Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Problém obchodního cestujícího

Bc. Jan Lokvenc

Diplomová práce

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2017/2018**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jan Lokvenc**
Osobní číslo:	**I16231**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Problém obchodního cestujícího**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

Problém obchodního cestujícího (Travelling Salesman Problem) je optimalizační problém, matematicky vyjadřující a zobecňující úlohu nalezení nejkratší možné cesty procházející všemi zadanými body na mapě. V teorii grafů zní úkol: "V daném ohodnoceném úplném grafu najděte nejkratší Hamiltonovskou kružnici." Problém je, že s rostoucím počtem měst počet možných cest velice rychle narůstá, a tím se doba potřebná k propočtu hrubou silou na počítačích stává zcela neúnosnou už při několika málo desítkách uzlů. Proto se používají heuristické metody. Cílem práce bude rešerše těchto metod, návrh a vypracování aplikace pro nalezení minimální Hamiltonovské kružnice.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná**

Seznam odborné literatury:

BALAKRISHNAN, V. *Schaum's outline of theory and problems of graph theory*. **New York: McGraw-Hill, c1997, viii, 293 p. ISBN 00-700-5489-4.**

Vedoucí diplomové práce: **RNDr. Josef Rak, Ph.D.**
Katedra matematiky a fyziky

Datum zadání diplomové práce: **30. října 2017**
Termín odevzdání diplomové práce: **18. května 2018**

Ing. Zdeněk Němec, Ph.D.
děkan

L.S.

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2017

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 18. 5. 2018

Jan Lokvenc

PODĚKOVÁNÍ

Rád bych poděkoval RNDr. Josefu Rakovi, Ph.D. za užitečné rady při zpracování, Vladimíru Fialovi za konzultaci a rodičům za podporu během studia.

ANOTACE

Tato práce se zabývá principy řešení optimalizační úlohy známé jako Problém obchodního cestujícího. První část je teoretickým úvodem do problematiky vázané na teorii složitosti a optimalizace. Dále je uveden popis principu užití a implementace metody větví a mezí, vybraných heuristických postupů a optimalizačního nástroje Gurobi optimizer.

KLÍČOVÁ SLOVA

teorie grafů, Problém obchodního cestujícího, celočíselné programování, optimalizace, branch and bound, heuristika, Gurobi optimizer

TITLE

Travelling salesman problem

ANNOTATION

This thesis deals with various solving principles of optimization problem known as the travelling salesman problem. The first part is a theoretical introduction to the problem related to the theory of complexity and optimization. It also describes the principles of using and implementing the branch and bound method, selected heuristics and the optimization tool Gurobi optimizer.

KEYWORDS

graph theory, Travelling salesman problem, integer programing, optimalization, branch and bound, heuristics, Gurobi optimizer

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK

PC	Personal computer
TSP	Travelling salesman problem
LP	Linear programing
MIP	Mixed integer programing
IP	Integer programing
ADT	Abstract data type
FIFO	Data type Queue
LIFO	Data type Stack
P	Polynomial
NP	Non-Deterministic Polynomial
B&B	Branch and Bound
LK	Lin-Kernighan
SW	Software
VPN	Virtual privat network
BeFS	Best first search
BFS	Breadth first search
DFS	Depth first search

ÚVOD

Hlavním tématem diplomové práce je matematická optimalizační úloha Problém obchodního cestujícího, která matematicky zobecňuje úlohu hledání nejkratší cesty mezi všemi zadanými vrcholy na mapě s cílem cesty v prvním navštíveném.

V první části práce bude čtenář seznámen se základním matematickým aparátem, nutným k pochopení formální definice úlohy dle teorie grafů a uveden do problematiky matematické optimalizace, včetně kategorizace úloh s návazností na složitost řešení.

Dále bude provedena analýza úlohy s uvedením základních historických milníků v hledání dostatečně výkonného algoritmu pro řešení rozsáhlých instancí úlohy s následnou formulací v podobě matematického modelu.

Poslední část bude věnována popisu metod vhodných pro řešení úlohy obchodního cestujícího, představení existujícího optimalizačního nástroje Gurobi optimizer a v neposlední řadě bude popsána implementace aplikace určené k demonstraci metod řešení.

1 TEORIE GRAFŮ

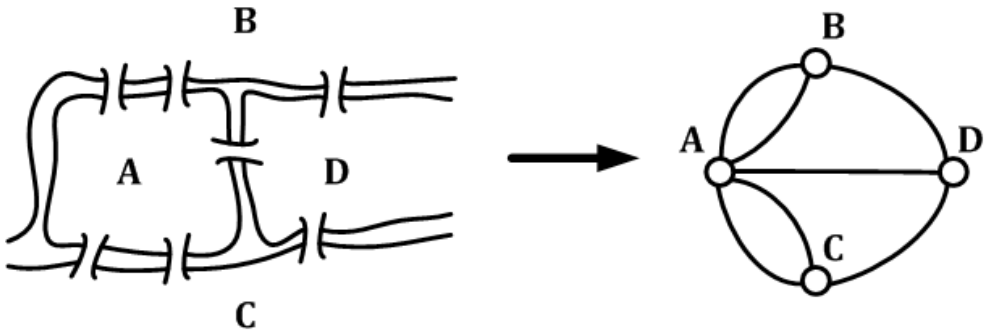
Teorie grafů je jednou z disciplín diskrétní matematiky, která se vyznačuje geometrickým přístupem zkoumání objektu. Základem je matematická struktura graf a jeho zobecnění. Grafem rozumíme objekt složen z množin vrcholů a hran, který má za úkol modelovat reálné situace abstrahované od nepodstatných detailů [1].

1.1 Významné úlohy

První úlohou teorie grafů bývá označovaná matematická hádanka zvaná Sedm mostů města Královce, kde při hledání řešení v roce 1736 matematik Leonhard Euler využil právě grafu pro zobecnění úlohy, aby definoval podmínky pro možné řešení. Další významnou úlohou v historii je hra irského šlechtice sira Williama Hamiltona z roku 1856 a asi nejslavnější potom Problém čtyř barev, který stále na důkaz svého řešení čeká.

1.1.1 Sedm mostů města Královce

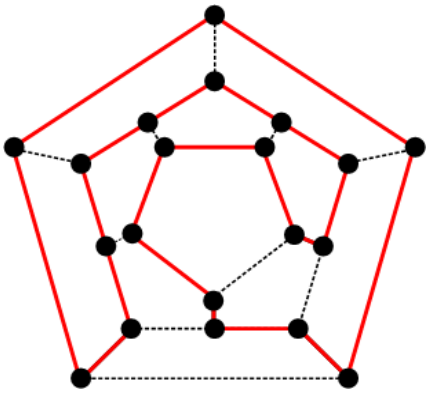
Cílem úlohy sedmi mostů je projít všechny mosty ve městě právě jednou a vrátit se na výchozí místo tak, aby žádný z mostů nebyl navštíven dvakrát. Tuto úlohu lze jednoduše převést do podoby grafu. Řešením je v grafu nalézt tzv. eulerovský tah.



Obrázek 1:Převod úlohy sedmi mostů na graf

1.1.2 Hamiltonova hra

Hamiltonova hra, v anglickém jazyku známa jako The Icosian[1] Game spočívá v nalezení posloupnosti hran a vrcholů, kde každý vrchol dvanáctistěnu bude obsažen právě jednou, tedy vytvořit v grafu hamiltonovskou kružnici.



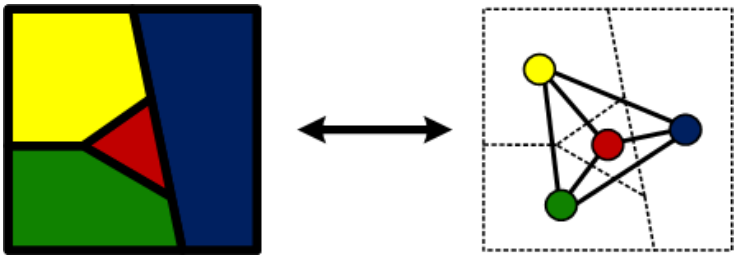
Obrázek 2: Icosian game

1.1.3 Problém čtyř barev

Pravděpodobně nejznámější úloha teorie grafů, nazývána Problémem čtyř barev, bývá neformálně definována takto:

„Stačí pro zabarvení jakékoli politické mapy čtyři barvy tak, aby žádný ze států nesousedil se státem stejné barvy?“

Mapu lze vyjádřit grafem tak, že každou zemi převedeme na vrchol a mezi dvěma vrcholy zakreslíme hranu právě tehdy, když odpovídající země (vrcholy) spolu sousedí (incidují). Pro řešení poté musí platit, že žádné incidující vrcholy nesmí být obarveny stejnou barvou.



Obrázek 3: Problém čtyř barev v grafu

[1] Icosian z řeckého icosa, tedy dvacet. Dvanáctistěn má dvacet vrcholů.

Úlohu čtyř barev se podařilo vyřešit experimentálním způsobem až v roce 1976 dvojici Kenneth Appel a Wolfgang Haken. Přes nové způsoby řešení a zdokonalování algoritmů, nikdy nebyl formulován žádný matematický důkaz pro řešení této úlohy [4].

1.2 Základní pojmy

Definice 1.1 Nechť V je množinou všech vrcholů a E množinou všech hran. Grafem G potom označujeme uspořádanou dvojici:

$$G = (V, E), kde\ E \to V \times V.$$

Pozn.: Pokud existuje $e \in E$, kterému je přiřazen více než jeden obraz $v \in V \times V$, pak se jedná o zobrazení mnohoznačné a graf G nazýváme multigrafem.

Definice 1.2 Graf G nazýváme orientovaným resp. neorientovaným, pokud hrany grafu jsou určeny uspořádanou resp. neuspořádanou dvojicí vrcholů.

Pozn.: Neuspořádaná dvojice $\{x, y\}$ je dvouprvkovou množinou obsahující prvky x a y, kde platí:

$$\{x, y\} = \{y, x\}.$$

Pozn.: Uspořádaná dvojice (x, y) je určena dvouprvkovou množinou, kde je definováno pořadí jednotlivých prvků.

$$(x, y) = \{\{x\}, \{x, y\}\}.$$

Definice 1.3 Graf G je ohodnocený, pokud existuje funkce $d: E(G) \to (0, \infty)$, která každé hraně přiřazuje hodnotu tzv. hranového ohodnocení.

Pozn.: Označme $V(G)\ resp.\ E(G)$ množinou všech vrcholů resp. hran grafu G.

Definice 1.4 Úplným grafem G se označuje takový, kde jsou všechny možné dvojice různých vrcholů spojeny hranou.

$$|E(G)| = \binom{|V(G)|}{2} = \frac{|V(G)|(|V(G) - 1|)}{2}$$

Pozn.: Počet všech prvků množiny M se značí $|M|$.

Definice 1.5 Každý souvislý graf, který neobsahuje kružnici, se nazývá stromem.

