ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA DOPRAVNÍ

Alona Talska

Problém obchodního cestujícího, aplikace v dopravních a logistických systémech

Bakalářská práce

2019

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta dopravní
děkan
Konviktská 20, 110 00 Praha 1

K617 ..... **Ústav logistiky a managementu dopravy**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení studenta (včetně titulů):

Alona Talska

Kód studijního programu a studijní obor studenta:

B 3710 – LOG – Logistika a řízení dopravních procesů

Název tématu (česky): **Problém obchodního cestujícího, aplikace v dopravních a logistických systémech**

Název tématu (anglicky): Travelling Salesmann Problem, Applications in Transport and Logistics

Zásady pro vypracování

Při zpracování bakalářské práce se řiďte následujícími pokyny:

- Problém obchodního cestujícího, formulace a klasifikace úloh
- Exaktní, heuristické a metaheuristické metody řešení a jejich omezení
- Příklady aplikace TSP v dopravě a logistice
- Analýza existujícího komerčního SW na trhu, výhody a nevýhody
- Analýza kvality řešení metod řešení s využitím TSPLIB instances
- Závěry a zhodnocení analýz

ČVUT FD

Rozsah grafických prací: podle pokynů vedoucího bakalářské práce

Rozsah průvodní zprávy: minimálně 35 stran textu (včetně obrázků, grafů a tabulek, které jsou součástí průvodní zprávy)

Seznam odborné literatury: VOLEK, J., LINDA, B.: Teorie grafů - aplikace v dopravě a veřejné správě. Pardubice: Univerzita Pardubice, 2012, 190 s. ISBN 978-80-7395-225-9.

DYNTAR, J.: Návrh a optimalizace dodavatelských systémů s využitím dynamické simulace. Praha: FinEco, 2018, 203 s. ISBN 978-80-86590-15-8.

Vedoucí bakalářské práce: **doc. Ing. Josef Volek, CSc.**

Datum zadání bakalářské práce: **30. června 2018**
(datum prvního zadání této práce, které musí být nejpozději 10 měsíců před datem prvního předpokládaného odevzdání této práce vyplývajícího ze standardní doby studia)

Datum odevzdání bakalářské práce: **2. prosince 2019**
a) datum prvního předpokládaného odevzdání práce vyplývající ze standardní doby studia a z doporučeného časového plánu studia
b) v případě odkladu odevzdání práce následující datum odevzdání práce vyplývající z doporučeného časového plánu studia

L. S.

doc. Ing. Tomáš Horák, Ph.D.
vedoucí
Ústavu logistiky a managementu dopravy

doc. Ing. Pavel Hrubeš, Ph.D.
děkan fakulty

Potvrzuji převzetí zadání bakalářské práce.

Alona Talska
jméno a podpis studenta

V Praze dne ..... 11. září 2019

Poděkování

Na tomto místě bych ráda poděkovala všem, kteří mi poskytli podklady pro vypracování této práce. Zvláště pak děkuji panu doc. Ing. Josefovi Volkovi CSc. vedoucímu mé bakalářské práce, za trpělivost, konzultace a odborné rady, které mi poskytoval po celou dobu její tvorby. V neposlední řadě chci poděkovat svým blízkým a rodině za morální a materiální podporu, které se mi dostává po celou dobu studia.

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci, zpracovanou na závěr studia na ČVUT v Praze Fakultě dopravní.

Prohlašuji, že jsem předloženou práci vypracovala samostatně a že jsem uvedla veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Nemám závažný důvod proti užívání tohoto školního díla ve smyslu § 60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 2. 12. 2019 .....

Alona Talska

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA DOPRAVNÍ

PROBLÉM OBCHODNÍHO CESTUJÍCÍHO, APLIKACE V DOPRAVNÍCH A LOGISTICKÝCH SYSTÉMECH

Bakalářská práce

Prosinec 2019

Alona Talska

ABSTRAKT

Předmětem předložené bakalářské práce „Problém obchodního cestujícího, aplikace v dopravních a logistických systémech“ je problém obchodního cestujícího, formulace a klasifikace úloh. Dále rozdělení metod řešení do základních skupin s popisem jejich vlastností a příklady aplikace TSP v dopravě a logistice. Praktická část se následně týká analýzy existujícího komerčního software na trhu a analýzy kvality řešení metod s využitím TSPLIB instances.

ABSTRACT

The subject of this bachelor thesis "Travelling Salesman Problem, Applications in Transport and Logistics" is the travelling salesman problem, formulation and classification of problems. Further follows division of the solution methods into basic groups with description of their characteristics and examples of application of TSP in transport and logistics. The practical part includes the analysis of existing commercial software on the market and the analysis of the quality of each method solution using TSPLIB instances.

KLÍČOVÁ SLOVA

problém obchodního cestujícího, doprava, logistika, metody řešení, softwarové řešení, kvalita řešení

KEYWORDS

travelling salesman problem, transport, logistics, solution methods, software solution, quality of solution

Obsah

SEZNAM POUŽITÝCH ZKRATEK

TSP	Travelling Salesman Problem (úloha obchodního cestujícího)
STSP	Symmetric Travelling Salesman Problem (symetrická TSP)
ATSP	Asymmetric Travelling Salesman Problem (asymetrická TSP)
TSPLIB	TSP Library (knihovna úloh obchodního cestujícího)
NP-úlohy	nedeterministické polynomiální úlohy
HK	Hamiltonovská kružnice
ACO	Ant Colony Optimization (Optimalizace Mravenčí kolonie)
SW	Software
ERP	Enterprise Resource Planning (podnikový informační systém)
CRM	Customer Relationship Management (řízení vztahů se zákazníky)
TM	Transport Management (dopravní management)
SAP	Systems, Applications and Products in Data Processing

Úvod

Problém obchodního cestujícího (TSP) patří v současnosti mezi nejznámější optimalizační úlohy. Úkolem obchodního cestujícího je projet všemi místy na mapě právě jednou a následně se vrátit do výchozího místa. Výdaje cestujícího na cestu mají být minimální. Úloha je komplikovaná tím, že cestující smí každým městem projet právě jednou. Optimální řešení (tedy nejkratší Hamiltonovská kružnice v daném hranově ohodnoceném grafu) problému šetří čas a s ním spojené náklady.

Výpočet optimálního řešení TSP je náročný na čas, protože úloha patři k tzv. NP-úplným úlohám. To znamená, že se tradiční metody nejsou schopny dostat k výsledku v požadovaném čase. V případě TSP se jedná například o exaktní algoritmy matematického programování apod. Dochází k tomu z toho důvodu, že s rostoucím počtem měst počet přípustných řešení cest rychle narůstá a tím se doba potřebná k výpočtu v reálném čase stává zcela neúnosnou už při řádově desítkách měst. Z toho vyplývá neřešitelnost úlohy exaktními algoritmy v rozumném čase.[1]

Při řešení NP-úplných úloh se proto z praktických důvodů využívají jednoduché heuristiky a metaheuristické algoritmy. Řešení získaná pomocí heuristik a metaheuristik nezaručují optimálnost, ale mohou se k optimálnímu řešení v závislosti na kvalitě použité metody optimálnímu řešení významně přiblížit. U některých sofistikovaných metod může odchylka od optima činit méně než 5 %. Použití heuristických metod je proto jedinou schůdnou cestou, jak řešit úlohy tohoto typu v praxi.[11]

Právě popis a porovnaní kvality řešení exaktních, heuristických a metaheuristických metod a také nástrojů řešení je cílem této práce.

V první kapitole je popsána historie vzniku TSP, obecná formulace problému, klasifikace úloh, kterých existuje velké množství v různých oborech společenské praxe, a jejich aplikace. Práce je zaměřena zejména na oblast využití TSP v dopravní praxi. Dále jsou uvedeny exaktní, heuristické a metaheuristické metody řešení a jejich omezení. Ve třetí kapitole jsou popsány příklady aplikace TSP v dopravě a logistice, čtvrtá kapitola práce se bude zabývat analýzou existujícího komerčního software pro řešení TSP na trhu. Poté následuje analýza kvality metod řešení pomocí TSPLIB instances a samozřejmě závěr práce.

Základní pojmy teorie grafů

Problém obchodního cestujícího lze snadno a srozumitelně formulovat s pomocí neorientovaného hranově ohodnoceného grafu, který je v teorii grafů definován jako konečná množina vrcholů a hran. Dále budou v práci používány následující pojmy:

V Teorii grafu reálný svět se převádí na graf, a proto se rozlišuji orientované a neorientované grafy.

Graf je uspořádaná trojice $G = (V, X, p)$, kde V je množina vrcholů, X je množina hran a zobrazení p je incidence grafů G. Pro TSP vždycky uvažujeme neorientovaný graf.

Orientovaný graf je uspořádaná trojice $D = [V, Y, p]$, kde Y je množina orientovaných hran, V je množina všech uspořádaných dvojíc vrcholů $[u, v]$, p je prosté zobrazení množiny hran do množiny všech uspořádaných dvojic $[u, v]$, $u, v \in V, u \neq v$.

Pokud pro hranu v orientovaném grafu platí $p\,[h] = [u, v]$, nazýváme počáteční vrchol u **výchozím,** koncový vrchol v **koncovým.**

Incidence p přiřazuje každé hraně grafu neuspořádanou dvojici vrcholů. Platí-li pro incidenci hrany $h \in X, p(h) = (u, v)$, hrana h inciduje s vrcholem u a vrcholem v. Vrcholy u, v nazýváme přilehlými vrcholy. Podobné platí pro hrany – hrany incidující s vrcholem se nazývají přilehlé hrany.

Vrcholy grafu jsou zjednodušeným modelem objektů reálného světa, vrcholy mohou vyjadřovat například osídlení (obce, města), významné body dopravních a logistických systémů (železniční stanice, zastávky, terminály MHD, logistická centra apod.). Vrcholy zpravidla značíme malými písmeny (většínou (v)) latinské abecedy s možností indexace pro odlišení. Množiny vrcholu značíme W, *V, U.*

Hrana spojuje dva uzly v grafu. Hrana je jednoduchým modelem například úseku komunikace, které spojuji uzly sítě atd. Existují hrany orientované (mají směr, kterým je možné se mezi uzly pohybovat) a neorientované (mezi uzly se lze pohybovat obousměrně). Množiny hran grafu značíme *X, Y.*

Stupeň vrcholu (označujeme $st(v)$) je dán počtem hran incidujících s vrcholem $v \in V$. Vrcholy se poté rozlišují na vrcholy se sudým a lichým stupni, podle počtu incidujících hran. Vrchol, se kterým inciduje *sudý počet hran* je tedy *sudý vrchol* a vrchol, se kterým inciduje *lichý*

počet hran je *lichý vrchol*. Pro každý graf též platí následující vztah mezi počtem hran a stupni vrcholu:

$$\sum_{v_i \in V} st(v) = 2q$$

Prazdným grafem nazývame graf, pro který platí: $V = \emptyset, X = \emptyset$. Prázdný graf budeme značit $G = \emptyset$.

Kompletním (úplným) grafem nazýváme graf, ve kterém je každý vrchol přilehlý k ostatním vrcholům grafu. Tyto grafy označujeme K_n.

Graf $G = (V,\ X,\ p)$ nazveme **vrcholově (hranově) ohodnoceným** grafem, pokud existuje funkce $o(v)$ (resp. $o(h)$), která přiradí každému vrcholu $v \in V$ (hraně $h \in X$) nezáporné číslo, které vyjadřuje určitou kvantitativní nebo kvalitativní vlastnost vrcholu nebo hrany. Grafy mohou být vrcholově i hranově ohodnocené. Váha hrany c má stejný význam jako termín ohodnocení hrany.[1]

Klika grafu je každý maximální úplný podgraf grafu G. Klika se označuje Q.[34]

Množina sousedů vrcholů $\Gamma(u) \subset V$ je pro každý vrchol definována následovně:

$$\Gamma(u) = \{\, v \in V : \exists h \in X, p\,(h) = (u, v)\}$$

Následující pojmy platí u *neorientovaných* grafů:

- **Orientovaný sled** je střídavá posloupnost vrcholů a orientovaných hran začínající v počátečním vrcholu sledu a končící v koncovém vrcholu orientovaného sledu.
- **Tah** je sled, ve kterém se neopakuje žádná hrana.
- **Dráha** je orientovaný sled, ve kterém se neopakují vrcholy.
- **Cyklus** je uzavřená dráha, ve které je počáteční a koncový vrchol totožný.
- **Cesta** je tah, ve kterém se neopakuje žádný vrchol. Cestu mezi dvěma vrcholy $u, v \in V$ označujeme $m(u, v)$.

Kružnice je uzavřená cesta.

Okružní cesta je cesta, která končí a začíná ve stejném vrcholu grafu.

Délka cesty mezi dvěma vrcholy je definována jako součet ohodnocení hran cesty.

Matice přilehlosti je čtvercová 0/1 matice, která vyjadřuje, zda jsou vrcholy přilehlé, nebo nejsou.

Matice incidence vyjadřuje incidenci vrcholů s hranami.

Matice přímých vzdáleností grafu G je matice $D = (d_{ij})_{i,j1}^{n}$, jejíž prvky jsou definovány následovně:

- $d_{ij} = o(h), jestliže\ \exists h \in X, pro\ kterou\ p(h) = (v_i, v_j)$,
- $d_{ij} = \infty, jestliže\ \nexists h \in X, pro\ kterou\ p(h) = (v_i, v_j)$,
- $d_{ij} = 0, pro\ kterou\ i = j$.

Graf je jednoznačně určen maticí přímých vzdáleností.

Vzdálenost dvou vrcholů $u, v \in V$ v hranově ohodnoceném grafu $G = (V, X, p)$ je definována jako $d(u,v) = \min_{m(u,v)\in M}\{\sum_{h\epsilon m(u,v)} o(h)\}$, kde *M* je množina všech cest mezi vrcholy u a v.

Kostra grafu $G = (V, X)$ je graf $T = (W, Y) | W = V, Y \subset X$, který je stromem. V každem souvislém neorientovaném grafu existuje alespoň jedna kostra. V grafu s n vrcholy má každá kostra právě $n - 1$ hran.

V hranově ohodnocených grafech definujeme minimální kostru grafu. Minimální kostrou souvislého, hranově ohodnoceného grafu je kostra, pro kterou je součet ohodnocení hran minimální.

Souvislý graf je graf, v kterém mezi libovolnou dvojici jeho vrcholů u, v existuje alespoň jedna cesta.

