:

- $\gamma = \infty$
  Tato hodnota vyjadřuje, že přímá cesta mezi sklady není povolena. Řešení TSP, pokud existuje, pak obsahuje m oddělených cest začínajících a končících v příslušném depu.

- $\gamma = -\infty$
  Za předpokladu, že m je libovolné číslo větší než nejmenší možný počet vozidel, pro který má VRP přípustné řešení, bude řešení TSP obsahovat všechny možné přímé trasy mezi depy, což povede na použití minimálního počtu vozidel. Jinými slovy řešení odpovídá problému VRP s úkolem nalézt nejmenší možný počet vozidel, při kterém jsou obslouženi všichni zákazníci. Zejména pokud je γ hodně velké, ale ne nekonečné záporné číslo, pak se při řešení TSP nejprve minimalizuje počet vozidel, až poté délka trasy.

- $\gamma = 0$
  Zde není zvýhodněno ani penalizováno odebrání vozidla. Řešení TSP odpovídá problému VRP s úkolem co nejlevněji obsloužit všechny zákazníky.

- $\gamma = -\varepsilon$
  Pokud jsou z finančního hlediska fixní náklady vozidla ekvivalentní zkrácení trasy o ε ujetých kilometrů za den, pak řešení TSP odpovídá problému VRP, kde se minimalizují celkové náklady (fixní i variabilní).

Takto formulovaná úloha TSP je těžší než stejně velká úloha klasického TSP, tj. bude vždy časově náročnější než přímé řešení VRP, ale jak již bylo zmíněno, jsou pro ni známé efektivní aproximační algoritmy, více v [16].

8 Vztahy mezi úlohami v metrickém případě

V metrickém případě jsme schopni popsat vztah mezi optimálními řešeními jednotlivých úloh v případě, že jsou řešeny na stejném grafu.

8.1 Vztah mezi TSP a m-TSP

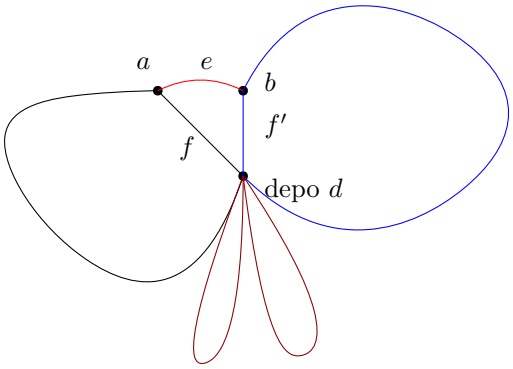
Ukážeme nyní souvislosti řešení úloh TSP a m-TSP v metrickém případě.

Věta 9. Buď G úplný ohodnocený neorientovaný graf. Buď $w\colon E \to R$ příslušné ohodnocení. Předpokládejme, že w splňuje Δ-nerovnost. Pak platí: hodnota optimálního řešení (minima) úlohy TSP je dolním omezením pro hodnotu optimálního řešení úlohy m-TSP (pro $m \geq 1$).

Navíc, pokud $m < m'$, je optimum (minimum) úlohy m-TSP nejvýše rovno optimu úlohy m'-TSP

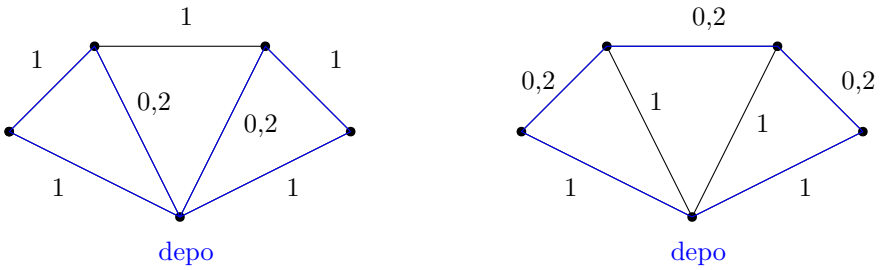
Důkaz. Buď T trasa odpovídající optimu úlohy m-TSP. Jsou-li a a b dva vrcholy této trasy sousedící s depem d, pak odstraněním hran $ad = f$ a $bd = f'$ z T a přidáním hrany $ab = e$ získáme trasu T' s délkou $|T'| \leq |T|$ vzhledem k platnosti trojúhelníkové nerovnosti.

Zřejmě je $m \leq n - 1$, kde n je počet vrcholů uvažovaného grafu, viz obrázek 6. Chybějící hrany mají vysoké ohodnocení. □



Obrázek 6: Ilustrace k důkazu pro tvrzení o vztahu TSP a m-TSP.

Pro obecná ohodnocení mohou být optima úlohy TSP a m-TSP v různém vztahu, viz obrázek 7. V prvním případě neplatí Δ-nerovnost, ve druhém případě ano, modře jsou optimální okružní trasy.



Obrázek 7: Vliv trojúhelníkové nerovnosti na optimální řešení.

8.2 Vztahy pro CVRP

Zřejmě platí.

Tvrzení 4. Máme-li dvě úlohy CVRP lišící se pouze kapacitami vozidel Q a Q', pak je-li $Q \leq Q'$, tak je optimum úlohy s kapacitou Q' nejvýše rovno optimu úlohy s kapacitou Q.

9 Aproximační algoritmy

Z aproximačních algoritmů obdržíme přípustné řešení, které je blízké optimálnímu řešení, značíme OPT, a bylo získáno v krátkém čase. Kvalitu získaného řešení určuje funkce $\delta : \mathbf{Z}^+ \to \mathbf{Q}^+$ a mluvíme o δ-aproximačním algoritmu.

Definice 22. (δ-aproximační algoritmus)
Mějme minimalizační (maximalizační) problém P a funkci $\delta \geq 1$ ($\delta \leq 1$). Algoritmus $\mathcal{A}$ je δ-aproximační algoritmus pro problém P, pokud pro každou instanci I problému P dá algoritmus přípustné řešení s takové, že

$$f_P(I, s) \leq \delta(|I|) \cdot \mathrm{OPT}(I) \ (f_P(I, s) \geq \delta(|I|) \cdot \mathrm{OPT}(I)),$$

a vyřeší ho v čase omezeném pevným polynomem v $|I|$, kde $|I|$ je velikost úlohy, kterou rozumíme počet bitů potřebný k jejímu zapsání, za předpokladu, že jsou všechna čísla v úloze zapsána binárně. Zřejmě, čím blíže je funkce δ k jedničce, tím lepší je algoritmus. Užitečné případy jsou především, pokud se pro něakou úlohu povede ukázat, že pro ni existuje aproximační algoritmus, kdy δ je konstantní funkce. Mluvíme pak o aproximačním algoritmu s konstantním faktorem δ. Např. pro metrickou úlohu TSP existuje $\frac{3}{2}-$aproximační algoritmus. Aproximačními algoritmy se podrobně zabývá V. V. Vazirani [36].

9.1 Aproximační schéma

Při praktických aplikacích se vyplatí místo přesného algoritmu použít aproximaci, obzvlášť u NP-těžkých problémů.

Definice 23. (Aproximační schéma)
Nechť P je NP-těžký optimalizační problém s účelovou funkcí f_P. Aproximačním schématem úlohy P nazveme algoritmus $\mathcal{A}$, který při vstupu (I, ε) dá řešení s tak, že

- $f_P(I, s) \leq (1 + \varepsilon) \cdot \mathrm{OPT}$, pokud P je minimalizační problém,
- $f_P(I, s) \geq (1 - \varepsilon) \cdot \mathrm{OPT}$, pokud P je maximalizační problém,

kde I je instance optimalizačního problému a $\varepsilon > 0$ je chybový parametr.

Algoritmus $\mathcal{A}$ nazveme polynomiální aproximační schéma (polynomial time approximation scheme - PTAS), pokud pro každé pevné $\varepsilon > 0$ proběhne v polynomiální čase v závislosti na velikosti úlohy I. Nevíme ale, jak čas průběhu algoritmu závisí na ε. Přidáním požadavku, aby čas běhu algoritmu závisel polynomiálně na velikosti úlohy I a na $1/\varepsilon$, získáme plně polynomiální aproximační schéma (fully polynomial time approximation scheme - FPTAS), podrobněji v [36].

10 Řešení problémů pomocí heuristik

Heuristika pochází z řeckého *heuristiké*, tj. umění hledat [15]. Při řešení optimalizačních úloh jde v zásadě o polynomiální postup, který neprobere všechny možnosti, ale jeho řešení může být optimální nebo velice blízké optimálnímu řešení dané úlohy. Výstupem heuristického algoritmu může být i nejednoznačná odpověď, kdy není zřejmé, zda řešení existuje, a nelze prokázat, že neexistuje. U heuristik je velmi často dokázáno, že představují δ-aproximační algoritmy [34], kde δ je rostoucí funkce, (tj. hodnota řešení určeného heuristikou může být libovolně daleko od hodnoty optimálního řešení). Heuristiky dělíme na konstrukční (postupně tvoří řešení) a vylepšující (vylepšují náhodně určené řešení).

Dalším rozšířením jsou tzv. metaheuristiky, např. evoluční algoritmy, které se inspirovaly u přírodních procesů genetiky. Zajímavým využitím heuristik a metaheuristik je vzít získané řešení jako startovací bod pro přesný algoritmus. Heuristik pro řešení TSP i CVRP existuje velké množství, viz [33], my popíšeme několik z nich.

10.1 Metoda nejbližšího souseda

Jednou z nejjednodušších heuristik pro TSP je metoda nejbližšího souseda (Nearest Neighbor), viz [32], kde začínáme v libovolném bodě a do trasy přidáváme nejbližší vrchol, který ještě nebyl navštíven. Po nalezení posledního volného vrcholu se vrátíme do výchozího, což většinou bývá dlouhá hrana, která způsobuje, že jsme nenalezli optimální trasu.

10.2 Metoda Clarke Wrighta

Metoda Clarke Wrighta (metoda CW) patří mezi nejznámější konstrukční heuristiky pro řešení VRP, ale dá se použít i pro klasický TSP, nebo m-TSP. Je založena na myšlence rezerv, které vznikají při spojování okružních tras, které začínají a končí ve výchozím bodu (depu) označeném jako vrchol 1. Při spojení tras $(1, \ldots, i, 1)$ a $(1, j, \ldots, 1)$ vznikne okružní trasa $(1, \ldots, i, j, \ldots, 1)$ a rezerva (ušetřená vzdálenost) $s_{ij} = c_{i1} + c_{1j} - c_{ij}$. Postup algoritmu se dá popsat dvěma kroky:

1. Napočítáme všechny rezervy s_{ij} pro $i, j = 1, \ldots, n$, $i \neq j$, a vytvoříme $n - 1$ okružních tras ve tvaru $(1, i, 1)$ pro $i = 2, \ldots, n$. Rezervy s_{ij} seřadíme v nerostoucím smyslu.

2. (Paralelní verze) Začneme od první rezervy ze seřazeného seznamu rezerv a zjistíme, zda existují dvě okružní trasy, z nichž jedna obsahuje

hranu $(1, j)$ a druhá hranu $(i, 1)$, které lze propojit tak, že vzniklá okružní trasa bude přípustná ve smyslu zadání úlohy. Je-li tomu tak, spojíme je vymazáním hran $(1, j)$, $(i, 1)$ a přidáním hrany (i, j). Postup opakujeme, dokud existují propojitelné okružní trasy.