Definice 1.6 Kostrou grafu $G(V, E)$ se označuje libovolný strom $T(V, E')$, kde $E' \subseteq E$.

Definice 1.7 Kořenovým stromem se označuje orientovaný graf, kde se kořenem označuje právě jeden vrchol grafu, do kterého nevedou žádné hrany. Do všech ostatních vrcholů grafu vede právě jedna hrana. Platí tedy, že všechny vrcholy grafu jsou z kořene orientovaně dostupné.

Definice 1.8 Sledem S z vrcholu $v_1$do vrcholu v_n se označuje posloupnost incidujících vrcholů a hran:

$$S = \{v_1, e_1, v_2, e_2, \dots, e_n, v_m\}.$$

Pozn.: Pokud $v_1 = v_n$, pak se nazývá sled cyklem.

Definice 1.9 Tahem se označuje sled, kde se každá hrana vyskytuje právě jednou.

Definice 1.10 Eulerovským tahem se označuje takový tah, který obsahuje všechny hrany grafu.

$$G = (V, E);\ \forall\, e\ \in E : e = \{x, y\}; x, y\ \in V;$$

$$S = \{v_1, e_1, v_2, e_2, \dots, e_n, v_m\};$$

$$|E| = n\ ;\ \forall i, j = 0, \dots\ , n\ ;\ i \neq j\colon e_i \neq e_j;$$

Pozn.: Eulerovský tah je uzavřeným resp. otevřeným, pokud platí $v_1 = v_n$ resp. $v_1 \neq v_n$.

Definice 1.11 Cestou se označuje takový tah, ve kterém se každý vrchol vyskytuje právě jednou. Pokud počáteční vrchol je roven koncovému vrcholu, pak se taková cesta označuje jako kružnice.

Definice 1.12 Hamiltonovská cesta je taková, která obsahuje všechny vrcholy grafu právě jednou.

$$G = (V, E);\ \forall\, e\ \in E : e = \{x, y\}; x, y\ \in V;$$

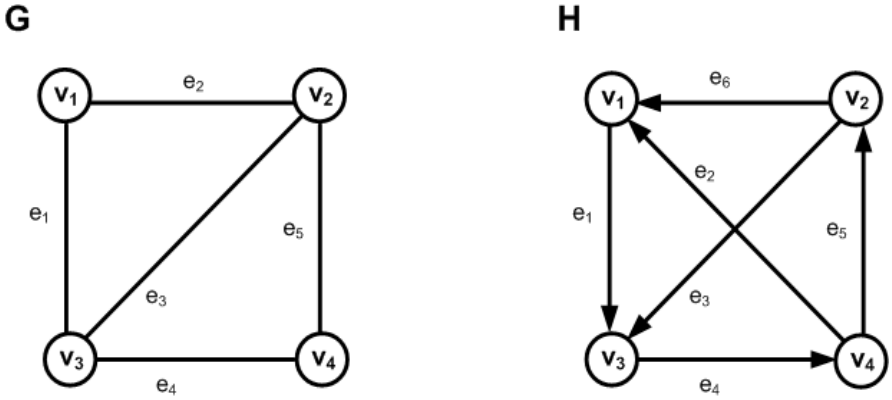
$$S = (v_1, e_1, v_2, e_2, \dots, e_n, v_m)\ ;$$

$$|V| = m\ ;\ \forall i, j = 0, \dots\ , m\ ;\ i \neq j\colon v_i \neq v_j;$$

Pozn.: Pokud v hamiltonovské cestě platí $v_1 = v_m$, pak se označuje jako hamiltonovská kružnice.

1.3 Reprezentace grafů

Bez újmy na obecnosti nebude následující teorie zahrnovat grafy s násobnými hranami, tedy multigrafy. Znázornění jednotlivých variant reprezentace bude demonstrováno na následujících grafech:



Obrázek 4:Příklad neorientovaného a orientovaného grafu

1.3.1 Matice sousednosti

Definice 1.13 Každému orientovanému resp. neorientovanému grafu G s určeným neměnným pořadím vrcholů $v_1, \dots, v_n$ lze přiřadit matici sousednosti M_G typu $n \times n$ dle předpisu:

$$m_{ij} = \begin{cases} 1, & pokud\ existuje\ hrana\ (v_i, v_j)\ resp.\ \{v_i, v_j\}, \\ 0, & pokud\ neexistuje\ hrana\ (v_i, v_j)\ resp.\ \{v_i, v_j\}. \end{cases}$$

Pozn.: Matice sousednosti pro neorientované grafy je vždy symetrická.

$$M_G = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}, \qquad M_H = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

1.3.2 Matice incidence

Definice 1.14 Každému orientovanému grafu *G*, který neobsahuje smyčky[2], s určeným neměnným pořadím vrcholů $v_1, \dots, v_n$ a hran $e_1, \dots, e_m$ lze přiřadit matici incidence B_G typu $n \times m$ dle předpisu:

$$b_{ij} = \begin{cases} 1, & pokud\ v_i\ počátkem\ hrany\ e_j, \\ -1, & pokud\ v_i\ koncem\ hrany\ e_j, \\ 0, & v\ ostatních\ případech. \end{cases}$$

Varianta pro neorientované grafy bez smyček potom platí obdobný předpis:

$$b_{ij} = \begin{cases} 1, & pokud\ v_i inciduje\ s\ hranou\ e_{j,} \\ 0, & v\ ostatních\ případech. \end{cases}$$

Příklad:

$$B_G = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}, \qquad B_H = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & -1 \\ 1 & 0 & -1 & -1 & 0 & 0 \\ 0 & -1 & 0 & 1 & -1 & 0 \end{pmatrix}$$

[2] Hrana se začátkem i koncem ve stejném bodě.

2 TEORIE SLOŽITOSTI

Teorie složitosti je disciplína oboru matematiky a informatiky, která se snaží definovat výpočetní problémy, kategorizovat je do příslušných skupin a uvést vztahy mezi těmito skupinami.

2.1 Základní pojmy

Type úlohy označuje způsob, kterým jsou data reprezentována a v jakém uspořádání budou tvořit zadání úlohy, včetně popisu korektního výsledku a jeho vztahu k zadání.

Instance úlohy je označení pro konkrétní data, která splňují předpis úlohy konkrétního typu.

Algoritmus představuje teoretický princip řešení libovolného typu úlohy. Konkrétní realizace v libovolném programovacím jazyku se nazývá implementací.

2.2 Kategorizace algoritmů

Pro řešení konkrétního typu úlohy lze obecně použít více algoritmů. S měnícím se principem využívaných algoritmů se mění i jejich efektivita pro řešení dané úlohy. Pro vzájemné porovnání dvou algoritmů se využívá ukazatel asymptotické složitosti.

Nejčastějšími kritérii pro ohodnocení efektivity algoritmu jsou

- časové nároky,
- paměťové nároky.

2.2.1 Asymptotická složitost

Pro porovnání efektivnosti dvou algoritmů je třeba definovat ukazatel, který bude abstrahovat od veškerých konstant zahrnutých ve skutečné náročnosti algoritmu a zároveň bude zohledňovat všechny instance řešené úlohy.

Definice 2.4 Nechť $k \cdot f(n)$ je skutečnou složitostí algoritmu, kde n je velikost vstupní množiny dat a k reálnou konstantou[3]. Označme $\theta(g(n))$ asymptotickou složitost algoritmu, kde $g(n)$ je asymptotický odhad funkce $f(n)$, pro kterou platí [7]:

$$f(n) \in \theta(g(n)) \rightarrow \exists C, C', n_0 : C, C' > 0, \forall (n > n_0) : |C \cdot g(n)| \leq |f(n)| \leq |C' \cdot g(n)|.$$

Definice 2.5 Složitost v nejlepším možném případě nazýváme dolní asymptotickou složitostí algoritmu $\Omega(g(n))$, pro kterou platí:

$$f(n) \in \Omega(g(n)) \rightarrow \exists C, n_0 : C > 0, \forall (n > n_0) : |C \cdot g(n)| \leq |f(n)|.$$

Definice 2.6 Složitost v nejhorším možném případě nazýváme horní asymptotickou složitostí algoritmu $O(g(n))$, pro kterou platí:

$$f(n) \in O(g(n)) \rightarrow \exists C, n_0 : C > 0, \forall (n > n_0) : |f(n)| \leq |C \cdot g(n)|.$$

Pozn.: Pro asymptotická složitost algoritmu platí vlastnost:

$$f(n) \in \theta(g(n)) \rightarrow f(n) \in O(g(n)) \wedge f(n) \in \Omega(g(n)).$$

Dle asymptotické složitosti lze algoritmy kategorizovat do tříd složitosti. V závislosti na velikosti vstupních dat označené proměnnou n bude doba trvání algoritmu dle následné tabulky:

Tabulka 1: Vliv složitosti na trvání výpočtu [čas jedné operace]

Množství vstupních dat	**Horní asymptotická složitost**					
	$O(1)$	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(2^n)$	$O(n!)$
1	1	1	1	1	2	1
10	1	10	10	100	1024	3628800
100	1	100	200	10000	10^{30}	9×10^{157}
1000	1	1000	3000	1000000	10^{300}	∞
100000	1	100000	500000	10^{12}	∞	∞

[3] Vyjadřuje především kvalitu implementace, použitý jazyk, kompilátor a jiné.

2.3 Kategorizace úloh

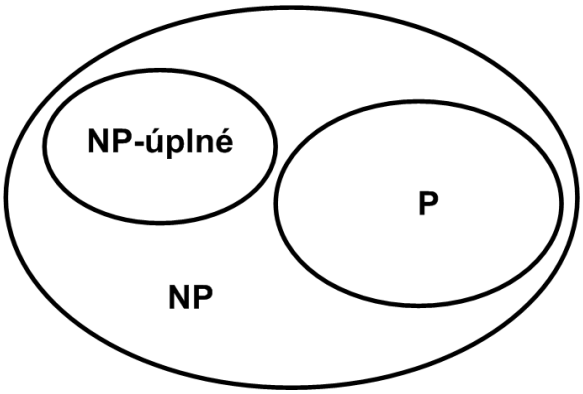
Jednotlivé úlohy lze kategorizovat dle jejich obtížnosti. Zjednodušeně lze říci, že obtížnost dané úlohy je určena složitostí nejefektivnějšího algoritmu, který dosahuje korektního výsledku dané úlohy.

Definice 2.7 Třídou P (Polynomial) obtížných úloh rozumíme množinu úloh, pro které existuje algoritmus, jehož časová složitost je shora asymptoticky omezená libovolným polynomem.

Definice 2.8 Třídou NP (Non-Deterministic Polynomial) obtížných úloh rozumíme množinu úloh, pro které nebyl popsán algoritmus k jejich řešení v polynomiálním čase, lze však správnost výsledku v polynomiálním čase ověřit.

Definice 2.9 Pokud lze polynomiálně redukovat[4] libovolnou úlohu z NP na jednu konkrétní, pak tuto úlohu označujeme jako NP-úplnou.

