BAKALÁŘSKÁ PRÁCE

Využití matematických metod pro optimalizaci přepravních tras

The Use of Mathematical Methods to Optimize Transport Routes

STUDIJNÍ PROGRAM

Ekonomika a management

STUDIJNÍ OBOR

Řízení a ekonomika průmyslového podniku

VEDOUCÍ PRÁCE

RNDr. Pavel Ludvík, Ph.D.

HOJDAROVÁ

ADÉLA

2019

HOJDAROVÁ, Adéla. *Využití matematických metod pro optimalizaci přepravních tras*. Praha: ČVUT 2019. Bakalářská práce. České vysoké učení technické v Praze, Masarykův ústav vyšších studií.

MASARYKŮV ÚSTAV
VYŠŠÍCH STUDIÍ
ČVUT V PRAZE

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracovala samostatně. Dále prohlašuji, že jsem všechny použité zdroje správně a úplně citovala a uvádím je v přiloženém seznamu použité literatury.
Nemám závažný důvod proti zpřístupňování této závěrečné práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) v platném znění.

V Praze dne: 16. 05. 2019 Podpis:

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: Hojdarová Jméno: Adéla Osobní číslo: 461762

Fakulta/ústav: Masarykův ústav vyšších studií (MÚVS)

Zadávající katedra/ústav: Oddělení ekonomických studií

Studijní program: Ekonomika a management

Studijní obor: Řízení a ekonomika průmyslového podniku

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Využití matematických metod pro optimalizaci přepravních tras

Název bakalářské práce anglicky:

The Use of Mathematical Methods to Optimize Transport Routes

Pokyny pro vypracování:

Cílem práce je využití vybraných matematických metod pro výpočet optimálních přepravních tras společnosti Schreiner Transporte, s.r.o. a následná komparace a zhodnocení získaných dat s daty původními.
Přínosem bude návrh na eventuální snížení provozních nákladů dané společnosti pomocí zoptimalizovaných přepravních tras.
OSNOVA: 1. Úvod; 2. Teoretická část - LP, VRP, teorie grafů, metody použitelné k řešení problému např. VAM, Švastova metoda, genetický algoritmus, Clark - Wright; 3. Software; 4. Praktická část - představení dané společnosti, představení daných tras, implementace metod, zhodnocení výsledků; 5. Závěr

Seznam doporučené literatury:

1. B. Golden, S. Raghvan, and E. Wasil. The Vehicle Routing Problem: Latest Advances And New Challenges, 2008
2. CRAW I.: Linear Optimization and Numerical Analysis; http://www.maths.abdn.ac.uk/igc/tch/index/mx3503/notes.pdf
3. GUDEHUS, Timm; KOTZAB, Herbert. Comprehensive Logistics, 1. vyd. Hamburg: Springer - Verlag Berlin Heidelberg, 2009
4. VOLEK, J., LINDA, B. Teorie grafů - aplikace v dopravě a veřejné správě. Pardubice: Univerzita Pardubice, 2012

Jméno a pracoviště vedoucí(ho) bakalářské práce:

RNDr. Pavel Ludvík, Ph.D., MÚVS ČVUT v Praze, oddělení ekonomických studií

Jméno a pracoviště konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: 5. 12. 2018 Termín odevzdání bakalářské práce: 5. 5. 2019

Platnost zadání bakalářské práce: 30. 9. 2020

Podpis vedoucí(ho) práce | Podpis vedoucí(ho) ústavu/katedry | Podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

10-03-2019

Datum převzetí zadání | Podpis studenta(ky)

Poděkování

Děkuji vedoucímu své bakalářské práce panu RNDr. Pavlu Ludvíkovi, Ph.D. za poskytování podnětných návrhů v průběhu vypracovávání této práce. Dále bych chtěla poděkovat panu Mgr. Jiřímu Škvárovi za cenné rady. Oběma pak děkuji za ochotu a trpělivost.

Abstrakt

Jak již z názvu práce vyplývá, je zaměřena na problematiku týkající se optimalizace logistických služeb pomocí vybraných matematických metod. Cílem práce je vysvětlení Problému obchodního cestujícího a vytvoření programů, které budou počítat metodu Mravenčí kolonie a Simulovaného žíhání. Obsah tohoto tématu je značně rozsáhlý, proto se zaměřuje do hloubky jen na některé bloky, a to především na popis Problému obchodního cestujícího, a to z několika hledisek. Nejprve z hlediska historického kontextu pro lepší pochopení souvislostí a metod, dále se zaměřuje na různé modifikace tohoto problému. Jeho grafické znázornění a řešení pomocí Teorie grafů. Následně je tento problém popsán z hlediska výpočetních metod. Pro tuto část práce byl vybrán zástupce z řady exaktních metod, zástupce z řad metod heuristických a metaheuristických. Zásadní částí práce je výpočet optimálních přepravních tras pomocí vytvořených programů, porovnání jednotlivých výsledků mezi sebou a následná komparace získaných dat s daty původními.

Klíčová slova

Problém obchodního cestujícího, optimalizace, teorie grafů, heuristiky, exaktní metody, mravenčí kolonie, simulované žíhání

Abstract

As the title of the thesis suggests, it is focused on problems related to optimization of logistic services using selected mathematical methods. The aim of the thesis is to explain the Traveling salesman problem and to create programs that will calculate the Ant colony method and Simulated Annealing method. The content of this topic is very extensive, so the thesis focuses in depth only on some blocks, especially on the description of the Traveling Salesman Problem from several points of view. Firstly, from a historical background, for a better understanding of the context and methods,

it also focuses on various modifications to this problem. Its graphical representation and solution by means of Graph Theory. Subsequently, this problem is described in terms of computational methods. For this part of the work, a representative from a number of exact methods and representatives from a number of heuristic, metaheuristic methods was chosen. The basic part of the thesis is the calculation of optimal transport routes by means of created programs, comparison of individual results with each other and the subsequent comparison of obtained data with the original data.

Key words

Traveling Salesman Problem, Graph Theory, Ant Colony, Simulated Annealing, Optimization, Heuristics, Exact Methods

Obsah

Úvod

Problém obchodního cestujícího je dnes všeobecně známý, i když o jeho počátcích mnoho nevíme. Například nelze přesně říct, jak vznikl jeho název. Historické pozadí, kterému se budu věnovat v první kapitole, nám také pomůže seznámit se s daným problémem v obrysech předtím, než přejdeme k hlubší analýze. Ve druhé kapitole je rozepsán matematický pohled na tento problém. Kapitola pokračuje stručným popisem modifikací zadání úlohy TSP. Dále jsou stručně popsány různé metody skupin řešení; exaktní metody, heuristiky a metaheuristiky.

TEORETICKÁ ČÁST

1 FORMULACE TSP

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 tu ale háček, protože už při 85 městech (což je pro představu jen o pár víc, než počet okresních měst v ČR) je těchto potenciálních cest k prozkoumání víc, než kolik je ve viditelném vesmíru atomů.[1] Čímž se z původního zadání, které se nám jevilo celkem snadné stává problém, který už nemá tak jednoduché řešení. Vyvstává zde otázka, proč se vůbec ale tímto problémem zabývat, vždyť bychom si při nejhorším trochu zajeli. Hlavní důvody jsou zde dva. Hledání nejkratší spojnice mezi mnoha body se využívá v celé řadě oborů, od výroby mikročipů po plánování pohybu Hubbleova teleskopu, a používáním pokročilých metod hledání se ročně ušetří stovky miliard korun po celém světě.

1.1 Historické pozadí

Problému si všiml už v roce 1930 australský matematik a ekonom Karl Menger, který jako první rozšířil povědomí o složitosti TSP mezi matematickou veřejnost v té době se však spoléhalo pouze na řešení hrubou silou, to se Mengerovi moc nelíbilo. Problém později oficiálně formuloval Hassler Whitney na semináři na Princetonské univerzitě v roce 1934.[2] O celých 28 let později vyhlásila velká americká společnost Procter & Gamble reklamní kampaň, součástí kampaně byla soutěž o 10 000 dolarů. Pravidla zněla: Představte si, že Toody a Muldoon chtějí cestovat po Spojených státech, navštívit každé z 33 míst na soutěžní mapě a přitom ujet co nejkratší vzdálenost. Vaším úkolem je naplánovat trasu, která bude postupovat od města k městu a bude co do vzdálenosti nejkratší okružní cestou z Chicaga zase zpět do Chicaga. Problém v té době zůstal nevyřešen. Jen pro představu, řešení problému se 33 městy hrubou silou by trvalo přibližně 28 bilionů let. V roce 1954 tři matematici

[1] COOK, William. *Po stopách obchodního cestujícího: matematika na hranicích možností*. S. 12.
[2] GRECO F., Traveling salesman problem. S. 13

z prestižního výzkumného centra RAND Corporation v Santa Monice George Dantzig, Ray Fulkerson a Selmer Johnson řešili problém o instanci 48 měst. Problém vyřešili pouze za užití papíru a tužky díky vynalézavé aplikaci lineárního programování. Tým z RAND Corp sice vyřešil problém cesty přes 48 států, ale nevyřešil samotnou otázku TSP. Řešením otázky TSP rozumíme efektivní algoritmus, tj. posloupnost kroků, jejichž provedení vždy vede k nalezení optimální cesty a to bez ohledu na to, jak složité je zadání úlohy.[3] Za efektivní algoritmus by se dal považovat právě takový, který provede svou práci v čase úměrném k n^k, kde k je nějaké přirozené číslo a n je počet měst. Pro demonstraci dobrého a špatného algoritmu uvádím tabulku.

	n = 10	*n=25*	*n=50*	*n=100*
n^3	0,000 001 s	0,000 02 s	0,000 1 s	0,001 s
2^n	0,000 001 s	0,03 s	13 dní	40 bil let

Tabulka 1 Doba běhu počítače s rychlostí 10^9 operací za sekundu. Zdroj: [2]