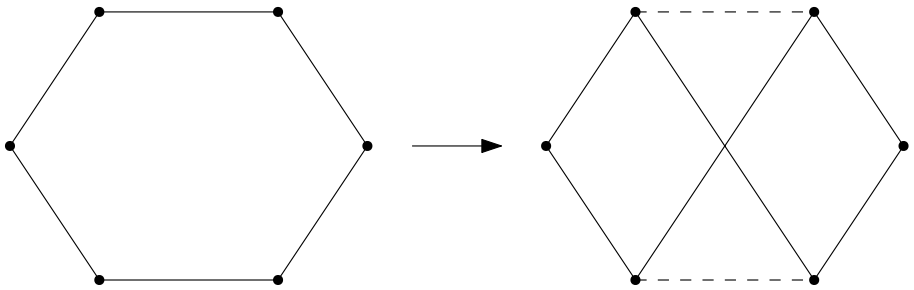
2. (Sekvenční verze) Uvažujeme postupně každou trasu $(1, i, \ldots, j, 1)$. Určíme první rezervu s_{ki} nebo s_{jl} pro přípustné spojení aktuální trasy s jinou, která obsahuje hranu $(k, 1)$, nebo hranu $(1, l)$. Provedeme spojení tras a postup opakujeme. Pokud již není žádné spojení možné, uvažujeme další trasu.

Při porovnání druhých kroků se ukázalo, že je lepší používat paralelní verzi [33]. Metodu CW s paralelní verzí jsme naimplementovali v softwaru MATLAB.

Metoda má ale také své nedostatky [16]. A to především fakt, že počáteční řešení (krok 1) je nepřípustné, jelikož je obvykle počet vozidel menší než počet zákazníků. Také nerozlišuje mezi úlohami najít nejlevnější trasy a nebo minimalizovat počet vozidel.

10.3 Lin-Kernighanův algoritmus

Lin a Kaernighan představili svůj algoritmus v roce 1971. Vylepšili jeden z algoritmů, který patří mezi algoritmy řešící symetrický TSP tak, že vylepšuje náhodně zkonstruovanou trasu na daných vrcholech. Jde o λ-opt algoritmus, který v každém kroku nahradí λ hran trasy jinými λ hranami tak, že výsledná trasa je kratší, pro $\lambda = 2$ je takové prohození na obrázku 8. Výměny probíhají, dokud již není žádné další vylepšení možné. Tento algoritmus je založený na myšlence λ-optimality.



Obrázek 8: Prohození hran pro 2-opt algoritmus.

Tvrzení 5. O okružní trase řekneme, že je λ-opt (λ-optimální), jestliže už není možné dalším prohozením λ hran získat kratší trasu.

Nevýhodou je, že hodnota λ musí být předem známa. Je těžké určit λ tak, abychom našli kompromis mezi rychlostí výpočtu a kvalitou získaného řešení.

Lin a Kernighan tuto nevýhodu odstranili zavedením variabilního λ-opt algoritmu, kde se hodnota λ určuje v každém iteračním kroku. Základní myšlenkou algoritmu je, že máme náhodně zvolenou počáteční okružní trasu T a snažíme se najít dvě množiny hran, označme je $X = \{x_1, \ldots, x_r\} \in T$ a $Y = \{y_1, \ldots, y_r\} \notin T$, tak, že když hrany z množiny X vymažeme a nahradíme je hranami z Y, dostaneme lepší okružní trasu. Množiny X a Y se tvoří hranu po hraně za dodržení následujících podmínek.

- Hrany x_i a y_i musejí mít společný vrchol a stejně tak y_i a x_{i+1}. Označíme-li t_1 jeden z koncových vrcholů hrany x_1, tak dále obecně platí předpis $x_i := (t_{2i-1}, t_{2i})$, $y_i := (t_{2i}, t_{2i+1})$ a $x_{i+1} = (t_{2i+1}, t_{2i+2})$ pro $i \geq 1$.

  Nutnou, ale ne postačující, podmínkou, aby výměna hran X s Y vyústila v trasu, je, že řetězec $(x_1, y_1, x_2, \ldots, x_r, y_r)$ je uzavřený, tedy $y_r = (t_{2r}, t_1)$.

- Hranu $x_i = (t_{2i-1}, t_{2i})$ vybíráme tak, že pokud je t_{2i} spojeno s t_1, výsledný řetězec je trasou. Toto kritérium přípustnosti se používá pro $i \geq 3$ a zaručuje možnost uzavření trasy.

- Hranu y_i vybíráme s ohledem na hodnotu zisku G_i, který chceme kladný. Zisk z výměny hran x_i a y_i definujeme jako $g_i = c(x_i) - c(y_i)$, G_i je pak součtem $g_1 + g_2 + \cdots + g_i$.

- Množiny X a Y nemají žádnou společnou hranu.

Základní algoritmus lze popsat následujícími kroky.

Algoritmus 1. (LK)

1. Vygenerujeme náhodnou počáteční trasu T.
2. $i = 1$, vybereme vrchol t_1.
3. Vybereme hranu $x_1 = (t_1, t_2) \in T$.
4. Vybereme hranu $y_i = (t_2, t_3) \notin T$ tak, že $G_1 > 0$. Pokud taková není, jdeme na krok 12.
5. $i = i + 1$.
6. Vybereme $x_i = (t_{2i-1}, t_{2i}) \in T$ tak, že

a) je-li t_{2i} spojeno s t_1, výsledná konfigurace je okružní trasa T', a

b) $x_i \neq y_s \ \forall s < i$.

Pokud je T' lepší trasa než T, volíme $T = T'$ a jdeme na krok 2.

7. Vybereme $y_i = (t_{2i}, t_{2i+1}) \notin T$ tak, že

   a) $G_i > 0$,

   b) $y_i \neq x_s \ \forall s \leq i$, a

   c) x_{i+1} existuje.

   Pokus takové y_i existuje, jdeme na krok 5.

8. Existuje-li nevyzkoušená varianta pro y_2, $i = 2$ a jdeme na krok 7.

9. Existuje-li nevyzkoušená varianta pro x_2, $i = 2$ a jdeme na krok 6.

10. Existuje-li nevyzkoušená varianta pro y_1, $i = 1$ a jdeme na krok 4.

11. Existuje-li nevyzkoušená varianta pro x_1, $i = 1$ a jdeme na krok 3.

12. Existuje-li nevyzkoušená varianta pro t_1, jdeme na krok 2.

13. Stop, (nebo jdeme na krok 1).

Nyní se spíše vužívá LKH algortimus, kde H odkazuje na K. Helsgauna, který ho ještě vylepšil, viz [13].

10.3.1 (λ, α)-postoptimalizace

U Lin-Kernighanova algoritmu byl zaveden pojem λ-optimality v souvislosti s λ-opt algoritmem pro řešení úlohy TSP. Při řešení úloh CVRP hrají roli nejen délky hran, ale rovněž poptávky míst.

Zavedeme zde proto analogicky α-okolí řešení $\mathbf{x}$ jako množinu všech řešení získatelných z $\mathbf{x}$ pomocí a-opt operace.

Operaci a-opt definujeme pro úlohu CVRP následovně. Mějme úplný graf G s vrcholovým ohodnocením d_i (poptávky míst), kapacitu vozidla Q a počet vozidel m. Uvažujme pokrytí (rozklad) vrcholů grafu G vzájemně disjunktními množinami C_i, $i = 1, \ldots, m$. Množiny C_i tedy splňují $C_i \cap C_j = \emptyset$ a $\bigcup_{i=1}^{m} C_i = V(G)$. Navíc $\sum_{j=1}^{|C_i|} d_j \leq Q$ pro každé $i = 1, \ldots, m$.

Vezměme a-prvkovou podmnožinu D množiny $V(G)$. Pro různá i, j uvažujme $D \cap C_i$ a $D \cap C_j$ a bez újmy na obecnosti nechť $|D \cap C_i| \leq |D \cap C_j|$.

Provedeme výměny každé ze všech podmnožin množiny $D \cap C_i$ se všemi podmnožinami množiny $D \cap C_j$ s tím, že po výměně jsou splněny kapacitní omezení pro nové podmnožiny C_i' a C_j', tj. platí pak $\sum_{k=1}^{|C_i'|} p_k \leq Q$ a $\sum_{j=1}^{|C_i'|} p_j \leq Q$.

Definice 24. Řekneme, že řešení úlohy CVRP je (λ, α)-optimální, pokud v jeho λ-okolí a α-okolí neexistuje řešení s lepší hodnotou cílové funkce.

Jedná se o pojem lokální optimálnosti řešení. Ačkoli je (λ, α)-okolí relativně veliké, při konstantních λ a α je prohledání tohoto okolí polynomiálním algoritmem. Navíc často $\lambda = 2$ a v praktické aplikaci v části 14 rovněž volíme $\alpha = 2$. (V praktické části ukážeme, že řešení z LKH řešiče není obecně α-optimální.)

10.4 Evoluční metody s operátorem EAX

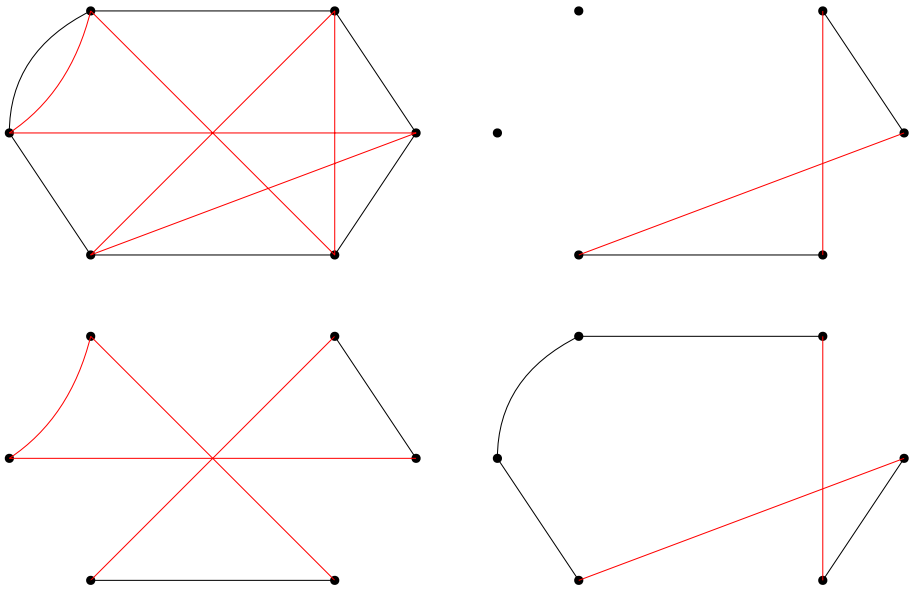
Zkratka EAX znamená Edge-Assembly Cross-over, do češtiny přeloženo jako křížení skupin hran. Tzv. operátor EAX se využívá v evolučních algoritmech, které jsou navrženy pro řešení TSP. Evoluční algoritmy jsou návody, jak z rodičů (některých možných řešení) určit potomka/ky (nová lepší řešení). EAX byl vytvořen Nagatou a Kobayashim [30] a je brán jako efektivní křížení pro TSP. Postup křížení hran lze popsat takto:

1. Na vrcholech daného grafu si označíme pár rodičů jako trasu A (černá) a trasu B (červená) a definujeme graf G_{AB}, viz obrázek 9 vlevo nahoře, který je tvořený sloučením obou tras.