Hamiltonovská kružnice je podgraf grafu, který je kružnicí a obsahuje všechny vrcholy grafu. HK můžeme definovat jako souvislý pravidelný graf druhého stupně obsahující všechny vrcholy grafu.

Minimální/maximální HK je kružnice, jejíž součet ohodnocení hran je minimální/maximální.

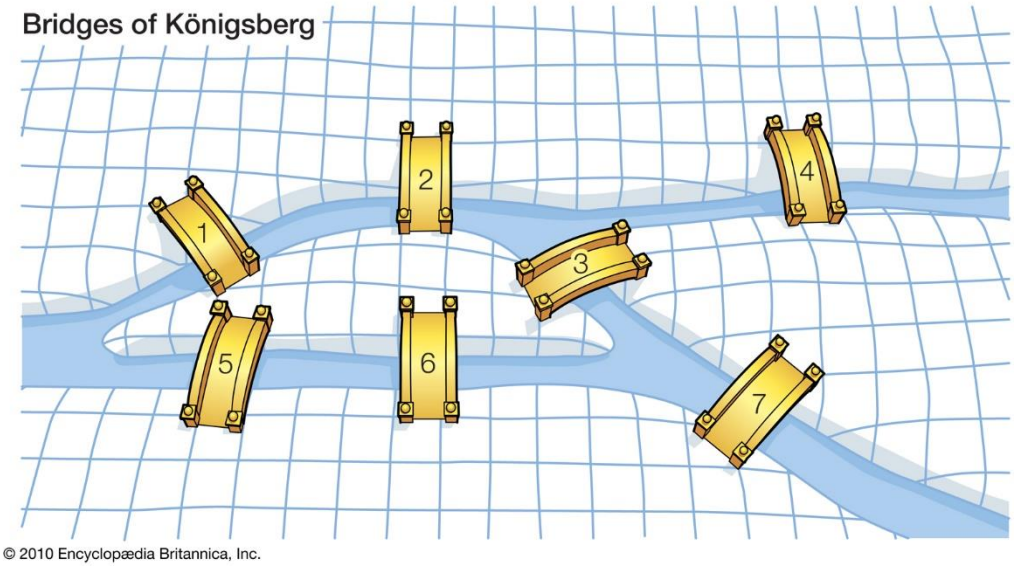
Otevřená cesta se nazývá **hamiltonovská cesta**, obsahuje-li všechny vrcholy (a tudíž všechny vrcholy právě jedenkrát).[1]

Hamiltonovský cyklus je cyklus, který obsahuje každý vrchol grafu.[37]

1 Problém obchodního cestujícího, formulace, klasifikace úloh

1.1 Historie TSP

Historie TSP sahá až do 18. století, kdy začal daný problém studovat jako první švýcarský matematik Leonard Euler tak, že formuloval a vyřešil problém sedmi mostů města Královce. Historické město Královec se rozkládalo na obou březích řeky Pregolji, přičemž uprostřed řeky byly dva ostrovy, které spojovalo se zbytkem města sedm mostů, viz následující obrázek.

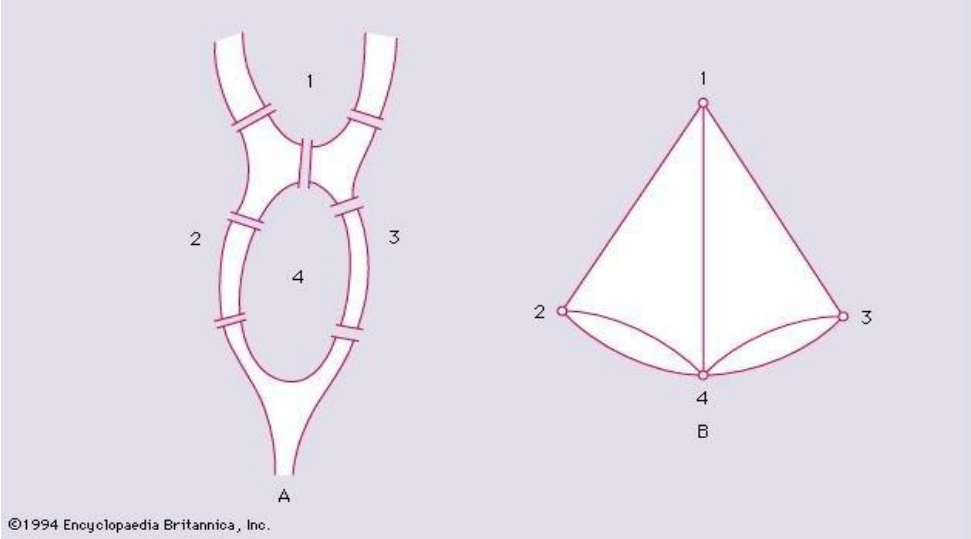


Obrázek 1: Sedm mostů města Královce [17]

Euler se snažil zjistit, zda by bylo možné projít všechny mosty tak, aby se každý most prošel právě jednou a aby se cestující vrátil do výchozího místa. Vyjádřil břehy jako vrcholy a mosty jako hrany mezi nimi, čímž vytvořil graf, který můžeme vidět na obrázku 2.

Úlohu můžeme v teorii grafů formulovat následovně:

- Zvol začátek cesty v libovolném vrcholu grafu.
- Postupně projdi všemi hranami tak, abys prošel každou hranou právě jednou.
- Cestu ukonči ve vrcholu, ve kterém jsi začal.



Obrázek 2: Schéma reprezentující břehy a mosty [19]

Euler dokázal, že projít každým mostem pouze jednou a vrátit se do původního místa není možné, protože graf, který reprezentuje sedm mostů, nesplňuje nutnou podmínku – totiž aby každý vrchol byl sudého stupně. Graf tedy nelze projit eulerovským tahem.[1]

Další známá úloha teorie grafů, kterou řešil Euler, se nazývá „úloha jezdce“ nebo „jezdcova procházka“. Představme si typickou šachovnici, která má 64 polí a figurku jezdce, která se pohybuje podle šachových pravidel po prázdné šachovnici. Úkolem jezdce je projít všechna pole, přičemž na každé smí vstoupit právě jednou. Následně se musí vrátit posledním tahem na původní pole.

Pole, ze kterého jezdec startuje, je označeno jako výchozí vrchol a pole, do kterého se bude přemisťovat, je označeno jako dočasně-koncový uzel. (Finálním koncovým uzlem je zde startovací pole po projití všech polí šachovnice. Jezdec v něm svou úlohu začíná i končí.) Dva uzly se spojí hranou ve chvíli, kdy se jezdec přesune ze startovacího pole do následujícího.

Tímto způsobem musí jezdec projít celou šachovnici a vrátit se do startovacího pole. Jinými slovy se musí najit Hamiltonovská kružnice, podobně jako v TSP.

Po Eulerovi se grafy na dlouhou dobu zůstaly mimo zájem matematiků. Vrátil se k nim v 19. století německý fyzik Gustav Kirchhoff, který formuloval zákony, platné v elektrických obvodech a slouží k výpočtu napětí a proudů v jednotlivých větvích obvodu po mocí koster grafu.[2]

V roce 1856 irský matematik William Hamilton vytvořil hru „Icosian game“, někdy také nazývanou „Hamilton’s Icosian“. Hamilton připojil ke každému vrcholu dodekaedru jméno některé

ze světových metropolí a požadoval, abychom vyšli z některého města, postupovali po hranách, prošli každým městem právě jednou, a nakonec se vrátili do výchozího místa. Jinými slovy bylo úkolem hráčů pospojovat všechny vrcholy pravidelného dvanáctistěnu tak, aby byl každý vrchol použit právě jednou a vytvořila se HK.[3]

1.2 **Formulace TSP**

Obecná formulace problému zní následovně: je dáno několik měst a vzdálenosti mezi nimi. Úkolem obchodního cestujícího je projit těmito městy a následně se vrátit do výchozího města s minimálními výdaji na cestu a zároveň navštívit každé město právě jednou. Formálně se jedná o nalezení nejkratší hamiltonovské kružnice v úplném ohodnoceném grafu.

TSP může být snadno definována jako úloha teorie grafů následovně: nechť $G = (V, A)$ je graf, kde V označuje množinu vrcholu (které reprezentují města) a A je množina hran s nezápornými ohodnoceními (nebo vzdálenostmi) matice $C = (c_{ij})$. Řešení spočívá v určení nejkratší kružnice, která prochází všemi vrcholy právě jednou (Hamiltonovské kružnice) v daném grafu.

Formulace TSP je také obecně známa jako Euklidovské TSP (viz kapitola 1.4), ve kterém je očekávána symetrická vzdálenost matice $C = (c_{ij})$, kde platí $c_{ij} = c_{ji}$ pro všechna $i, j \in V$ a kde platí trojúhelníková nerovnost, tedy $c_{ik} < c_{ij} + c_{jk}$ pro všechny $i, j, k \in V$.

TSP je symetrická, pokud je definována na úplném neorientovaném grafu. V případě, jestli TSP je definovaná na orientovaném grafu, jedná se o asymetrickou TSP (podrobněji se o symetrickém a asymetrickém TSP zaobírají kapitoly 1.3.1 a 1.3.2).[4]

1.2.1 Formulace dle Dantzig-Fulkerson-Johnsona

Označíme města čísly 1 až *n* a v matici vzdáleností definujeme:

$$x_{ij} = \begin{cases} 1\ pokud\ cesta\ vede\ z\ města\ i\ do\ města\ j \\ 0\ \ jinak \end{cases}$$

To znamená, že hodnota x_{ij} je bivalentní a může tedy nabývat pouze dvou hodnot. 0, pokud hrana není zařazena do výsledného cyklu a 1, pokud je hrana součástí výsledného cyklu (vede z města *i* do města *j*).

Nechť c_{ij} je vzdálenost z města i do města j. To znamená, že můžeme zapsat TSP jako následující problém celočíselného lineárního programovaní, kde hodnota účelové funkce vyjadřuje celkovou délku zařazených hran:

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

Podmínky (2) a (3) zabezpečí, že v každém vrcholu právě jedna hrana končí, a právě jedna hrana začíná (z každého a zároveň do každého vrcholu se pojede právě jednou).

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

$$\sum_{j=1, i \neq j}^{n} x_{ij} = 1, i = 1, \dots, n; \tag{3}$$

Omezení (4) zaručí, že se řešení skládá pouze z jednoho cyklu a odstraní se případné podcykly. Zároveň znamená, že počet hran, které mohou být zařazeny do kliky, je definován množinou vrcholů S a nemůže přesáhnout $S - 1$.[5][38]

$$\sum_{i \in Q} \sum_{j \in Q} x_{ij} \leq |S| - 1;\ \forall S \subseteq \{2, \dots, n\}, |S| \geq 2 \tag{4}$$

1.2.2 Formulace dle Miller-Tucker-Zemlina

Hodnota účelové funkce (5) vyjadřuje celkovou délku zařazených hran:

$$min \sum_{i}^{n} \sum_{j}^{n} c_{ij} x_{ij} \tag{5}$$

Podmínky (6) a (7) zabezpečí, že v každém vrcholu právě jedna hrana končí, a právě jedna hrana začíná (z každého a zároveň do každého vrcholu se pojede právě jednou).

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1, \dots, n; \tag{6}$$

$$\sum_{i=1}^{n} x_{ij} = 1, j = 1, \dots, n; \tag{7}$$

Podmínka (8) se nazývá „smyčková podmínka“ a slouží k odstranění podcyklů. Proměnné v_i označuji pořadí, ve kterém města i jsou navštívena $(i \neq 1)$.

$$v_i - v_j + n x_{ij} \leq n - 1;\ i = 2, \dots, n, j = 2, \dots, n;\ i \neq j; \tag{8}$$

Podmínka (9) vyjadřuje skutečnost, že hodnota x_{ij} je bivalentní, bivalentní a může tedy nabývat dvou hodnot: 0 pokud hrana není zařazena do výsledného cyklu, 1 pokud je hrana součástí výsledného cyklu (vede z města *i* do města *j*).[7][38]

$$x_{ij} \in \{0,1\};\ i = 1, \dots, n;\ i = 1, \dots, n. \tag{9}$$

1.3 Klasifikace úloh obchodního cestujícího

1.3.1 Symetrická úloha obchodního cestujícího (TSP)

Podle druhu matice vzdálenosti se TSP dělí do dvou typů. Pokud je matice vzdálenosti symetrická (graf G je neorientovaný, a hranou lze procházet v obou směrech), tak se TSP nazývá „symetrická úloha obchodního cestujícího“ (STSP).

Symetrii úloh lze jednoduše ukázat na matici vzdáleností níže na obrázku č.3. V matici jsou v řádcích a sloupcích označeny vrcholy (města) a vzdálenost d_{ij} mezi nimi. Na hlavní diagonále matice se nachází nuly, v ostatních polích pak délky minimálních cest mezi jednotlivými vrcholy.

	V1	V2	V3	V4
V1	0	d_{12}	d_{13}	d_{14}
V2	d_{21}	0	d_{23}	d_{24}
V3	d_{31}	d_{32}	0	d_{34}
V4	d_{41}	d_{42}	d_{43}	0

Obrázek 3: Matice vzdáleností [autorka 2019]

1.3.2 Asymetrická úloha obchodního cestujícího (ATSP)

V případě, kdy matice vzdáleností není symetrická (ekvivalentně: graf G je orientovaný), jde o asymetrickou úlohu obchodního cestujícího. V ATSP existuje různé ohodnocení cest mezi dvěma vrcholy ($c_{ij} \neq c_{ji}$). S takovým případem se můžeme setkat, pokud budeme cestovat městem, ve kterém jsou jednosměrné ulice, nebo pokud se stane dopravní nehoda a některé ulice budou v jednom směru neprůjezdné.

Vzhledem k tomu, že každý neorientovaný graf je možné vnímat jako orientovaný, pomocí zdvojení hran (každou neorientovanou hranu lze vnímat jako dvojici protisměrně orientovaných hran) můžeme vnímat STSP jako speciální případ ATSP.

1.3.3 Modifikace úlohy obchodního cestujícího

Úloha obchodního cestujícího s časovými okny (TSP with Time Windows)

Na rozdíl od klasického TSP je ještě omezen tím, že každému vrcholu (zákazníkovi) je přiřazeno tzv. časové okno (období, kdy ho musí obchodní cestující navštívit). Časové okno i-tého zákazníka je časový interval mezi nejdříve možným začátkem obsluhy a nejpozdějším možným koncem obsluhy.