1.2 Současný rekord

V současné době je vyřešen problém 85 900 bran na počítačovém čipu. Řešení bylo umožněno až po zpřístupnění implementace Letchfordova separačního algoritmu do Concordu. Výpočet byl proveden se skupinou 96 počítačů s duálními procesory. Celková výpočetní doba byla 84,8 let. Poslední útok na problém 85 900 bran na počítačovém čipu se započal v roce 2005 a teprve v roce 2009 byl oficiálně publikován program i datové zprávy, které ověřili správnost výpočtu. Datové záznamy obsahují popis použitých řezných rovin a řešení duálních LP úloh pro každý dílčí problém při prohledávání metodou větví a řezů. Samotný počítačový program má 6 646 řádků v jazyce C a určuje, jak

[3] COOK, William. *Po stopách obchodního cestujícího: matematika na hranicích možností*. S. 15.

procházet dílčí problémy a počítat duální řešení, na jejichž základě je možné odříznout příslušné větve prohledávacího stromu.[4]

1.2.1 TSP - SVĚT

Jedním z aktuálně nevyřešených problémů je problém s názvem „World" a jak již název napovídá, jde o to najít optimální cestu mezi všemi místy na světě, která jsou registrována jako osídlená, bylo zde zařazeno i několik výzkumných základen na Antarktidě.[5] Každé město je specifikováno zeměpisnou šířkou a délkou. Počet dohromady čítá 1 904 711 míst. Aktuální rekord je 7 512 218 268 km, tento výpočet byl ověřen 5. června 2007. Výsledek ještě není optimální, na optimální se stále čeká.[6]

1.2.2 TSP – HVĚZDY

Druhým z aktuálně nevyřešených problémů je problém označený „Hvězda". Skládá se z 526 280 881 nebeských objektů v katalogu A2.0 Námořní observatoře Spojených států amerických.[7] S úlohou se manipuluje vcelku jen velice obtížně, dokonce i polynomiální čas n^2 je zde ohromné číslo, když si uvědomíme, že n se pohybuje v řádech pěti set milionů. Proto je současným cílem vyvinout metody, jak úlohu rozdělit na dílčí problémy, ty vyřešit jednotlivě, a pak dát opět rozumným způsobem dohromady aby se docílilo optimálního výsledku.

---

[4] APPLEGATE, David L. *The traveling salesman problem: a computational study*. S.78.
[5] COOK, William. *Po stopách obchodního cestujícího: matematika na hranicích možností. S. 187.*
[6] APPLEGATE, David L. *The traveling salesman problem: a computational study. S. 89*
[7] COOK, William. *Po stopách obchodního cestujícího: matematika na hranicích možností. S. 189*

2 Matematický pohled na problém

Z matematického hlediska jsou vstupní data znázorněna ohodnoceným grafem, kde uzly jsou jednotlivá města a ohodnocené hrany jsou vzdálenosti mezi těmito městy. Úkolem obchodního cestujícího je efektivně nalézt nejkratší hamiltonovskou kružnici (tj. co nejkratší cestu).

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

Definice 1.1 Nechť je x n-členný vektor proměnných, množina $B = \{0,1\}$, d n-členný vektor cen proměnných, d_x 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 1.2 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:

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

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

za podmínek: $$\sum_{j=1}^{n} x_{ij} = 1; i = 1, \ldots, n$$

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

První dvě podmínky zaručují, že bude každý uzel navštíven právě jednou.

2.1 Modifikace úloh

Základní verze problému je taková, že obchodní cestující smí navštívit každé město právě jednou, tedy nemůže žádné město vynechat a zároveň se nesmí vrátit do žádného s výjimkou počátečního města, ve kterém musí také svou cestu zakončit. Problém může být eventuálně upraven tak, že se může vrátit do města, ve kterém již byl, pokud to je pro jeho cestu výhodnější.

Metrický problém je takový, pokud pro všechna města (i, j, k) platí trojúhelníková nerovnost $C_{ik} \leq C_{ij} + C_{jk}$, kde C_{ik} je vzdálenost mezi body i a k.

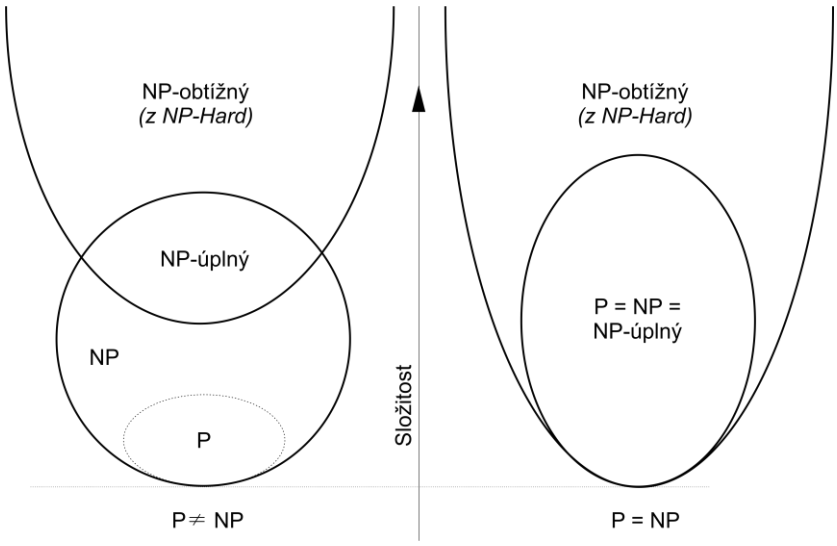
U **symetrického problému** platí, že $C_{ij} = C_{ji}$ Je to situace, kdy potřebujeme zajistit, aby obchodní cestující prošel města po co nejkratší dráze, tzn. hodnoty hran jsou vzdálenosti. V tomto případě nezáleží, jakým se pohybujeme směrem.

Euklidovský problém je takový, kde odpovídají hodnoty C_{ij} vzdálenostem bodů v rovině. Euklidovský problém je jak symetrický, tak metrický.

Asymetrický problém se liší v tom, že C_{ij} se nerovná C_{ji}. Lze si to představit jako cestu kopcovitým terénem, kdy cesta z kopce trvá kratší dobu než cesta do kopce. Hodnoty hran jsou v tomto případě doby trvání cesty z jednoho města do druhého.

2.2 Třídy složitosti

Úlohy řadíme do několika tříd, roztřídění můžete vidět na Eulerově diagramu, který je uveden níže.



Obrázek 1 Eulerův diagram. Zdroj: [11]

2.2.1 P versus NP

Mnoho problémů, které vznikají v praxi, jako je například problém nejkratší cesty, jsou spíše problémy optimalizační než problémy rozhodovací. Každý takový problém však implicitně zahrnuje nekonečně mnoho problémů spojených s rozhodováním. Zaznamenali jsme tři vztahy inkluze mezi třídami P, NP a co-NP a je přirozené se ptát, zda jsou tyto inkluze správné. Protože P = NP pouze pokud P = co-NP, zde vyvstávají tyto dvě základní otázky, obě byly představovány jako dohady.[8]

COOK-EDMONS-LEVINŮV DOHAD

$$P \neq NP$$

EDMONSŮV DOHAD

$$P = NP \cap co - NP$$

Jeden z nevyřešených problémů v oblasti počítačových věd určuje, zda existují otázky, jejichž odpověď může být rychle zkontrolována, ale které vyžadují neuvěřitelně dlouhou dobu k vyřešení jakýmkoli přímým postupem. Problémy, jako je výše uvedené, se zdají být takového druhu, ale zatím nikdo nedokázal dokázat, že některý z nich je opravdu tak obtížný, jak se zdá, tj. Že skutečně neexistuje žádný způsob, jak vygenerovat odpověď s pomocí počítače. Stephen Cook a Leonid Levin formulovali samostatně v roce 1971 problém P, což je snadné nalezení versus NP, což je snadno kontrolovatelný.[9]

[8] BONDY, J. A. a U. S. R. MURTY. *Graph theory. S. 125.*
[9] P vs NP Problem | Clay Mathematics Institute. The Clay Mathematics Institute | Clay Mathematics Institute [online]. Copyright © Clay Mathematics Institute [cit. 22.04.2019]. Dostupné z: http://claymath.org/millennium-problems/p-vs-np-problem

3 Teorie grafů

Mnoho reálných situací lze znázornit pomocí grafu či sítě, neboli pomocí množiny bodů (uzlů, vrcholů,…) a hran, tj. spojnic mezi nimi. Uzly mohou představovat distribuční centra a hrany zas přímé trasy mezi nimi.

Definice 3.1 Grafem rozumíme uspořádanou dvojici, která se skládá z množiny uzlů tzv. vrcholů a množiny hran, přičemž hrany jsou dvojice uzlů.

Graf je možno znázornit v rovině tak, že uzly jsou znázorněny body (kolečky, ovály, rámečky, trojúhelníčky,…)a hrany jejich spojnicemi.

3.1 Graf a jeho součásti

Jak již bylo řečeno výše, graf se skládá z uzlů a hran. Hrana může být orientovaná nebo neorientovaná. Pokud je hrana neorientovaná znamená to, že pořadí vrcholů není určeno, tj. jedná se o neuspořádanou dvojici uzlů, hrana nemá daný směr a umožňuje tedy obousměrný pohyb mezi uzly, které spojuje. Naproti tomu hrana orientovaná je uspořádaná dvojice uzlů, kde je jejich pořadí určeno. To znamená, že má hrana definován směr od prvního uzlu (tzv. počátečního uzlu hrany) ke druhému (tzv. koncovému uzlu hrany) a průchod hranou je umožněn pouze v tomto pořadí. Při znázorňování se pro orientované hrany používají orientované úsečky či oblouky s šipkami, pro neorientované hrany se používají také úsečku nebo oblouky ale již bez šipek.