2. Rozdělíme hrany v G_{AB} na AB-cykly, tj. na kružnice na G_{AB}, kde se střídají hrany z trasy A s hranami z trasy B. Příkladem takového AB-cyklu je kružnice na obrázku 9 vpravo nahoře.

3. Vybereme určitý počet AB-cyklů podle zvoleného pravidla a jejich hrany uložíme do množiny E. Např. vložíme do množiny E AB-cyklus z předchozího kroku. Z trasy A odebereme její hrany, které jsou obsaženy v množině E, a přidáme k ní hrany trasy B z množiny E, viz obrázek 9 vpravo dole. Nebo lze stejným způsobem vyměnit hrany v trase B, což je na obrázku 9 vlevo dole.

4. Pokud po výměně hran vzniknou podtrasy, tj. výsledkem není hamiltonovská kružnice, pospojujeme je v přípustnou trasu např. 2-opt algoritmem.



Obrázek 9: Aplikace operátoru křížení EAX na černou a červenou hamiltonovskou kružnici.

Pravidel, jak vybrat AB-cykly do množiny E je několik, viz [30]. Je možné vybrat pouze jeden AB-cyklus, nebo náhodně či specifickým postupem několik z nich.

11 Uložení nákladu

Po rozvrhnutí optimální trasy pro rozvoz, je dalším krokem optimální a bezpečné uložení nákladu do vozidla tak, aby byla úložná plocha využita efektivně, a aby se zboží nepoškodilo. Jednou ze základních a známých úloh je úloha o batohu, kde neřešíme, jak předměty do batohu naskládat, ale jde nám o to, aby naložené věci nepřekročily nosnost batohu a byly co nejužitečnější. Dalším krokem jsou například úlohy o plnění zásobníků, které se dělí na 1D, 2D a 3D problémy.

11.1 Úloha o batohu

V angličtině je tato úloha známá pod pojmem Knapsack problem (KP). V úloze máme n předmětů, které chceme naložit do batohu o nosnosti L. U každého předmětu je známa jeho hmotnost w_i a užitek u_i, $i = 1, \ldots, n$. Zavedeme binární proměnnou x_i, která označuje, zda je daný předmět vložen

do batohu či nikoli. Úlohu zapíšeme pomocí ILP v následujícím tvaru:

$$\max \sum_{i=1}^{n} u_i x_i, \tag{64}$$

$$\sum_{i=1}^{n} w_i x_i \leq L, \tag{65}$$

$$x_i \in \{0, 1\}. \tag{66}$$

Jde tedy o úlohu, kde má matice $\mathbf{A}$ pouze jeden řádek. Úloha je často používána jako příklad pro ukázku metody větví a mezí 5.1 nebo na demonstraci dynamického programování.

11.1.1 Dynamické programování pro úlohu o batohu

Dynamické programování je založeno na rozdělení problému na podproblémy a kombinací jejich řešení získat celkové řešení. Důležitý je požadavek, že pokud mají podproblémy nějakou část společnou, je řešena pouze jednou. Výsledky podproblémů jsou zaznamenávány do tabulky a dají se tak zpětně použít. Postup algoritmu si rozdělíme do několika kroků:

1. Rozdělení problému na podproblémy.
   Vytvoříme si matici V typu $(n+1) \times (L+1)$. Jednotlivé prvky (pole) označíme $V[i, w]$, které pro $1 \leq i \leq n$ a $0 \leq w \leq L$ budou obsahovat maximální užitek z naložených předmětů váhy nejvýše w. Výsledný užitek bude obsažen na pozici $V[n, L]$.

2. Rekurzivní určení optimálního řešení.
   Na začátku nastavíme

   $$\begin{aligned} V[0, w] &= 0, & 0 \leq w \leq L, \\ V[i, w] &= -\infty, & w \leq 0. \end{aligned}$$

   Další hodnoty pro $1 \leq i \leq n$ a $0 \leq w \leq L$ určujeme z rekurzivního vztahu

   $$V[i, w] = \max\{V[i-1, w], u_i + V[i-1, w-w_i]\}.$$

   Pokud je větší hodnota $V[i-1, w]$, znamená to, že předmět i do batohu nevkládáme a naopak.

   Výpočet provádíme řádek po řádku, až dostaneme konečnou hodnotu celkového užitku na pozici $V[n, L]$.

3. Určení naložených předmětů
   Ve výše popsaných krocích se nikam neukládá, jaké předměty jsou vloženy do batohu, a výsledek říká pouze hodnotu maximálního užitku. Vytvoříme pomocnou matici k, kde na pozici $k[i, w]$ zaznamenáme, zda jsme na pozici $V[i, w]$ přidali předmět i, $k[i, w] = 1$ či nikoliv, $k[i, w] = 0$. Množinu naložených předmětů T určíme následujícím algoritmem:

$$
\begin{aligned}
&K = L;\\
&\text{for } (i = n, \ldots, 1)\\
&\quad \text{if } (k[i, K] == 1)\\
&\qquad \{i \in T;\\
&\qquad K = K - w_i;\}
\end{aligned}
$$

Podrobněji je dynamické programování popsáno v [5].

Příklad 2. Mějme batoh o kapacitě $L = 10$ a čtyři předměty s příslušnými hmotnostmi a užitky.

i	1	2	3	4
u_i	10	40	30	50
w_i	5	4	6	3

Pomocí dynamického programování jsme získali matici V:

$i \backslash w$	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0_0	0_0	0_0	0_0	0_0	10_1	10_1	10_1	10_1	10_1	10_1
2	0_0	0_0	0_0	0_0	40_1	40_1	40_1	40_1	40_1	50_1	50_1
3	0_0	0_0	0_0	0_0	40_0	40_0	40_0	40_0	40_0	50_0	70_1
4	0_0	0_0	0_0	50_1	50_1	50_1	50_1	90_1	90_1	90_1	90_1

Červené indexy vyznačují hodnoty matice $k[i, w]$, z které jsme určili, že do batohu byly vloženy předměty 2 a 4 s celkovým užitkem 90. Dynamické programování jsme též implementovali v softwaru MATLAB. Pomocí něj pak v praktické části odhadujeme minimální potřebný počet vozidel pro úlohu CVRP.

11.2 FPTAS pro úlohu o batohu

Budeme předpokládat, že všechna čísla vyskytující se v úloze I jsou zapsána binárně. Připomeňme, že velikostí úlohy $|I|$ rozumíme počet bitů potřebný

k jejímu zapsání. Pokud budou čísla v úloze malá, bude velikost úlohy polynomiální, a tedy i čas, po který algoritmus poběží. Získání FPTAS (plně polynomiálního aproximačního schématu) 9.1 je založeno na myšlence, že pokud vynecháme určitý počet nejméně důležitých bitů v závislosti na hodnotě parametru ε, můžeme na upravené hodnoty pohlížet jako na čísla ohraničená polynomem v n a $1/\varepsilon$. To nám umožní získat řešení, kde celkový užitek je alespoň $(1-\varepsilon)$-násobek optimální hodnoty, a které je nalezeno v čase ohraničeném polynomem v n a $1/\varepsilon$.

Algoritmus 2. (FPTAS pro úlohu o batohu)

1. Pro dané $\varepsilon > 0$ buď $K = \dfrac{\varepsilon U}{n}$, kde $U = \max\{u_i\}$, $i = 1, \ldots, n$.
2. Pro každý předmět i určíme $u'_i = \left\lfloor \dfrac{u_i}{K} \right\rfloor$.
3. Pro užitky u'_i použijeme dynamické programování k nalezení nejužitečnější množiny S'.
4. Výstup S'.

Dá se dokázat následující věta, viz [36].

Věta 10. Algoritmus 2 je plně polynomiální aproximační schéma pro úlohu o batohu.

11.3 Plnění zásobníků

V úloze o batohu nebereme v potaz, jak předměty do batohu ukládáme. A to by mohlo způsobit, že užitečnou věc poškodíme, je tedy vhodné klást podmínky i na uložení předmětů. Plnění zásobníků, anglicky Bin Packing Problem (BPP), bere tyto podmínky v potaz. Touto problematikou se zabývá např. diplomová práce [32]. Podle toho v kolika rozměrech se liší předměty k uložení, rozlišujeme jedno, dvou a trojdimenzionální problém. Tyto problémy se většinou řeší pomocí heuristik, na bázi hladových algoritmů, které se podle předzpracování předmětů k uložení dělí na dvě skupiny:

- **On-line metody:** O ukládaných předmětech dopředu nic nevíme a postupně je ukládáme do zásobníků.
- **Off-line metody:** Před nakládáním známe přesný počet předmětů a jejich rozměry. Podle toho si je můžeme dopředu připravit a dosáhnout tak lepšího řešení.

11.3.1 1D problém

Nejjednodušším problémem je BBP, kde se předměty liší pouze v jednom rozměru, např. ve výšce, která nesmí být vyšší než je výška používaných zásobníků.

Definice 25. 1D-BPP
Mějme n předmětů s výškou h_j, šířkou w_j a hloubkou d_j, $j = 1, \ldots, n$, pro které platí $h_j \leq H$, $w_j = W$ a $d_j = D$, kde H, W a D jsou konstanty a H je výška zásobníku.

Pro představu uvedeme úlohu, kde máme n předmětů s výškou $h_j \in (0, 1]$, a chceme je uložit do co nejméně zásobníků o jednotkové velikosti. K řešení můžeme použít heuristiku First-Fit (on-line metoda), kde bereme předměty v libovolném pořadí a umisťujeme je do prvního zásobníku, kam se vejdou, není-li místo v žádném z částečně naplněných zásobníků, umístíme předmět do nového prázdného. Pokud je výsledkem, že použijeme m zásobníků, bude alespoň $m - 1$ z nich z více jak poloviny plných, tj.

$$\sum_{i=1}^{n} a_i > \frac{m-1}{2}. \tag{67}$$

Víme, že součet velikostí je dolní mez pro optimum, pak $m - 1 < OPT$, tzn.

$$m \leq 2OPT. \tag{68}$$

Jedná se tedy o 2-aproximační algoritmus. Platí následující věta, viz [36].

Věta 11. Pro libovolné $\varepsilon > 0$ neexistuje pro problém plnění zásobníků aproximační algoritmus, který by zaručoval $(3/2 - \varepsilon)$-násobek optima, za předpokladu P $\neq$ NP.

Jako další on-line heuristiky lze uvést Last-Fit, která je opakem k First-Fit, Best-Fit, kde vkládáme do prvního možného nejvíce zaplněného zásobníku, nebo Worst-Fit, která je opakem k Best-Fit. Off-line heuristiky vycházejí z jmenovaných on-line heuristik, kde před samotným vkládáním nejprve předměty sestupně seřadíme.

11.3.2 2D problém

Předměty se liší ve dvou rozměrech a praktickým příkladem může být skládání zboží na palety.