Pozn.: Vyřešení právě jedné NP-úplné úlohy by znamenalo odpověď na mileniální otázku, zda $NP = P \; resp. \; NP \neq P$.



Obrázek 5:Kategorizace úloh a vztahů mezi nimi

[4] Existuje polynomiální algoritmus, který převede vstupní data jedné úlohy na vstup druhé úlohy tak, že výsledky obou instancí budou totožné.

3 OPTIMALIZACE

Optimalizace je matematickou disciplínou, která se zabývá hledáním minimální resp. maximální hodnoty účelové funkce v optimalizačních úlohách.

Definice 3.1 Nechť U je univerzum všech potenciálních řešení, $P \subseteq U$ množinou všech přípustných řešení a funkce $f: U \rightarrow \mathbb{R}$ účelovou funkcí. Řešením obecné optimalizační úlohy je potom $\vec{x} \in P$, pro které platí:

$$f(\vec{x}) \leq f(x),\ \forall x \in P$$

Pozn.: Obecnou minimalizační úlohu lze převést na maximalizační vynásobením účelové funkce číslem -1.

3.1 Kategorizace úloh

Vzhledem k tomu, že obecná optimalizační úloha svým popisem pokrývá široké spektrum problémů, je vhodné dále klasifikovat jednotlivé úlohy do konkrétnějších skupin, dle charakteru účelové funkce a omezujících podmínek (lineární, kvadratické, nelineární, nehladká, globální). Tento přístup je výhodný z důvodu možnosti využití charakteristických metod řešení úloh, které spadají do konkrétní skupiny.

Definice 3.2 Nechť x je n-členný vektor proměnných, d n-členný vektor cen těchto proměnných, dx lineární funkcí a $Ax \leq b$ soustavou lineárních nerovnic omezujících podmínek, kde A je maticí koeficientů omezení a b vektorem pravých stran. Poté lze úlohu lineárního programování zapsat ve tvaru [9]:

$$min\{dx: Ax \leq b, x \in \mathbb{R}_+^n\}$$

Pozn.: V úloze LP lze bez újmy na obecnosti veškeré omezující podmínky definovat pomocí rovnic, neboť je možné každou nerovnost převést na rovnici pomocí přidání tzv. přídatné proměnné.

Definice 3.3 Pokud v úloze lineárního programování existuje vektorový podprostor $y \subseteq x \wedge y \in \mathbb{Z}_+$, pak se jedná o úlohu smíšeného lineárního celočíselného programování (MIP).

Definice 3.4 Pokud v úloze lineárního programování platí $x \in \mathbb{Z}_+$, pak se jedná o úlohu lineárního celočíselného programování (IP).

Definice 3.5 Jestliže všechny celočíselné proměnné mohou nabývat pouze hodnot z množiny $B = \{0,1\}$, pak je úloha bivalentní.

3.2 Exaktní algoritmy

Exaktní algoritmy jsou takové, které při řešení optimalizační úlohy vždy vrátí optimální řešení. Mezi takové algoritmy pro řešení TSP lze řadit algoritmy hrubé síly, jejichž princip spočívá v postupném projití všech možných variant, nebo algoritmus Branch and Bound celočíselného programování.

3.3 Heuristiky

Většina optimalizačních úloh, kde proměnné nabývají pouze diskrétních hodnot, náleží do NP-třídy obtížnosti úloh. Pro tyto úlohy tedy neexistuje deterministický algoritmus řešící libovolnou instance úlohy v možném časovém horizontu.

Heuristické metody řešení umožnují dosáhnout výsledku obtížných úloh v relativně krátkém čase za použití navržené strategie výpočtu. Obecně tyto metody nezaručují dosáhnutí optimálního řešení, avšak ve většině případů poskytují dobré řešení v závislosti na použití správně navržené heuristické strategie.

Konkrétní heuristiky jsou navrhovány pro konkrétní typy optimalizačních úloh, které umí řešit. Obecně se tedy nejedná o univerzální metody řešení. Mezi neznámější pro řešení TSP patří hladový algoritmus, metody vkládání nebo k-opt.

Definice 3.6 Nechť heuristická metoda dosahuje řešení $g(x)$ optimalizační úlohy U, které je nejvýše k-násobkem optimálního řešení $f(x)$. Konstanta k se potom označuje jako aproximační faktor této heuristické metody [10].

Pozn.: Heuristické metody lze využít pro zrychlení částí deterministických algoritmus.

3.4 Metaheuristiky

Metaheuristiky jsou obecné heuristické postupy, kterými lze řešit libovolný typ optimalizační úlohy. Jedná se v podstatě o univerzální heuristiky, mezi které patří například genetické algoritmy nebo metody lokálního hledání.

Genetické algoritmy jsou založeny na myšlence využití evoluční biologie pro hledání optimálního řešení úlohy za použití dědičnosti, selekce, křížení a mutace. Základem je tzv. populace, tedy množina přípustných řešení. Každému jedinci populace je spočtena tzv. fitness funkce, která vyjadřuje kvalitu řešení dané úlohy. Selekcí jedinců s nejvyšší hodnotou fitness

funkce vznikne výběr vhodný ke křížení. Křížením a případnou mutací vznikne nová generace jedinců nesoucí rysy svých předků. Tento proces se opakuje, dokud není nalezen jedinec s přijatelnou hodnotou účelové funkce.

Metody lokálního hledání jsou založeny na prohledávání stavového prostoru v okolí výchozího zvoleného uzlu. Problém těchto metod je v možném uváznutí na hodnotě lokálního extrému.

4 PROBLÉM OBCHODNÍHO CESTUJÍCÍHO

Problém obchodního cestujícího, zkráceně TSP z anglického Travelling salesman problem, je matematická optimalizační úloha. Přes zdánlivou jednoduchost zadání, které by neformálně mohlo znít: „Navštivte nejkratší cestou všechna města tak, aby bylo každé město navštívené právě jednou a cesta končila ve stejném městě, kde začala.“, je řešení této úlohy zcela netriviální. Již při několika desítkách měst narůstá obtíž rozměrů. Obecně bývá úloha TSP označována otázkou milénia.

4.1 Složitost

Problém obchodního cestujícího, na první pohled triviální úloha, kde stačí projít všechna přípustná řešení a vybrat jedno optimální. V čem je tedy ten problém? Řekněme, že instance, pro kterou budeme hledat optimální cestu, obsahuje 10 měst k navštívení. Označme množinu měst jako vektor m, kde každá položka označuje právě jedno město. Každé řešení úlohy lze reprezentovat jako cyklickou posloupnost měst:

$$m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8, m_9, m_{10}.$$

Každou takovou posloupnost lze označit jako permutaci všech hodnot vektoru m. Pro přehlednost řekněme, že cesta bude vždy začínat právě v m_1. Pro výběr druhého města v pořadí existuje 9 možností, dalšího 8 a tak dále. Celkový počet možných cest je tedy $9 \times 8 \times 7 \times \ldots \times 3 \times 2 \times 1$, což je 9 faktoriál.

Pro zjednodušení lze uvažovat, že vzdálenost mezi městy je v obou směrech cesty stejná. Potom tedy platí rovnost cest:

$$m_1, m_2, \ldots, m_9, m_{10} = m_{10}, m_9, \ldots, m_2, m_1.$$

Z této symetrie lze odvodit, že k nalezení optimálního řešení je nutno projít právě polovinu všech možných permutací, tedy $\frac{9!}{2}$, což je přesně 181 440 možných cest.

V 21. století se většina populace spolehne na strojový výpočet. Při řešení instance velikosti 10 lze dosáhnout řešení poměrně snadno. Co se ale stane, pokud se velikost instance změní na 50, 100 nebo 1000 měst?

Pokud budeme uvažovat o řešení instance o velikosti pouhých 50 měst hrubou silou, počet všech možných permutací bude přibližně 3×10^{64}. Pokud budeme mít k dispozici superpočítač Titan z Oak Ridge National Laboratory, disponující 560 640 jádry a 710 144 GB operační

paměti, který je schopen provést $17{,}5 \times 10^{15}$ aritmetických výpočtů za sekundu [14], pak bude výpočet trvat přibližně 27 179 821 700 369 645 575 125 trilionů let, a to pouze za předpokladu jedné aritmetické operace na kontrolu jedné cesty.

4.2 Historie

Ačkoliv přesná historie úlohy TSP není zcela známá, lze se domnívat, že se již několik desítek až stovek let dříve s touto úlohou v praxi setkávali například kazatelé na svých cestách nebo obchodní cestující. Z této analogie pravděpodobné pochází název úlohy „Problém obchodního cestujícího“.

První zmínku o TSP, jako matematické optimalizační úloze, uvedl Merrill Flood ve svém článku z roku 1956, kde napsal: „Problém formuloval v roce 1934 Hassler Whitney na semináři na Princetonské univerzitě.“ [3].

Řešení instancí TSP se v průběhu let díky dokonalejším algoritmům a zvyšování výpočetního výkonu neustále zlepšuje. Přesto největším objevem v historii zůstává princip lineární programování (LP) G. Dantziga a později algoritmus větví a mezí založený na LP výzkumného týmu ve složení Dantzig, Fulkerson, Johnson, který definoval směr řešení rozsáhlých instancí TSP. Výpočetní sílu navrženého algoritmu ručně demonstrovali na instanci velikosti 42 měst států USA [2].

Postupným zlepšováním metody větví a mezí, například kombinováním s metodou řezných nadrovin, pro výpočet těsnějších odhadů nebo zavedením heuristických postupů do částí exaktních algoritmů, se dosáhlo řešení instancí o velikosti 120 měst (Groetschel, 1977), 532 měst (Padberg, Rinaldi, 1987) nebo 15 112 měst (Applegate, Bixby, Chvátal, Cook, 2001) [3].

Dosavadní rekordní vyřešená instance TSP patří výzkumnému týmu ve složení D. Applegate, R. Bixby, V. Chvátal, W. Cook, D. Espinoza, M. Goycoolea, K. Helsgaun s neuvěřitelnou velikostí 89 900 měst.

4.3 Matematický model

TSP je matematickou optimalizační úlohou, kterou lze kategorizovat jako bivalentní úlohu lineárního celočíselného programování (0-1 IP).

Definice 4.1 Nechť je x n-členný vektor proměnných, množina $B = \{0, 1\}$, d n-členný vektor cen proměnných, dx lineární účelovou funkcí a $Ax \leq b$ soustavou lineárních nerovnic omezujících podmínek, kde A je maticí koeficientů omezení a b vektorem pravých stran. Poté lze bivalentní úlohu lineárního celočíselného programování zapsat ve tvaru:

$$min\{dx: Ax \leq b, x \in B^n\}$$

Definice 4.2 Problémem obchodního cestujícího je úloha hledání minimální hamiltonovské kružnice v úplném ohodnoceném grafu.