Má-li graf všechny hrany neorientované, nazývá se neorientovaný graf, má-li všechny hrany orientované, pak jde o graf orientovaný. Obsahuje-li jak orientované, tak i neorientované hrany, jedná se o graf částečně orientovaný. Tomuto se lze naštěstí ve většině případů vyhnout tak, že ho nahradíme grafem orientovaným a to tak, že namísto každé neorientované hrany dáme dvojici navzájem opačně orientovaných hran. Ty budou mít stejné uzly, ale jejich pořadí bude opačné.

Pro graf se používá matematické značení $G = (V, E)$ v případě neorientovaného grafu a $G = (V, A)$ v případě orientovaného grafu, kde G je graf, V množina jeho uzlů (podle anglického pojmu „vertex") a E množina neorientovaných hran (podle anglického pojmu „edge"), respektive A množina orientovaných hran (podle anglického pojmu „arc"). Je-li hrana *u* a *v*, používáme pro neorientovanou hranu v souladu s matematickými zvyklostmi zápis se složenými množinovými závorkami, tj. $e = \{u, v\}$ – neuspořádaná dvojice je vlastně dvouprvková množina, zatímco pro orientovanou hranu budeme používat závorky kulaté, tj. $e = (u, v)$.[10]

Příklad 1 $G = (V(G), E(G))$

kde: $V(G) = \{u, v, w, x, y\}$
$E(G) = \{a, b, c, d, e, f, g, h\}$

a ψ_G je definováno jako

$$\psi_G(a) = uv \quad \psi_G(b) = uu \quad \psi_G(c) = uw \quad \psi_G(d) = wx$$
$$\psi_G(e) = vx \quad \psi_G(f) = wx \quad \psi_G(g) = ux \quad \psi_G(h) = xy$$

Příklad 2 $H = (V(H), E(H))$

kde: $V(H) = \{v_0, v_1, v_2, v_3, v_4, v_5\}$
$E(H) = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}\}$

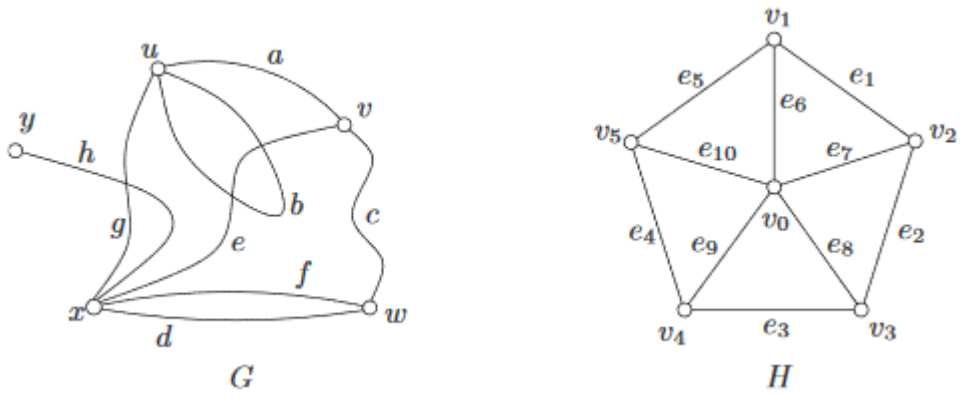
a ψ_H je definováno jako

$$\psi_H(e_1) = v_1v_2 \quad \psi_H(e_2) = v_2v_3 \quad \psi_H(e_3) = v_3v_4 \quad \psi_H(e_4) = v_4v_5 \quad \psi_H(e_5) = v_5v_1$$
$$\psi_H(e_6) = v_0v_1 \quad \psi_H(e_7) = v_0v_2 \quad \psi_H(e_8) = v_0v_3 \quad \psi_H(e_9) = v_0v_4 \quad \psi_H(e_{10}) = v_0v_5$$

Grafy jsou pojmenovány právě slovem grafy z toho důvodu, že je můžeme graficky reprezentovat. A je to právě grafická reprezentace grafů, která nám pomáhá porozumět spoustě jejich vlastností. Schémata grafů G a H můžeme vidět níže.

---

[10] KUČERA, Petr. *Modely teorie grafů I*. S.7.
Ψ (psí) je dnes již běžně nepoužívané písmeno v cyrilice



Obrázek 3.1 Diagramy grafů G a H. Zdroj: [5]

Neexistuje jediný správný způsob kreslení grafu; relativní polohy bodů reprezentujících vrcholy a tvary čar reprezentujících hrany obvykle nemají žádný význam. Na obr. 1.1 jsou hrany G znázorněny křivkami a křivky H přímými segmenty. Diagram grafu pouze zobrazuje vztah incidence. Často však kreslíme diagram grafu a odkazujeme na něj jako na samotný graf; ve stejném duchu nazýváme jeho body „vrcholy" a jeho linie „hrany". Pokud hrana e obsahuje uzel v, říkáme, že e a v spolu incidují. Uzly, které incidují se stejnou hranou, se nazývají sousední uzly nebo stručněji sousedé. Množina všech sousedů uzlu v se pak nazývá okolí uzlu v.

Počet hran, které s uzlen incidují, se nazývá stupeň uzlu. Počet hran, pro které je daný uzel počátečním, resp. koncovým, se nazývá výstupní, resp. vstupní stupeň uzlu. Uzel se vstupním stupněm bývá označován následujícími pojmy: počáteční uzel grafu, vstupní uzel, vstup. Uzel s výstupním stupněm 0 zase pojmy: koncový uzel grafu, výstupní uzel, výstup. Při některých aplikacích je potřeba, aby se v grafu vyskytovala více než jedna hrana incidující se stejnou dvojicí uzlů a pokud je orientovaná, také se stejnou orientací. Takovýto graf se nazýváme multigrafem.

Kvantitativní charakter modelů teorie grafů se docílí zavedením ohodnocených grafů, tento graf se nazývá hranově ohodnocený, pokud je každé hraně přiřazena nějaká hodnota a uzlově ohodnocený, pokud je určitá hodnota přiřazena každému uzlu. Mají-li hrany skutečně kvantitativní charakter, bývají buďto obecně označovány jako váhy

nebo je můžeme nazývat přímo podle jejich ekonomické interpretace, tj. např. vzdálenost mezi uzly, množství zboží, které je možno po hraně přepravit, čas potřebný na průchod hranou apod. Někdy má ohodnocený graf pouze nominální (kvalitativní) charakter, například když jsou hrany (uzly) rozděleny do určitých tříd. V těchto případech se namísto pojmu váha používá pojem barva, to může znamenat, že každá třída je označená jinou barvou.

3.2 Grafy orientované a neorientované

V první části této podkapitoly se budu věnovat neorientovaným grafům a jeho součástem, ve druhé části pak grafům orientovaným.

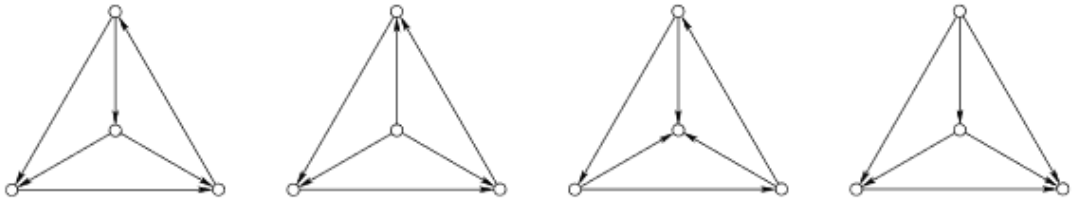
Neorientované grafy

Neorientovaná cesta, nebo také řetěz, je graf, který má uzly $v_1, v_2, \dots, v_n$, a hrany $e_1 = (v_1, v_2)$, $e_2 = (v_2, v_3), \dots, e_{n-1} = (v_{n-1}, v_n)$. Je tedy tvořen posloupností na sebe navazujících hran. Má dva uzly stupně 1: v_1 a v_n, které budeme nazývat jako počáteční, nebo koncové uzly řetězu. Všechny ostatní uzly jsou stupně 2.

Kružnice je graf, který se liší od řetězu tím, že má navíc hranu tvořenou jeho koncovými uzly, tj. má uzly $v_1, v_2, \dots, v_n$, a hrany $e_1 = \{v_1, v_2\}, e_2 = \{v_2, v_3\}, \dots, e_{n-1} = \{v_{n-1}, v_n\}, e_n = \{v_n, v_1\}$. Kružnice nemá nikde počátek ani konec a všechny uzly jsou si rovnocené.

Orientované grafy

Orientovaná cesta je graf, který má uzly $v_1, v_2, \dots, v_n$, a hrany $e_1 = (v_1, v_2)$, $e_2 = (v_2, v_3), \dots, e_{n-1} = (v_{n-1}, v_n)$.



Obrázek 3.2 Orientované grafy. Zdroj [5]

Cyklus se liší od cesty tím, že má navíc hranu tvořenou koncovým uzlem cesty v_n a počátečním v_1, tj. má uzly $v_1, v_2, \ldots, v_n$, a hrany $e_1 = (v_1, v_2)$, $e_2 = (v_2, v_3), \ldots, e_{n-1} = (v_{n-1}, v_n), e_n = (v_n, v_1)$. To znamená, že nemá žádný počáteční ani koncový uzel.

Acyklický graf je takový, jehož podgrafem není žádný cyklus.

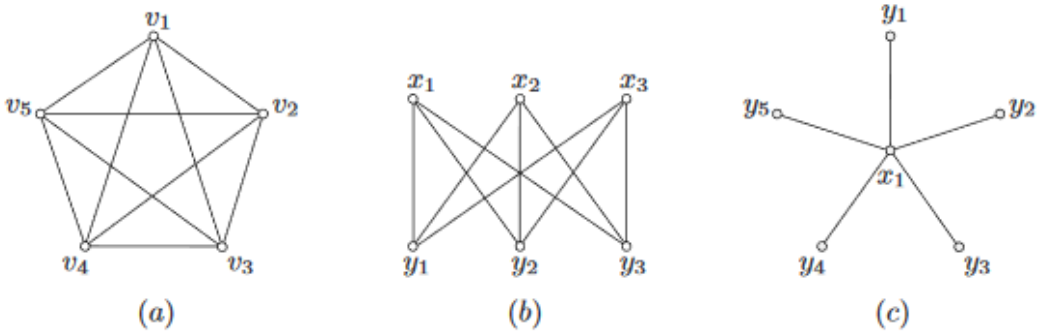
Síťový graf je orientovaný, souvislý, nezáporně ohodnocený graf, která má právě jeden vstup a právě jeden výstup.