Definice 26. 2D-BPP
Mějme n předmětů s výškou h_j, šířkou w_j a hloubkou d_j, $j = 1, \ldots, n$, pro které platí $h_j \leq H$, $w_j \leq W$ a $d_j = D$, kde H, W a D jsou konstanty, H je výška zásobníku a W jeho šířka.

K řešení můžeme použít heuristiku Bottom Left Fit. Předměty si nejprve sestupně seřadíme podle výšky h_j a poté je začneme vkládat do zásobníku od levého spodního rohu vedle sebe, dokud jejich společná šířka nepřesáhne šířku W. Tak vytvoříme první úroveň a další začne ve výšce prvního předmětu z předchozí úrovně. Postup opakujeme tak, abychom nepřekročili výšku H.

Předměty v jednotlivých úrovních se dají poté ještě přeskupit pomocí algoritmů pro 1D-BPP.

Tuto heuristiku jsme naimplementovali v softwaru MATLAB, viz implementační část 14.

11.3.3 3D problém

Definice 27. 3D-BPP
Mějme n předmětů s výškou h_j, šířkou w_j a hloubkou d_j, $j = 1, \ldots, n$, pro které, za předpokladu, že předměty nerotují, platí $h_j \leq H$, $w_j \leq W$ a $d_j \leq D$, kde H, W a D jsou konstanty, H je výška zásobníku, W šířka a D jeho hloubka.

12 Dynamické modely přepravních úloh

Výše uvedené modely byly formulovány jako statické, tj. model je kompletně formulován ještě před jeho řešením. Sestavení kompletního modelu pro větší úlohy zabírá příliš mnoho paměti.

Uvedeme jednu z užitečných technik, která v případě, že počet omezení je velký, výrazně snižuje paměťové nároky.

12.1 Odložená omezení

V řadě modelů využívajících popis pomocí LP, resp. MILP, jsou omezující podmínky sestaveny např. pro všechny podmnožiny množiny vrcholů uvažovaného grafu (např. i v uvedených modelech přepravních/transportních úloh).

Při velkém množství omezujících podmínek, které roste exponenciálně s počtem vrcholů uvažovaného grafu (počet všech podmnožin n prvkové množiny je 2^n), je jen jejich vygenerování téměř nemožné.

Užitečnou technikou v takovém případě je možnost rozdělení omezujících podmínek do dvou skupin na základní, které jsou součástí statického popisu modelu, a dynamickou, která obsahuje podmínky generované ad hoc. V případě, že stávající řešení nesplňuje tuto podmínku, je podmínka přeřazena do statické skupiny. Přeřazená podmínka tak závisí na aktuálním řešení.

Velmi často tak není nutné uvažovat všechny podmínky modelu, i v případě dopočtení úlohy do optima.

Těmto dynamicky ad hoc dovytvářeným podmínkám se říká odložené podmínky (angl. lazy constraints). S jejich pomocí lze také např. řešit okružní dopravní problém 6.4 s různými kapacitami vozidel (kterému jsme se nevěnovali).

V dalším bude tato technika využita jak v případě, že omezujících podmínek není exponenciálně mnoho (označí se tak podmínky ve statickém popisu modelu), tak v případě dynamicky dogenerovávaných podmínek.

V případě ILP a MILP úloh může být při použití této techniky problémem vůbec nalezení přípustného řešení. Její výhodou je nízká paměťová náročnost výpočtu.

Tento typ techniky není popsán v běžných textech o optimalizaci. Popsány jsou v manuálech např. systémů CPLEX [6] a Gurobi [12].

12.2 Dynamický MILP model pro úlohu CVRP

V této části popíšeme navržený postup s využitím odložených podmínek, který jsme rovněž využili u DFJ modelu úlohy TSP.

Dynamický model DFJ typu Idea je založena na převodu úlohy m-TSP na TSP pomocí vytvoření umělých dep (viz sekce 7). Pro uvažovanou úlohu CVRP odhadneme počet nutných vozidel na základě úlohy o batohu 11.1.1 popsané výše. S tímto počtem vozidel řešíme pak úlohu mTSP s tím, že se při hledání tras kontroluje dodržení kapacit nasazených vozidel na těchto trasách. Pokud je kapacita na určité trase překročena, je sestavena příslušná omezující podmínka a tato trasa je v dalším postupu eliminována.

Máme zadánu úlohu CVRP, tj. je dán ohodnocený graf, vrchol určující depo, jsou zadané poptávky vrcholů, kapacita vozidla (jednotlivých vozidel). Postup je pak následující.

1. Urči řešením úlohy bin-packing (úloha o batohu) nutný počet vozidel.
2. Je-li nutný počet vozidel vyšší než počet dostupných vozidel, skonči.
3. Napočti řešení úlohy m-TSP se získaným počtem vozidel. Výsledné řešení použij jako vstup do úlohy CVRP.

4. Sestav příslušný model DFJ pro rozšířenou matici (získanou přidáním umělých dep k původní úloze) bez SEC podmínek. Toto lze pro neorientovaný i orientovaný graf.

5. Iterativně řeš úlohu pomocí metody větví a mezí.

6. Při nalezení celočíselného řešení ověř:

   - podmínky eliminace podtras (SEC),
   - podmínky dodržení kapacit na podtrasách.

7. V případě porušení přidej podmínky k modelu.

8. Pokračuj v metodě větví a mezí.

Krok 3 je volitelný.

V případě grafu, jehož hranové ohodnocení splňuje trojúhelníkovou nerovnost, by při použití většího než nutného počtu vozidel mělo příslušné řešení vyšší hodnotu. Není tedy v tomto případě třeba měnit počet použitých vozidel (se stejnou kapacitou).

Kroky založené na úloze bin-packing jsou popsané spolu s implementací dynamického programování v sekci 11.1.1.

13 Software pro řešení dopravních optimalizačních úloh

Ke zpracování dat a pro vytvoření programů jsme nejprve zvolili software MATLAB. Z pseudoreálných dat byla vybrána množina, která velikostí odpovídá rozvozům větší firmy se dvěma depy za jeden den. K prvnímu depu je přiřazeno 221 zákazníků a k druhému 361. Jsou známy GPS souřadnice dep i zákazníků a poptávky zákazníků. Na základě těchto údajů byl ke každému skladu vytvořen úplný neorientovaný ohodnocený graf, kde jako cena hrany byla zvolena vzdálenost mezi zákazníky.

Podle tvarů modelů jsme napsali jednotlivé programy, které mají za úkol sestrojit matici $\mathbf{A}$, vektor pravých stran $\mathbf{b}$ a vektory omezující proměnné modelu. Matice jsme odvodili na základě tvarů matic pro malé úlohy, jelikož pokud se dodržuje stejný vzorec vytváření rovnic, mají matice svůj charakteristický tvar. Např. rovnice tvaru $\sum_{j=1}^{n} x_{ij} = 1$, $i = 1, \ldots, n$, $i \neq j$ jsou řazeny

postupně podle i

$$\begin{aligned} x_{12} + x_{13} + \cdots + x_{1n} &= 1 \\ x_{21} + x_{23} + \cdots + x_{2n} &= 1 \\ &\vdots \\ x_{n1} + x_{n2} + \cdots + x_{nn-1} &= 1. \end{aligned}$$

Samotné sestavení matic pro rozsáhlé problémy zabírá mnoho času i paměti, nemluvě o výpočtu řešení. To se projevilo při pokusu spustit programy na osobním počítači HP 250 G3 s procesorem INTEL Pentium (N3530 s frekvencí 2 160 MHz) a pamětí RAM 4GB. Proto byla vyzkoušena varianta, že se sestavené matice vkládají do softwaru Gurobi, který nabízí rozhraní i pro MATLAB, která také není vhodná pro slabší počítače.

Sestrojili jsme programy pro modely symetrického TSP (DFJ, MTZ, GG), m-TSP (m-MTZ), symetrický CVRP a tokový CVRP. Také program dynamického programování pro úlohu KP, který jsme využili pro stanovení minimálního počtu vozidel pro úlohy m-TSP a CVRP. Z heuristik jsme sepsali program pro metodu Clark-Wrighta a Bottom Left Fit heuristiku k řešení 2D BPP. Vzhledem k výsledkům je jasné, že rozhodnutí využívat software MATLAB nebylo úplně šťastné. Vyzkoušeli jsme proto i jiné softwary a programy, které jsou rychlejší a efektivnější. Programy jsme spouštěli na stroji `pilleus` katedry matematiky Západočeské univerzity v Plzni. Jedná se o obecné řešiče Gurobi, CPLEX, MOSEK a řešiče LKH 3 Solver a Concorde Solver, které jsou zaměřené na TSP a jeho zobecnění.

13.1 Gurobi Optimization

Vývojáři Gurobi si dávají za úkol poskytnout co možná nejlepší a nejefektivnější software pro řešení distribučních optimalizačních úloh. Pro studenty je dostupná bezplatná akademická licence. Výhodou je, že Gurobi obsahuje rozšíření pro spolupráci s jinými softwary a programovacími jazyky, jako je C, C++, Java, .NET, Python, MATLAB nebo R. Na webových stránkách [12] se můžeme dočíst, že v benchmarkingových průzkumech vychází Gurobi jako:

- nejrychlejší mezi řešiči lineárního programování,
- nejrychlejší mezi řešiči smíšeného celočíselného programování,
- nejrychlejší při řešení smíšených QP/QC problémů,

- nejrychlejší v detekci neřešitelnosti modelu.

Gurobi obsahuje více než 100 volitelných parametrů, které umožňují upravit výkon podle konkrétního modelu, a také nástroj automatického ladění (Automatic Tuning Tool), který efektivně prozkoumá různá nastavení parametrů a vrátí návrh pro zadaný model.

13.2 CPLEX Optimizer

CPLEX je produktem IBM analytiků [6] a poskytuje flexibilní, vysoce výkonné řešiče lineárního programování, smíšeného celočíselného programování, kvadratického programování a kvadraticky omezeného programování. Při řešení obtížných problémů mohou algoritmy pro smíšené celočíselné programování využívat více procesorů. Mezi kladné vlastnosti patří to, že poskytuje vysoký výkon a škálovatelnost, vyvíjí a zavádí nové optimalizační modely a je dostupný pro testování zdarma přes Community Edition. Zaměřuje se na podnikové rozhodovací procesy, jako operační, taktické a strategické plánování. Lze ho využít ve výrobě, energetice, veřejných službách, financích a v logistice. Uživatel může pracovat v grafickém uživatelském prostředí na míru a má přístup k dalším pokročilým funkcím, jako je prediktivní analýza a strojové učení, aniž by se musel starat o základní technologii optimalizace, více v dokumentaci [7].

13.3 MOSEK

Optimalizační software MOSEK [22] je navržen, aby byl schopen řešit rozsáhlé problémy lineárního, konvexního kvadratického, smíšeného a semidefinitního programování. Velikost problému je limitována pouze velikostí dostupné paměti, nabízí primární a duální simplexovou metodu pro úlohy LP, a metodu větví a mezí a řezů pro smíšené celočíselné programování, disponuje vysoce účinným předřešičem ke snížení velikosti problému před samotnou optimalizací.