Úloha s více návštěvami (TSP with multiple visits)

V této úloze chceme najít cestu obchodního cestujícího, která začíná ve výchozím vrcholu grafu G a cestující projde každý vrchol *alespoň* jednou. Poté se vrátí do výchozího uzlu takovým způsobem, že celková délka cesty je minimalizována. Od klasického TSP se liší tím, že zde cestující může navštěvovat vrcholy vícekrát, pokud mu to zkrátí celkovou délku cesty.

Maximální úloha obchodního cestujícího (MAX TSP)

Narozdíl od klasického TSP je zde cílem najít cestu v grafu G, kde je součet vah hran co největší. Tedy nalézt nejdelší možnou cestu. MAX TSP lze řešit jako TSP s nahrazením každé váhy hrany jejím opačným číslem. Pokud požadujeme, aby váhy hran byly nezáporné, může být ke každé z vah hrany přidána konstanta beze změny optimálního řešení problému. Tim se jedná o minimalizační úlohu, kde každou hranu ohodnotíme vahou, která se rovná záporné hodnotě ohodnocení hrany. Minimalizační úloha se tak převádí na maximalizační.

Úzkoprofilová úloha obchodního cestujícího (Bottleneck TSP)

V případě „Bottleneck“ je cílem najít takovou cestu v grafu *G*, aby největší váha hrany při cestě byla co nejmenší. Bottleneck TSP lze formulovat jako TSP s exponenciálně velkými váhami hran.[5]

1.4 Metriky

Metrika popisuje a umožnuje formálním způsobem definovat pojem vzdálenosti dvou bodů v prostoru nebo časoprostoru, zároveň umožňuje vypočítat různé vlastnosti prostoru.

Dělí se na tři základní skupiny, jejichž popis následuje.[13]

1.4.1 Geometrická metrika

V této metrice platí trojúhelníková nerovnost, podle které součet délek dvou stran trojúhelníku nemůže být menší než délka třetí strany. To znamená, že cesta z A do B a následně do C není kratší než cesta z A přímo do C.

V metrickém prostoru M s metrikou d má trojúhelníková nerovnost tvar:

$$d\,(uw) \leq d(uv) + d(vw), \forall\, u, v, w \in V(G)$$

To znamená, že vzdálenost u a w není větší než součet vzdálenosti z u do v a vzdálenosti z v do w.[13]

1.4.2 Euklidovská metrika

Zvláštní typ metrické TSP, jelikož vzdálenosti v rovině ctí trojúhelníkovou nerovnost. Navíc v Euklidovském TSP vzdálenost mezi dvěma body (městy) rozumíme jako Euklidovskou vzdálenost (klasická přímá vzdálenost mezi dvěma body v Euklidovském prostoru).[13] Euklidovská metrika je daná vztahem:

$$d(p,q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \ldots + (q_n - p_n)^2} = \sqrt{\sum_{i=q}^{n} (q_i - p_i)^2},$$

kde $p = (p_1, p_2, \ldots, p_n)$ a $q = (q_1, q_2, \ldots, q_n)$ jsou body v Euklidovském prostoru, vzdálenost *d* z *q* do *p*, nebo z *p* do *q* je dána Pythagorovou větou: $d^2 = q^2 + p^2$.[39]

1.4.3 Manhattanská metrika

TSP metrika, v němž platí pravidla Manhattanské (též Newyorské) metriky, která funguje podle pravoúhlého systému ulic na Manhattanu v New Yorku a která odpovídá představě nejkratší vzdálenosti, kterou musí urazit automobil při cestě z jedné křižovatky na jinou – předpokládáme, že systém ulic je pravoúhlý.[13]

2 Exaktní, heuristické a metaheuristické metody

Existují různé metody řešení TSP – lze je dělit do tří skupin: exaktní metody, heuristické a metaheuristické. Tyto metody se liší výpočetní náročností, rychlostí a kvalitou výsledného řešení. Exaktní metody jsou vhodné pouze pro malý počet vrcholů grafu. Je to z toho důvodu, že – jak bylo zmíněno dříve – TSP patří mezí NP-úplné úlohy, kde je těžké dodržet optimální výsledek v polynomiálním čase. S růstem rozměru grafu roste výpočetní složitost, proto se často používají alternativní (heuristické a metaheuristické) metody řešení.

Řešení získané pomocí heuristik a metaheuristik však nemusí dát optimální řešení, ale může se k optimálnímu řešení přiblížit. I když řešení nemusí být optimální, výhoda takových metod je v mnohem kratším výpočetním čase. „Heuristic“ znamená „hledat“ nebo „objevovat metodou pokusu a omylu“. Další krok ve vývoji heuristických algoritmů jsou takzvané metaheuristické algoritmy. Zde „meta“ znamená „nad“ nebo „vyšší úroveň“ a tyto algoritmy obecně dávají lepší výsledky, protože nekončí v lokálním minimu.

Rozdíl mezi jednoduchou heuristikou a sofistikovanou metaheuristikou je docela velký, především co se týče kvality a přiblížení se k optimálnímu řešení. Jednoduché heuristiky zpravidla končí v lokálním minimu, zatímco metaheuristiky dokážou optimum hledat na celé množině přípustných řešení.

Nejmladší skupinou algoritmů jsou algoritmy inspirované přírodou, které jsou většinou metaheuristické a v posledních letech si získaly hodně popularity. [6]

2.1 Exaktní metody

Jak již bylo zmíněno dříve, exaktní metody vždy naleznou optimální řešení (pokud existuje), ale pro úlohy většího rozsahu jsou nevhodné, protože doba výpočtu je příliš velká, nebo nelze tohoto řešení dosáhnout kvůli časové náročnosti. S růstem počtu míst úlohy rychle narůstá výpočetní složitost, a proto pro úlohy většího rozsahu je nutné použít alternativní metody řešení.

Mezí exaktní metody patří metoda větvení a mezí a její variace (metoda větvení a řezu, metoda větvení a oceňovaní, metoda větvení, řezu a oceňovaní), dále metody lineárního programování a metoda hrubé sily prozkoumávání všech možných přípustných řešení. V této kapitole bude popsána metoda Branch&Bound (větvení a mezi) jako jedna z nejjednodušších exaktních metod a metoda zpětného prohledávání (backtracking).

2.1.1 Branch-and-bound (metoda větvení a mezí)

Metoda větvení a mezí je metoda pro řešení úloh celočíselného programování. Tato metoda používá strategii hledání do šířky ve stromu řešení, který je postupně vytvářen. Metoda větvení a mezí se snaží odřezat neperspektivní větve, které prokazatelně neobsahují optimální řešení v tzv. stromu řešení (neboli stromu problému), a tím snížit exponenciální složitost algoritmu. Větve se odřezávají na základě znalosti optimalizačního kritéria a omezujících podmínek.

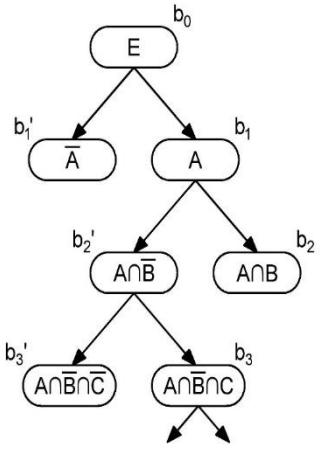
Snahou je ořezat co nejvíce větví, které jsou co nejblíže kmeni, ale zároveň neuříznout větev, ve které se nachází optimální řešení.

$E = \{S_1, S_2, \dots, S_n\}$ je množina řešení některé úlohy diskrétní optimalizace a f je funkce definovaná na této množině. Je potřeba nalézt podmnožinu $E_m \subset E$, ve které funkce dosahuje minima (jestliže existuje).

S pomocí vlastnosti P_A je možné rozdělit E na dvě podmnožiny A a $\bar{A}$, přičemž $A \cap \bar{A} = \{\emptyset\}, A \cup \bar{A} = \mathrm{E}$.

Předpokládejme, že dokážeme nalézt spodní hranici b_0 hodnot funkce f na množině řešení E. Dále předpokládejme, že dokážeme stanovit i spodní hranice $b_1 \geq b_0, b'_1 \geq b_0$ funkce f na A, resp. $\bar{A}$.

Vlastnosti P_B a P_C i další též umožňují rozdělit E na dvě disjunktivní podmnožiny. Potom vlastnostem $P_B \wedge P_A, \overline{P_B} \wedge P_A, \overline{P_A} \wedge P_B, \overline{P_A} \wedge \overline{P_B}$ odpovídají podmnožiny $B \cap A, \bar{B} \cap A, \bar{A} \cap B, \bar{A} \cap \bar{B}$. Každou podmnožinu považujeme za vrchol grafu. Tímto způsobem je možné vytvořit graf, který se nazývá strom a je znázorněn na obrázku 4.



Obrázek 4: Vytváření stromu [1]

Postup vytváření stromu je následující:

Krok 1:

- Předpokládejme, že jsme sestavili část stromu využitím k-vlastnosti $P_1, P_2, \dots, P_k$ a našli spodní hranici funkce f této části.

Krok 2:

- Vybereme visící vrchol stromu, který má nejnižší hranici.

Krok 3:

- S pomocí k vlastností nebo $k+1$ vlastností získáme další dva nové vrcholy, pro které určíme spodní hranici b.
- Pro vrcholy stromu reprezentující podmnožiny množiny řešení E platí následující vlastnost: sjednocením všech vrcholů (visících) v každé fázi dostaneme původní množinu řešení E. Pro strom na obrázku tedy platí:

$$E = \bar{A} \cup (A \cap B) \cup (A \cap \bar{B} \cap \bar{C}) \cup (A \cap \bar{B} \cap C).$$

Pokud výsledkem daného procesu je visící vrchol, který představuje jednoprvkovou množinu, potom funkce f nabývá na této množině minimální hodnotu.[1]

Nevýhodou této metody je její výpočetní složitost. Jak už bylo zmíněno dříve, je nepoužitelná pro úlohy většího rozměru, protože její časová náročnost roste exponenciálně s lineárním růstem úlohy. Aby byl čas řešení větších úloh únosný, používají se alternativní metody, které budou zmíněny dále.

2.1.2 Metoda zpětného prohledávání – backtracking

Metoda zpětného prohledávání (lze se setkat s názvy metoda pokusu a omylů, metoda hledaní s návratem, metoda prohledávání do hloubky) je založena na prohledávání stavového prostoru problému. Backtracking hledá optimální řešení rekurzivně, postupně sestavuje řešení kousek po kousku a odstraňuje řešení, které nesplňují určité podmínky nebo omezení. Některá řešení jsou vyloučena i bez jejich vyzkoušení. Dílčí problémy jsou řešeny jeden po druhém, čímž se dostaneme k nejlepšímu možnému řešení.[14]

2.2 Heuristické metody

Jak již bylo poznamenáno, TSP patří mezi NP-úplné problémy, které jsou obtížně řešitelné exaktními metodami, využívají se tedy zejména v případech větších rozměrů řešených úloh. Získaná řešení nemusí být vždy optimální, ale mohou se k nim přiblížit a to v rozumném čase. Následující podkapitoly jsou věnovány popisu heuristických algoritmů.

Existuji jednoduché (konstrukční, záměnné, vypouštěcí heuristiky) a metaheuristické metody.

2.2.1 Konstrukční heuristiky

Konstrukční heuristiky jsou využívány za účelem vytvoření řešení úlohy, bez ohledu na „nejlepší řešení“.[8]

Hladový algoritmus

Hladový algoritmus je jednoduchý heuristický algoritmus, který však většinou nevykazuje příliš dobré výsledky. Tento algoritmus je velmi podobný algoritmu hledání minimální kostry grafu. V každém svém kroku algoritmus vybírá lokální minimum (hranu s nejmenším ohodnocením) a existuje možnost, že tak nalezne i minimum globální. Vybranou hranu následně algoritmus zařadí do výsledné cesty. Tento algoritmus byl vyvinut pro reseni ulohy TSP na kompletním grafů. Aby byla splněna podmínka Hamiltonovského cyklu, všechny hrany zařazené do výsledné cesty musí splňovat tyto podmínky:

- po přidání hrany nevznikne kružnice s menším počtem hran než *n*;
- do/z jednoho vrcholu nevchází/nevychází více než jedna hrana;
- počet hran zařazených se rovná počtu vrcholů grafu.

Potom algoritmus spojí první a poslední vrchol cesty, a tím vznikne HK.

Algoritmus lze shrnout do následujících kroků:

Uvažujeme hranově ohodnocený graf $K_n = (V, X)$. V grafu K_n najdeme podgraf $H = (W, Y)$, který bude HK. Kružnice může začit v libovolném nebo zadaném vrcholu grafu (v počátečním vrcholu).

Krok 1:

- Zvolíme počáteční vrchol kružnice, $v_p \in V$.
- Položíme $W = \emptyset, Y = \emptyset$.
- Počáteční vrchol zařadíme do $W (v_p \rightarrow W)$, položíme $v_s = v_p$.
- Určíme $\Gamma(v_p)$ množinu přilehlých (sousedních) vrcholů v_p.
- Položíme $\bar{\Gamma}(v_p) = \Gamma(v_p) \backslash W$.

Krok 2:

- Vybereme hranu $h^* \in X | \, p\,(h^*) = (v_p, v_i), v_i \in \bar{\Gamma}\,(v_p)$ s minimálním ohodnocením:

$o(h^*) = \min\limits_{v_i \epsilon \Gamma(v_p)} \{o(v_p, v_i)\}$ a zařadíme ji do HK $h^* \rightarrow Y$, vrchol v_i zařadíme do W.

Krok 3:

- Položíme $v_p = v_i$ a určíme $\bar{\Gamma}(v_p) = \Gamma(v_p) \backslash W$.
- Je-li $\bar{\Gamma}(v_p) = \{\emptyset\}$, zařadíme poslední hranu uzavírající kružnici $(v_p, v_s) \rightarrow Y$ a pokračujeme krokem 4. Je-li $\bar{\Gamma}(v_p) \neq \{\emptyset\}$, pokračujeme krokem 2.

Krok 4:

- Sečteme ohodnocení hran zaražených do HK. Součet je hodnotou minimální HK.[1]

Metoda nejbližšího souseda

Jednou z nejjednodušších heuristik je právě metoda nejbližšího souseda. Základem této metody je hladový algoritmus, který vytvoří HK tak, že se v každém kroku algoritmu vyberou dva nejbližší vrcholy a spojí se. Po zpracování všech vrcholů v grafu se cesta uzavře.

Všechny hrany zařazené do výsledné cesty musí splňovat tyto dvě podmínky:

- po přidání hrany nevzniká kružnice;
- z jednoho vrcholu nevychází více než jedna hrana a nevhází víc než jedna.