3.3 Speciální typy grafů

Některé typy grafů hrají v teorii grafů významné role.

Graf je **bipartijní**, pokud jeho množina vrcholů může být rozdělena do dvou podmnožin X a Y tak, že každá hrana má jeden konec v X a jeden konec v Y; taková dvojice (X, Y) se nazývá bipartice grafu a jeho části X a Y. Označujeme bipartijní graf G s bipartií (X, Y) podle G $[X, Y]$. Je-li G $[X, Y]$ jednoduché a každý vrchol v X je spojen s každým vrcholem v Y, pak se G nazývá úplný bipartijní graf. Hvězda

je kompletní bipartijní graf G $[X,Y]$ s $|X| = 1$ nebo $|Y| = 1$.[12] Obrázek 3.2 ukazuje diagramy úplného grafu, kompletní bipartijní graf a hvězdu.

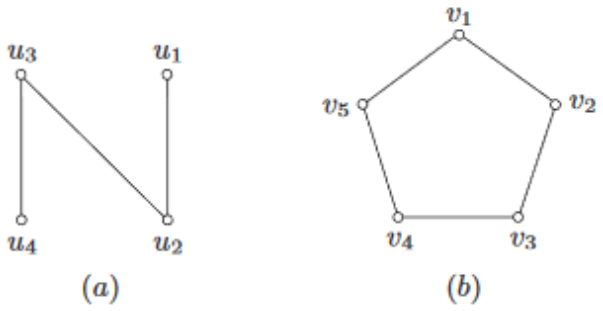


Obrázek 2.3 Diagramy grafu úplného, bipartijního a graf hvězdy. Zdroj [5]

Jako příklad bipartního grafu v praxi zde můžu uvést kompletní bipartní graf z obr. 3.3 (b), kde x_1, x_2, x_3 jsou jednotlivý dodavatelé (tj. první část grafu) a y_1, y_2, y_3 jsou zákazníci (tj. druhá část grafu), bez stanovení omezujících podmínek, bychom mohli jednotlivé zákazníky a dodavatele spojit právě takto.

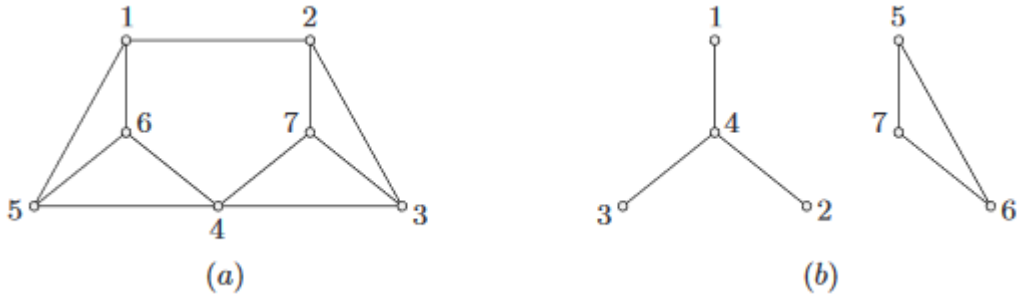
[12] BONDY, J. A. a U. S. R. MURTY. *Graph theory. S. 116*

Cesta je jednoduchý graf, jehož vrcholy mohou být uspořádány v lineární posloupnosti takovým způsobem, že dva vrcholy jsou sousedící, pokud jsou v sekvenci po sobě jdoucí, a jinak než takto jsou nesousedící. Stejně tak cyklus tří nebo více vrcholů je jednoduchý graf, jehož vrcholy mohou být uspořádány v cyklické sekvenci takovým způsobem, že dva vrcholy jsou sousedící, jsou-li v sekvenci po sobě jdoucí a jsou nesousedící jinak než takto; cyklus na jednom vrcholu sestává z jediného vrcholu se smyčkou a cyklus na dvou vrcholech sestává ze dvou vrcholů spojených dvojicí paralelních hran. Délka dráhy nebo cyklu je počet jejích hran. Cesta nebo cyklus délky k se nazývá k-dráha cyklu; dráha nebo cyklus je podle parity k. A 3-cyklus často nazýváme jako trojúhelník, 4-cyklus čtyřúhelník, 5-cyklus pětiúhelník, 6-cyklus šestiúhelník, a tak dále. Obrázek 3.4 znázorňuje cestu délky 3 a 5-cyklus.



Obrázek 3.3 Diagram grafu délky 3 a 5-cyklus. Zdroj: [5]

Souvislý graf je takový, v němž pro jeho každé dva uzly existuje jako podgraf řetěz, jehož jsou tyto uzly koncovými. V opačném případě se graf nazývá nesouvislým. Příklady souvislých a nesouvislých grafů jsou zobrazeny na obr. 3.4.



Obrázek 3.4 Souvislý a nesouvislý graf. Zdroj [5]

Komponentou grafu je pak každý jeho maximální souvislý (indukovaný) podgraf. Nesouvislý graf je disjunktním sjednocením svých komponent.

Strom je souvislý graf, jehož žádným podgrafem není kružnice. Částečný graf grafu G, který je strom, se nazývá kostra grafu G.

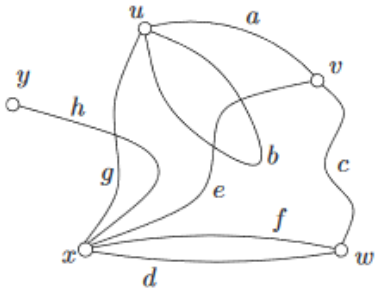
Rovinný graf je takový, který lze v rovině zakreslit tak, že se žádné dvě jeho hrany neprotínají.

3.4 Matice incidence a sousednosti

I když jsou kresby vhodným prostředkem pro určování grafů, nejsou jednoznačně vhodné pro ukládání grafů do počítačů, ani pro použití matematických metod pro studium jejich vlastností. Pro tyto účely uvažujeme dvě matice spojené s grafem, jeho incidenční matici a jeho matici přilehlosti. Nechť je G graf s množinou vrcholů V a množinou hran E. Matice incidence G je pak matice $n \times m$ $MG: = (m_{ve})$, kde m_{ve} značí počet $(0, 1\, nebo\, 2)$ kolikrát vrchol v a hrana e incidují. Je tedy

[15] KUČERA, Petr. *Modely teorie grafů I. S. 10*

zřejmé, že matice incidence je jen další cestou pro určování grafu. Matice sousednosti G je pak $n \times n$ matice $AG := (a_{uv})$, kde a_{uv} je počet hran spojujících vrcholy u a v, každá smyčka se počítá jako dvě hrany.[16] Matice grafu G na obr. 1.1 jsou znázorněny na obr. 3.6.



G

	a	b	c	d	e	f	g	h
u	1	2	0	0	0	0	1	0
v	1	0	1	0	1	0	0	0
w	0	0	1	1	0	1	0	0
x	0	0	0	1	1	1	1	1
y	0	0	0	0	0	0	0	1

M

	u	v	w	x	y
u	2	1	0	1	0
v	1	0	1	1	0
w	0	1	0	2	0
x	1	1	2	0	1
y	0	0	0	1	0

A

Obrázek 3.5 Matice incidence a sousednosti grafu G. Zdroj [5]

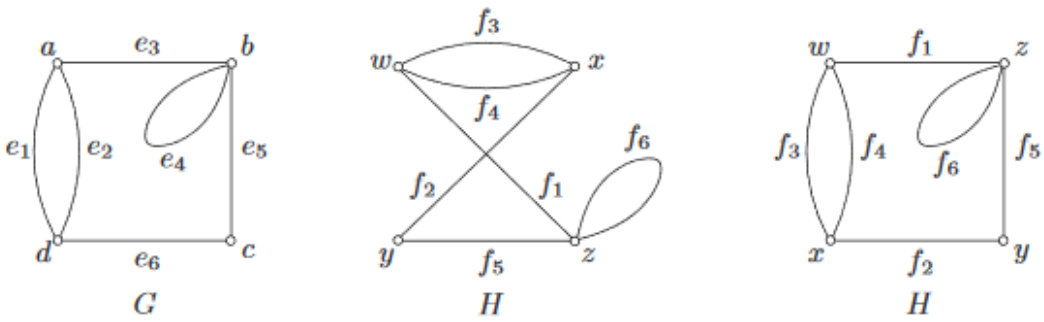
3.5 Isomorfismus a automorfismus

V této podkapitole se budu jen krátce zabývat isomorfismem a automorfismem a jako první začnu s isomorfismem.

Definice 3.2.

Dva grafy $G = (V, E)$ a $G' = (V', E')$ nazýváme **isomorfní**, jestliže existuje bijektivní (tj. vzájemně jednoznačné) zobrazení $f: V \to V'$ tak, že platí: $\{x, y\}$ je prvkem E, právě když $\{f(x), f(y)\}$ je prvkem E'. Příklady isomorfních grafů jsou uvedeny na obr. 3.7.