Pro možnost řešení konkrétních úloh optimalizace je nejprve nutné ústní formulaci úlohy převést do matematického modelu, který odpovídá tvaru optimalizační úlohy[5]. Matematický model úlohy TSP může být popsán účelovou funkcí ve tvaru:

$$f = min \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij} ,$$

kde:

$$x_{ij} = \begin{cases} 1, & pokud\ hrana\ mezi\ vrcholy\ i\ a\ j\ je\ součástí\ řešení \\ 0, & pokud\ hrana\ mezi\ vrcholy\ i\ a\ j\ není\ součástí\ řešení \end{cases}$$

$$d_{ij}\ je\ hodnota\ ohodnocení\ hrany\ mezi\ vrcholy\ i\ a\ j$$

za podmínek:

$$\sum_{j=1}^{n} x_{ij} = 1; i = 1, \dots, n$$

$$\sum_{i=1}^{n} x_{ij} = 1; j = 1, \dots, n$$

[5] Pro jednu úlohu z pravidla existuje více možných korektních matematických modelů.

5 BRANCH AND BOUND

Branch and Bound, zkráceně B&B, je nazýván algoritmus řešení úloh celočíselného programování, kde řešení spočívá ve větvení úlohy na jednodušší podúlohy, pomocí kterých se získá řešení úlohy hlavní. Během větvení dochází k prořezávání podúloh, aby se předešlo zbytečnému procházení větví výpočtu, které prokazatelně neobsahují optimální řešení.

5.1 Větvení úlohy

Definice 5.1 Nechť P je množina všech přípustných řešení optimalizační úlohy U, kterou označme $P(U)$. Optimalizační úlohu U, kde f je účelovou funkcí, lze rozdělit na n úloh $U_1, \dots, U_n$, kde účelovou funkcí je též f. Takové úlohy označme jako podúlohy úlohy U, pro které platí:

$$P(U) = P(U_1) \cup \dots \cup P(U_n).$$

Pozn.: Pokud $n = 2$, pak se mluví o dichotomickém jinak polytomickém větvení [12].

Věta 5.2 [8] Řešení optimalizační úlohy U je rovno nejlepšímu řešení libovolné podúlohy z $U_1, \dots, U_n$. Pokud pro libovolnou podúlohu U_i platí že:

1. Známe optimální řešení úlohy U_i,
2. nebo dokážeme, že úloha U_i nemá přípůstné řešení,
3. nebo dokážeme, že žádné přípustné řešení není výhodnější než dosud nalezené přípustné řešení úlohy U,

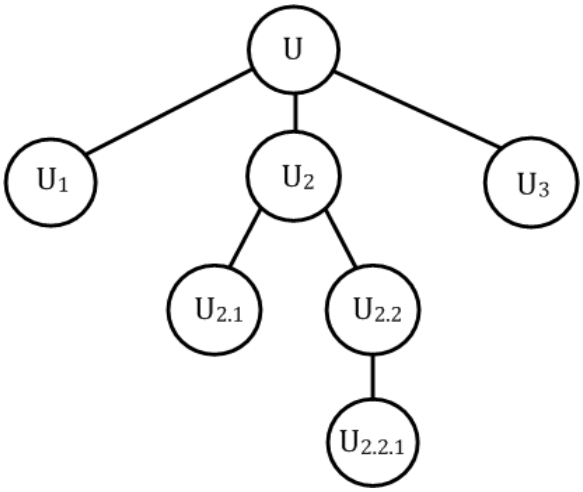
pak je známé optimální řešení úlohy U.

Pozn.: Je důležité, aby podúlohy byly snazší než hlavní úloha, přičemž je výhodné, aby $P(U_1), \dots, P(U_n)$ byly vzájemně disjunktní.

Pozn.: Ve většině případů jsou jednotlivé podúlohy U_i stále příliš obtížné k řešení, a proto se dále obdobným způsobem větví na podúlohy ve tvaru $U_{i1}, \dots, U_{in}$, čímž vzniká tzv. strom řešení.

5.2 Strom řešení

Definice 5.3 Pokud lze úlohu větvit dle (5.1), pak všechny jednotlivé podúlohy možno označit jako vrcholy stromového grafu, kde kořenem je hlavní úloha U. Takový graf se označuje jako strom řešení.



Obrázek 6: Princip budování stromu řešení

Definice 5.4 Jednotlivé listy stromu řešení lze označit jako mrtvé resp. živé pokud splňují jednu ze tří podmínek (5.2) resp. pokud je nutné hledat optimální řešení pomocí dalšího větvení.

Pozn.: Pokud jsou všechny listy stromu řešení mrtvé, pak je řešení úlohy u konce.

Strom řešení je vlastně strukturou k-cestného vyhledávacího stromu, ve kterém hledáme prvek s minimální hodnotou odhadu optima, který je zároveň listem. Tento list je zároveň optimální řešení hlavní úlohy.

5.3 Omezující funkce

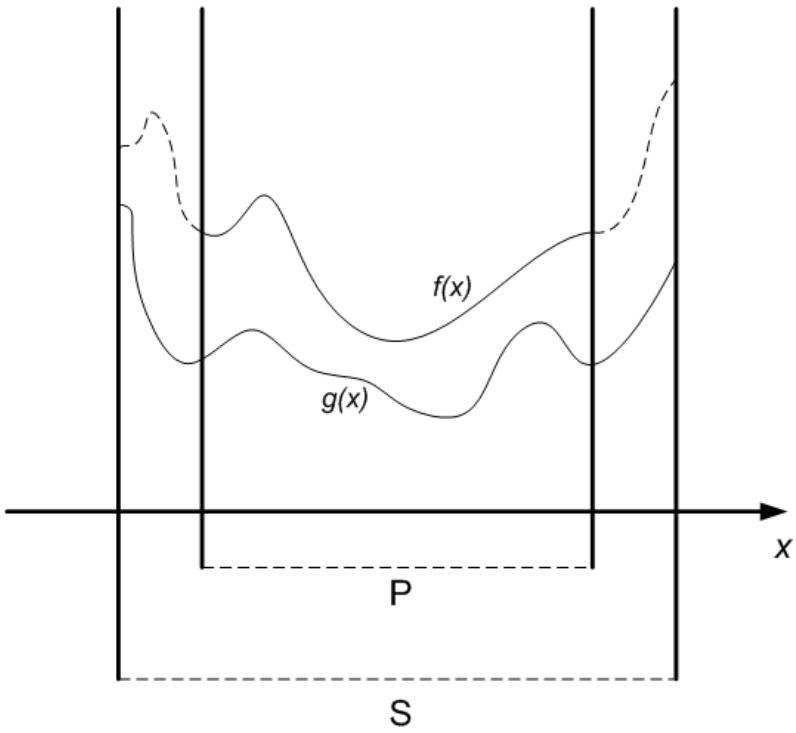
Věta 5.5 [12] Nechť U je optimalizační úloha definována na univerzu S, kde P je množnou přípustných řešení a f účelovou funkcí. Pro omezující funkci $g(x)$ platí:

$$\min g(x), x \in S \leq \begin{Bmatrix} \min f(x), x \in S \\ \min g(x), x \in P \end{Bmatrix} \leq \min f(x), x \in P.$$

Omezující funkci označujeme jako silnou resp. slabou, pokud je svojí hodnotou blízká resp. vzdálená hodnotě optima.

Pokud označíme $\min g(x), x \in P$ jako optimální řešení relaxované[6] úlohy U', které splňuje všechny omezující podmínky původní úlohy, jedná se zároveň o optimální řešení hlavní úlohy.

Optimální řešení relaxované úlohy se využívá jako odhad optimálního řešení v kontextu metody B&B.



Obrázek 7: Omezující funkce

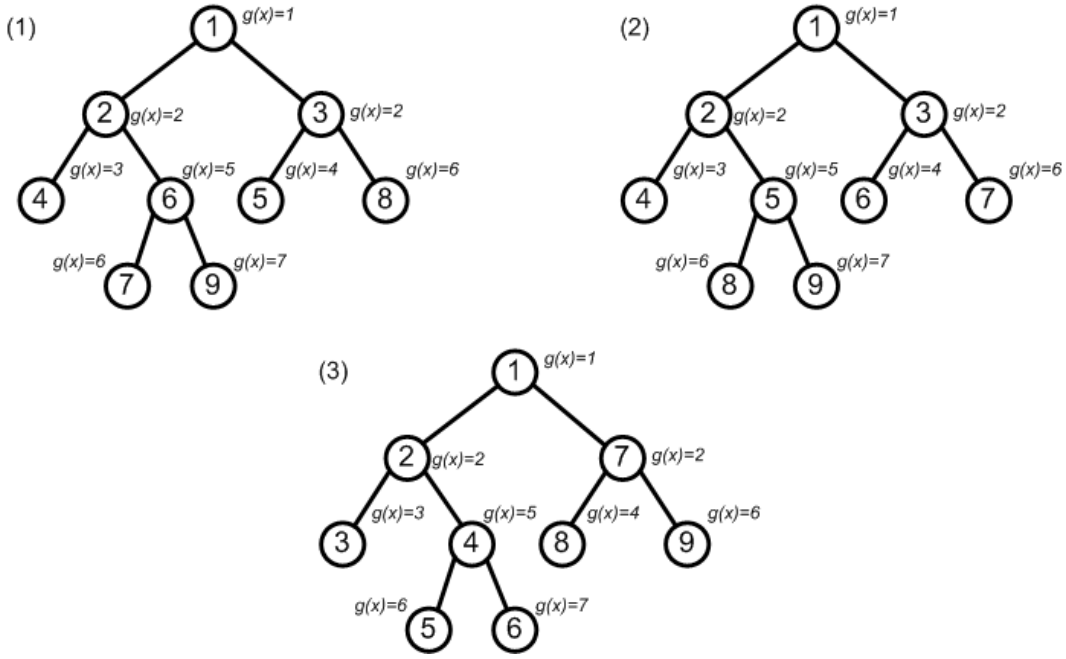
5.4 Výběr úlohy

Strategie pro výběr úlohy ve stromu řešení k umrtvení nebo větvení je dalším důležitým kritériem pro rychlé nalezení optimálního řešení. Mezi nejznámější patří následující metody výběru [12]:

- **Best first search (BeFS)**, který vybere živý list s nejmenší hodnotou odhadu optimálního řešení. Pomocnou strukturou je ADT prioritní fronta. Tento postup se tedy snaží projít pouze větev stromu, ve které se, dle odhadu, s největší pravděpodobností nachází optimální řešení.

[6] Relaxovaná úloha v sobě nezahrnuje všechny omezující podmínky původní úlohy. Obvykle se vynechávají podmínky, které radikálně zvětšují časové nároky řešení.