Po splnění podmínek se zařadí hrana s minimálním ohodnocením. Pokud hrana nevyhovuje alespoň jedné z podmínek, algoritmus zkusí další hranu z daného uzlu, která má jinak nejmenší ohodnocení. Pak se přesune do uzlu, do kterého vedla nejkratší hrana splňující všechny podmínky, a postup se opakuje, dokud cesta se neuzavře návratem do výchozího vrcholů. Algoritmus lze shrnout do následujících kroků:

Krok 1:

- Zvolíme libovolný počáteční vrchol $v_j \in V$, položíme $l = j \; a \; W = \{1, .., n\} \backslash \{j\}$.

Krok 2:

- Dokud $W = \emptyset$, opakujeme:
    1) vybereme vrchol $v_j, j \in W$ tak, že platí $c_{lj} = \min\{c_{li} \mid i \in W\}$
    2) spojíme vrcholy v_l a v_j a položíme $W = W \backslash \{j\}$ a $l = j$

Krok 3:

- Pro uzavření Hamiltonovského cyklu spojíme vrchol v_n a vrchol v_j, který jsme vybrali první.

Metoda nejbližšího souseda je rychlá a jednoduchá, ale – stejně jako hladový algoritmus – nemusí vést k dobrým výsledkům. Jednou z možností vylepšení výsledků algoritmu je možnost vybrat si jako počáteční uzel postupně všechny uzly, spustit algoritmus pro každý z nich a následně vybrat nejlepší řešení. Takový postup zlepší výsledek, ale čas řešení úlohy vzroste.[8]

Vkládací metoda

Další z řady intuitivních přístupů k danému problému je metoda vkládací.[8] Tato metoda začíná cyklem s malou podskupinou vrcholů a pokračuje přibíráním zbývajících vrcholů. Tento cyklus se zvětšuje do té doby, dokud se nezapojí všechny vrcholy grafu. Tím je vytvořen Hamiltonovský cyklus. Přesný postup je následující

Krok 1:

- Vybereme počáteční cyklus n vrcholů $v_1, v_2, \dots, v_n (\, n \geq 1)$, $W = V \backslash \{v_1, v_2, \dots, v_n\}$.

Krok 2:

- Dokud $W = \emptyset$, opakujeme:
    1) vybereme vrchol $v_j, j \in W$ podle zadaného kritéria
    2) vložíme v_j na některou pozici v cyklu a položíme $W = W \backslash \{j\}$

Existuje ale několik možností pro implementaci tohoto vkládacího postupu. Hlavním rozdílem mezi nimi je kritérium výběru vrcholu j v první části druhého kroku. Počáteční cyklus se může skládat minimálně ze tří vrcholů nebo se (v degenerovaných případech) skládá ze smyčky $(n = 1)$ nebo z hrany $(n = 2)$. Vybraný vrchol se vkládá do cesty tak, aby prodloužení délky cyklu bylo minimální.

Existuje několik možností pro prodloužení aktuálního cyklu. Říkáme, že vrchol je vrcholem zařazeným do cyklu, pokud je již obsažen v částečném Hamiltonovském cyklu. Pro $j \in W$, kde $d_{min}(j) = \min\{c_{ij} | \, i \in V \setminus W\}$.

- **Nejbližší vkládání:** Vložíme vrchol, který má nejmenší vzdálenost od vrcholů cyklu, tedy vybereme $j \in W$, kde $d_{min}(j) = \min\{d_{min}(l) | \, l \in W\}$.
- **Nejvzdálenější vkládání:** Vložíme vrchol, jehož nejmenší vzdálenost od vrcholu cyklu je maximální, tj. vybereme $j \in W$, kde $d_{min}(j) = \max\{d_{min}(l) | \, l \in W\}$.
- **Nejlevnější vkládaní:** Vložíme vrchol tak, aby prodloužení délky cyklu bylo co nejmenší.
- **Náhodné vkládání:** Vrchol vybereme náhodně a vložíme ho na co nejlepší pozice.

Tyto heuristiky však nejsou vhodné pro úlohy větších rozměrů z důvodu velké paměťové náročnosti.[8]

Metoda minimální kostry

Metoda minimální kostry je založená na algoritmu hledání minimální kostry grafu a následné transformaci na Hamiltonovský cyklus. Existuje několik přístupů a algoritmů pro hledání minimální kostry grafu.

Borůvkův algoritmus

Borůvkův algoritmus pochází z roku 1926 jako metoda pro konstrukci efektivní elektrické sítě na Moravě. Nejprvé byl objeven Otakarem Borůvkou a pak znovuobjeven jinými vědci, mezi které patřil například Georges Sollin a proto se o algoritmu občas lze dočíst jako o Sollinově algoritmu. Je nejstarším z algoritmů pro sestrojení minimální kostry. Borůvkův algoritmus postupně spojuje komponenty souvislosti (na počátku jsou vrcholy komponentami souvislosti) do větších a větších podskupin. V každém kroku spoji komponenty souvislosti hranou s co nejmenším ohodnocením, dokud nezůstane pouze jediná skupina vrcholů, což bude právě hledaná minimální kostra.[36] Postup algoritmu je následující:

Krok 1:

- Na začátku algoritmu každý vrchol je komponentou souvislosti

Krok 2:

- Pokud existuje vice, než jedná komponenta, opakujeme pro každou komponentu:
    1. Najdeme hranu s co nejmenším ohodnocením, která spojuje komponentu s jinou komponentou.

2. Přidáme tuto hranu do minimální kostry, a spojíme dvě komponenty pomocí této hrany.

Krok 3:

- Algoritmus končí v chvíli, kdy zůstane jenom jedná komponenta. Tato komponenta je minimální kostrou grafů.[35]

Kruskalův algoritmus

Tento algoritmus je založen na principu hladového algoritmu. Na začátku se hrany seřadí podle ohodnocení od nejmenšího do největšího a následně se zařazují do kostry tak, že jejich zařazením nevznikne kružnice. Hrany s co nejmenším ohodnocením se zařazují až do té chvíle, kdy hrany spojí vrcholy celého grafu. Postup algoritmu je následující.

Krok 1:

- Seřadíme hrany grafu podle ohodnocení. Do kostry vložíme všechny vrcholy grafu $W = V$, položíme $Y = \{\emptyset\}$.

Krok 2:

- Vybereme hranu $h^* \in X$ s minimálním ohodnocením a zařadíme ji do kostry grafu $(h^* \rightarrow Y)$.

$$o(h^*) = \min_{h \in X/Y} \{o(h)\}$$

Krok 3:

- Z dosud nevybraných hran seznamu vybereme další hranu s minimálním ohodnocením. Do kostry ji zařadíme v případě, že jejím zařazením nevznikne kružnice. V opačném případě vybereme další hranu.

Krok 4:

- Podle kroku 3 postupujeme do té doby, dokud je možné vybrat alespoň jednu hranu, jejíž zaražení nezpůsobí vznik kružnice, jinak přejdeme na krok 5.

Krok 5:

- Sečteme ohodnocení vybraných hran, hodnota součtu je hodnotou minimální kostry grafu.[1]

Jarníkův-Primův algoritmus

Tento algoritmus byl objeven v roce 1930 Vojtěchem Jarníkem a následně znovuobjeven Robertem Primem v roce 1957 a také Edsgerem Dijkstrou v roce 1959. Hrany se na začátku neseřazují, kostra se začíná vytvářet z jednoho vrcholu. Poté algoritmus v každém kroku připojí vrchol, mezi nímž a již propojenou částí grafu je hrana s minimální délkou. Tím dojde ke zvětšení velikosti kostry až do chvíle, kdy se vyčerpají všechny vrcholy. Postup algoritmu je následující.

Krok 1:

- Položíme $W = \{\emptyset\}, Y = \{\emptyset\}$.

Krok 2:

- Vybereme hranu $h^* \in X \setminus Y \mid p(h^*) = (u, v)$ s minimálním ohodnocením a zařadíme ji do kostry $T = (W, Y)(h^* \to Y; u, v \to W)$.

$$o(h^*) = \min_{h \in X/Y} \{o(h)\}$$

Krok 3:

- Určíme množiny sousedů – $\Gamma(v_i)$ pro všechny vrcholy $v_i \in W$ a také určíme množinu $\Gamma(W) = \cup\ \Gamma(v_i) \setminus W$.

Krok 4:

- Množiny hran $h \in X \setminus Y \mid p\,(h) = (v_i, v_j), v_i \in W, v_j \in = \cup\ \Gamma(W) \setminus W$ vybereme hranu s minimálním ohodnocením a zařadíme ji do kostry, pokud jejím zařazením nevznikne kružnice $(h^* \to Y; v_j \to W)$.

Krok 5:

- Krok 3 a 4 opakujeme do té doby, dokud $W \neq V$, jinak pokračujeme krokem 6.

Krok 6:

- Sečteme ohodnocení vybraných hran; hodnota součtu je hodnotou minimální kostry grafů.[1]

Christofidova metoda

Algoritmus pro nalezení optimálního řešení TSP s využitím minimální kostry grafu publikoval Nicos Christofides, matematik a profesor finanční matematiky. Christofidova metoda se liší od ostatních algoritmů tím, že většina heuristických algoritmů zaručuje nejhorší výsledek ve dvojnásobné výši oproti optimu (algoritmem nalezená trasa je v nejhorším případě dvakrát delší než skutečná optimální trasa). Christofidesova metoda garantuje, že nejhorší možná trasa má vůči optimu poměr 3/2 (tedy že nejhorší nalezená trasa je o polovinu delší než délka optimální trasy).

Tato metoda nejprve vyhledá minimální kostru grafu a pak ji transformuje na Hamiltonovský cyklus. Postup algoritmu je následující.

Krok 1:

- Vytvoříme minimální kostru *T* grafu *G* (např. pomocí Borůvkova algoritmu).

Krok 2:

- Vytvoříme podmnožinu vrcholu *O* grafu *T*, která obsahuje pouze vrcholy s lichým stupněm.

Krok 3:

- Na množině *O* vytvoříme minimální perfektní párování.

Krok 4:

- Hrany minimální kostry grafu *T* a minimálního perfektního párování sjednotíme do jedné množiny, ve které je každý vrchol vrcholem sudého stupně.

Krok 5:

- Ze sjednocené množiny vytvoříme Eulerovský cyklus.

Krok 6:

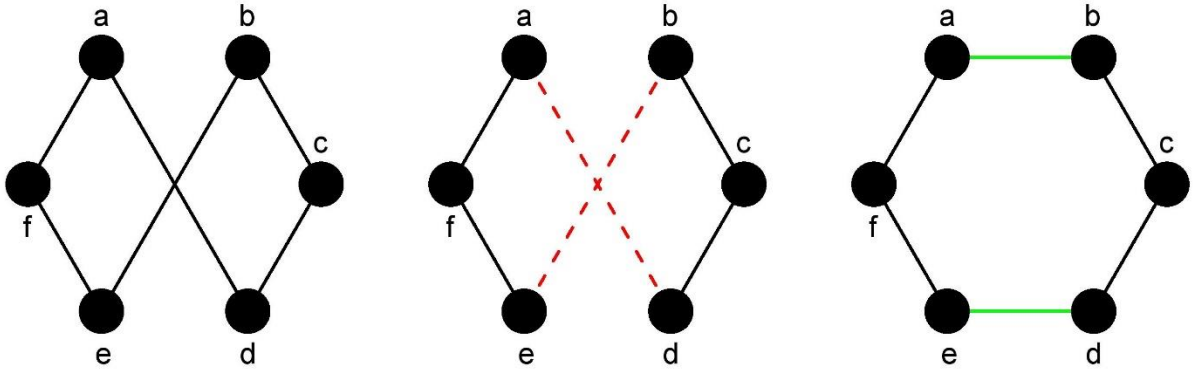
- Eulerovský cyklus transformujeme na Hamiltonovský cyklus tím, že vytvoří posloupnost po sobě jdoucích vrcholů v Eulerovském cyklu a vynechají se opakující se vrcholy.[40]

2.2.2 Zlepšovací / záměnné heuristiky

Metoda 2-opt

Tato metoda je založena na lokálním prohledávání. Metoda 2-opt nehledá řešení TSP, ale slouží ke zlepšování již existujícího řešení TSP.

Pokud se Hamiltonovský cyklus křižuje sám se sebou, dvě křižující se hrany jsou odstraněny a nahrazeny jinými dvěmi hranami tak, aby nově vzniklá cesta tvořila Hamiltonovský cyklus s lepším ohodnocením. Nejprve se najde počáteční řešení TSP (např. pomocí Borůvkova algoritmu, algoritmu nejbližšího souseda aj.), poté algoritmus začne toto řešení zlepšovat tak, že zamění dvě hrany v celkové cestě a otestuje, zda je ohodnocení nové cesty lepší než ohodnocení původní cesty. Princip této metody je znázorněn na obrázku 5.



Obrázek 5: Vyměna hran metodou 2-opt [autorka 2019]

Postup algoritmu je následující:

Krok 1:

- Nechť T je libovolný Hamiltonovský cyklus.

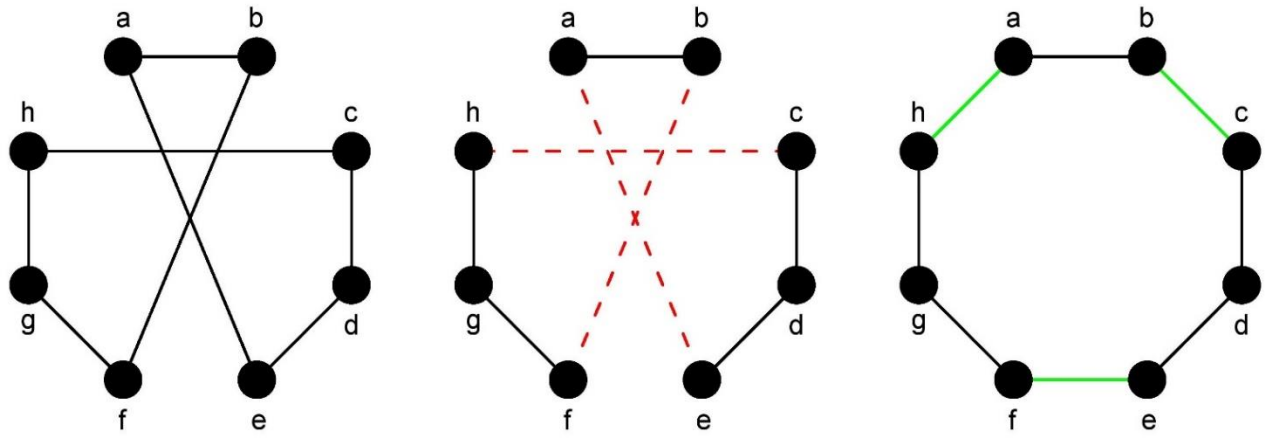
Krok 2:

- Pro každý vrchol i opakujeme následující kroky, až do chvíle, kdy vznikne chyba:
  a) vybereme vrchol i.
  b) prozkoumáme všechny kroky 2-opt které zahrnují hranu mezi vrcholem a jeho nástupcem v cyklu. Ze všech řešení vybereme to, které poskytuje nejkratší délku v cyklu. Pokud neexistuje žádné možné zkrácení, hlásíme chybu pro i.