[16] BONDY, J. A. a U. S. R. MURTY. *Graph theory. S. 118*



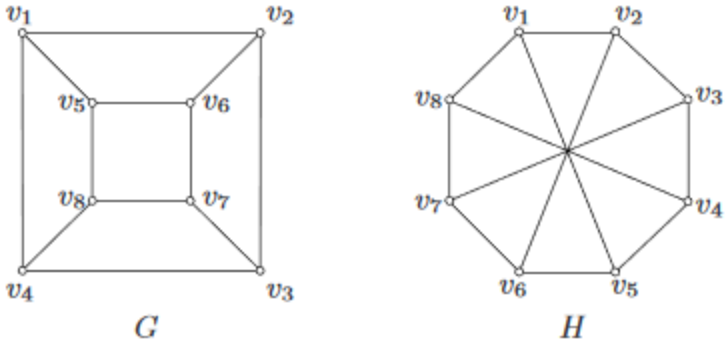
Obrázek 3.6 Grafy isomorfní. Zdroj: [5]

Abychom si pro pořádek ukázali, že jsou grafy G a H opravdu isomorfní, provedeme zobrazení.

$$\theta \coloneqq \begin{pmatrix} a & b & c & d \\ w & z & y & x \end{pmatrix} \qquad \emptyset \coloneqq \begin{pmatrix} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \\ f_3 & f_4 & f_1 & f_6 & f_5 & f_2 \end{pmatrix}$$

Naopak jako příklad grafů, které nejsou isomorfní, nám poslouží následující obrázek.

Isomorfismus grafu sama na sebe se nazývá **automorfismus**. Je to tedy zobrazení, pro které platí, že se jedná buď o identitu, nebo o permutaci uzlů zachovávající strukturu grafu.

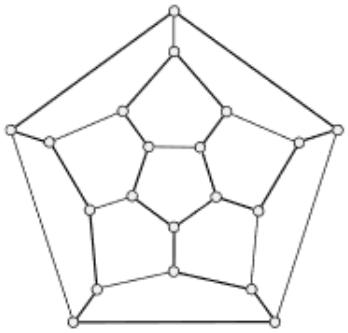


Obrázek 3.7 Neisomorfní grafy. Zdroj: [5]

3.6 Hamiltonova hra a kružnice

Řekněme si, že cesta nebo cyklus, který obsahuje každý vrchol grafu, se nazývá Hamiltonova cesta nebo Hamiltonův cyklus grafu. Takové cesty a cykly jsou pojmenovány po sirovi Williamovi Rowanovi Hamiltonovi, který v dopise svému příteli Gravesinovi 1856 popsal matematickou hru na dodecahedronu(tj. dvanáctistěnu). Hamilton studoval problém cesty, která má projít všech dvacet vrcholů pravidelného dvanáctistěnu. Hamilton načrtl graf, který odpovídá řešenému problému a nazval ho Icosian (tj. z řeckého „eikosi", dvacet). Vrcholy v grafu představují hrany v pravidelném dvanáctistěnu a malá kolečka představují jeho vrcholy.[18]

[18] COOK, William. *Po stopách obchodního cestujícího: matematika na hranicích možností. S. 41*



Obrázek 3.8 Hamiltonův graf: Icosian. Zdroj: [5]

Hamiltonova cesta po vrcholech a hranách dvanáctistěnu se dá tedy reprezentovat jako cesta v grafu. Dopis, ve kterém Hamilton popisoval svou hru, zněl přibližně následovně.

> Stejně jako v drobném článku, který jsem vám poslal minule, uvažujme tři symboly: $\iota, \kappa, \lambda,$ které splňují následující rovnice:
>
> $$\iota^2 = 1,\ \kappa^3 = 1,\ \lambda^5 = 1,\ \lambda = \iota\kappa$$
>
> Nejprve musíme na jednom či dvou příkladech ukázat, že takto definované symboly mají nezvyklé, ale přesně určené vlastnosti, které z nich činí legitimní nástroj formálního kalkulu: každý výsledek dosažený s jejich pomocí (soudím tak z mnoha výsledků, které jsem zatím získal) lze snadno zajímavě interpretovat s odkazem na cestu mezi stěnami či vrcholy v jednom z pevných těles uvažovaných ve starověké geometrii. Zjistil jsem, že někteří mladí lidé nacházejí zálibu v jisté matematické hře, kterou lze v mém grafu hrát: první hráč zasune špendlík do libovolných pěti po sobě jdoucích vrcholů, např. $abcde$ nebo $abcde'$, a úkolem druhého hráče je doplnit dalších patnáct špendlíků tak, aby pokryl všechny zbývající body a skončil přesně vedle prvního špendlíku, který zasunul první

hráč. Připomínám současně, že podle teoretických výsledků v tomto dopise může druhý hráč vždy takovou cestu nalézt.[20]

Hamiltonova kružnice je uzavřená cesta v grafu, která prochází každým vrcholem právě jednou. A úloha rozhodnout, jestli v daném grafu existuje hamiltonovská kružnice, či nikoli, je příkladem NP – úplného problému, a tento problém v sobě tedy obsahuje prakticky celou složitost TSP.

Následující zjednodušená podmínka se ukazuje jako překvapivě užitečná.

Věta: Nechť S je množina vrcholů hamiltonovského grafu G

pak, $c(G - S) \leq |S|$

Pokud tedy rovnost ve větě platí, pak každá z $|S|$ komponent $G - S$ je sledovatelná a každý Hamiltonův cyklus G obsahuje v každé z těchto složek Hamiltonovu cestu.[21]

4 Lineární programování

Na úvod této kapitoly je potřeba podotknout, že se dříve pro název lineárního programování používal termín „lineární optimalizace".

[20] COOK, William. *Po stopách obchodního cestujícího: matematika na hranicích možností. S. 46*
[21] BONDY, J. A. a U. S. R. MURTY. *Graph theory. S.471*

Otázkou existence řešení úlohy lineárního programování se zabývá již teorie lineárních nerovností zpracována na počátku minulého století matematikem J. Farkasem. Zásadní rozvoj lineárního programování však vyvolala až nutnost řešit složité ekonomické problémy, s nimiž se setkávaly válčící země během druhé světové války. Zejména tedy pak po druhé světové válce, přibližně v té době zrovna mladý Dantzig dostal lákavou nabídku, která ho měla udržet v Pentagonu. Jeho úkolem bylo mechanizovat procesy spojené s vojenským plánováním, Dantzig se do úkolu pustil s plnou vervou a vytvořil tak novou teorii s převratnými důsledky – lineární programování, krátce LP.[22]

Lineární programování se zabývá speciálními úlohami na vázané extrémy funkcí více proměnných. Specifikum těchto úloh spočívá v tom, že jde o extrémy lineárních funkcí limitovaných podmínkami ve tvaru lineárních rovnic a nerovností.

Kompletní řešení TSP se zakládá nejen v nalezení nejlepší cesty v množině bodů, ale také v podání důkazu, že nalezená cesta je skutečně tou nejlepší ze všech možných cest. Užití hrubé síly, tedy prozkoumání všech těchto možných cest sice v principu splňuje tato kritéria, ale není matematicky elegantní, a také často z časového hlediska není prakticky proveditelné. Lineární programování umí najít způsob jak optimální cestu nalézt a jak obhájit, že daná cesta je optimální, aniž bychom kvůli tomu museli zkontrolovat všechny ostatní cesty. Dokáže potvrdit tvrzení „žádná cesta těmito body nemůže být kratší než X". Číslo X zde chápeme jako takové měřítko optimálnosti cesty, jestliže tedy dokážeme najít cestu délky X, můžeme si být jisti, že jsme našli právě tu optimální. Jinými slovy se jedná o způsob, jak získat jisté záruky kvality cesty na základě vhodné kombinace jednoduchých pravidel. Existence takové záruky vyplývá ze slabé věty o dualitě, té se budu věnovat více později.

[22] REINELT, G. *The traveling salesman: computational solutions for TSP applications. S.214*

4.1 Formulace LP

Základní úlohu lineárního programování můžeme vyjádřit následujícím způsobem (zde uvedeno konkrétně pro nalezení maxima lineární funkce). Občas se úloha lineárního programování neformuluje pro maximum, nýbrž pro minimum účelové funkce.

Nalezněte maximum lineární funkce:

$$z = c_1x_1 + c_2x_2 + \cdots + c_nx_n \qquad 4.1$$

Za podmínek:

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \qquad 4.2$$

$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2$$

$$\vdots$$

$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m$$

$$x_1 \geq 0 \qquad (i = 1,2,3,\dots,n) \qquad 4.3$$

Lineární funkci z (4.1), jejíž maximum hledáme, nazýváme účelovou funkcí. Dané rovnice (ad. 4.2) nazýváme vlastní omezení úlohy (neboli podmínky). Dané nerovnosti (4.3) nazýváme podmínkami nezápornosti. Proměnné x_i, kde $i = 1,2,\dots,n$, charakterizují objemy procesů, které se mají realizovat, má-li být daný úkol splněn.

Lineární programování koresponduje takovému případu, kdy jsou všechny funkce (tj. účelová funkce i všechna vlastní omezení)

lineární. V lineárním programování se vedle linearity všech funkcí předpokládá, že jsou i všechny koeficienty úlohy (tj. koeficienty všech funkcí) reálná čísla.