Nejsilnější stránkou MOSEKu je nejmodernější optimalizace pomocí metod vnitřního bodu pro spojité lineární, kvadratické a kuželové problémy. Nabízí rozšíření pro C, Javu, .NET, Python, toolbox pro MATLAB, balíček pro R a prostředí pro příkazovou řádku. Neumí řešit nekonvexní úlohy. Zvládá pouze konvexní problémy obsahující jedno nebo více celočíselných omezení.

13.4 LKH Solver

Tento software je efektivní implementací heuristiky Lin-Kernighanova algoritmu pro řešení STSP, viz 10.3. Byl schopen nalézt optimální řešení pro

všechny velké úlohy, na kterých byl zkoušen. Mezi problémy byl i problém 7 397 měst, tj. největší netriviální problém, pro který bylo nalezeno optimální řešení. Navíc dokázal najít lepší řešení u problémů s neznámým optimem, např. problém 85 900. Existují verze 1, 2 a v roce 2017 bylo vydáno rozšíření programu na verzi LKH-3, která je schopná řešit více problémů. Z předchozí verze jsou to problémy STSP, ATSP, problém hamiltonovské kružnice a cesty. Přibyly výše zmíněné problémy m-TSP, CVRP a jejich další rozšíření, více na [14].

13.5 Concorde TSP Solver

Concorde je počítačový program pro řešení STSP a některých příbuzných optimalizačních problémů. Program je napsaný v programovacím jazyce ANSI C a je dostupný pro akademický výzkum. Stejně jako LKH byl vyzkoušen na knihovně vyřešených velkých problémů, kde největší byl problém 85 900 měst. Více na stránkách [2].

14 Praktické výsledky

Výpočty byly prováděny na stroji `pilleus` katedry matematiky Západočeské univerzity v Plzni. Stroj je vybaven 128GB operační paměti a 24 procesory (Inte(R) Xeon(R) CPU E5-2630 v2 s frekvencí 2.60GHz) a systémem Debian GNU/Linux 3.2.86-1.

Vysvětlivka k výsledkům V tabulkách odpovídá označení *barrier* využití metody vnitřního bodu, *dualsimp* využití duální simplexové metody, *primsimp* použití primární simplexové metody a *standard* výchozímu nastavení řešiče. Uvedené metody byly řešičem využity ve všech navštívených vrcholech prohledávacího stromu.

V ostatních případech je použito standardní nastavení parametrů.

Doby výpočtů označené hvězdičkou odpovídají času přerušení výpočtu z důvodu překročení časového limitu.

Poznámka k paralelizaci CPLEX jako jediný z řešičů velmi intenzivně využívá paralelní výpočty, velmi často všechny dostupné procesory. Doby výpočtů u CPLEX odpovídají času výpočtu z pohledu uživatele a nikoli čistému procesorovému času sečtenému ze všech využitých procesorů.

Gurobi a MOSEK využívá paralelizaci ve standardním nastavení velmi jemně, většinou jen jeden procesor. Ostatní programy paralelizaci nevyužívají.

14.1 TSP

Pro úlohu TSP byly otestovány jednotlivé ILP (resp. MILP) modely za použití řešičů Gurobi, CPLEX a MOSEK. Z heuristických metod byly použity CW (implementovaná v práci), LKH a implementace EAX z [10] (ta na základě podmínek využití jen na vybranou přednastavenou instanci). Jako exaktní řešič byl použit software Concorde.

Velikostí úlohy v názvech sekcí se rozumí počet vrcholů uvažovaného ohodnoceného úplného orientovaného grafu.

Postup testování byl následující: v případě využití statického MILP modelu byl tento model vygenerován v prostředí MATLAB. Model byl pak řešen uvedeným příslušným řešičem přímo. U dynamicky generovaného modelu bylo nutné využít programovací jazyk C++, jelikož v prostředí MATLAB není možné dynamicky s řešiči pracovat.

Poznámka k DFJ modelu Tento model obsahuje exponenciální množství omezení (podmínky eliminace podtras). Z tohoto důvodu byla využita technika odložených omezení (lazy constraints), při které se podmínky generovaly dynamicky dle potřeby. Nebyl tedy sestaven přímo statický model ILP.

Při této dynamické technice není možné standardní řešiče používat jako „černou skříňku“. Tuto techniku tak bylo nutné implementovat v C++ s interaktivním voláním řešiče Gurobi. Vzhledem k náročnosti nebyly zatím zkoušeny ostatní řešiče.

Tento přístup u DFJ modelu úlohy TSP byl využit u všech dále uvedených testovaných instancí.

Byla zde zároveň testována různá nastavení s ohledem na rozšiřující testy tohoto přístupu u CVRP modelu.

14.1.1 Velikost 37

Metoda	Výsledné řešení	Doba výpočtu
MTZ/Gurobi	150	144,77s
MTZ/CPLEX	150	118,62s
MTZ/Mosek	150	31 786,0s
GG/Gurobi	150	2,43s
GG/CPLEX	150	23,34s
GG/Mosek	150	9 939,1s
DFJ/Gurobi+standard	150	0,57s
DFJ/Gurobi+barrier	150	0,43s
DFJ/Gurobi+dualsimp	150	0,16s
DFJ/Gurobi+primsimpl	150	0,15s
CW	151	0,09s
LKH-3	150	42,5s
Concorde	150	0,14s

Získaná řešení s hodnotou 150 jsou optimální (prokázáno řešičem Gurobi, CPLEX a Concorde).

14.1.2 Velikost 74

Metoda	Výsledné řešení	Doba výpočtu
MTZ/Gurobi	194	10h*
MTZ/CPLEX	194	10h*
MTZ/Mosek	217	10h*
GG/Gurobi	194	35,92s
GG/CPLEX	194	27,52s
GG/Mosek	211	10h*
DFJ/Gurobi+standard	194	2,67s
DFJ/Gurobi+barrier	194	2,95s
DFJ/Gurobi+dualsimp	194	2,00s
DFJ/Gurobi+primsimpl	194	2,33s
CW	208	0,37s
LKH-3	194	164,5s
Concorde	194	0,21s

Získaná řešení s hodnotou 194 jsou optimální (prokázáno řešičem Gurobi, CPLEX u modelu DFJ a Concorde).

U modelu MTZ bylo řešiči Gurobi a CPLEX nalezeno optimální řešení, ale v uvedeném časovém limitu nebyla prokázána jeho optimalita.

14.1.3 Velikost 111

Metoda	Výsledné řešení	Doba výpočtu
MTZ/Gurobi	243	10h*
MTZ/CPLEX	239	10h*
MTZ/Mosek	287	10h*
GG/Gurobi	239	97,54s
GG/CPLEX	239	432,10s
GG/Mosek	278	10h*
DFJ/Gurobi+standard	239	9,43s
DFJ/Gurobi+barrier	239	57,90s
DFJ/Gurobi+dualsimp	239	4,17s
DFJ/Gurobi+primsimpl	239	9,36s
CW	257	0,66s
LKH-3	239	684,10s
Concorde	239	0,94s

Získaná řešení s hodnotou 239 jsou optimální (prokázáno řešičem Gurobi, CPLEX a Concorde).

14.1.4 Velikost 148

Metoda	Výsledné řešení	Doba výpočtu
MTZ/Gurobi	303	10h*
MTZ/CPLEX	286	10h*
MTZ/Mosek	317	10h*
GG/Gurobi	286	374,11s
GG/CPLEX	286	962,86s
GG/Mosek	354	10h*
DFJ/Gurobi+standard	286	144,64s
DFJ/Gurobi+barrier	286	330,78s
DFJ/Gurobi+dualsimp	286	75,71s
DFJ/Gurobi+primsimpl	286	25,48s
CW	304	1,72s
LKH-3	286	893,1s
Concorde	286	1,74s

Získaná řešení s hodnotou 286 jsou optimální (prokázáno řešičem Gurobi, CPLEX a Concorde). U modelu MTZ CPLEX nalezl optimální řešení, ale neprokázal jeho optimalitu.

14.1.5 Velikost 185

Metoda	Výsledné řešení	Doba výpočtu
MTZ/Gurobi	343	10h*
MTZ/CPLEX	334	10h*
MTZ/Mosek	421	10h*
GG/Gurobi	325	702,35s
GG/CPLEX	325	5 096,83s
GG/Mosek	398	10h*
DFJ/Gurobi+standard	325	94,31s
DFJ/Gurobi+barrier	325	595,77s
DFJ/Gurobi+dualsimp	325	75,03s
DFJ/Gurobi+primsimpl	325	54,97s
CW	344	3,60s
LKH-3	325	2 401,9s
Concorde	325	1,32s

Získaná řešení s hodnotou 325 jsou optimální (prokázáno řešičem Gurobi, CPLEX a Concorde).

14.1.6 Velikost 222

Metoda	Výsledné řešení	Doba výpočtu
MTZ/Gurobi	393	10h*
MTZ/CPLEX	380	10h*
MTZ/Mosek	531	10h*
GG/Gurobi	369	1 531,09s
GG/CPLEX	369	16 626,84s
GG/Mosek	483	10h*
DFJ/Gurobi+standard	369	111,51s
DFJ/Gurobi+barrier	369	1 334,18s
DFJ/Gurobi+dualsimp	369	125,84s
DFJ/Gurobi+primsimpl	369	232,74s
CW	392	7,01s
LKH-3	369	2 815,9s
Concorde	369	2,95s

Získaná řešení s hodnotou 369 jsou optimální (prokázáno řešičem Gurobi a Concorde). Paměťové nároky se pohybovaly u řešiče Gurobi při DFJ modelu zhruba okolo 110MB, u LKH-3 okolo 5MB a u Concorde 3MB.

Sestavení modelu GG v MATLABu zabralo 80GB, sestavení modelu MTZ 35GB.

14.1.7 Velikost 362

Vzhledem k dlouhým výpočetním časům a špatným výsledným hodnotám, nejsou již uváděny modely MTZ a GG.

Metoda	Výsledné řešení	Doba výpočtu
DFJ/Gurobi+standard	536	1 914,43s
DFJ/Gurobi+barrier	536	1 483,09s
DFJ/Gurobi+dualsimp	536	1 948,5s
DFJ/Gurobi+primsimpl	536	1 038,19s
CW	570	41,90s
LKH-3	536	4 241,4s
Concorde	536	7,19s

Získaná řešení s hodnotou 536 jsou optimální. LKH-3 je schopen nalézt optimální řešení v čase 0,55s.

14.1.8 Velikost 532

Následující instance `att532` je převzata z TSPlib [20]. Slouží pro srovnání programu LKH-3, dostupné implementace využívající operátor křížení EAX a program Concorde. Úloha byla pro srovnání vyřešena i DFJ modelem.

Metoda	Výsledné řešení	Doba výpočtu
DFJ/Gurobi+standard	27686	16 284,94s
EAX	27686	105,65s
LKH-3	27868	184,8s
Concorde	27686	38,94s

U EAX a LKH-3 je uveden celkový čas pro 10 pokusů. Program Concorde zároveň v uvedeném čase prokázal optimalitu řešení (a rovněž řešič Gurobi u DFJ modelu). Modely MTZ a GG již nebyly testovány z důvodu nedosažení rozumných výsledků u menších instancí.