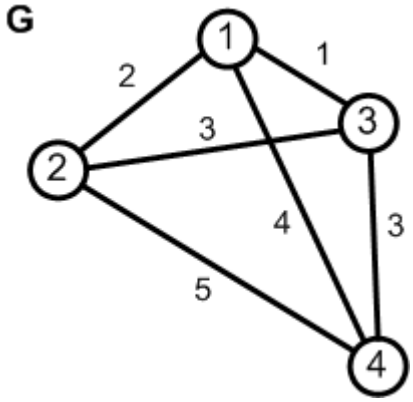
- **Breadth first search (BFS)** je princip výběru založený na průchodu stromu řešení do šířky. Pomocnou strukturou je ADT fronta. Tento postup se velice podobá výpočtu hrubé síly. Postupně prochází všechny větve stromu řešení.
- **Depth first search (DFS)** je princip založený na průchodu stromu řešení do hloubky.Pomocnou strukturou je ADT zásobník. Tento postup nejprve najde první přípustné řešení, které později zlepšuje.



Obrázek 8: Pořadí dle (1) BeFS, (2) BFS, (3) DFS

5.5 Aplikace B&B na TSP

Příklady této kapitoly budou demonstrovány na následujícím grafu:



Obrázek 9: Ohodnocený graf

5.5.1 Paměťová reprezentace

Definice 5.6 Každému ohodnocenému grafu G s určeným neměnným pořadím vrcholů $v_1, \dots, v_n$ lze přiradit matici cen D typu $n \times n$ dle předpisu:

$$d_{ij} = \begin{cases} d(v_i, v_j), & pokud\ existuje\ hrana\ mezi\ v_i\ a\ v_j \\ \infty, & pokud\ neexistuje\ hrana\ mezi\ v_i\ a\ v_j \end{cases}$$

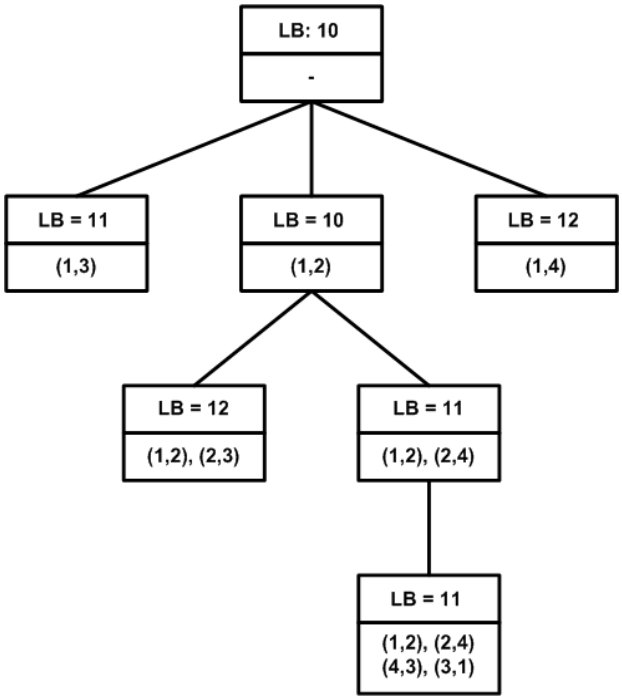
Pozn.: Pokud $i = j$, pak $d_{ij} = \infty$.

Příklad:

$$D = \begin{pmatrix} \infty & 2 & 1 & 4 \\ 2 & \infty & 3 & 5 \\ 1 & 3 & \infty & 3 \\ 4 & 5 & 3 & \infty \end{pmatrix}$$

5.5.2 Strom řešení

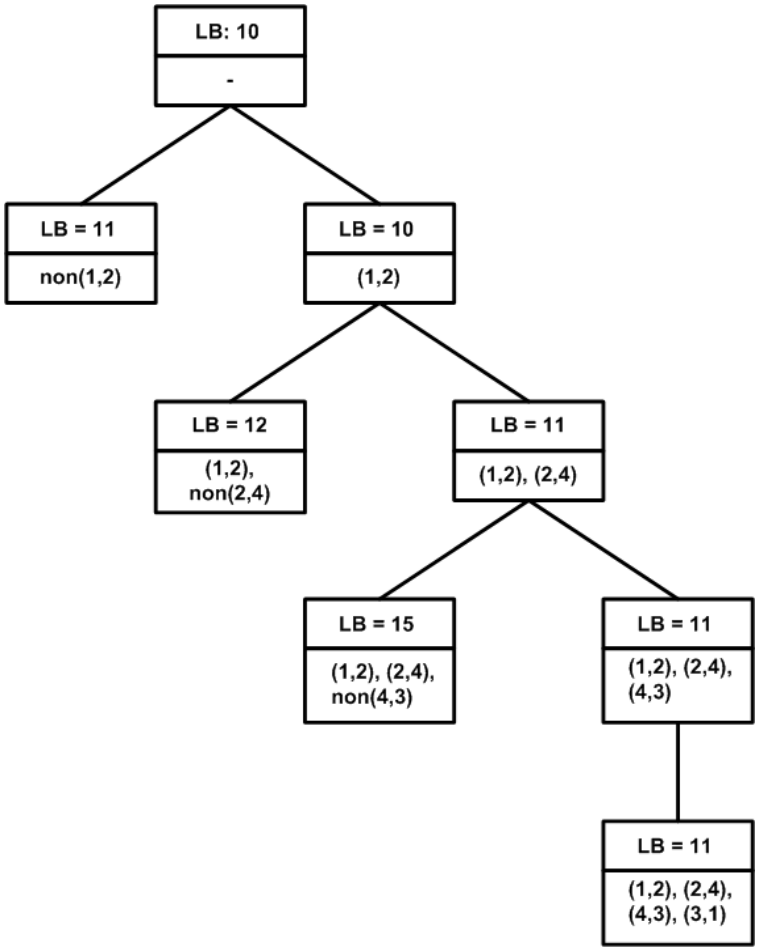
Definice 5.7 Nechť T je k-cestným stromem řešení, který vznikne polytomickým dělením úlohy TSP, kde vrcholy označují konkrétní podúlohu a nesou informace o dosud projitých hranách a příslušných minimálních odhadech optima.



Obrázek 10: K-cestný strom řešení

Pozn.: Zařazené hrany jsou hamiltonovskou cestou projité části grafu.

Pozn.: Pro řešení TSP lze využít i binárního stromu řešení, kde jsou v pravé resp. levé větvi všechna potenciální řešení, které obsahují resp. neobsahují příslušnou hranu.



Obrázek 11: Binární strom řešení

5.5.3 Výpočet omezující funkce

Definice 5.8 Nechť U je instance úlohy TSP reprezentované maticí cen D a vektorem řešení s. Podúloha U' vzniklá zařazením hrany $e = (i, j)$ do s' bude reprezentována submaticí D' matice cen D, vzniklé přidáním omezující podmínky cesty zpět:

$$d'_{ji} = \infty$$

a následného odebrání i-tého řádku, j-tého sloupce.

Příklad: Zařazená hrana $e = (1,3)$.

$$D = \begin{pmatrix} \infty & 2 & 1 & 4 \\ 2 & \infty & 3 & 5 \\ 1 & 3 & \infty & 3 \\ 4 & 5 & 3 & \infty \end{pmatrix} \rightarrow \begin{pmatrix} \infty & 2 & 1 & 4 \\ 2 & \infty & 3 & 5 \\ \infty & 3 & \infty & 3 \\ 4 & 5 & 3 & \infty \end{pmatrix} \rightarrow D' = \begin{pmatrix} 2 & \infty & 5 \\ \infty & 3 & 3 \\ 4 & 5 & \infty \end{pmatrix}$$

Definice 5.9 Nechť D je maticí cen úlohy U. K matici D lze přiřadit redukovanou matici R stejného typu dle předpisu:

$$r_{ij} = d_{ij} - r_i - c_j,$$

kde:

$$r_i = min_j d_{ij};\ j = 1, \dots, n$$

$$c_j = min_i(d_{ij} - r_i);\ i = 1, \dots, n.$$

Příklad: Redukce matice D

$$D = \begin{pmatrix} \infty & 2 & 1 & 4 \\ 2 & \infty & 3 & 5 \\ 1 & 3 & \infty & 3 \\ 4 & 5 & 3 & \infty \end{pmatrix} \rightarrow \begin{pmatrix} \infty & 1 & 0 & 3 \\ 0 & \infty & 1 & 3 \\ 0 & 2 & \infty & 2 \\ 1 & 2 & 0 & \infty \end{pmatrix} \rightarrow R' = \begin{pmatrix} \infty & 0 & 0 & 1 \\ 0 & \infty & 1 & 1 \\ 0 & 1 & \infty & 0 \\ 1 & 1 & 0 & \infty \end{pmatrix}$$

$$r = (1, 2, 1, 3), c = (0, 1, 0, 2)$$

Nechť s' je k-členný vektor všech zařazených hran v řešení dané úlohy U'. Minimální odhad řešení LB úlohy U' je:

$$LB = \sum_{i=1}^{k} s_i + \sum_{i=1}^{n} r_i + \sum_{i=1}^{n} c_i$$

5.5.4 Postup algoritmu

Po navržení strategie pro řešení dílčích problémů popsaných výše je postup pro řešení TSP pomocí B&B následující:

1. Alokuj prioritní frontu $PrioQ$ (strom řešení)
2. Vypočti LB hlavní úlohy U a vlož do $PrioQ$ (kořen)
3. Vyjmi úlohu U' s nejmenší hodnotou LB z fronty $PrioQ$
4. Pokud lze získat optimální řešení U', ukonči algoritmus
5. Rozděl úlohu U' na podúlohy $U'_1, \dots, U'_n$ dle dělící strategie
6. Vypočítej LB pro $U'_1, \dots, U'_n$
7. Zařaď $U'_1, \dots, U'_n$ do $PrioQ$
8. Zpět na 3. krok

5.5.5 Další metody odhadu

Zvýšení výkonu implementovaného algoritmu lze dosáhnout především pomocí správně navržené omezující funkce, která bude poskytovat silné odhady.

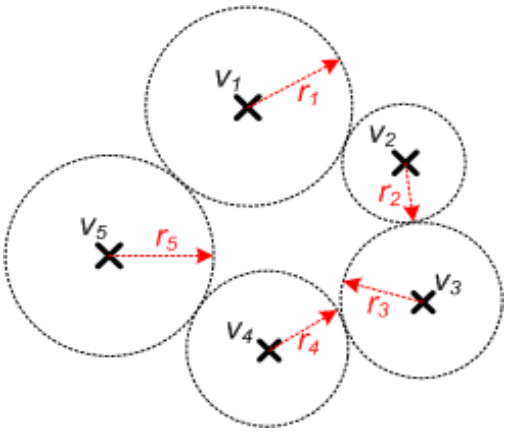
Definice 5.11 Mějme úlohu euklidovského[7] TSP nad grafem G. Pro výpočet odhadu optimálního řešení úlohy lze použít metodu kontrolních zón, která spočívá v nalezení optima úlohy lineárního programování [10]:

$$max \sum_{i=1}^{n} 2r_i,$$

kde:

$$r_i + r_j \leq d_{ij} \; \forall i, j \; ; \; r_i \geq 0 \; \forall i.$$

Pozn.: Pro velké instance TSP se stává tato metoda výpočetně příliš náročná.