Jestliže některé koeficienty lineárně závisí na určitých parametrech, mluvíme o tzv. úloze parametrického lineárního programování.[23]

Jak již bylo zmíněno úloha lineárního programování je specifickým případem úlohy matematického programování, kterou lze matematicky formulovat jako úlohu nalezení extrému tj. maxima nebo minima, kde maximem rozumíme matematickou funkci, jejíž hodnota představuje nejnižší hodnotu ze všech vstupních parametrů. Funkce provádí porovnání jednotlivých parametrů a výsledkem je hodnota toho parametru, který se při porovnání se všemi ostatními jeví jako nejnižší a minimem zas naopak rozumíme matematickou funkci, jejíž funkční hodnota představuje nejvyšší hodnotu ze všech vstupních parametrů. Funkce provádí porovnání jednotlivých parametrů a výsledkem je hodnota toho parametru, který se při komparaci se všemi ostatními jeví jako nejvyšší, na množině řešení soustavy rovnic a nerovností. Tato funkce se nazývá účelová (též cílová nebo kriteriální) funkce. Nerovnosti lze převést na rovnice o větším počtu neznámých a řešení úloh minimalizačních lze převést na maximalizační

4.2 Maticový zápis úlohy LP

Úlohu LP lze též zapsat ve zkráceném maticovém tvaru:

$$\begin{Bmatrix} max \\ min \end{Bmatrix} \quad \mathbf{c}^{T}\mathbf{x}; \text{ účelová funkce}$$

za podmínek: $$\mathbf{Ax} \begin{Bmatrix} \leq \\ = \\ \geq \end{Bmatrix} \mathbf{b};$$

[23] SCHRIJVER, Alexander. *Theory of linear and integer programming*.s 68

$$\mathbf{x}\ \{\geq \mathbf{0}\};$$

A matice typu $m \cdot n$; c^T řádkový vektor typu $1 \cdot n$; **b** sloupcový vektor typu $m \cdot 1$; **x** vektor neznámých typu $n \cdot 1$; (nad tělesem reálných čísel **R**)

$\mathbf{c}^{\mathrm{T}} = (c_1, \ldots, c_n)$; $\mathbf{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$; $\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$; $\mathbf{A} = \begin{pmatrix} \boldsymbol{a_{11}} & \cdots & \boldsymbol{a_{1n}} \\ \cdots & \cdots & \cdots \\ \boldsymbol{a_{m1}} & \cdots & \boldsymbol{a_{mn}} \end{pmatrix}$

Na množině vektorů lze evidentním způsobem zavést částečné uspořádání.

Nerovnice lze převést na rovnice přidáním dalších doplňkových proměnných.

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i \Leftrightarrow \sum_{j=1}^{n} a_{ij}x_j + x_i' = b_i;\ x_i' \geq 0; \quad \sum_{j=1}^{n} a_{ij}x_j \geq b_i \Leftrightarrow \sum_{j=1}^{n} a_{ij}x_j - x_i' = b_i;\ x_i' \geq 0;$$

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \Leftrightarrow \sum_{j=1}^{n} a_{ij}x_j \geq b_i \wedge \sum_{j=1}^{n} a_{ij}x_j \leq b_i;$$

$$max\ \boldsymbol{c}^T\boldsymbol{x} = -min(-\boldsymbol{c}^T\boldsymbol{x});$$

Pokud na některé proměnné není kladen požadavek nezápornosti, lze provést tento rozklad:

$$x_j = x_j^+ - x_j^-;\ x_j^+ = \max(x;0), x_j^- = \max(-x;0),\ x_j^+ \geq 0, x_j^- \geq 0;$$

To znamená, že každou úlohu lineárního programování lze převést na tvar:

$$min\ \boldsymbol{c}^T\boldsymbol{x}; za\ podm. \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}$$

nebo na tvar:

$$min\ \boldsymbol{c}^T\boldsymbol{x}; za\ podm. \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}$$

Vynásobíme-li nerovnost (-1), znaménko nerovnosti se otočí.

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i \Leftrightarrow -\sum_{j=1}^{n} a_{ij}x_j \geq -b_i; \quad \sum_{j=1}^{n} a_{ij}x_j \geq b_i \Leftrightarrow -\sum_{j=1}^{n} a_{ij}x_j \leq -b_i;$$

4.3 Řešení úlohy LP

Přípustné řešení

Jde o takové řešení soustavy lineárních rovnic (4.2), které splňuje podmínky nezápornosti (4.3).

Optimální řešení

Je právě takové přípustné řešení, které maximalizuje účelovou funkci vyznačenou (4.1).

Degenerované řešení

Základní řešení, které má více než $n-m$ nulových složek.

Základní řešení

Jedná se o přípustné řešení, které má nejvýše tolik kladných složek, kolik je lineárně nezávislých rovnic tvořících vlastní omezení (ad. 4.2), (v našem případě nejvýše m kladných složek a nejméně $n-m$ nulových složek za předpokladu, že je $n > m$). Sloupce koeficientů v matici **A** odpovídající kladným složkám základního řešení jsou lineárně nezávislými vektory.

Nechť $h(A) = m$. Základní řešení úlohy lineárního programování v rovnicovém tvaru nazýváme nedegenerované, jestliže má právě m kladných složek. Můžeme říct, že úloha lineárního programování v

rovnicovém tvaru je nedegenerovaná, jsou-li nedegenerovaná všechna její základní řešení.

4.4 Simplexový algoritmus

Dantzig představil lineární programování a také simplexový algoritmus na své přednášce v roce 1948 na Wisconsinské univerzitě. Ačkoli se mohlo v průběhu let zdát, že je tento algoritmus překonaný, jeho nová implementace z konce 80. Let, která zahrnovala programy CPLEX a OSL, potvrdila, že starý algoritmus má ještě ledacos do sebe. V roce 2000 byl původní Dantzigův algoritmus vyhlášen jedním z deseti nejlepších algoritmů století.[24]

Nalezení výchozího řešení

Převedení SLN na SLR v kanonickém tvaru s nezápornou pravou stranou všech rovnic; tj. $b_i \geq 0, i = 1, 2, \dots, m$

Při zadání úlohy ve tvaru $A\vec{x} \leq \vec{b}, b_i \geq 0, i = 1, 2, \dots, m$
Převedením obdržíme:

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n + x_1' = b1 \qquad x_1' = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n + x_2' = b2 \qquad x_2' = b_2$$

$$\vdots$$

$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \qquad x_m' = bm \qquad x_m' = b_m$$

$$x_1 = x_2 = \cdots = x_n = 0$$

Základní výchozí řešení je ve tvaru $\vec{x} = (0, \dots, 0;\ b_1, b_2, \dots, b_m)$

[24]COOK, William. *Po stopách obchodního cestujícího: matematika na hranicích možností. S. 120*

Řešení je přípustné, neboť $b_i \geq 0$ (tj. vyhovuje podmínkám nezápornosti).

Při zadání úlohy, kdy některé $b_i < 0$, danou rovnici násobíme (-1)

Při zadání úlohy ve tvaru $A\vec{x} \geq \vec{b}, b_i \geq 0, i = 1, 2, \ldots, n$
SLN převedeme na SLR, která ale není v kanonickém tvaru.

$$x_1 - 2x_2 \geq 5 \qquad x_1 - 2x_2 - x_1' = 5$$

$$2x_1 + 3x_2 \geq 2 \qquad 2x_1 + 3x_2 - x_2' = 5$$

Rovnice nelze násobit (-1), protože bychom obdrželi zápornou pravou stranu a řešení by tak nebylo přípustné. Výchozí přípustné řešení lze získat zavedením pomocných proměnných a pomocné účelové funkce.

Poté co převedeme soustavu omezujících podmínek na rovnicový kanonický tvar pomocí doplňkových a pomocných proměnných. Sestavíme simplexovou tabulku; do simplexových tabulek zaznamenáváme celý průběh výpočtu.

Zákl. proměnné	x_1	x_2	x_1'	x_2'	b_i	$p = \frac{b_i}{a_{ik}}, pro\ a_{ik} \geq 0$
x_1' x_2'						
z						max

Tabulka 4.1 Simplexova tabulka. Zdroj [9]

Základní proměnné
Matice soustavy
Sloupec absolutních členů rovnic $b_i \geq 0$
Sloupec pro určení klíčového řádku
Koeficienty anulované účelové rovnice
Hodnota účelové funkce příslušná danému základnímu přípustnému řešení.

4.5 LP DUALITA

Jedním z důležitých výsledků teorie lineárního programování je poznatek, že s každou maximalizační úlohou je také automaticky spjata úloha minimalizační. Minimalizační úloha je danou maximalizační úlohou jednoznačně určena a má i velmi užitečné vlastnosti. Protože není podstatné, zda se zabýváme maximalizací či minimalizací, je i s každou minimalizační úlohou podobným způsobem spjata i úloha maximalizační. Poznatky o těchto vztazích mají důležitou roli právě při zjišťování existence optimálních řešení a jsou široce využívány v takových oblastech, jako je například ekonomie. Ke každé úloze týkající se lineárního programování (tzv. primární úloze) lze nějakým způsobem přiřadit jinou úlohu lineárního programování, a to úlohu sdruženou neboli tedy duální.

Duální úloha je sestavena z identických konstant. Dvojice sdružených úloh spolu úzce souvisí. Má řadu zajímavých společných vlastností, a to jak z hlediska výpočetní interpretace, tak i z hlediska ekonomické interpretace. Dualita představuje vzájemný vztah. Jakákoliv úloha z dvojice duálních úloh může být brána jako primární. Je dobré si uvědomit, že přívlastky primární a duální jsou podmíněné tím, ze které úlohy vycházíme. Každému vlastnímu omezení primární úlohy odpovídá jedna duální proměnná a každé podmínce nezápornosti primární úlohy odpovídá jedno duální omezení.[25]

Primární matematický model
Účelová funkce → max (resp. Min)
Vlastní omezení primární úlohy (počet m)
Podmínky nezápornosti prim. proměnných (počet n)

Duální matematický model
Účelová funkce → min (resp. max)
Podmínky nezápornosti duálních proměnných (počet m)
Vlastní omezení duální úlohy (počet n)

[25] LUENBERGER, David G. a David G. LUENBERGER. *Linear and nonlinear programming. S. 89*

$$z = \sum_{j=1}^{n} c_j\, x_j \rightarrow \max \qquad\qquad f = \sum_{i=1}^{m} b_j\, u_j \rightarrow min$$

Získáme duální proměnné y_i pro každou primární podmínku A_i.

$$y_i \in \mathbb{R} \text{ pro} \sum_{j=1}^{n} a_{ij} x_j = b_i$$

$$y_i \geq 0 \text{ pro} \sum_{j=1}^{n} a_{ij} x_j \leq b_i$$

Duální podmínka pro každou primární proměnnou x_j:

$$\sum_{i=1}^{m} a_{ij} y_i = c_j \text{ pro } x_j \in \mathbb{R}$$

$$\sum_{i=1}^{m} a_{ij} y_i \geq c_j \text{ pro } x_j \geq 0, \text{pokud primární maximalizujeme}$$

$$\sum_{i=1}^{m} a_{ij} y_i \leq c_j \text{ pro } x_j \geq 0, \text{pokud primární minimalizujeme}$$