14.1.9 Komentář k výsledkům

Z tabulek je vidět, že z klasických (DFJ, GG, MTZ) modelů si při vzrůstající velikosti úlohy TSP nejlépe vedl DFJ model, který ve své formulaci má

exponenciální počet omezení. Jako metoda jeho řešení byla ovšem zvolena technika odložených podmínek, která tuto nevýhodu velmi zdatně odstraňuje. Tímto modelem a přístupem tak lze do optimality řešit i středně velké úlohy TSP se stovkami míst. U MTZ modelu, který ačkoli má polynomiální počet omezení, byla dokonce i teoreticky dokázána značná slabost jeho LP relaxace [11]. Ta je zde podpořena velmi dlouhými výpočetními dobami. U GG modelu je rovněž vidět exponenciální nárůst výpočetní doby.

Tyto problémy s klasickými modely se v poslední době daří úspěšně odstranit metodou generování sloupců, která je ovšem implementačně náročná.

Při nutnosti rychlejšího získání kvalitního výsledku je tak možné sáhnout po velmi sofistikovaných heuristických metodách. Jednou z nejuznávanějších je pak metoda programu LKH-3. Další možností je využití evolučních metod využívajících operátor křížení EAX. Pokud je třeba získat přípustné řešení rychle, je možné volit např. heuristiku CW, která ovšem nedává řešení blízké optimálnímu.

Jako ideální řešič se však pro takto veliké úlohy ukázal program Concorde. Ten je založený na sofistikovaném propojení heuristik s řešičem ILP a dává tak zároveň informaci o optimalitě řešení. Zároveň využívá zanedbatelné množství paměti, pro uvedené úlohy v řádu MB.

14.2 m-TSP

Úlohy m-TSP byly řešeny při využití ILP, resp. MILP modelů převodem na úlohu TSP uvedeným v části 7.

Jak bylo uvedeno výše, je možné u úlohy m-TSP uvažovat dva přístupy:

1. minimalizace celkových nákladů (počet vozidel je pak proměnná s horním omezením daným dostupným počtem vozidel)

2. minimalizace nákladů při předepsaném fixním počtu vozidel.

Použité matice vzdáleností (cen) splňují trojúhelníkovou nerovnost. Tedy, jak bylo dokázáno v části 8.1, při řešení úlohy minimalizace celkových nákladů bychom dostali jako výsledek odpovídající řešení úlohy TSP uvedené v předchozí sekci.

Náš přístup byl proto následující. Vzhledem k řešení úlohy CVRP na stejných grafech (se zadanými kapacitami a poptávkami navíc), se určil minimální počet vozidel k řešení úlohy CVRP pomocí implementace úlohy KP (výpočet proběhl v řádu sekund) a tento se použil v odpovídajících úlohách m-TSP uvedených dále. Získaná optimální řešení úloh m-TSP pak představují dolní odhad na hodnotu řešení úlohy CVRP.

Poznámka Z hodnocení byl vynechán řešič MOSEK, který alespoň v základním nastavení u TSP nepodával dobrá řešení. Čas uvedený u LKH odpovídá celkové době 10ti opakování.

Clarke Wrightova metoda Při použití této metody na matici vzdáleností upravenou podle převodu m-TSP na TSP (přidáním potřebného počtu kopií depa) vycházely nerealistické výsledky. Nejsou proto uváděny. Např. pro úlohu velikosti 37 s 8 depy a ohodnocením hran mezi depy 100 000 vyšla délka 600 171.

14.2.1 Velikost 37

Zde bylo použito 8 vozidel.

Metoda	Výsledné řešení	Doba výpočtu
m-MTZ/Gurobi	185	26,80s
m-MTZ/CPLEX	185	14,11s
DFJ/Gurobi+standard	185	0,09s
DFJ/Gurobi+barrier	185	0,29s
DFJ/Gurobi+dualsimp	185	0,08s
DFJ/Gurobi+primsimpl	185	0,07s
LKH-3	185	343,6s
Concorde	185	0,40s

14.2.2 Velikost 74

Zde bylo využito 8 vozidel.

Metoda	Výsledné řešení	Doba výpočtu
m-MTZ/Gurobi	216	4 148,74s
m-MTZ/CPLEX	216	7 343,83s
DFJ/Gurobi+standard	216	0,81s
DFJ/Gurobi+barrier	216	9,28s
DFJ/Gurobi+dualsimp	216	1,10s
DFJ/Gurobi+primsimpl	216	0,63s
LKH-3	216	517,7s
Concorde	216	0,58s

14.2.3 Velikost 111

Zde bylo využito 8 vozidel.

Metoda	Výsledné řešení	Doba výpočtu
m-MTZ/Gurobi	255	10h*
m-MTZ/CPLEX	255	10h*
DFJ/Gurobi+standard	255	7,75s
DFJ/Gurobi+barrier	255	32,39s
DFJ/Gurobi+dualsimp	255	8,57s
DFJ/Gurobi+primsimpl	255	6,71s
LKH-3	255	993,8s
Concorde	255	1,11s

14.2.4 Velikost 148

Zde bylo využito 9 vozidel.

Metoda	Výsledné řešení	Doba výpočtu
m-MTZ/Gurobi	310	10h*
m-MTZ/CPLEX	309	10h*
DFJ/Gurobi+standard	305	63,15s
DFJ/Gurobi+barrier	305	214,49s
DFJ/Gurobi+dualsimp	305	67,74s
DFJ/Gurobi+primsimpl	305	39,56s
LKH-3	305	1 035,9s
Concorde	305	1,71s

14.2.5 Velikost 185

Zde bylo použito 10 vozidel.

Metoda	Výsledné řešení	Doba výpočtu
m-MTZ/Gurobi	348	10h*
m-MTZ/CPLEX	351	10h*
DFJ/Gurobi+standard	347	215,52s
DFJ/Gurobi+barrier	347	352,42s
DFJ/Gurobi+dualsimp	347	232,34s
DFJ/Gurobi+primsimpl	347	85,65s
LKH-3	347	3 517,8s
Concorde	347	2,64s

Řešení s hodnotou 347 je optimální.

14.2.6 Velikost 222

Zde bylo použito 12 vozidel.

Metoda	Výsledné řešení	Doba výpočtu
m-MTZ/Gurobi	414	10h*
m-MTZ/CPLEX	403	10h*
DFJ/Gurobi+standard	395	37,82s
DFJ/Gurobi+barrier	395	895,95s
DFJ/Gurobi+dualsimp	395	39,07s
DFJ/Gurobi+primsimpl	395	72,19s
LKH-3	395	3 752,5s
Concorde	395	4,90s

U modelu m-MTZ zabralo jeho vytváření v MATLABu 30GB paměti. Řešení s hodnotou 395 je optimální.

14.2.7 Velikost 362

Model m-MTZ se kvůli paměťovým nárokům MATLABu nepodařilo vygenerovat.

Bylo použito 18 vozidel.

Metoda	Výsledné řešení	Doba výpočtu
DFJ/Gurobi+standard	596	2 531,31s
DFJ/Gurobi+barrier	596	26 270,18s
DFJ/Gurobi+dualsimp	596	2 520,15s
DFJ/Gurobi+primsimpl	596	2 466,70s
LKH-3	596	42 016,8s
Concorde	596	71,90s

Řešení s hodnotou 596 jsou optimální (prokázáno Gurobi a Concorde).

Čas u LKH je uveden pro 10 opakování, ale trasa s délkou 596 byla nalezena již po 2,16s v prvním běhu.

14.2.8 Komentář k výsledkům

Jelikož vstupní matice splňuje trojúhelníkovou nerovnost (a hrany mají nenulové ohodnocení), jsou hodnoty optimálních hodnot úlohy m-TSP vždy vyšší

než u odpovídající úlohy TSP. Hodnota optima m-TSP s daným počtem vozidel je pak dolním odhadem na optimum příslušné CVRP úlohy.

Pro řešení LP relaxace ILP úlohy se jako vhodná metoda jeví metoda vnitřního bodu. Oproti celkové době řešení úlohy ILP je doba řešení její LP relaxace zanedbatelná.

Jako vhodná metoda pro ostatní vrcholy výpočetního stromu se prokázala klasická simplexová metoda nebo duální simplexová metoda.

14.3 CVRP

Jako model ILP formulace úlohy je použit tokový model. Důvodem je, že jeho formulace není oproti jiným modelům exponenciální a dále nevyžaduje (aspoň v prvním přiblížení použitém zde) pokročilé metody ILP (metoda generování sloupců).

U úlohy rozměru 222 má ILP formulace cca 50 tisíc omezení a 50 tisíc celočíselných (0/1) proměnných.

U LKH metody bylo provedeno vždy 10 restartů metody, u každého s 10 000 iteracemi.

Vzhledem k dlouhým výpočetním časům bylo u použitých řešičů CPLEX a Gurobi (v tabulkách níže uvedený jako GRB) využívány pouze přednastavené hodnoty parametrů. Výpočetní časy nejsou uvedeny přímo v tabulkách, neboť u některých přístupů byl využit ruční zásah v průběhu výpočtů.

Námi implementovaná metoda dynamického MILP modelu je označena jako dynMILP. Metoda funguje v přijatelném čase s ukázáním optimality pro grafy do zhruba 90ti vrcholů.

Kombinace CPLEX+GRB znamená použití výsledného řešení z řešiče CPLEX jako vstupního řešení pro Gurobi. Obdobně kombinace LKH+GRB znamená vložení výsledku z LKH do řešiče Gurobi jako počáteční řešení.

Od velikosti úloh 111 skončily řešiče CPLEX a Gurobi na nedostatek paměti.

Poznámka k SCVRP modelu U modelu SCVRP (symetrický CVRP) je počet omezení exponenciální. Bohužel se nepodařilo v časových limitech (4 týdny) vygenerovat jeho popis v MATLABu ani pro velikost úlohy 37 (pouze pro ilustrativní příklad 1 velikosti 10), výsledky proto nejsou uvedeny. Technika odložených podmínek u něj nebyla z časových důvodů implementována.

14.3.1 Velikost 37

CW	GRB	CPLEX	CPLEX+GRB	LKH	LKH+GRB	Dolní odhad	dynMILP
257	256	256	256	256	256	240 (GRB)	256

Výpočetní čas se pohyboval okolo 10 minut u ILP formulace (CPLEX, Gurobi), u LKH (10 pokusů) 2,8 hodiny – každý pokus zde nalezl optimální trasu. Následné prokázání optimality řešení s hodnotou 256 pomocí Gurobi trvalo zhruba 4 týdny. Námi implementovaná metoda dynamického MILP modelu zabrala 99,37s a zároveň prokázala optimalitu. Řešení CW zabralo 0,23s.

Ve všech řešeních bylo využito 8 vozidel.

14.3.2 Velikost 74

CW	GRB	CPLEX	CPLEX+GRB	LKH	LKH+GRB	Dolní odhad	dynMILP
314	315	325	319	306	306	219 (GRB)	306

Doba výpočtu LKH byla 12,1 hodin. Výpočty CPLEX a Gurobi skončily po zhruba 3 dnech na nedostatek paměti. Metoda dynMILP zabrala 6,5 dne spolu s ukázáním optimality řešení. Výpočet CW zabral 0,74s.

V řešeních bylo využito 8 vozidel.