Krok 3:

- Získané nové T je výsledkem algoritmu.[8]

Existuje také metoda 3-opt, která funguje velice podobně jako metoda 2-opt, jen místo dvou hran jsou zaměněny tři. Metoda 3-opt je účinnější, ale i náročnější než 2-opt. Princip metody 3-opt je znázorněn na obrázku 6:



Obrázek 6: Vyměna hran metodou 3-opt [autorka 2019]

Na stejném konceptu metody k-opt je založen i Lin-Kernighanův algoritmus (jeden z nejlepších algoritmů pro řešení TSP). Lin-Kernighanův algoritmus se přizpůsobí a v každém kroku rozhoduje, kolik hran je potřeba vyměnit, aby celková cesta byla kratší.[8]

2.3 **Metaheuristické metody**

Metaheuristické metody stejně jako heuristiky nezaručují optimální řešení. Heuristiky jsou speciálně vytvořené pro určité typy úloh, zatímco metaheuristiky lze definovat jako obecné algoritmy pro řešení obtížných úloh.

2.3.1 Genetický algoritmus

Genetické algoritmy jsou součástí evolučních algoritmů a jsou inspirované Darwinovou evoluční teorií.

Jejich základem je tvorba generací různých řešení daného problému, čímž se dojde až k optimálnímu řešení. Základem algoritmu jsou jedinci, jejichž kvalitu můžeme ohodnotit pomocí hodnoty fitness funkce. Čím lepší má jedinec hodnotu fitnes, tím větší má šanci přežít a reprodukovat se. Aktuální množina jedinců tvoří generaci, ze které následně pomocí křížení

a reprodukce vzniká nová generace. V té jsou vlastnosti jedinců částečně získány po rodičích, přičemž jsou částečně ovlivněny náhodnými mutacemi v procesu reprodukce.

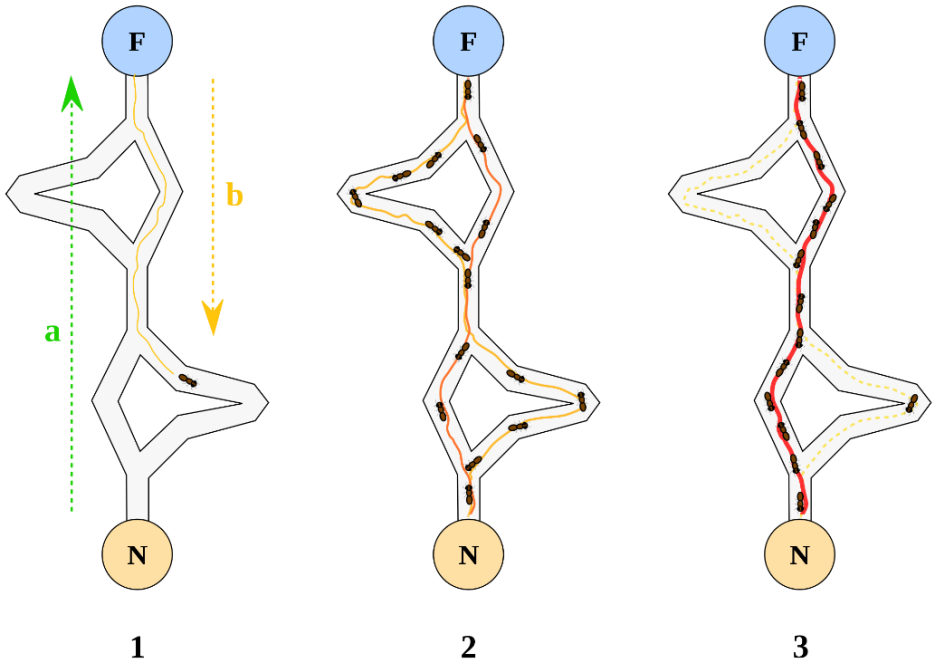
Po desítkách opakování se vyvine populace s jedinci, kteří mají vysoké ohodnocení a reprezentují optimální řešení daného problému. Může se stát i to, že celá populace v procesu vývoje zdegeneruje a nejlepší jedinec bude reprezentovat pouze lokální optimum. K tomu dochází z toho důvodu, že podobně jako v přírodě, v sobě tento evoluční proces zahrnuje značný díl náhodností.[9]

Obecné schéma algoritmu vypadá následovně:

1) Vynuluj hodnotu počítadla generací $t = 0$.
2) Náhodně vygeneruj počáteční populaci $P(0)$.
3) Vypočítej ohodnocení (fitness) každého individua v počáteční populaci $P(0)$.
4) Vyber dvojice individuí z populace $P(t)$ a vytvoř jejich potomky $P'(t)$.
5) Ohodnoť nově vytvořená individua v $P'(t)$.
6) Vytvoř novou populaci $P(t+1)$ z původní populace $P(t)$ a množiny potomků $P'(t)$.
7) Zvětši hodnoty počítadla generací o jedničku $(t \coloneqq t+1)$.
8) Vypočítej ohodnocení $(fitness)$ každého individua v populaci $P(t)$.
9) Pokud t je rovno maximálnímu počtu generací nebo je splněno jiné ukončovací kritérium, vrať jako výsledek populaci $P(t)$; jinak pokračuj krokem 4.[9]

2.3.2 Ant Colony Optimization – Optimalizace Mravenčí kolonie

Optimalizace Ant colony pracuje s myšlenkou, že hledání potravy mnoha druhů mravenců je založeno na nepřímé komunikaci zprostředkované prostředím. Na cestě mezi mraveništěm a zdrojem potravy totiž mravenci vytvářejí chemickou stopu v podobě feromonů. Tu ostatní mravenci cítí a s vyšší pravděpodobností volí spíše cestu se silnou koncentrací feromonů, čímž dojde k jejímu opětovnému posílení. Pokud ovšem feromonová stopa není udržovaná, vyprchá a zmizí.[10] Mravenci se řídí stopou s nejsilnější koncentrací feromonů a po určité době najdou optimální (nejkratší) cestu od mraveniště k potravě. Pak preferují právě tuto trasu, protože čím kratší je trasa, tím víckrát ji mravenci stihnou použít a tím se stopa feromonu ještě více posílí.



Obrázek 7: Princip ACO - feromony [16]

Pro TSP dá se využívat ACO tak, že máme ze začátku nějaký počet mravenců, například 20. Náhodně je umístíme ve vrcholech a dovolíme jim pohybovat se libovolně mezi nimi. Ovšem, pro mravence je zakázané opětovně navštěvovat vrchol, ve kterém se již jednou nacházel s výjimkou, pokud se jedná o vracení do původního vrcholu a uzavření cesty. Mravenec, který prošel nejkratší cestou, po sobě zanechává stopu feromonu nepřímo úměrnou délce jeho cesty. Stezku s nejsilnější stopou feromonů poté budou preferovat ostatní mravenci. Tento proces se bude opakovat, dokud nebude nalezena nejkratší cesta.[22]

2.3.3 Tabu search

Metoda Tabu Search je založena na metodě lokálního prohledávání. Tabu Search se snaží odstranit problém uváznuti v lokálním optimu, k čemuž často dochází u heuristických metod. Pro vyvedení algoritmu z lokálního optima se využívá tabu seznam. Do tohoto seznamu se ukládají všechna řešení, ke kterým se algoritmus nesmí vracet, protože už byla použita. Tabu seznam má však omezenou kapacitu, takže nejstarší vložené transformace se ze seznamu vymazávají při vkládání nových. Použití tabu seznamu způsobí, že při dosazení lokálního extrému bude algoritmus vychýlen a bude muset pokračovat v prohledávání nových oblastí.[10]

3 Aplikace TSP v logistice a dopravě

Aplikace TSP lze spatřit v mnohých odvětvích lidské činnosti. Její uplatnění sahá od vyzvednutí dětí školním autobusem, přes výrobu elektroniky, typografii, až do návrhů GNSS.[22] V této kapitole se však budeme zabývat aplikací TSP v oborech logistiky a dopravy. V logistice lze využít TSP například pro snadnou dopravní obsluhu území, řešení distribučních i rozvozních úloh nebo zásobován.

3.1 **Okružní jízdy**

Předpokládejme, že *n* zákazníků vyžaduje určité množství zboží a dodavatel musí uspokojit všechny požadavky pomocí určitého počtu vozidel. Vozidla se musí přiřadit k zákazníkům a musí se vytvořit časový plán dodávek pro každé vozidlo tak, aby nebyla překročena kapacita vozidel a celková přepravní vzdálenost byla minimalizována.

Tento problém je řešitelný jako TSP, pokud neexistují žádná časová a kapacitní omezení a pokud je počet nákladních vozidel fixní. V tomto případě se úloha nazývá TSP s více obchodními cestujícími.[8]

3.2 **Problém při výběru objednávek ve skladech**

Tento problém je spojen s manipulací s materiálem ve skladu. Předpokládejme, že pracovníci ve skladu obdrží objednávku libovolných položek. Následně tedy musí shromáždit všechny položky pro jejich expedici ze skladu k zákazníkovi.

Umístění položek ve skladu odpovídá vrcholům grafu. Vzdálenost mezi dvěma vrcholy je dána časem potřebným k přemístění vozidla z jednoho místa na druhé. Problém nalezení nejkratší trasy pracovníka, a s tím spojenou minimální dobou kompletace objednávky, lze nyní vyřešit jako TSP.[8]

Obrázek 8: Amazon sklad [12]

3.3 **Repasování motorů s plynovou turbínou**

K této aplikaci dochází při repasování proudových motoru letadel. Pro zajištění rovnoměrného proudění vzduchu turbínami jsou v každém stupni turbíny umístěné sestavy trysek pro rozvaděči lopatky. Tyto lopatky mají individuální vlastnosti, a správné umístění lopatek kolem obvodu může přinést podstatné výhody (snížení vibraci, zvýšení rovnoměrnosti proudění, snížení spotřeby paliva). Problém nejlepšího umístění lopatek způsobem lze modelovat jako TSP se speciální účelovou funkcí.[8]

3.4 **Návrh motorů pro letoun – Boeing 777**

O této aplikaci není z důvodu průmyslového tajemství publikováno mnoho detailů. Nicméně lze konstatovat, že zde byly využity genetické algoritmy pro optimalizaci parametrů proudového motoru, který byl navržen pro letoun Boeing 777 s důrazem na minimální spotřebu paliva. První návrh byl vytvořen za pomoci klasických technik, genetické algoritmy byly využity následně pro doladění některých parametrů.

Díky aplikaci genetických algoritmů se podařilo dosáhnout téměř 2,5 % úspory paliva. Tento fakt ekonomicky znamená zhruba 2 miliony dolarů úspory ročně pro jeden letoun.[20]

3.5 **Sběr mincí**

Jednou ze starších aplikací TSP je plánovaní sběru mincí z telefonních automatů v určitém regionu. Pro tento případ byla použita modifikovaná verze Lin-Kerninghanovy metody. Modifikace zde byla provedena s cílem zvládnutí zvláštností při cestovaní městem (například jednosměrné ulice, dopravní nehody...). Pokud by zde nedošlo k úpravám (tedy zůstal by předpoklad symetrické úlohy), algoritmus by nebyl schopen v praxi pracovat, protože by předpokládal náklady na cestu z *x* do *y* jsou stejné jako *y* do *x*. To je však v praxi téměř nedosažitelné, protože existuje značné množství excesů, které mohou průběh sběru ovlivnit.[21]

4 Analýza existujícího komerčního SW

Pro provedení analýzy existujícího komerčního softwaru bylo kontaktováno množství národních i nadnárodních firem. Dotaz zněl, zda využívají nějaký, a případně jaký, komerční SW pro plánování oběhů jejich vozidel. Dále byli též požádáni o sdělení faktů, které u programů považují za přínosné a případně i které považují za nedostatky.

Zde se nachází seznam firem, které byly kontaktovány s dotazem na užívaný SW:

- Česká pošta, s.p.
- Staropramen s.r.o.
- Red Bull Česká republika s.r.o.
- Direct Parcel Distribution CZ s.r.o.
- ESA s.r.o.
- DB Schenker s.r.o.
- PPL s.r.o.
- DHL Express (Czech Republic) Ltd.
- Velká pecka s.r.o. (rohlik.cz)
- Košík.cz s.r.o.
- ŠKODA AUTO a.s.
- AutoMax Group s.r.o.
- UNIPETROL RPA s.r.o. (Benzina)
- OMV Česká republika s.r.o.
- damejidlo.cz s.r.o.
- DoDo Czech s.r.o.
- Auto Kelly a.s.
- Douglas s.r.o.
- Plzeňský Prazdroj a.s.
- PEPSICO CZ s.r.o.
- TPL Czech spol. s.r.o.
- Seznam.cz, a.s.

4.1 Získané odpovědi

Z celkem 22 kontaktovaných společností dorazilo zpět 9 odpovědí. Vzhledem k tomu, že průzkum probíhal v letních měsících, může zde docházet k částečnému zkreslení z důvodů dovolených a podobně. V následující tabulce jsou shrnuty získané informace o využívaném softwaru od jednotlivých společností.

Tabulka 1: Odpovědi společností

Jméno společnosti	Odpověď
Česká pošta, s.p.	Odkázáno na jiný kontakt, dále bez odpovědi.
Staropramen s.r.o.	
Red Bull Česká republika s.r.o.	
Direct Parcel Distribution CZ s.r.o.	
ESA s.r.o.	
DB Schenker s.r.o.	
PPL s.r.o.	
DHL Express (Czech Republic) Ltd.	Vlastní systémy, nespadají do žádané kategorie.
Velká pecka s.r.o. (rohlik.cz)	Využívají systém Tasha od společnosti SolverTech.
Košík.cz s.r.o.	Přeposláno na logistické oddělení, dále bez odpovědi.
ŠKODA AUTO a.s.	Přeposláno na logistické oddělení, dále bez odpovědi.
AutoMax Group s.r.o.	Odkázáno na DB Schenker s.r.o.
UNIPETROL RPA s.r.o. (Benzina)	
OMV Česká republika s.r.o.	
damejidlo.cz s.r.o.	Přeposláno na logistické oddělení, dále bez odpovědi.
DoDo Czech s.r.o.	
Auto Kelly a.s.	
Douglas s.r.o.	
Plzeňský Prazdroj a.s.	Z časových důvodů se nemohou věnovat požadavku.
PEPSICO CZ s.r.o.	
TPL Czech spol. s.r.o.	Využívají systém Rinkai Routing.
Seznam.cz, a.s.	