Obrázek 12: Princip metody kontrolních zón

Definice 5.12 Mějme úlohu metrického[8] TSP nad grafem G. Pro výpočet odhadu optimálního řešení úlohy lze využít tzv. minimální l-strom, který se skládá z minimální kostry grafu G bez libovolného vrcholu v_0 a dvou nejkratších hran, které incidují s v_0. Označováno jako Held-Karp metoda [16].

Pozn.: Hodnota odhadu dle Held-Karp metody bývá ≈ 90 % optima [10].

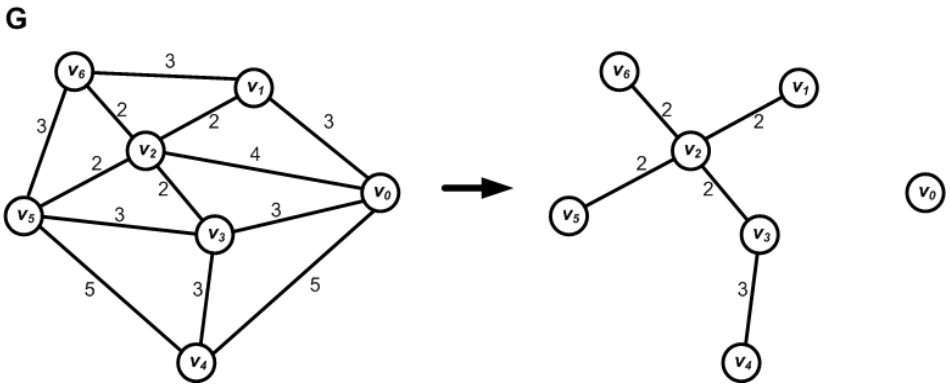
---

[7] Existuje obousměrná symetrie vzdáleností mezi všemi vrcholy. Formálně tedy platí $d_{ij} = d_{ji}$.

[8] Platí zde trojúhelníková nerovnost. Formálně tedy platí $d_{ij} \leq d_{ik} + d_{jk}$.

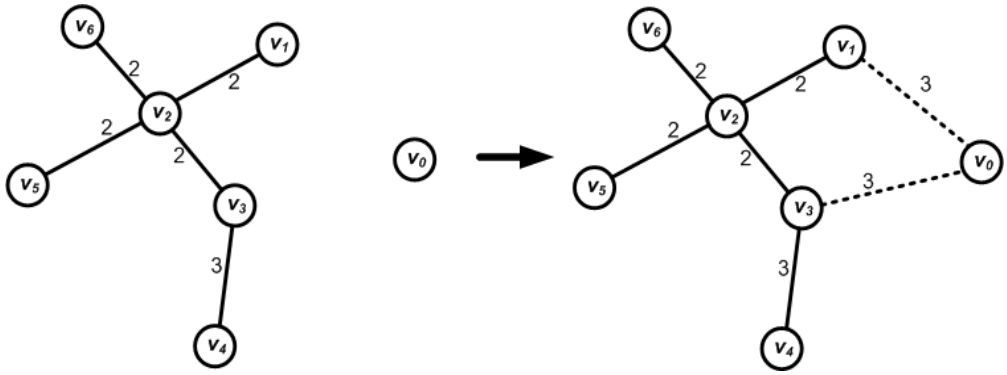
Příklad: Vstupem Held-Karp metody je graf *G* a výstupem hodnota minimálního odhadu *LB*.

1. Získáme kostru grafu *G* bez vrcholu v_0 viz Obrázek 13 např. pomocí Kruskalova algoritmu.



Obrázek 13: Minimální kostra

2. Ze vzniklé kostry získáme minimální l-strom přidáním dvou nejkratších hran, které incidují s vrcholem v_0 viz Obrázek 14.



Obrázek 14: Minimální l-strom

3. Hodnotu odhadu je sumou hranového ohodnocení všech hran minimálního l-stromu.

$$LB = 2 + 2 + 2 + 2 + 3 + 3 + 3 = 17$$

6 METODA LIN–KERNIGHAN

Metoda Lin-Kernighan (LK), občas označována jako k-opt algoritmus, je heuristická strategie navržená pro řešení TSP. Jejím základním principem je postupné zlepšování dosazené hamiltonovské kružnice, pomocí výměny obecně k hran v každé iteraci algoritmu a následné kontroly zlepšení řešení [13].

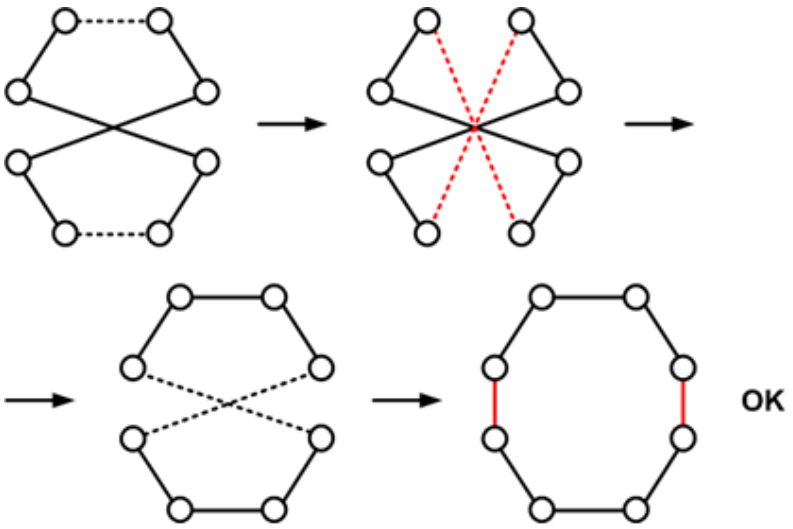
S rostoucím k se zvyšuje přesnost výsledného řešení na úkor výpočetní složitosti. Problémem je počet možných přeuspořádání k hran.

Tabulka 2: Počet možných přeuspořádání k-opt, zdroj: [10]

k	2	3	4	5	6	7	8
Počet přeuspořádání	1	4	20	148	1 358	15 104	198 144

6.1 Algoritmus 2-opt

Algoritmus 2-opt je konkrétní variantou metody Lin-Kernighan, která pracuje v každém kroku právě s dvěma hranami, které mění s cílem najít lepší řešení. Složitost algoritmu je $O(n^2)$.



Obrázek 15: Princip algoritmu 2-opt

Pozn.: Pokud při nalezení zlepšení řešení aplikujeme algoritmus opět od začátku nad tímto zlepšeným řešením, dosáhne algoritmus lepších výsledků, ale složitost bude $O\left(n^2 log_2(n)\right)$.

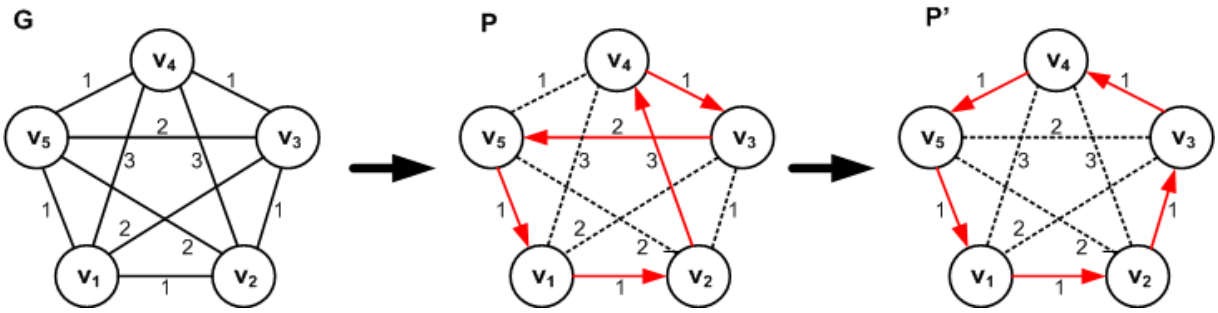
Nechť je dána výchozí hamiltonovská kružnice H v úplném neorientovaném grafu G jako posloupnost hran ve tvaru $E_H = \{(v_1, v_2), (v_2, v_3), \dots, (v_{n-1}, v_n), (v_n, v_1)\}$. Postup algoritmu je potom následující:

1. Vyber dvě hrany $e_1 = (v_i, v_{i+1}), e_2 = (v_j, v_{j+1})$, kde $i + 1 < j$
2. Získej hamiltonovskou kružnici H' úpravou vybraných hran dle $e_1' = (v_i, v_j), e_2' = (v_{i+1}, v_{j+1})$
3. Urči změnu ohodnocení $\Delta_H = d_{i,j} + d_{i+1,j+1} - d_{i,i+1} - d_{j,j+1}$.
4. Pokud $\Delta_H < 0$, pak $H = H'$
5. Pokud pro všechny možné kombinace hran nenastalo $\Delta_H < 0$ pak ukonči algoritmus
6. Zpět na 1. krok

Pozn.: Počáteční řešení je vhodné hledat pomocí jiné heuristické metody například metodou vkládání nejbližšího souseda.

Příklad: Nechť vstupem algoritmu je graf G a přípustné řešení posloupnost vrcholů P.

$$G = \begin{pmatrix} \infty & 1 & 2 & 3 & 1 \\ 1 & \infty & 1 & 2 & 2 \\ 2 & 1 & \infty & 1 & 2 \\ 3 & 2 & 1 & \infty & 1 \\ 1 & 2 & 2 & 1 & \infty \end{pmatrix}, P = \{v_1, v_2, v_4, v_3, v_5\}$$

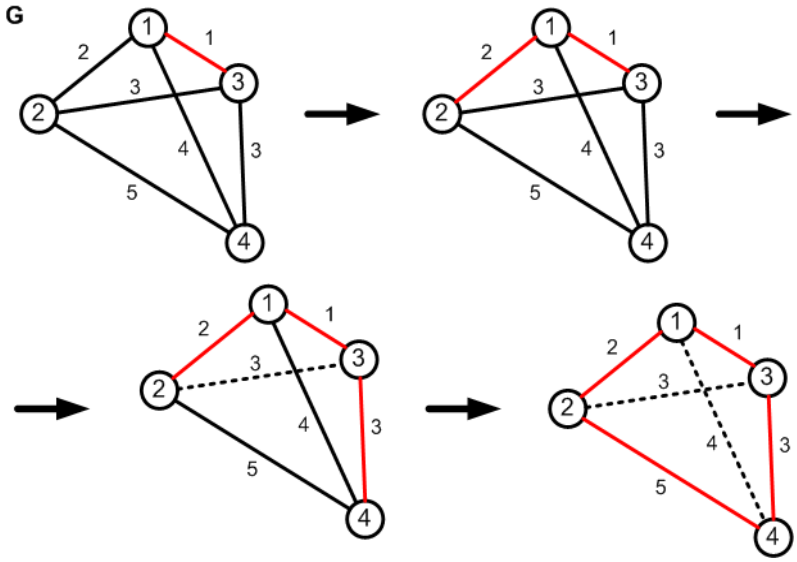


Obrázek 16: Vstup/výstup algoritmu 2-opt

Výstupem algoritmu je potom posloupnost vrcholů $P' = \{v_1, v_2, v_3, v_4, v_5\}$. V tomto konkrétním případě se jedná o řešení optimální.