14.3.3 Velikost 111

CW	GRB	CPLEX	CPLEX+GRB	LKH	LKH+GRB	Dolní odhad
366	379	410	359	340	340	241 (GRB)

Doba výpočtu LKH byla 21,6 hodin. Výpočet CW zabral 3,04s.

V řešeních bylo využito 8 vozidel.

14.3.4 Velikost 148

CW	GRB	CPLEX	CPLEX+GRB	LKH	LKH+GRB	Dolní odhad
440	501	480	453	412	412	270 (GRB)

Doba výpočtu LKH byla 42,2 hodin (cca 1,8 dne). Výpočet CW trval 6,98s.

V řešeních bylo využito 9 vozidel.

14.3.5 Velikost 185

CW	GRB	CPLEX	CPLEX+GRB	LKH	LKH+GRB	Dolní odhad
483	648	493	476	460	460	335 (GRB)

Doby výpočtu u LKH byla 67,3 hodin (cca 2,8 dne). Výpočet CW zabral 13,84s.

V řešeních bylo využito 10 vozidel.

14.3.6 Velikost 222

CW	GRB	CPLEX	CPLEX+GRB	LKH	LKH+GRB	Dolní odhad	LKH+$(2,2)$-opt
552	557	561	559	516	515	378 (GRB)	515

Paměť využitá při GRB nebo CPLEX 40GB, čas výpočtu LKH+GRB 470695s (cca 5,5 dne). Doba výpočtu LKH byla 89,5 hodin (cca 3,7 dne). Výpočet CW zabral 25,95s.

V řešeních bylo využito 12 vozidel.

Poznamenejme, že ačkoli pro LKH bylo využito vždy 10 restartů, v každém z nich bylo nalezeno řešení s hodnotou velmi blízkou uvedené nejlepší hodnotě, speciálně vždy bylo lepší než nejlepší řešení nalezené pomocí ILP formulace.

Rozbor výsledku programu LKH-3 Řešení získané programem LKH-3 bylo analyzováno na λ-optimalitu. Ve výsledku bylo 12 tras odpovídajících použití 12ti vozidel. Na množině vrcholů každé trasy s přidaným depem byla zvlášť řešena úloha TSP. Cílem bylo rozhodnout, zda na příslušných množinách vrcholů neexistuje jiná trasa s délkou kratší než získanou z LKH-3.

Trasa	Řešení TSP
1	28
2	46
3	48
4	35
5	65
6	18
7	7
8	65
9	69
10	79
11	2
12	54
Délka celkem	516

Řešení získané z LKH-3 má délku 516, je tedy λ-optimální (dokonce pro všechny možné hodnoty λ) na každé z uvažovaných podmnožin (podtras). Ovšem není α-optimální (dokonce pro $\alpha = 2$), jak je vidět z výsledků (LKH+GRB našlo řešení o 1 lepší).

14.3.7 Velikost 362

Zde uvedeme pouze heuristiky CW a LKH.

CW	LKH-3+GPX2	LKH-3+IPX
955	897	905

GPX2 a IPX jsou dva možné operátory křížení tras, které program LKH rovněž umožňuje využívat. U všech předchozích výpočtů je použit IPX operátor.

Bylo opět použito 10 restartů. Doby výpočtu byly 5729821,5s (66,3 dne) u GPX2, 5194198,4s (60,1 dne) u IPX, tj. zhruba 6,5 dne na jeden běh programu. Výpočet CW zabral 128,39s.

Ačkoli jsou zde výpočetní doby LKH značné, velmi dobré řešení (lepší než CW) je nalezeno velmi rychle již v první fázi běhu (po několika sekundách); zbylá doba je věnována jeho vylepšování.

Počet použitých vozidel byl vždy 18.

14.3.8 Komentář k výsledkům

Jak je vidět z výsledků (doby výpočtů u použitých modelů v řádech dnů, resp. pouze hrubý odhad z metody CW), úloha CVRP je daleko obtížnější

než úloha m-TSP nebo TSP. Při výpočtech je třeba vzít do úvahy, že požadavek na zlepšení dobrého řešení znamená velké množství dodatečných výpočtů, případně kombinaci velmi sofistikovaných metod (to je dáno NP-těžkostí úlohy).

14.4 Poznámky k použitým řešičům

Během testování se ukázalo několik předností a slabostí používaných řešičů:

- CPLEX si vedl lépe v počátečních fázích optimalizace než Gurobi, hlavně při celočíselných koeficientech matice omezení (rovnosti i nerovnosti).
- CPLEX většinou dával nepatrně lepší dolní odhad na optimální řešení (při minimalizačních úlohách).
- Gurobi využívá méně paměti, méně využívá paralelizaci, ale celková kvalita řešení a čistá doba výpočtu je lepší než u CPLEX (čistou dobou výpočtu myslíme součet výpočetních časů na všech využitých procesorech).
- MOSEK alespoň v základním nastavení nedával příliš dobrá řešení; využívá ovšem zároveň velmi málo operační paměti (řádově pouze jednotky GB i pro rozsáhlé úlohy).
- LKH-3 našel v téměř všech případech optimální řešení v prvních několika sekundách.

Oba řešiče Gurobi a CPLEX po dlouhé době výpočtu začaly vyčerpávat operační paměť (desítky GB), což byl důvod k použité metodě restartu těchto řešičů.

Při řešení úloh ILP vzniklých z tokového modelu výpočty u rozsáhlejších úloh zahrnovaly průchod cca 10 miliony vrcholů prohledávacího stromu (metody mezí a větví) a až cca 300 milionů iterací simplexové metody nebo metody vnitřního bodu.

Na opačné straně výpočetní náročnosti je pak implementovaná metoda CW (Clarka-Wrighta), která i při relativní jednoduchosti dává aspoň hrubý horní odhad výsledných délek.

14.5 Interpretace proměnných z modelu

Program Gurobi si sám vytváří proměnné ve tvaru C_q, $q = 1, \ldots, n$, z kterých není na první pohled jasné, o kterou hranu se jedná. V modelech, kde pracujeme s $n(n-1)$ proměnnými, tj. s plnou symetrickou maticí, lze převést tyto proměnné na proměnné ve tvaru x_{ij} následujícím postupem:

$$t = \left\lfloor \frac{q}{n-1} \right\rfloor, \quad l = q \mod n-1,$$

- $t > l$

$$i = t + 1 \quad j = l + 1$$

,

- $t \leq l$

$$i = t + 1 \quad j = l + 2$$

.

14.6 2D BPP

Pro problém 2D BPP byla v prostředí MATLAB implementována heuristika Bottom Left Fit, která byla spuštěna na osobním počítači a výpočet proběhl v čase 0.1985s. V následující tabulce je uvedeno číslo výrobku, jeho rozměry, počet kusů k naložení, obsah plochy 1 kusu k naložení a počet palet, na kterých se nachází daný výrobek.

Výrobek	Výška [mm]	Šířka [mm]	Délka [mm]	Kusů	Obsah š x d	Palet
1089816	400	381	381	1	145 161	1
1089548	400	381	381	1	145 161	1
1089393	255	363	363	4	131 769	1
1091544	180	432	312	9	134 784	3
1089770	290	400	300	20	120 000	4
1091605	290	400	300	20	120 000	4
1091466	290	400	300	80	120 000	14
1091504	290	400	300	20	120 000	4
1089561	290	400	300	20	120 000	4
1089841	171	404	268	72	108 272	18
1091483	115	404	268	27	108 272	7
1091665	171	404	268	4	108 272	2
1091668	171	404	268	4	108 272	2
1091666	171	404	268	4	108 272	2
1091477	171	404	268	9	108 272	3
1092329	171	404	268	144	108 272	36
1089573	171	404	268	9	108 272	3
1091481	171	404	268	9	108 272	3
1089516	171	404	268	9	108 272	3
1091511	120	400	265	1	106 000	1
1089530	330	300	200	16	60 000	2
1092200	229	255	187	18	47 685	2
1092196	229	255	187	18	47 685	2
1092199	229	255	187	36	47 685	4
1092197	229	255	187	18	47 685	2
1089927	325	264	176	76	46 464	6
1091484	271	278	142	20	39 476	2

V další tabulce je uvedeno z kolika procent jsou palety zaplněny.

Paleta	Zaplnění [%]	Paleta	Zaplnění [%]	Paleta	Zaplnění [%]	Paleta	Zaplnění [%]
1	85,15	30	75,00	59	45,11	88	45,11
2	56,16	31	45,11	60	45,11	89	45,11
3	56,16	32	45,11	61	45,11	90	45,11
4	64,04	33	45,11	62	45,11	91	45,11
5	75,00	34	45,11	63	45,11	92	45,11
6	75,00	35	45,11	64	45,11	93	45,11
7	75,00	36	45,11	65	45,11	94	45,11
8	75,00	37	45,11	66	45,11	95	45,11
9	75,00	38	45,11	67	45,11	96	45,11
10	75,00	39	45,11	68	45,11	97	45,11
11	75,00	40	45,11	69	45,11	98	45,11
12	75,00	41	45,11	70	45,11	99	45,11
13	75,00	42	45,11	71	45,11	100	45,11
14	75,00	43	45,11	72	45,11	101	45,11
15	75,00	44	45,11	73	45,11	102	45,11
16	75,00	45	45,11	74	45,11	103	76,13
17	75,00	46	45,11	75	45,11	104	93,59
18	75,00	47	45,11	76	45,11	105	79,48
19	75,00	48	45,11	77	45,11	106	79,48
20	75,00	49	45,11	78	45,11	107	79,48
21	75,00	50	45,11	79	45,11	108	79,48
22	75,00	51	45,11	80	45,11	109	79,48
23	75,00	52	45,11	81	45,11	110	78,08
24	75,00	53	45,11	82	45,11	111	77,44
25	75,00	54	45,11	83	45,11	112	77,44
26	75,00	55	45,11	84	45,11	113	77,44
27	75,00	56	45,11	85	45,11	114	77,44
28	75,00	57	45,11	86	45,11	115	66,52
29	75,00	58	45,11	87	45,11	116	20,56

15 Závěr

Jedním z našich cílů bylo vyzkoušet jeden ze složitějších modelů, který se využívá v přepravní logistice, a to okružní dopravní problém s kapacitami (CVRP).

Úloha CVRP vzniká spojením dvou optimalizačních problémů: TSP a KP (resp. BPP). Je možné na ni pak pohlížet jako na „dvourozměrnou“ úlohu, kdy jedním rozměrem jsou vzdálenosti (TSP část) a druhým kapacity (KP část).

Mezistupněm mezi TSP a CVRP je úloha m-TSP. Při té se uvažuje vícero

obchodních cestujících, ale neuvažují se kapacity vozidel a poptávky míst. Úloha se dá převést na úlohu TSP s počtem vrcholů daným součtem počtu původních míst a počtu vozidel m, kde depo rozdělíme na m vrcholů se stejnými souřadnicemi. Z praktického pohledu je tedy situace podobná jako u TSP.

Nejprve jsou zpracovány modely úlohy CVRP založené na ILP, resp. MILP. Jejich přímé řešení na zvolených středně velkých úlohách je bez použití pokročilých technik velmi zdlouhavé. Z praktického pohledu je rovněž problém, že dnešní, i když velmi pokročilé, řešiče úloh LP a ILP, nenabízejí přímo možnost využití dekompozičních metod u LP a je nutné je donaprogramovat použitím vestavěných funkcí.