Účelová funkce

$$\min \sum_{i=1}^{m} b_i y_i \text{ , pokud primární je } \max c^T x$$

$$\max \sum_{i=1}^{m} b_i y_i, \text{ pokud primární je } \min c^T x$$

Speciální případy pro: $a \in \mathbb{R}^{m \times n}, c \in \mathbb{R}^n,\ b \in \mathbb{R}^m$

$$\text{(P)} \quad \begin{array}{l} \max c^T x \\ x \in \mathbb{R}^n \\ Ax \leq b \end{array} \qquad\qquad \text{(D)} \quad \begin{array}{l} \min b^T y \\ y \geq 0 \\ A^T y = c \end{array}$$

$$\text{(P̄)} \quad \begin{array}{l} \max c^T x \\ x \geq 0 \\ Ax \leq b \end{array} \qquad\qquad \text{(D̄)} \quad \begin{array}{l} \min b^T y \\ y \geq 0 \\ A^T y \geq C \end{array}$$

4.5.1 Slabá věta o dualitě

Nechť x a y jsou přípustná řešení (P) a (D), pak $c^T x \leq b^T y$

Důkaz:

$$y^T Ax \leq x^T b \; z\,(P), y \geq 0$$
$$y^T ax \; = c^T x \; z\,(D)$$

Hodnota účelové funkce minimalizační úlohy v kterémkoli přípustném řešení je horní mezí hodnot účelové funkce duálně sdružené maximalizační úlohy na množině všech jejích přípustných řešení a obdobně hodnota účelové funkce maximalizační úlohy v kterémkoli přípustném řešení je dolní mezí hodnot účelové funkce duálně sdružené minimalizační úlohy na množině všech jejích přípustných řešení.[26]

4.5.2 Silná věta o dualitě

Má-li jedna z duálně sdružených úloh optimální řešení, má optimální řešení i úloha druhá, přičemž optimální hodnoty účelových funkcí jsou si rovny.[27]

Pro předchozí příklad mohou tedy nastat následující situace:

- ani (P) ani (D) nemá přípustné řešení
- jedna z nich je neomezená a druhá je nepřípustná
- (P) i (D) mají přípustné řešení, pak existují optimální řešení x^* a y^* a platí $c^T x^* = b^T y^*$

5 Heuristiky

Heuristika je zkusmé řešení problému. Tyto metody jsou založeny na způsobu prohledávání prostoru přípustných řešení. Většinou nezaručují nalezení optimálního řešení. Jako příklad z této kategorie může být např. náhodné prohledávání. Funguje na principu náhodného generování a ohodnocování X z množiny přípustných řešení.

[26] SCHRIJVER, Alexander. *Theory of linear and integer programming. S. 165*
[27] SCHRIJVER, Alexander. *Theory of linear and integer programming. S. 167*

6 Metaheuristiky

V této kapitole se budu věnovat popisu metaheuristických metod. Jmenovitě se jedná o simulované žíhání a mravenčí kolonie. Tyto metody budu později používat pro samotný výpočet praktického příkladu. Na začátek bych chtěla uvést, že se jedná o numerické metody a nejsou tedy úplně přesné. Výsledek konverguje k minimu.

6.1 Simulované žíhání

Metoda simulovaného žíhání patří mezi stochastické metody optimalizace, přestože její základ je ve fyzice. Většinou bývá předmětem studia počítačové fyziky. Tato metoda se zrodila v 80. letech, přibližně v té době, kdy se aplikovala analogie z fyzikální podstaty žíhání tuhých těles. To spočívá v tom, že je těleso zahřáto na vysokou teplotu a následně zas pozvolna ochlazováno – tím se předchází defektům ve struktuře tělesa (částice se dostanou do rovnovážné polohy). Defekty krystalické mřížky mají při vysoké teplotě vysokou pravděpodobnost zániku a právě pomalé ochlazování systému zabezpečí, že pravděpodobnost vzniku nových defektů klesá.

Předpokládejme, že proces ochlazování je dostatečně pomalý. Potom pro každou teplotu T je žíhané těleso v rovnovážném stavu a je popsán Boltzmanovým rozdělením pravděpodobnosti, že těleso je ve stavu s energií Ei při uvedené teplotě T.[28]

$$W_T(E_i) = \frac{e^{-\frac{E_i}{k_B T}}}{\sum_i e^{-\frac{E_i}{k_B T}}}$$

[28] PEARL, Judea. *Heuristics: intelligent search strategies for computer problem solving*. S. 120

Uvedené Boltzmanovo rozdělení upřednostňuje stavy s nižší energií při klesající teplotě. V případě, že se teplota blíží k nule, má stav s minimální energií nenulovou pravděpodobnost. V roce 1953 N. Metropolis a jeho kolegové nevrhli algoritmus založený na Metodě Monte Carlo. Když budeme předpokládat, že pro změnu stavu systému (změna řetězce kódujícího prohledávaný prostor) je použitelná je použita mutace generující řetězec z blízkého okolí, dalším předpokladem bude, že změna stavu je symetrická tj. pravděpodobnost toho, že se stav A změní na stav B, musí být stejná jako změna stavu z B na A. Pokud je rozdíl energie (např. rozdíl funkčních hodnot)mezi nagenerovaným stavem a aktuálním stavem záporný, potom proces pokračuje s novým právě nagenerovaným stavem. V opačném případě je pravděpodobnost přijetí nagenerovaného stavu určena vztahem:

$$p\,(\Delta\, E) = \; e^{-\frac{E_i}{k_B T}} \; .$$

Toto pravidlo nazýváme Metropolisovým kritériem a tato forma metody Monte Carlo se nazývá Metropolisův algoritmus. Pro to, aby byl algoritmus úspěšný je důležitá volba teploty, která by se z počátku měla volit vysoká (pro velké hodnoty T je pravděpodobnost přijetí blízká jedné (tj. přijmou se takřka všechny nové stavy). Po několika cyklech ji pak teprve budeme snižovat podle vzthahu:

$$T^{k+1} = \beta T^{k},$$

kde: $0 < \beta < 1$, je konstantou určující rychlost ochlazování. Většinou se T_0 volí tak, aby pravděpodobnost přijetí byla 0,8, parametr β pak nastavíme na hodnotu 0,95.

Konkrétní algoritmus aplikovaný na TSP (symetrický) je v příloze této práce.

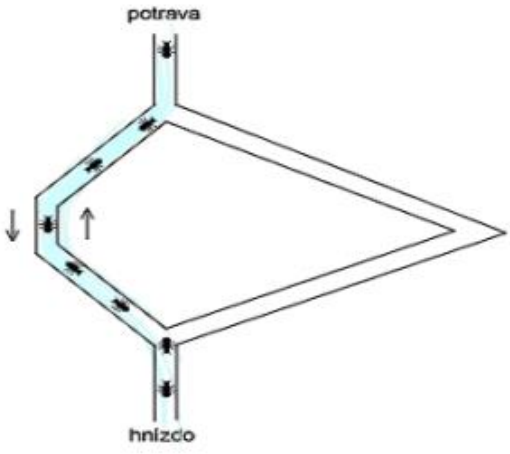
Pr	Pravděpodobnost nahrazení stávajícího řetězce novým
x*	Řešení
x'	Nové řešení
T	Teplota
α	Koeficient snižování teploty
k	Počet úspěšných pokusů

Tabulka 2 Popis proměnných algoritmu

6.2 Mravenčí kolonie

Optimalizace pomocí mravenčí kolonie nebo také ACO - *Ant Colony Optimalization, je* vhodný způsob k řešení složitých úloh. Tento přístup je poměrně moderní, jeho rozvoj se započal v 90. letech minulého století. Tento algoritmus využívá rojovou inteligenci (kolonie mravenců)je tedy inspirován skutečnými mravenci (jejich koloniemi).

Mravenci se při pohybu za potravou orientují podle pomalu se vypařujících feromonových stop, které za sebou zanechává každý mravenec z kolonie. Mravenci se přirozeně vydávají směrem, kde je feromonová stopa silnější.



Obrázek 9 Binární most s nestejně dlouhými rameny. Zdroj [7]

Šance, že si mravenec vybere kratší cestu je přímo úměrná délkám obou ramen.

Pro fungování algoritmu je důležitý pojem umělého mravence. Umělý mravenec je struktura vytvořená pro implementaci algoritmu. S reálným mravencem není totožný, pouze se inspiruje jeho chováním.

Konkrétní a celý algoritmus aplikovaný na TSP (symetrický) je podobně jako ten předchozí v příloze této práce.

τ	Hodnota feromonu
m	Počet mravenců
T	Cesta
T+	Nejkratší cesta
q	Náhodná veličina s rovnoměrným rozložením[0,1]
q_0	Nastavitelný parametr z intervalu [0,1]
j	Město
J_i^k	Města dosud nenavštívená mravencem
η	Dohlednost
p	Pravděpodobnost přechodu
L	Délka cesty
L+	Délka nejkratší cesty
ρ	Odpařování feromonu

Tabulka 3 Popis proměnných algoritmu

PRAKTICKÁ ČÁST

7 Představení původních tras

V této kapitole budou představeny dvě okružní trasy. Tyto trasy jsou reálným případem vybrané společnosti. Obě níže uvedené trasy jsou vedeny napříč různými státy. Trasy se v průběhu let vyvíjely podle potřeb zákazníků. Společnost používá pro obě cesty jeden typ tahače, a to sice: Mercedes Benz Tahač Návěsu Actros 1845 LS. Parametry vozu jsou uvedeny v následující tabulce, tyto údaje později slouží k výpočtu nákladů na jednotlivé trasy.