Společnosti, které odpověděly, že využívají vlastní software, bohužel z bezpečnostních důvodů a autorských práv odmítly sdělovat bližší informace. Jediné, co mohly poskytnout byly obecné informace o jeho původu, případně kontakt na vývojáře. Z nich však všichni sídlí v zahraničí (Indie, Malajsie...) a nepodařilo se od nich získat zpětnou vazbu.

4.2 Systémy SAP, Helios, Rinkai a SolverTech

Software pro optimalizaci dopravy pomáhá ušetřít čas a peníze tím, že vypočítá optimalní trasy rozvozu s ohledem na různá omezení a zváštnosti, které se vyskytují v oblasti logistiky a dopravy. V této kapitole je popsán systém SAP TM od německé společností SAP, která vyvíjí, prodává a implementuje ERP systémy pro podniky různé velikosti a různých oborů. Tento nástroj má mnoho užitečných funkcí, z pohledu TSP je však nejdůležitější optimalizátor VRS, který dokáže naplánovat jízdy a jejich rozvrh. Dále je popsán ERP systém Helios, který je popularní na českém trhu a jeho verze pro velké a střední firmy Helios GREEN nabízí modul na plánování tras. Následně je popsán nástroj Rinkai Routing, na který odkázala společnost TPL Czech spol. s.r.o. Ta jej využívá k rozvozům pro PEPSICO v Plzeňském kraji. Následuje popis systému Tascha od české společnosti SolverTech s.r.o., na který odkázala společnost Velká pecka s.r.o. (rohlik.cz).

4.2.1 SAP

SAP je německá firma, která vyvíjí, prodává a implementuje ERP systémy pro podniky různé velikosti a různých oborů. Systém SAP propojuje všechny důležité obchodní funkce tak, že obsahuje moduly, které se zabývají financemi, CRM, prodejem, skladováním a podobným. Moduly je možno instalovat samostatně a případně dále propojovat, jejich přehled následuje.

- **SAP LE** (Logistics Execution) – zabývá se logistickými procesy ve společnosti.
- **SAP MM** (Material Management) - poskytuje možnosti správy materiálů, zásob a skladů.
- **SAP PP** (Production Planning) - sleduje a zaznamenává toky výrobního procesu, například plánované a skutečné náklady.
- **SAP PM/PS** (Plant Maintenance and Project System) – zahrnuje v sobě dva typy údržby, Preventivní (planování a realizace údržby) a Korektivní (náhodný stroj se rozbil a je nutné jej opravit).
- **SAP FI** (Financial Accounting) – pomáhá analyzovat finanční stav firmy na trhu.
- **SAP CO** (Controlling) - podporuje plánování, vykazování a monitorování operací firem.
- **SAP HR** (Human Resources) – zaznamenává informace o pracovnících po celou dobu jejich pracovního poměru.
- **SAP SD** (Sales and Distribution) – slouží především pro příjem a zpracování zákaznických požadavků.

Obrázek 9: Moduly SAP [25]

SAP TM

SAP TM pokrývá celou škálu druhů dopravy (letecká, oceánská, železniční a silniční) a je navržen pro poskytovatele přepravních a logistických služeb. Jedná se o samostatný systém a obvykle je nainstalován odděleně od SAP ERP, však lze jej snadno propojit. Přestože většina instalací SAP TM je prováděna se zavedeným SAP ERP, je technicky možné použít SAP TM i v prostředí mimo SAP ERP. Obecně však platí, že obsahuje všechny funkce SAP ERP LE a k tomu mnoho dalších funkcí, jako je komplexní správa zón a jízdních pruhů, výběrové řízení na nákladní dopravu, portál pro spolupráci, optimalizace dopravy a tras.[28]

Součástí systému jsou následující čtyři optimalizační nástroje:

- **VSR** (Vehicle Scheduling & Routing) Optimizer – plánuje dodávky nejlepším možným způsobem podle dostupných vozidel a tras.
- **Load Optimizer** – uspořádá palety a balíky ve vozidle s dodržením různých pravidel, například pravidel stohovatelnosti.
- **Carrier Selection** – Přiřazuje dopravce pro každou zásilku s ohledem na náklady, alokace a obchodní podíly.
- **Strategic Freight Management** – hodnotí nabídky dopravců pro dlouhodobé smlouvy na základě nákladů, kapacity a rizika.[27]

Obrázek 10: Modul SAP TM [41]

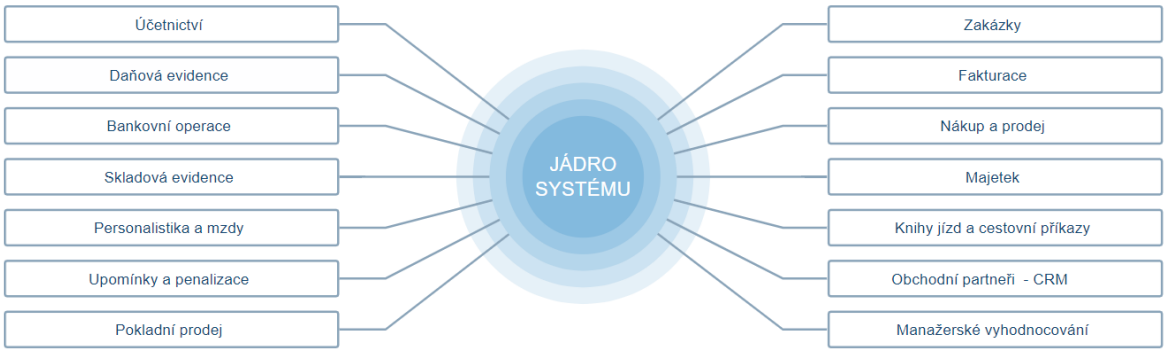
Optimalizátor VSR řeší kapacitně omezenou úlohu okružních jízd, která je zobecněným problémem obchodního cestujícího, a proto tedy následuje podrobnější popis právě tohoto nástroje.

Jeho účelem je zajistit *plánování jizd* a *plánování rozvrhu*. Optimalizátor musí směrovat zásilky tak, že najde nejlepší možnou cestu, kterou musí řidič projet mezi zákazníky, aby splnil všechna časová omezení. Dalším krokem je *plánování rovrhu*. Čas začátku a konce činností je přiřazen pro každé z vozidel a míst. Řešení tohoto problému musí být nalezeno s ohledem na všechna omezení a preference. Nakonec je pro každé vozidlo a každé místo vytvořen „seznam úkolů“ spolu se specifickými časovými razítky, která říkají, kdy začít a kdy ukončit nakládku / vykládku apod.[27]

4.2.2 Helios

Helios je ERP systém od společnosti Asseco Solutions a.s., která se zabývá tvorbou těchto systémů již od roku 1990. Systémy Helios mají několik stupňů, které se liší v závislosti na velikosti společnosti, která je využívá. Rozdíly jsou například ve složitosti zpracování, ukládání dat a každý systém se skládá z různých modulů, přehled modulu lze vidět na obrázku 11. Celkem se jedná o tři různé stupně, viz následující přehled.

- **Helios GREEN** – pro velké a střední firmy. Snadno se přizpůsobí požadavkům firmy a pokrývá všechna oborová řešení.
- **Helios ORANGE** – pro střední a menší firmy. Poskytuje aktuální přehled o situaci na trhu.
- **Helios RED** – pro podnikatele a malé firmy.



Obrázek 11: Přehled modulů systému Helios [24]

V sektoru služeb dopravy a přepravy nabízí následující funkcionality:

- Plánování jízd
- Technický stav vozidla
- Ekonomika provozu vozidel
- Silniční daň
- Pojistné události
- Dodací podmínky INCOTERMS

- Protokol o všech operacích

Helios GREEN narozdíl od Helios ORANGE a Helios RED nabízí plánovaní tras, přičemž Helios ORANGE nabízí jen plánování jízd. Helios RED poté pouze knihu jízd, sloužící k záznamu jízd dopravními prostředky. Součástí stupně Helios GREEN je také modul „Logistika a Sklady“, který umožnuje evidovat skladové zásoby, sledovat jejich pohyb, šarže, výrobní čísla, jakost a podobně.

V oblasti dopravy dále umožňuje:

- Evidence vozidel
- Plánování tras a jednotlivých jízd
- Přiřazení jednotlivých dokladů (poptávka, dodací list, FV) k jednotlivým jízdám
- Evidence vratek
- Vyhodnocení jednotlivých rozvozů

Společnosti, které využívají SW Helios, jsou například: SILO TRANS s.r.o., MATRIX a.s., TEDDIES s.r.o., SKANSKA a.s.[24]

4.2.3 Rinkai Routing

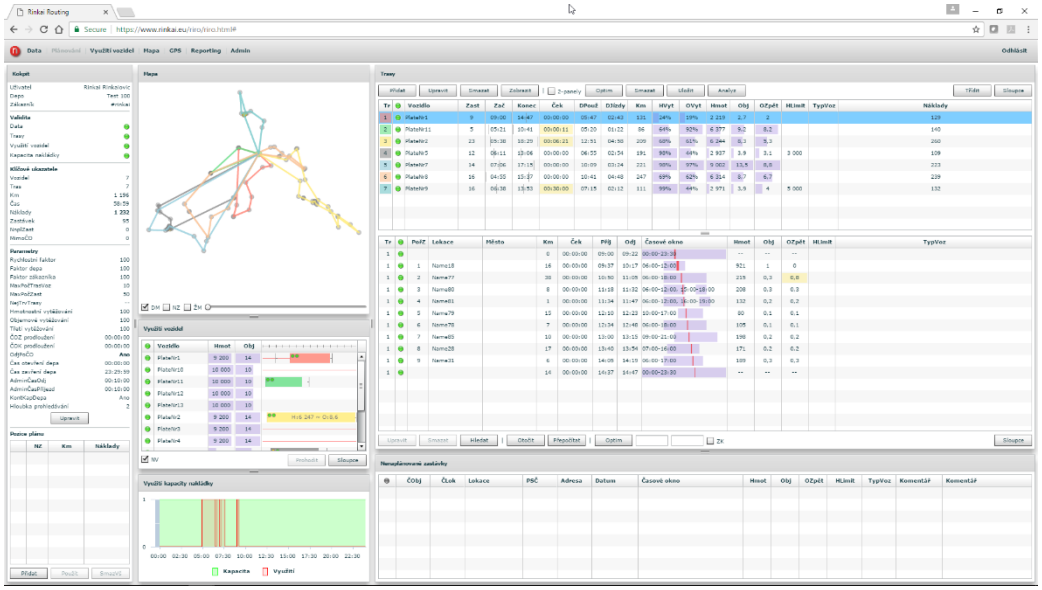
Na tento nástroj odkázala společnost TPL Czech spol. s.r.o. V praxi se nesetkávají s problémy ze strany softwaru, ale spíše ze strany klientů, kteří do systému zadávají adresy špatným způsobem, tudíž je následně nelze zapracovat do správného oběhu vozidel.

Rinkai Routing je logistický optimalizační nástroj vyvinutý společností Rinkai s.r.o. Umožnuje vytvořit efektivní plán dopravy, maximalizující vytížení vozového parku a minimalizující přepravní náklady, na základě zákaznických objednávek, respektující přepravní a zákaznická omezení. Další firmy, které využívají nástroj Rinkai Routing jsou například: Alza.cz, IKEA, Svijany a Maso Brejcha.

Nejprve je potřeba zadat zákaznické objednávky do systému, což lze udělat manuálně nebo automaticky z ERP systému. Systém dále vytvoří optimální trasy pro rozvoz. Při optimalizaci využívá digitální mapy obsahující informace o mýtných poplatcích, průjezdnostech a rychlostech úseků pro různé typy vozidel apod. Trasy lze plánovat denně nebo předem na delší časový úsek (dny, týdny, měsíce).[23]

Společnost Rinkai dále uvádí tato výhody užívání jejich systému:

- Snížení přepravních nákladů typicky o 10-20 %.
- Zrychlení přípravy plánu dopravy.
- Zlepšení zákaznického servisu.
- Snadný odhad dopadu změny struktury vozového parku[23]



Obrázek 12: Uživatelské rozhranní systému Rinkai [23]

Rinkai Routing se dle webových stránek společnosti také umí postarat o:

- časová okna zákazníků
- tonážová omezení
- omezeních na určitý typ vozidla
- časová dostupnost vozidel
- kapacita nakládky
- průjezdnost silnic pro různá vozidla[23]

4.2.4 SolverTech Tasha

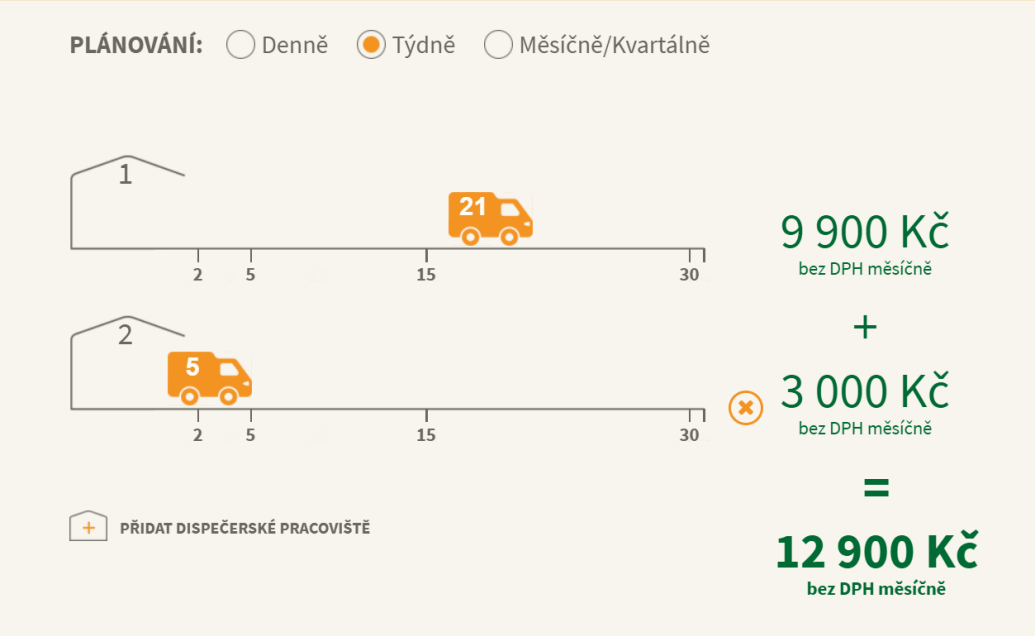
Na tento nástroj odkázala společnost Velká pecka s.r.o (provozující online supermarket rohlik.cz). Dle tvrzení společnosti denně plánují téměř 10000 objednávek a standardizovaný software jim poskytuje stabilní, predikovatelné výsledky. Dále umožňuje výrazně rychleji škálovat ve stávajících i v nových rozvozových lokalitách. Za malou nevýhodu by se dala považovat náročnost systému na údržbu, aby byl vždy aktuální.