7 HLADOVÝ ALGORITMUS

Hladový algoritmus, v anglickém jazyce označován jako Greedy algorithm, je heuristická metoda sestavující řešení úlohy po částech, které se postupně spojují v celek. V případě instance úlohy TSP velikosti n se „hladově“ vybírá vždy nejkratší hrana grafu, kterou lze přidat do řešení, aniž by byla porušena podmínka stupně dva všech vrcholů a zároveň vytvoření kružnice nižšího stupně než n. Složitost algoritmu je $O(n^2 \log_{10} n)$.



Obrázek 17: Princip hladového algoritmu

Příklad: Vstupem algoritmu je množina hran reprezentována maticí cen grafu *G* (obr. 15) a výstupem je posloupnost hran *P*.

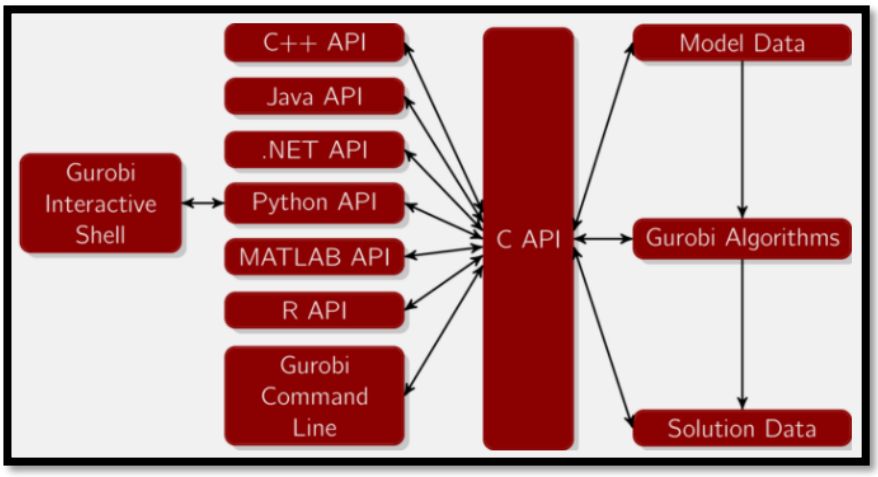
$$D_G = \begin{pmatrix} \infty & 2 & 1 & 4 \\ 2 & \infty & 3 & 5 \\ 1 & 3 & \infty & 3 \\ 4 & 5 & 3 & \infty \end{pmatrix} \rightarrow P = \{\{1,3\},\{1,2\},\{3,4\},\{2,4\}\}.$$

Postup algoritmu pro instanci TSP velikosti n:

1. Setřiď množinu všech hran $edges$ dle ohodnocení
2. Vyjmi nejkratší hranu e z $edges$
3. Pokud se zařazením hrany e stupeň libovolného vrchol bude větší než 2 nebo vznikne kružnice stupně menšího n, zpět na 2. krok
4. Pokud zařazením hrany e vznikne kružnice stupně n, ukonči algoritmus
5. Zpět na 2. krok

8 GUROBI

Gurobi optimizer je komerční[9] optimalizační řešitel, který je vyvíjen s důrazem na vysoký výkon výpočtu s využitím nejmodernějších technologií. Rychlostí dosažení řešení řady typů úloh patří k předním světovým nástrojům. Lze jej použít k řešení úloh lineárního a kvadratického programování a dalších. Jedná se o robustní optimalizační nástroj, který podporuje aplikační rozhraní k řadě programovacím jazykům jako například Java, C++, Python nebo MATLAB a lze jej také přilinkovat k modelovacím jazykům AIMMS, AMPL a dalším.



Obrázek 18: Architektura Gurobi optimizer [15]

Při použití Gurobi optimizer je uživateli poskytováno tzv. flexibilní licencování, které umožňuje finanční přizpůsobení řadě projektům, a to včetně podpory klient-server služeb a cloudu pro multiplatformní provoz na výpočetně výkonnějších strojích.

Pozn.: Při využití licence pro akademické účely je nutné při autentizaci uživatele být připojen na univerzitní VPN.

[9] Existují bezplatné licence pro akademické použití.

8.1 Příklad

Mějme matematický model bivalentní celočíselné optimalizační úlohy ***ExampleMIP*** popsané účelovou funkcí:

$$f = max(x + 2y + 3z),$$

kde:

$$x, y, z \in \{0, 1\},$$

za podmínek:

$$x + 2y + z \leq 4,$$

$$x + 2y \geq 1.$$

Vzhledem k robustnosti Gurobi optimizeru a detailní programátorské dokumentaci vztahující se ke konkrétnímu aplikačnímu rozhraní, není nutné popisovat jednotlivé kroky implementace. Zdrojový kód pro řešení demonstrační úlohy viz Příloha A.

Postup řešení:

```
1. Definice výpočetního prostředí
2. Definice účelové funkce
3. Definice omezujících podmínek
4. Vlastní výpočet
5. Zobrazení výsledku
```

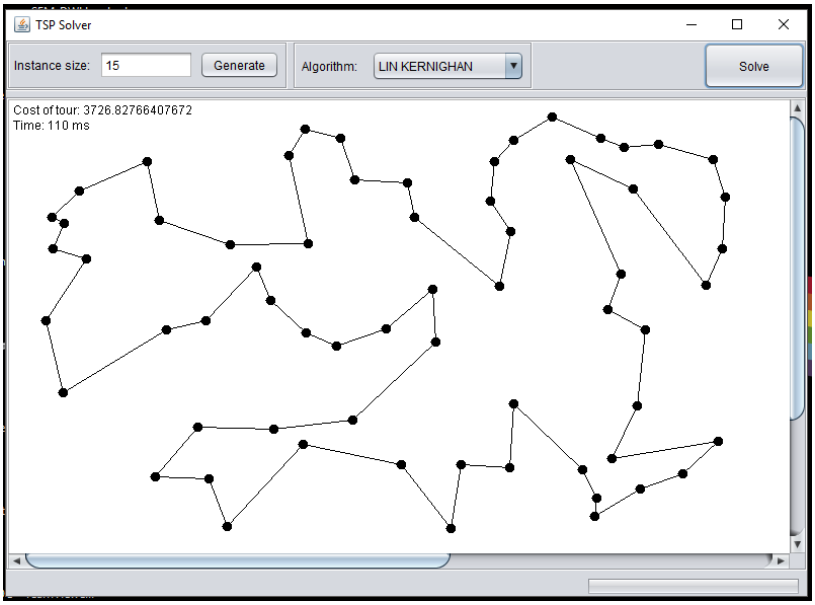
9 APLIKACE

V rámci diplomové práce byla implementována intuitivní aplikace určená k vizualizaci dosažených výsledků pomocí uživatelem zvolené metody řešení na vytvořené instanci úlohy TSP s názvem TSP Solver.

9.1 Uživatelská dokumentace

Aplikace disponuje grafickým rozhraním pro jednoduchou interakce s uživatelem, které nabízí uživatelskou a automatickou variantu vytváření instance úlohy TSP. V rámci aplikace je možnost dynamické volby výpočetního algoritmu.

Výstupem aplikace je grafické zobrazení řešení zadané instance úlohy včetně zobrazení délky cesty a času výpočtu. Informace se současně zapisují do logovací textového souboru pro možnost dalšího zpracování.



Obrázek 19: Grafické rozhraní

Generate slouží k automatickému generování instance o uživatelem definované velikosti. Jedná se o automatickou alternativu postupného klikání na vykreslovací plochu.

Algorithm nabízí možnost dynamického výběru používaného algoritmu pro řešení aktuální instance úlohy. Poskytuje následující volby:

1. GUROBI, řešení úlohy pomocí externí knihovny Gurobi optimizer,
2. BRANCH AND BOUND, implementovanou variantu metody větví a mezí,

3. LIN-KERNIGHAN, implementovanou variantu 2-opt algoritmu,
4. GREEDY, implementovanou variantu hladového algoritmu.

Solve spouští výpočet aktuálně zadané instance úlohy řešené pomocí vybraného algoritmu.

Cost of tour zobrazuje informace o délce cesty aktuální vyřešené instance úlohy.

Time informuje uživatele o čase výpočtu řešení aktuální vyřešené instance úlohy. Hodnota je uvedena v milisekundách.

9.2 Architektura

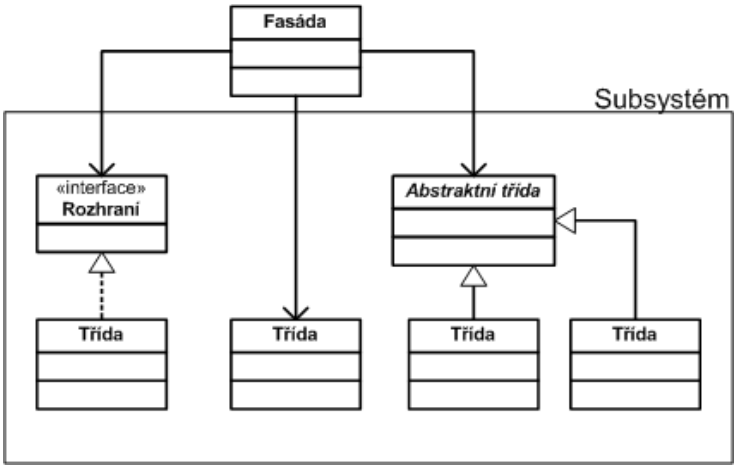
Pro minimalizaci duplicit kódu a oddělení logiky aplikace od grafického rozhraní, což vede k jednoduší modifikace nebo případné výměně logických částí programu, jsou použity principy objektového programování a vybrané návrhové vzory.

Návrhový vzor označuje v objektově orientovaném programování tzv. Best practise postup, pro řešení určitého druhu problému, který zajišťuje lepší čitelnost zdrojového kódu a ve většině případů pomáhá předcházet chybám v logickém návrhu aplikace.

Architektura aplikace popsána v jazyce UML viz Příloha B.

9.2.1 Fasáda

Fasáda je návrhový vzor, který umožnuje vytvořit rozhraní k subsystému, který je složený z logicky souvisejících tříd. Razantně zvyšuje přehlednost výsledného zdrojového kódu aplikace, který umožňuje dělit do aplikačních vrstev a subsystémů.

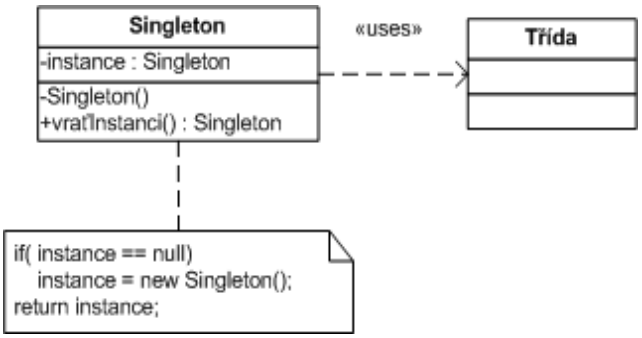


Obrázek 20: Návrhový vzor fasáda

V aplikaci TSP Solver je fasáda použita pro oddělení výpočetní logiky od grafického zobrazení formou aplikačního rozhraní (API).