Jako doplněk klasických modelů byl v diplomové práci zpracován jednoduchý model úlohy CVRP založený na dynamické formulaci úlohy. Inspirací zde byl program Concorde, který je dosud již zhruba 25 let nejúspěšnějším exaktním řešičem pro úlohu TSP. Zaveden a využit byl pojem (λ, α)-optimality pro úlohu CVRP. Uvedeno je srovnání řešení vybraných modelů na různě velikých úlohách pomocí dostupného softwaru pro úlohy MILP.

V oblasti úlohy KP byl zpracován přístup založený na dynamickém programování a v případě BPP heuristika (Bottom Left Fit) pro úlohu 2D BPP.

Všechny uvedené úlohy jsou tzv. NP-těžké. Z pohledu řešitelnosti je možné při stávajícím stupni poznání řešit úlohy TSP a m-TSP (díky převodu na TSP) do optimality pro grafy s desetitisíci vrcholy, viz knihovna TSP problémů [20]. U úlohy CVRP je zatím situace několikařádově horší, i přes potenciálně možný převod na TSP stejnou technikou jako m-TSP. Do optimality se dají řešit úlohy jen s málo desítkami až stovkami vrcholů.

Úlohy KP a BPP jsou z teoretického pohledu dobře aproximovatelné a rovněž řešitelné z praktického pohledu, např. dynamickým programováním. Pro úspěšné řešení a hlavně prokazování optimality je nutné používat pokročilé a sofistikované metody lineárního a celočíselného programování.

Od počátku byly nejen pro úlohu CVRP vyvíjeny heuristiky. Některé z nich jsou založeny na výše zmíněné „dvourozměrnosti“ úlohy, např. Fisherův-Jaikumarův algoritmus [9]. Většina z nich používá techniky lokálního prohledávání, případně v kombinaci s evolučními metodami křížení (crossover) a mutací (mutation).

Těchto heuristik existuje téměř nepřeberné množství. Diplomová práce se v této oblasti soustředila na výběr těch několika, které se ukázaly jako úspěšné a přinesly nové pohledy nebo nové techniky. Jedná se o metodu Clarke-Wrighta, metody využívající Lin-Kernighanův algoritmus nebo operátor EAX.

Praktické aplikace mají často vlastnost metriky (případně „téměř metriky“), je tedy zohledněn i přístup k úloze euklidovského TSP a euklidovského

CVRP a jsou zpracovány základní vlastnosti a vztah řešení těchto úloh.

V praktickém nasazení optimalizace je nutné využívat kombinaci různých přístupů v závislosti na požadavcích rychlosti/optimálnosti řešení. Pro uvažované typy úloh TSP, m-TSP a CVRP lze z hlediska rychlosti doporučit LKH-3, resp. příslušné metody, neboť program je pouze pro akademické využití.

Z hlediska ověření optimálnosti lze pro středně velké úlohy TSP, m-TSP doporučit model DFJ s využitím odložených podmínek. Pro rozsáhlé úlohy pak program Concorde, který je rovněž pouze pro akademické využití. Rychlost výpočtů je samozřejmě také ovlivněna typem procesoru a velikostí operační paměti použitého stroje.

U úlohy KP, BPP (2D-BPP, 3D-BPP) lze pak využít dynamické programování a příslušné 2D a 3D pakovací heuristiky.

Reference

[1] P. Berka, Přednáška Evoluční algoritmy, Vysoká škola ekonomická v Praze, [online]. [cit. 2019-7-20]. Dostupné z: https://sorry.vse.cz/-berka/docs/izi456/kap_5.5.pdf.

[2] Concorde TSP Solver, [online]. [cit. 2019-7-20]. Dostupné z: http://-www.math.uwaterloo.ca/tsp/concorde/index.html.

[3] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, A. Schrijver, *Combinatorial Optimization*, John Wiley & Sons, New York, NY, 1998. ISBN: 9780471558941.

[4] J.-F. Cordeau, G. Laporte, M. W.P. Savelsbergh, D. Vigo, *Chapter 6 Vehicle Routing*, Editor(s): C. Barnhart, G.Laporte, Handbooks in Operations Research and Management Science, Elsevier, Volume 14, 2007, Pages 367-428. ISBN: 9780444513465.

[5] T. H. Corman, Ch. E. Leiserson, R. L. Rivest, *Introduction to Algorithms*, 2. vydání, Cambridge: MIT press, 1989. ISBN: 0-262-03141-8.

[6] CPLEX Optimizer, [online]. [cit. 2019-7-20]. Dostupné z: https://-www.ibm.com/analytics/cplex-optimizer.

[7] Dokumentace CPLEX Optimizer, [online]. [cit. 2019-20-7]. Dostupné z: https://www.ibm.com/downloads/cas/V1WDO6OR.

[8] G. B. Dantzig, *Linear Programming and Extensions*, 2. vydání s korekcemi, Princeton University Press, Princeton, NJ, 1965. ISBN: 0-691-08000-3.

[9] Fisherův-Jaikumarův algoritmus, [online]. [cit. 2019-7-29]. Dostupné z: http://neo.lcc.uma.es/vrp/solution-methods/heuristics/cluster-first-route-second-method/#FisherJaikumar.

[10] Genetický algoritmus pro TSP [online]. [cit. 2019-7-29]. Dostupné z: https://github.com/sugia/GA-for-TSP.

[11] S. Gollowitzer, I. Ljubić, *MIP models for connected facility location: A theoretical and computational study*, Computers & Operations Research 38 (2), 2011, 435-449. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0305054810001334

[12] Gurobi Optimization, [online]. [cit. 2019-7-20]. Dostupné z: www.gurobi.com.

[13] K. Helsgaun, *An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic*, Roskilde University, Dánsko, 1998. Dostupné z: http://akira.ruc.dk/~keld/research/LKH/LKH-2.0/DOC/LKH_REPORT.pdf.

[14] K. Helsgaun, *An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Traveling Salesman and Vehicle Routing Problems*, Roskilde University, Dánsko, 2017, [online]. [cit. 2019-7-20]. Dostupné z: http://akira.ruc.dk/ keld/research/LKH-3/.

[15] Pojem heuristika, [online]. [cit. 2019-7-20]. Dostupné z: https://encyklopedie.soc.cas.cz/w/Heuristika.

[16] N. Christofides, *The vehicle routing problem*, RAIRO - Operations Research - Recherche Opérationnelle, Tome 10 (1976) no. V1, pp. 55-70, [online]. [cit. 2019-7-20]. Dostupné z: http://www.numdam.org/item/RO_1976__10_1_55_0/.

[17] V. Chvátal, *Linear Programming*, W. H. Freeman and Co., New York, NY, 1983. ISBN: 0716715872.

[18] O. H. Ibarra, C. E. Kim, *Fast approximation algorithms for the knapsack and sum of subset problems*, Journal of the ACM, 1975, 22:463-468.

[19] H. Karloff, *Linear Programming*, Birkhäuser, Boston, MA, 1991. ISBN: 3-7643-3561-0.

[20] Knihovna úloh TSP [online]. [cit. 2019-28-7]. Dostupné z: https://-www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/.

[21] R. Matai, S. P. Singh, M. L. Mittal, *Traveling Salesman Problem: An Overview of Applications, Formulations, and Solution Approaches*,InTech, 2010. ISBN: 978-953-307-426-9.

[22] MOSEK optimization software, [online]. [cit. 2019-7-20]. Dostupné z: https://www.mosek.com/products/mosek/.

[23] P. Munari, T. Dollevoet, R. Spliet, *A generalized formulation for vehicle routing problems*, arXiv:1606.01935v2 [math.OC], září 2017.

[24] Ch. H. Papadimitriou, K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, 1982. ISBN: 0-13-152462-3.

[25] P. Pavlíková, *Z historie lineárního programování - George B. Dantzig (1914-2005)*, Pokroky matematiky, fyziky a astronomie, Vol. 53 (2008), No. 3, 188–198, [online]. [cit. 2019-7-20]. Dostupné z: https://dml.cz-/bitstream/handle/10338.dmlcz/141858/PokrokyMFA_53-200832.pdf

[26] Přednášky z diskrétní optimalizace z finské univerzity LUT, [online]. [cit. 2019-7-20]. Dostupné z: http://www.mafy.lut.fi/study-/DiscreteOpt/CH6.pdf.

[27] Přednášky z předmětu Teorie grafů, diskrétní optimalizace a výpočetní složitost 1, Západočeská univerzita v Plzni, [online]. [cit. 2019-7-20]. Dostupné z: http://najada.fav.zcu.cz/ryjacek/KMA_TGD1.htm.

[28] Přednášky z předmětu Teorie grafů, diskrétní optimalizace 2, Západočeská univerzita v Plzni, [online]. [cit. 2019-7-20]. Dostupné z: http://najada.fav.zcu.cz/ryjacek/KMA_TGD2.htm.

[29] R. Roberti, P. Toth, *Models and algorithms for the Asymmetric Traveling Salesman Problem: an experimental comparison*, Journal of Mechanical Systems for Transportation and Logistics, 2012. 1. 113-133. 10.1007/s13676-012-0010-0.

[30] T. P. Runarsson, H.-G. Beyer, E. Burke, J. J. Merelo-Guervós, L. D. Whitley, X. Yao, *Parallel Problem Solving from Nature - PPSN IX, 9th International Conference, Reykjavik, Iceland, September 9-13, 2006, Proceedings*. ISBN: 978-3-540-38990-3.

[31] A. Schrijver, *Theory of Linear and Integer Programming*, John Wiley & Sons, New York, NY, 1986. ISBN: 0-471-90854-1

[32] K. Steinmetz, *Bin Packing Problem*, Diplomová práce, Univerzita Pardubice, Fakulta elektrotechniky a informatiky, 2013, [onlone]. [cit. 2019-7-20]. Dostupné z: https://dk.upce.cz/bitstream/handle/10195/53977/SteinmetzK_Bin-Packing_JM_2013.pdf?sequence=6&isAllowed=y.

[33] R. Toth, D. Vigo, *The Vehicle Routing Problem*, SIAM monographs on discrete mathematics and applications, Philadelphia, 2002. ISBN: 0-89871-579-2.

[34] V. Turau, *Algorithmische Graphentheorie*, 4. urozšířené a přepracované vydání, De Gruyter Studium, 2015. ISBN: 978-3-110-41727-2.

[35] Urban Operations Research, chapter 6, Applications of Network Models, Euclidean TSP, [online]. [cit. 2019-7-20]. Dostupné z: http://web.mit.edu/urban_or_book/www/book/chapter6/6.4.7.html.

[36] V. V. Vazirani, *Approximation Algorithms*, Springer, New York, 2001. ISBN: 3-540-65367-8.

[37] L. A. Wolsey, *Integer programming*, Wiley-Interscience series in discrete mathematics and optimization, J. Wiley & sons, New York (N.Y.), Chichester, Weinheim, 1998. ISBN: 978-0471283669.

[38] D. G. Luenberger, Y. Ye, *Linear and Nonlinear Programming*, 3. vydání, Springer Science+Business Media, LLC, New York, NY, 2008, ISBN 978-0-387-74502-2.