Systém Tasha od české společnosti SolverTech s.r.o tedy obecně slouží k optimalizaci tras a slibuje snížení nákladů na dopravu o 7-15 %. Aplikace je určena pro zásilkové služby, velkoobchod, e-shopy atd.

Mezi hlavní funkce systému patří:

- Denně aktualizované mapy pro rozvoz (počítá s uzavírkami a omezeními)
- Návrh rozvozního plánu s ohledem na tonáž a rozměr vozidel, mýtné, rychlostní omezení.
- Propojení s ERP systémy (SAP, Helios a podobné)
- Přehledné podklady k fakturaci (počet újetých kilometrů, povaha nakladu, typy vozidel, kombinované tarify řidičů)
- Plánování tras s ohledem na pracovní pravidla AETR (Evropská dohoda o práci osádek vozidel v mezinárodní silniční dopravě), vlastnosti nákladů a vozů.

Další společnosti, které využívají tento nástroj jsou například drogerie Teta, síť maloobchodů Hruška a výrobce pečiva a cukrovinek Adria Gold.[29]



Obrázek 13: Kalkulačka SolverTech Tasha [29]

4.3 Shrnutí

Všechny výše uvedené systémy jsou v praxi hojně využívány pro řešení problematiky okružních jízd, což je praktiky rozšířená aplikace TSP.

Nástroje však nelze rozumně srovnávat, a proto je kapitola řešena primárně jako analýza a popis systémů. SAP a Helios jsou ERP systémy a obsahují tudíž větší množství modulů pro řešení firemní problematiky. Systémy Rinkai a Tasha řeší především problematiku okružních jízd.

Pro optimální porovnání systémů by bylo vhodné postavit systémy před stejný problém a následně porovnat získané výsledky. To však nebylo možné z důvodu finanční náročnosti, jelikož systémy nejsou dostupné ve veřejně přístupných verzích. Při zjišťování cen systémů Rinkai Routing, Helios GREEN a SAP TM se nepodařilo zjistit kalkulaci. U všech však náklady značně závisí na rozsahu aplikace systému. Pouze systém SolverTech Tasha disponuje orientační kalkulačkou na svých webových stránkách, viz obrázek 13.[29]

5 Analýza kvality řešení metod s využitím TSPLIB instances

TSPLIB je knihovna vzorových příkladů TSP. Lze ji navštívit online z webových stránek německé Ruprechto-Karlovy univerzity v Heidelbergu. Obsahuje různé druhy TSP a existující optimální řešení pro některé problémy. Mezi ty patří například symetrický TSP, problém Hamiltonova cyklu, asymetrický TSP, sekvenční problém a problém kapacitně omezených okružních jízd.[30]

Pro otestování algoritmů bylo vytvořeno několik příkladů TSP, přičemž struktura vstupních datových souborů je vždy stejná. Každý se skládá z hlavičky, kde je název problému, jeho typ, stručný popis problému, váhy a počet vrcholů. Následuje seznam vrcholů a jejich pozice ve 2D prostoru ve formátu ID, souřadnice X, souřadnice Y, viz následující obrázek.[30]

```
NAME : att48
COMMENT : 48 capitals of the US (Padberg/Rinaldi)
TYPE : TSP
DIMENSION : 48
EDGE_WEIGHT_TYPE : ATT
NODE_COORD_SECTION
1 6734 1453
2 2233 10
3 5530 1424
.
.
.
48 3023 1942
EOF
```

Obrázek 14: Formát dat TSPLIB – úloha att48 [30]

Pro porovnání byly vybrány tři algoritmy. Z exaktních metod byl vybrán *backtracking* viz kapitola 2.1.2, z heuristických *hladový algoritmus* viz kapitola 2.2.1 a z metaheuristických metod byl vybrán algoritmus *ant colony* dle kapitoly 2.3.2.

Data použitá pro vstup byla namodelována dle úlohy att48 v TSPLIB. Využito bylo pouze prvních 17 měst z důvodu testování a porovnávání exaktní metody, která je výpočetně velice náročná. Knihovna TSPLIB nedisponuje TSP úlohami s dostatečně malým počtem vrcholů a nebylo by tedy možné provést srovnání jednotivých metod.

5.1 Výpočetní technika, software a použité algoritmy

Pro testování metod byl zvolen notebook Acer Nitro 5 (AN515-52-74FP) s těmito parametry:

- Procesor: Intel i7 8750H (2,2 / 4,1 GHz 6 jader / 12 vláken)
- Operační system: Windows 10 Pro 64bit
- Operační paměť: 16GB DDR4 SODIMM 2400 MHz
- Pevný disk: 512 GB SSD M.2 PCIe/NVMe

Program využitý pro testování byl MS Visual Studio v distribuci Community verze 2.2.3085.814.

```
void backtracking(std::vector<node>& nodes, std::vector<std::vector<float>>& weight_graph, int position, int count, float current_cost, float& ans)
{
    if (count == static_cast<int>(nodes.size()) && weight_graph[position][0] > 0)
    {
        if(current_cost + weight_graph[position][0] < ans)
        {
            ans = current_cost + weight_graph[position][0];
        }
        return;
    }

    for (auto i = 0; i < static_cast<int>(nodes.size()); i++) {
        if (!nodes[i].visited && weight_graph[position][i] > 0)
        {
            nodes[i].visited = true;
            backtracking(nodes, weight_graph, i, count + 1, current_cost + weight_graph[position][i], ans);
            nodes[i].visited = false;
        }
    }
}
```

Obrázek 15: Ukázka kódu backtracking [31]

Pro výpočetní část kódu algoritmu pro backtracking byl využit existující projekt na webových stránkách GeeksforGeeks zaměřených na počítačovou problematiku (úryvek kódu viz obrázek 15). Část načtení dat ve formátu TSPLIB a výstup byl upraven / vytvořen pro potřeby práce.[31]

Algoritmus pro metodu nejbližšího souseda byl vytvořen dle podkladů dostupných na webových stránkách ResearchGate (úryvek kódu viz obrázek 16). Vstupní data jsou zpracována obdobně jako tomu je pro algoritmus backtrackingu.

```
float greedy(std::vector<node>& nodes, std::vector<std::vector<float>>& weight_graph, int position, int count, float length)
{
        float current_lowest = INT32_MAX;
        int next_position;
        node* next_node = nullptr;

        for (auto i = 0; i < static_cast<int>(nodes.size()); i++)
        {
                if (weight_graph[position][i] < current_lowest && !nodes[i].visited)
                {
                        next_node = &nodes[i];
                        next_position = next_node->id;
                        current_lowest = weight_graph[position][i];
                }
        }

        if (count != static_cast<int>(nodes.size()) && next_node != nullptr)
        {
                next_node->visited = true;
                return greedy(nodes, weight_graph, next_position, count + 1, length + current_lowest);
        }
        return length + weight_graph[0][position];
}
```

Obrázek 16: Ukázka kódu hladového algoritmu [32]

Metoda ant colony, patří do skupiny metaheuristických algoritmů a je nejnáročnější na programování a optimalizaci. Z tohoto důvodu byl využit funkční, volně dostupný základ na webových stránkách GitHub, který byl následně upraven pro potřeby práce tak, aby podával požadované výsledky a omezila se chybovost. Populace mravenců byla nastavena na 50, maximální počet cest 20 a konstanta „odpařování feromonu“ na 0,5.[33] Úryvek kódu můžeme vidět na obrázku 17.

```
// runtime Structures and global variables
ACO::ACO()
{
    alpha = 1.0;
    beta = 5.0;
    qval = 100;
    maxTours = 20;
    antsPopulation = 50;
    evaporationRate = 0.5;
    best = INT_MAX;
}

// function init() - initializes the entire graph
void ACO::init()
{
    pheromones = new double*[gm.getNumberOfVertexes()];
    for (int i = 0; i < gm.getNumberOfVertexes(); i++)
    {
        pheromones[i] = new double[gm.getNumberOfVertexes()];
        for (int j = 0; j < gm.getNumberOfVertexes(); j++)
        {
            pheromones[i][j] = 1.0 / gm.getNumberOfVertexes();
        }
    }
```

Obrázek 17: Ukázka kódu mravenčí kolonie [33]

5.2 **Srovnání výsledků metod backtracking, nejbližší soused a ACO**

Následující tabulky shrnují získané informace z testování vybraných metod na redukované TSP úloze dle TSPLIB att48.tsp.[30] Při srovnávání všech metod najednou bylo záměrně zvoleno pouze prvních 17 měst, jak již bylo zmíněno, kvůli zahrnuté exaktní metodě.

Tabulka 2: Srovnání metod pro 2 až 5 uzlů

	Počet uzlů [-]	Délka cesty [j]	Doba výpočtu [s]
Backtracking	2	9453.31	0.00
Nejbližší soused	2	9453.31	0.00
Ant colony	2	9452.00	0.04
Backtracking	3	9518.43	0.00
Nejbližší soused	3	9518.43	0.00
Ant colony	3	9517.00	0.06
Backtracking	4	13104.69	0.00
Nejbližší soused	4	13165.94	0.00
Ant colony	4	13103.00	0.08
Backtracking	5	13199.20	0.00
Nejbližší soused	5	13877.78	0.00
Ant colony	5	13196.00	0.10

Tabulka 3: Srovnání metod pro 6 až 9 uzlů

	Počet uzlů [-]	Délka cesty [j]	Doba výpočtu [s]
Backtracking	6	18061.10	0.00
Nejbližší soused	6	18708.52	0.00
Ant colony	6	18058.00	0.12
Backtracking	7	18087.93	0.00
Nejbližší soused	7	19114.41	0.00
Ant colony	7	18084.00	0.14
Backtracking	8	18704.01	0.00
Nejbližší soused	8	20214.36	0.00
Ant colony	8	18700.00	0.16
Backtracking	9	18906.06	0.00
Nejbližší soused	9	20832.04	0.00
Ant colony	9	18901.00	0.18

Tabulka 4: Srovnání metod pro 10 až 13 uzlů

	Počet uzlů [-]	Délka cesty [j]	Doba výpočtu [s]
Backtracking	10	19520.30	0.00
Nejbližší soused	10	21179.55	0.00
Ant colony	10	19515.00	0.20
Backtracking	11	19593.57	4.12
Nejbližší soused	11	23465.91	0.00
Ant colony	11	19588.00	0.22
Backtracking	12	19614.59	46.43
Nejbližší soused	12	23465.96	0.00
Ant colony	12	19890.00	0.24
Backtracking	13	19730.07	582.00
Nejbližší soused	13	23712.17	0.00
Ant colony	13	20006.00	0.26

Tabulka 5: Srovnání metod pro 14 až 17 uzlů

	Počet uzlů [-]	Délka cesty [j]	Doba výpočtu [s]
Backtracking	14	neznámá	proces ukončen po 7 326 s bez výsledku
Nejbližší soused	14	24795.98	0.00
Ant colony	14	20440.00	0.28
Backtracking	15	neznámá	proces ukončen po 72 858 s bez výsledku
Nejbližší soused	15	24628.62	0.00
Ant colony	15	21105	0.3
Backtracking	16	netestováno	netestováno
Nejbližší soused	16	25522.09	0.00
Ant colony	16	21627.00	0.32
Backtracking	17	netestováno	netestováno
Nejbližší soused	17	26666.97	0.00
Ant colony	17	23127.00	0.34

Z výše uvedených výsledků se dá říct, že pokud se udržuje velice nízký počet uzlů, jsou si algoritmy téměř rovny v případě času potřebného pro výpočet i výsledné délky cesty. S narůstajícím počtem vrcholů se však projevují jejich nedostatky a výsledky se začínají značně odlišovat. Výpočet s časem uvedeným jako „0,00“ byl spočítán okamžitě a neprojevil se jakýmkoli zdržením.

Exaktní metoda *backtracking* se ukázala být schopná spočítat úlohu v rozumném čase pro nejvýše 12 uzlů. Výsledky v řádu stovek sekund se již nedají považovat za kvalitní vzhledem k nízkému počtu uzlů. Výpočet pro 14 a více uzlů se ukázal být naprosto neproveditelný v rozumném čase, neboť pro 14 uzlů výpočet trval 7326 s a poté byl ukončen z důvodu abnormální časové náročnosti. Pro 15 uzlů dopadl algoritmus ještě hůře, výpočet trval 72858 s a následně byl ukončen, jelikož se ani v takto obrovském časovém horizontu nedochýlil ke konci.

Heuristická metoda *nejbližšího souseda* se ukázala jako nesmírně rychlá pro počítačové zpracování výpočtu, nicméně s narůstajícím počtem uzlů se začala negativně projevovat její „hrubost“. Při velice nízkém počtu uzlů dosahovala obstojných výsledků, avšak dále se oproti zbývajícím dvěma značně odchylovala v délce cesty. Ta byla při 17 vrcholech téměř o 16 % delší oproti metodě ant colony a při nízkých počtech uzlů dosahovala o jednotky procent horších výsledků i oproti metodě backtrackingu.

Metaheuristická metoda *ant colony* dosahovala výborných výsledků po celou dobu testu. Jak již bylo zmíněno, populace mravenců byla nastavena na 50, maximální počet cest 20 a konstanta „odpařování feromonu“ na 0,5. Oproti ostatním dosahovala nejlepších výsledků až do 11 uzlů v síti. Poté se sice nepatrně zhoršila oproti backtrackingu (v řádu jednotek procent), ale čas potřebný pro její výpočet byl v porovnání zanedbatelný. Oproti metodě nejbližšího souseda dosahovala s narůstajícím počtem uzlů znatelně lepších výsledků při zanedbatelném časovém rozdílu.