9.2.2 Singleton

Singleton je návrhový vzor, který zabezpečuje, aby v aplikaci existovala instance dané třídy maximálně jednou. Ideálním příkladem použití je databázové připojení.

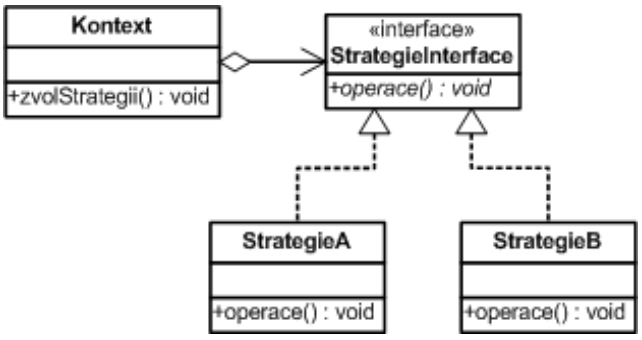


Obrázek 21: Návrhový vzor singleton

V aplikaci TSP Solver je singleton použit pro dostupnost jedinečného aplikačního rozhraní, které zpřístupňuje výpočetní logiku grafickému rozhraní.

9.2.3 Strategie

Strategie je návrhový vzor sloužící k dynamické výměně mezi rozdílnými implementacemi za běhu programu. Dle způsobu implementace může přepínání probíhat implicitně resp. explicitně.



Obrázek 22: Návrhový vzor strategie

V aplikaci TSP Solver je využíván tento návrhový vzor pro realizaci dynamické změny použitého algoritmu pro řešení úlohy.

9.3 Porovnání implementovaných algoritmů

V rámci aplikace jsou demonstrovány čtyři různé algoritmy. Porovnání složitosti z hlediska času a paměti znázorňuje následující tabulka:

Tabulka 3: Porovnání složitosti algoritmů

	Složitost	
	Čas	**Paměť**
Gurobi optimizer	–	–
B&B	$O(n!)$	$O(n!)$
2-opt	$O(n^2 \log_2 n)$	$O(n)$
Hladový algoritmus	$O(n^2 \log_{10} n)$	$O(n^2)$

Nejlepší dosažené časy řešení jednotlivých algoritmů, dle velikosti řešené instance znázorňuje následující tabulka:

Tabulka 4: Porovnání algoritmů dle času řešení [ms]

	Velikost instance							
	5	**8**	**11**	**15**	**30**	**50**	**100**	**200**
Gurobi optimizer	<1	2	2	3	52	80	1 983	85 439
B&B (BeFS)	<1	<1	<1	31	32 125	NaN	NaN	NaN
2-opt	<1	<1	<1	<1	<1	31	775	15 365
Hladový alg.	<1	<1	<1	<1	<1	<1	2	7
B&B (BFS)	4	43	168 630	NaN	NaN	NaN	NaN	NaN

Pozn.: Hodnota „NaN“ znamená, že čas běhu algoritmu je více než 200 000 ms.

Měření probíhalo na jednotném testovacím prostředí, kterým byl notebook ASUS K55V s dvoujádrovým procesorem Processor|Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz, kapacitou operační pamětí 4 GB (rychlostí 1600 ns), základní deskou ASUSTeK verze 1 a operačním systémem Operating system|Microsoft Windows 10 Education (version 10.0.14393; build 14393).

ZÁVĚR

V této diplomové práci na téma Problém obchodního cestujícího byl čtenář nejprve seznámen se základy teorie grafů včetně typických úloh a nastínění postupu řešení pro lepší představu hlavní problematiky.

Následně byla představena teorie složitosti, příslušné základní pojmy a kategorizace úloh dle složitosti s cílem uvést problém NP-úplných úloh. S tím souvisí navazující téma matematické optimalizace, definice obecné optimalizační úlohy, možné kategorizace dle vlastností konkrétní úlohy a přehled aplikovatelných způsobů řešení.

Další část je věnována neformální analýze úlohy obchodního cestujícího, historickému vývoji řešení včetně aktuálně dosažitelných výsledků. V návaznosti na tento úvod je formulován matematický model úlohy a představeny metody řešení s důrazem na vysvětlení principu algoritmu branch and bound lineárního celočíselného programování. Práce dále zahrnuje popsání principu heuristických metod Lin-Kernighan a hladového algoritmu. Dodatkem je poté uvedení existujícího řešitele optimalizačních úloh Gurobi optimizer jako externí knihovny včetně řešení úlohy.

Poslední část práce popisuje návrh architektury demonstrační aplikace představených algoritmů, včetně použitých návrhových vzorů a jejich principu užití. Tato část zároveň obsahuje uživatelskou dokumentaci a porovnání výsledků implementovaných algoritmů.

Tato práce seznamuje s hlavními principy řešení úlohy obchodního cestujícího a tím vybízí k řadě možných rozšíření. Z programátorského hlediska je nemalým úkolem implementace algoritmu větví a mezí pro paralelní zpracování ve více vláknech nebo efektivní implementace obecného k-opt algoritmu. Druhou cestou je studium heuristických metod s cílem navrhnout metodu nebo vylepšit stávající, kterou by bylo možné dosáhnout řešení s přijatelně malým aproximační faktorem, který bude zajišťovat dostatečnou přesnost pro většinu aplikací. V neposlední řadě je možné studovat další principy optimalizace metody větví a mezí s cílem dosažení výkonnější strategie algoritmu.

POUŽITÁ LITERATURA

[1] Graph theory. *Encyclopedia of Mathematics*. URL: http://www.encyclopediaofmath.org/index.php?title=Graph_theory&oldid=42252

[2] COOK, William. *The Traveling Salesman Problem* [online]. Last Updated: December 2016 [cit. 2018-02-10]. Dostupné z: http://www.math.uwaterloo.ca/tsp/index.html.

[3] COOK, William. *Po stopách obchodního cestujícího: matematika na hranicích možností.* Praha: Argo, 2012. ISBN 978-80-7363-412-4.

[4] BEČVÁŘ, Jindřich a Eduard FUCHS, *ed. Historie matematiky: sborník pro vyučující na středních školách.* Praha: Prometheus, 1997. Dějiny matematiky. ISBN 80-7196-046-2.

[5] DEMEL, Jiří. *Grafy a jejich aplikace.* Praha: Academia, 2002. ISBN 80-200-0990-6.

[6] PELIKÁN, Jan. *Diskrétní modely v operačním výzkumu.* Praha: Professional Publishing, 2001. ISBN 80-86419-17-7.

[7] SAVICKÝ, Petr. Výpočetní složitost I. Katedra logiky, 2016. Univerzita Karlova.

[8] DEMEL, Jiří. *Operační výzkum*. Katedra aplikované informatiky, Fakulta stavební, 2011. ČVUT.

[9] NEMHAUSER, George L. a Laurence A. WOLSEY. *Integer and combinatorial optimization*. New York: Wiley-Interscience, c1999. ISBN 0-471-35943-2.

[10] HLADÍK, Milan. *Celočíselné Programování*. Katedra aplikované matematiky, 2017. Univerzita Karlova.

[11] MATAIJA, Mirta, Mirjana RAKAMRIĆ ŠEGIĆ a Franciska JOZIĆ. *Solving the travelling salesman problemusing the Branch and Bound method*. In: Zbornik Veleučilišta u Rijeci. Vol. 4. 2016.

[12] CLAUSEN, Jens. *Branch and Bound Algoritms: Principles and Examples*, Department of Computer Science, University of Copenhagen, 1999. Denmark.

[13] HELSGAUN, Keld. *An Effective Implementation of K-opt Moves for the Lin-Kernighan TSP Heuristic*. Computer Science, No. 109, Roskilde University, 2006. Denmark.

[14] TITAN: OAK RIDGE NATIONAL LABORATORY: No. 1 system in November 2012. *TOP 500: The List.* [online]. 2012 [cit. 2018-04-27]. Dostupné z: https://www.top500.org/resources/top-systems/titan-oak-ridge-national-laboratory/

[15] *Gurobi Optimalization* [online]. [cit. 2018-04-27]. Dostupné z: http://www.gurobi.com

[16] VALENZUELA, Christine L. a Antonia J. JONES. *Estimating the Held-Karp lower bound for the geometric TSP*. School of computing and mathematics, 2001. University of Teesside.

PŘÍLOHY

PŘÍLOHA A – ZDROJOVÝ KÓD DEMOSTRAČNÍ ÚLOHY

```
1.  // import Gurobi knihovny
2.  import  gurobi.*;
3.  public  class  ExampleMIP  {
4.      public  static  void  main(String[]  args)  {
5.          try  {
6.              GRBEnv  env  =  new  GRBEnv("ExampleMTP.log");
7.              GRBModel  model  =  new  GRBModel(env);
8.
9.  // Zavedení proměnných
10.
11.             GRBVar  x  =  model.addVar(0.0,  1.0,  0.0,  GRB.BINARY,  "x");
12.             GRBVar  y  =  model.addVar(0.0,  1.0,  0.0,  GRB.BINARY,  "y");
13.             GRBVar  z  =  model.addVar(0.0,  1.0,  0.0,  GRB.BINARY,  "z");
14.
15. // Definice účelové funkce
16.
17.             GRBLinExpr  expr  =  new  GRBLinExpr();
18.             expr.addTerm(1.0,  x);
19.             expr.addTerm(2.0,  y);
20.             expr.addTerm(3.0,  z);
21.             model.setObjective(expr,  GRB.MAXIMIZE);
22.
23. // Přidání první podmínky x + 2 y + 3 z <= 4
24.
25.             expr  =  new  GRBLinExpr();
26.             expr.addTerm(1.0,  x);
27.             expr.addTerm(2.0,  y);
28.             expr.addTerm(1.0,  z);
29.             model.addConstr(expr,  GRB.LESS_EQUAL,  4.0,  "con_0");
30.
31. // Přidání druhé podmínky x + y >= 1
32.
33.             expr  =  new  GRBLinExpr();
34.             expr.addTerm(1.0,  x);
35.             expr.addTerm(2.0,  y);
36.             model.addConstr(expr,  GRB.GREATER_EQUAL,  1.0,  " con_1");
37.
38. // Řešení úlohy
39.
40.             model.optimize();
41.
42. // Vypsání hodnoty optimálního řešení
43.
44.             System.out.println("Optimal: " + model.get(GRB.DoubleAttr.ObjVal));
45.
46.             model.dispose();
47.             env.dispose();
48.         }
49.         catch  (GRBException  e)  {
50.             System.out.println("Error");
51.         }
52.     }
53. }
```

PŘÍLOHA B – ARCHITEKTURA APLIKACE V UML