5.3 **Srovnání metody nejbližšího souseda a ACO**

V tomto případě byla vyřazena exaktní metoda backtracking kvůli abnormální časové náročnosti. Testy, které proběhly, byly na úlohách pr229, ch130, gil262, dsj1000, pcb 1173 a ul1432. Zde se též porovnával dosažený výsledek s nejlepším dostupným řešením dle TSPLIB.[30]

Tabulka 6: Srovnání metody nejbližšího souseda a ACO

pr299	Ideální délka cesty [j]	Vypočtená délka cesty [j]	Doba výpočtu [s]
Nejbližší soused	48191	59899.02	0.00
Ant colony	48191	62593.00	5.68

kroA100	Ideální délka cesty [j]	Vypočtená délka cesty [j]	Doba výpočtu [s]
Nejbližší soused	21282	26856.39	0.00
Ant colony	21282	25682.00	2.00

gil262	Ideální délka cesty [j]	Vypočtená délka cesty [j]	Doba výpočtu [s]
Nejbližší soused	2378	3241.47	0.00
Ant colony	2378	3141.00	5.24

dsj1000	Ideální délka cesty [j]	Vypočtená délka cesty [j]	Doba výpočtu [s]
Nejbližší soused	18659688	24630950.00	0.00
Ant colony	18659688	29897000.00	200.00

pcb1173	Ideální délka cesty [j]	Vypočtená délka cesty [j]	Doba výpočtu [s]
Nejbližší soused	56892	70277.85	0.00
Ant colony	56892	87544.00	234.60

ul1432	Ideální délka cesty [j]	Vypočtená délka cesty [j]	Doba výpočtu [s]
Nejbližší soused	152970	188815.16	0.00
Ant colony	152970	251600.00	286.30

Nastavení metody ACO zde zůstalo stejné jako u předchozího testu (viz 5.1 a 5.3). Její výsledky však v tomto bodě nejsou zdaleka tak dobré, jako v předchozím. Pouze u úlohy gil262 a kroA100 dosáhla nepatrně lepších výsledků. U ostatních úloh byly výsledky značně horší a čas potřebný k výpočtu byl znatelně delší oproti metodě nejbližšího souseda. To však může být způsobeno především prvotním nastavením populace mravenců, maximálním počtem cest a konstantou „odpařování feromonu“. Pokud by se našla vhodnější kombinace, metoda by pravděpodobně dosáhla lepších výsledků – ta se však může úlohu od úlohy lišit a bylo by nutné vytvářet nastavení „na míru“ dané úloze. Metoda nejbližšího souseda zde překvapivě dosáhla lepších výsledků oproti ACO i přes svou „hrubou sílu“. Její nevýhodou však je, že ji nelze nijak ladit a její výsledek je konečný a není možno jej zlepšit, právě narozdíl od ACO.

6 Závěr

Předmětem této bakalářské práce je problém obchodního cestujícho a jeho aplikace v dopravních a logistických systémech. Řešení těchto úloh je značně náročné na čas, protože úloha patří k takzvaným NP-úplným úlohám. Cílem této bakalářské práce bylo popsat problém obchodního cestujícího, jeho formulace, klasifikace, historii a metody řešení problému. Také v prací byly popsany příklady aplikace TSP v dopravě a logistice. Praktická část se následně týká analýzy existujícího komerčního software na trhu a analýzy kvality řešení metod s využitím TSPLIB instances.

Práci lze rozdělit na dvě základní části. Teoretickou, do které spadají kapitoly 1 až 3 a praktickou, která zahrnuje kapitoly 4 a 5.

První kapitola je věnována popisu a historii TSP, formulacím problému, v práce jsou popsaný dvě nejznámější formulace – Dantzig-Fulkerson-Johnsona a Miller-Tucker-Zemlina, a k tomu ještě formulace TSP jako úlohy teorii grafu. TSP existuje v mnoha různých variacích, v první kapitole jsou zkratce popsány čtyři nejznámější ze všech.

V druhé kapitole jsou popsány nejznámější exaktní, heuristické a metaheuristické algoritmy, které lze využít pro řešení problému obchodního cestujícího. Tyto metody se značně liší výpočetní náročností, rychlostí a kvalitou výsledného řešení.

Ve třetí kapitole jsou rozebrány aplikace TSP v logistice a dopravě a čtvrtá kapitola se poté zabývá analýzou existujícího komerčního softwaru. Obsahuje průzkum a popis užívaného softwaru u společností v České republice – systémy SAP, Helios, Rinkai a SolverTech.

Pátá kapitola se zabývá analýzou kvality řešení s využitím knihovny problému obchodního cestujícího TSPLIB. V kapitole se nejprve porovnává exaktní (backtracking), heuristický (metoda nejbližšího souseda) a metaheuristický (ACO) algoritmus na redukované úloze TSPLIB instances. Následně pouze heuristický a metaheuristický na rozsáhlejších úlohách.

Po vyhodnocení provedených testů se potvrdilo, že exaktní algoritmus funguje přijatelně pouze na úlohách s menším počtem vrcholů. Přičemž se dále potvrdilo i to, že heuristiky a metaheuristiky skutečně dokážou úspěšně fungovat i na rozsáhlejších úlohách.

Doufám, že má práce bude pro čtenáře přínosem a do budoucna poslouží například i jako podklad k dalšímu zkoumání rozsáhlé problematiky úlohy obchodního cestujícího.

Zdroje

[1] VOLEK, Josef a Bohumil LINDA, VÍZNER, Filip, ed. *Teorie grafů: Aplikace v dopravě a veřejné správě*. Univerzita Pardubice: Univerzita Pardubice, Studentská 95, 532 10 Pardubice, 2012. ISBN 978-80-7395-225-9

[2] ŠIŠMA, Pavel, Teorie grafů 1736-1963. Praha: Prometheus, 1997. ISBN 80-7196-065-9

[3] SEDLÁČEK, Jiří, HANÁKOVÁ, Alena a Petr ČECH, ed. *Úvod do teorie grafů*. 3. vydání. Praha: Academia, 1981

[4] LAPORTE, G. A Concise Guide to the Traveling Salesman Problem. *The Journal of the Operational Research Society*[online]. 2016, **61**(1),[cit. 2019-11-11]. Dostupné z: https://www.jstor.org/stable/40540226?seq=1#metadata_info_tab_contents

[5] GUTIN, Gregory a Abraham P. PUNNEN, GUTIN, G., ed. *The Travelling Salesman Problem and Its Variations*. New York: Springer, 2002. ISBN 9780306482137

[6] YANG, Xin-She. *Nature-Inspired Optimization Algorithms*. London: Elsevier, 2014. ISBN 978-0-12-416743-8

[7] PATAKI, Gregor. The bad and the good-and-ugly: formulations for the travelling salesman problém. New York: Columbia University, 2000. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.34.7256&rep=rep1&type=pdf.

[8] JÜNGER, Michael, REINELT, Gerhard a RINALDI, Giovanni. The traveling salesman problem. 1994. Dostupné z: http://www.iasi.cnr.it/reports/R375/R375.pdf.

[9] HYNEK, Josef, SOMOGYI, Petr, ed. *Genetické algoritmy a genetické programování*. Praha: Grada Publishing, 2008. ISBN 978-80-247-2695-3

[10] DYNTAR, Jakub, KADEŘÁBKOVÁ, Božena, ed. *Návrh a optimalizace dodavatelských systémů s využitím dynamické simulace*. Praha: FinEco, 2018. ISBN 978-80-86590-15-8.

[11] *Travelling salesman problem*[online]. Wikipedia, 2019[cit. 2019-08-20]. Dostupné z: https://en.wikipedia.org/wiki/Travelling_salesman_problem

[12] DOBROVSKÝ, Pavel. *Podvodní sklady: Rybám vstup zakázán: Amazon*[online]. 12.října 2017[cit. 2019-08-20]. Dostupné z: https://www.abicko.cz/clanek/precti-si-technika/22162/podvodni-sklady-rybam-vstup-zakazan.html

[13] *Metrika* [online]. Wikipedia, 2019[cit. 2019-08-20]. Dostupné z: https://en.wikipedia.org/wiki/Travelling_salesman_problem#Metric

[14] *Backtracking | Introduction*[online]. Noida, India: GeeksforGeeks[cit. 2019-08-20]. Dostupné z: https://www.geeksforgeeks.org/backtracking-introduction/

[15] DYNTAR, Jakub. Návrh a optimalizace dodavatelských systémů s využitím dynamické simulace. Praha: FinEco, 2018, 203 s. ISBN 978-80-86590-15-8

[16] A hybrid algorithm of Ant Colony Optimization (ACO) and Iterated Local Search (ILS) for estimating electricity domestic consumption: Case of Turkey. *Science Direct*[online]. Nizozemsko: Elsevier, 2016, June 2016[cit. 2019-08-19]. Dostupné z: https://www.sciencedirect.com/science/article/abs/pii/S0142061515005840

[17] Sedm mostů města Královce. *Encyclopædia Britannica*[online]. Chicago: Encyclopædia Britannica, 2010[cit. 2019-08-16]. Dostupné z: https://www.britannica.com/science/Konigsberg-bridge-problem

[19] Schéma reprezentující břehy a mosty. *Encyclopædia Britannica*[online]. Chicago: Encyclopædia Britannica, 1994[cit. 2019-08-16]. Dostupné z: https://www.britannica.com/science/Konigsberg-bridge-problem

[20] V. Mařík, O. Štěpánková, J. Lažanský a kolektiv: Umělá inteligence (3), Academia, Praha, 2001. ISBN 80-200-0472-6

[21] *TSP Applications*[online]. Waterloo: University of Waterloo, 2007[cit. 2019-08-19]. Dostupné z: http://www.math.uwaterloo.ca/tsp/apps/index.html

[22] MATAI, Rajesh, Surya SINGH a Murari LAL MITTAL, DAVENDRA, Donald, ed. Travelling Salesman Problem: an Overview of Applications, Formulations, and Solution Approaches. *Travelling Salesman Problem*[online]. Rijeka: IntechOpen[cit. 2019-08-19]. ISBN DOI: 10.5772/12909. Dostupné z: https://www.intechopen.com/books/traveling-salesman-problem-theory-and-applications/traveling-salesman-problem-an-overview-of-applications-formulations-and-solution-approaches

[23] *Rinkai: Optimalizace dopravy*[online]. Digitální agentura Beneficio, 2017Rinkais.r.o.[cit. 2019-08-21]. Dostupné z: https://www.rinkai.cz/

[24] *HELIOS: Podnikový informační systém, ekonomický a účetní software, systém pro veřejnou správu*[online]. Sherwood, 2019[cit. 2019-08-21]. Dostupné z: https://www.helios.eu/

[25] *SAP Developer Workbench: The best resource for SAP and ABAP Knowhow*[online]. 2017[cit. 2019-08-23]. Dostupné z: https://www.dev-workbench.com/es/blog/what-is-sap/

[26] *Podnikové systémy SAP pro vaše podnikání*[online]. 2019[cit. 2019-08-23]. Dostupné z: https://www.sap.com

[27] *Secrets of the SAP Transportation Management Optimizer*[online]. 2018[cit. 2019-08-23]. Dostupné z: https://novigo.com/blog/secrets-of-the-sap-transportation-management-optimizer

[28] *XpertMinds: SAP Transportation Management – Frequently Asked Questions (FAQ)*[online]. 2016[cit. 2019-08-23]. Dostupné z: https://www.xpertminds.net/blog/2014/12/7/sap-transportation-management-frequently-asked-questions-faq

[29] *SolverTech: Efektivní plánování dopravy*[online]. 2019[cit. 2019-08-23]. Dostupné z:https://solvertech.cz/

[30] *TSPLIB*[online]. 1997[cit. 2019-08-23]. Dostupné z: http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html

[31] *GeeksforGeeks – A computer science portal for Geeks: Travelling Salesman Problem implementation using BackTracking*[online]. 2019[cit. 2019-08-25]. Dostupné z: https://www.geeksforgeeks.org/travelling-salesman-problem-implementation-using-backtracking/

[32] *ResearchGate: Implementation of Greedy Algorithm in Travel Salesman Problem*[online]. 2016[cit.2019-08-25]. Dostupné z: https://www.researchgate.net/publication/307856959_Implementation_of_Greedy_Algorithm_in_Travel_Salesman_Problem

[33] *GitHub: Project for university. Implementation of TSP algorithms (ant colony optimization, genetic algorithm)*.[online]. 2019[cit. 2019-08-23]. Dostupné z: https://github.com/jastka4/PEA_3

[34] *Algoritmy.net: Graf*[online]. 2019[cit. 2019-08-23]. Dostupné z: https://www.algoritmy.net/article/1369/Graf

[35] *Geeksforgeeks: Boruvka's algorithm*[online]. [cit. 2019-11-11]. Dostupné z: https://www.geeksforgeeks.org/boruvkas-algorithm-greedy-algo-9/

[36] *Borůvkův algoritmus*[online]. Wikipedia, 2019[cit. 2019-11-11]. Dostupné z https://cs.wikipedia.org/wiki/Bor%C5%AFvk%C5%AFv_algoritmus

[37] *Marie Demlová: Diskrétní matematika a logika pro KM. Přednašky[*online].[cit. 2019-11-11]. Dostupné z: http://math.feld.cvut.cz/demlova/teaching/dmc/pred100.pdf\

[38] SLAWEK, Tadeusz. A note on the Miller-Tucker-Zemlin model for the asymmetric traveling salesman problem. *BULLETIN OF THE POLISH ACADEMY OF SCIENCES TECHNICAL SCIENCES*[online]. 2016, January 2016, (3)[cit. 2019-11-11]. Dostupné z: https://www.researchgate.net/publication/308707033_A_note_on_the_Miller-Tucker-Zemlin_model_for_the_asymmetric_traveling_salesman_problem

[39] *Euclidian distance*[online]. Wikipedia, 2019[cit. 2019-11-11]. Dostupné z: https://en.wikipedia.org/wiki/Euclidean_distance

[40] *Christofides algorithm*[online]. Wikipedia, 2019[cit. 2019-11-11]. Dostupné z: https://en.wikipedia.org/wiki/Christofides_algorithm

[41] *What is SAP TM module* [online]. 2019[cit. 2019-11-11]. Dostupné z: https://blogs.sap.com/2019/02/18/what-is-sap-tm-module/

SEZNAM OBRÁZKŮ

SEZNAM TABULEK