Obsah motoru	12809 ccm
Výkon motoru	330 kW / 449 koní
Počet míst	2
Celková hmotnost	18 000 kg
Nosnost	9667 kg
Spotřeba	25.0l/100km
Emisní třída	Euro 6

Tabulka 3 Parametry tahače

7.1 Podmínky přepravy

Důležitým bodem pro výpočet tras jsou podmínky, které jsou stanoveny a pro obě trasy platí společně. První podmínkou je, aby řidič startoval vždy ze stejného místa (tj. depa) a do tohoto místa se poté, co projede všechny body trasy opět vrátil. Pro výpočet je potřeba dodržet dvojice sklad a zákazník. Aby nenastalo, že řidič pojede ze skladu do skladu a naopak. U obou úloh se zanedbávají kapacity skladů, kamion vyjede ze skladu vždy plný a u zákazníka zas vše vyloží. Lidský faktor je v úlohách řešen jen okrajově, a to z pohledu doby trvání každé z tras. S narůstající dobou ujetí každé z tras rostou i náklady na zaměstnance.

Jelikož jsou obě trasy mezistátní je třeba pro výpočet nákladů brát v potaz i mýtné v jednotlivých státech. Ceny mýtného pro každý jednotlivý stát uvádím v tabulce. Ceny jsou stanoveny danými státy s ohledem na parametry vozů.

Stát	ČR	SK	AUT	SUI	GER	HUN	POL
Cena/km	4,12 Kč	0,217 Eur	0,331 Eur	0,226 CHF	0,154 Eur	81,5 HUF	0,27 Eur

Tabulka 4 Ceny mýtného

7.2 První trasa

První trasa je vedena celkem přes šest různých států. Startovací depo se nachází v České republice, které je mimo jiné i skladem odkud vyráží kamion na cestu plný k prvnímu zákazníkovi. Na cestě je celkem 16 zastávek, kdy poslední je opět startovací depo. Souřadnice jednotlivých uzlů uvádím níže.

Souřadnice zastávek	Charakter zastávky
50.6444908, 13.8009325	Sklad
50.6388533, 15.2630058	Zákazník
50.072673, 14.536018	Sklad
48.294348, 16.896206	Zákazník
47.510793, 8.320976	Sklad
48.730445, 17.020783	Zákazník
46.818682, 16.847276	Sklad
46.828783, 16. 845303	Zákazník
46.366759, 17. 774400	Sklad
46.31295, 20.057976	Zákazník
51.469389, 21.152935	Sklad
49.950703, 20.255042	Zákazník
53.815211, 20.390317	Sklad
51.074286, 17.758050	Zákazník
52.300896, 21.040126	Sklad
50.08684, 19.931917	Zákazník
50.6444908, 13.8009325	Sklad

Tabulka 5 Souřadnice trasy č. 1

Pro představu uvádím mapu této trasy.

Obrázek 10 Mapa trasy č. 1

Vypočtené parametry této trasy uvádím pro přehlednost v tabulce.

Počet km	6552,5 km
Doba jízdy	69 h 31 min
Náklady	66 217, 301 Kč

Tabulka 6 Parametry trasy č. 1

7.3 Druhá trasa

Tato trasa je vedena celkem třemi státy a skládá se ze 13 zastávek. Počátek je opět v České republice, odkud vyjede kamion ze skladu k prvnímu zákazníkovi do Německa. Většina trasy se pak odehrává pouze zde. Poslední zastávkou na trase je zákazník v Lucembursku. Souřadnice trasy uvádím v tabulce.

Souřadnice	Charakter zastávky
50.6444908, 13.8009325	Sklad
50.141566, 11.806081	Zákazník
49.222743, 12.871393	Sklad
51.793328, 13.130070	zákazník
52.355983, 11.564308	Sklad

53.247054, 13.248462	zákazník
52.461132, 10.022068	Sklad
51.942722, 9.395165	zákazník
51.369468, 7.477489	Sklad
49.534607, 6.884097	zákazník
48.969155, 7.766405	Sklad
51.878718, 12.291082	zákazník
50.6444908, 13.8009325	Sklad

Tabulka 7 Parametry trasy č. 2

Pro představu jak vypadá tato trasa uvádím mapu.

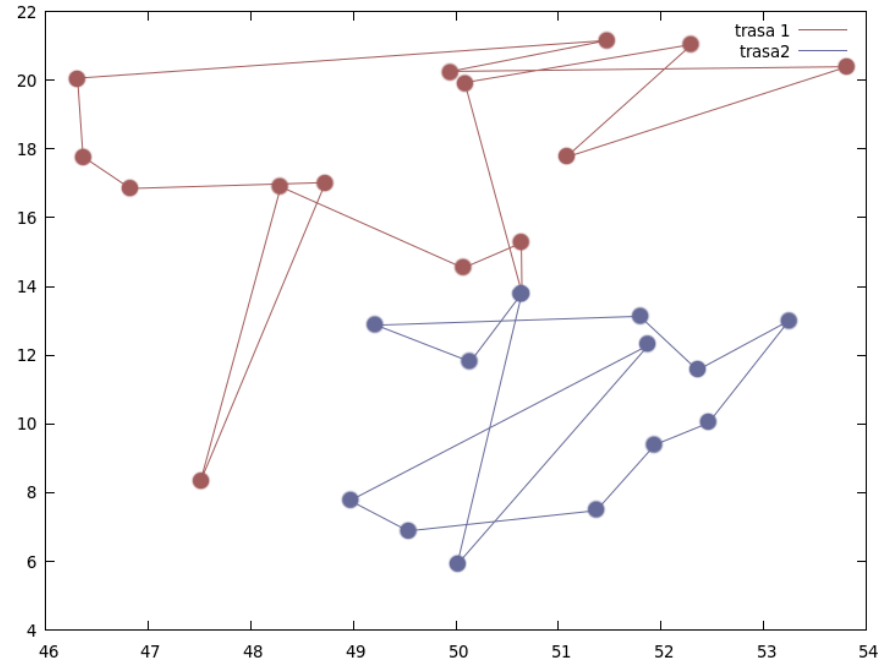
Obrázek 11 Mapa trasy č.2

Vypočtené parametry této trasy se uvádí pro přehlednost opět v tabulce.

Počet km	4 382, 2 km
Doba jízdy	47 h 07 min
Náklady	37 152,5 Kč

Tabulka 8 Parametry trasy č. 2

Pro budoucí pozorování změn tras byl vytvořen v programu gnuplot graf, který zaznamenává obě trasy.



Graf 1 Zobrazení obou tras v grafu

8 Výpočet nových tras

Tato část práce je stěžejní. Je potřeba vypočítat nové trasy, které by nesly nižší náklady. Všechny podmínky uvedeny výše však musí být zachovány. V obou případech se jedná o symetrickou úlohu TSP. Pro porovnání změn budeme počítat nové trasy původních, zadaných tras. Závěrem spojíme obě trasy v jednu a spočteme optimální trasu právě pro tuto nově vytvořenou trasu. Pro výpočty budu používat vlastní programy napsané v jazyce C++, které jsou součástí přílohy této práce. Programy jsou podrobně okomentovány a popsány, proto nejsou zmíněny zde. První program pracuje na základě algoritmu mravenčí kolonie a druhý simulovaného žíhání. Název vstupního souboru a parametry se nastavují přímo v kódu.

8.1 Matice vstupních tras

Nejprve je třeba sestavit vstupní matici pro každou z jednotlivých tras. Zde je třeba říct, že hrany (tj. trasy mezi uzly) jsou zde ohodnoceny jako náklady na cestu mezi sklady a zákazníky. Podmínka zachování principu sklad – zákazník – sklad je vyřešena v samotné matici, kde jsou na jednotlivá místa dosazeny vysoká čísla. Algoritmus prohledává nejnižší hodnoty a těmto se vyhne, právě takto zajistíme, aby řidič nejel ze skladu do skladu a naopak. Matice jednotlivých tras jsou přiloženy v příloze této práce.

8.2 Výsledky

Výsledky spočtených tras uvádím v tabulce pro každou metodu zvlášť. Nejprve uvedu pro metodu simulované žíhání, poté pro mravenčí kolonie.

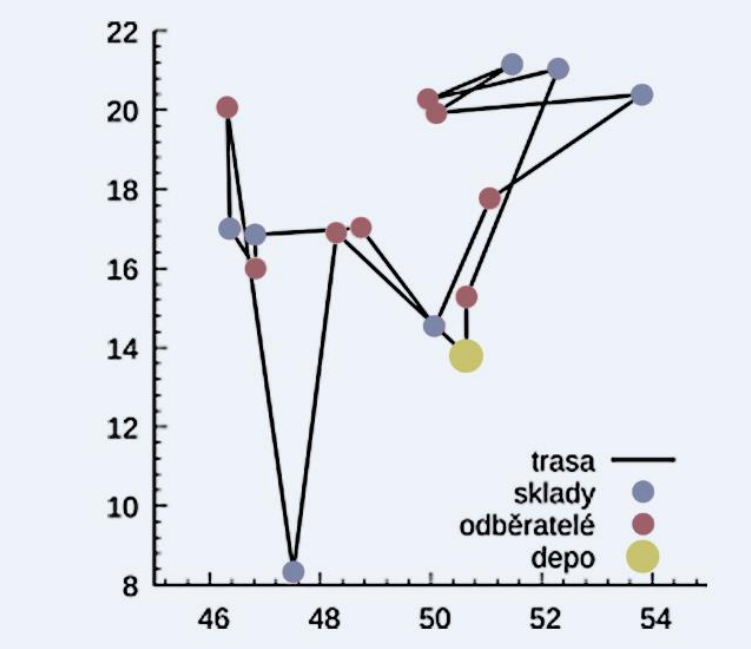
Simulované žíhání	
trasa	Cena
0 1 14 11 10 15 12 13 2 5 6 7 8 9 4 3 16	62666.6
0 3 4 5 6 7 8 11 10 9 2 1 12	32849.9
0 1 14 11 10 15 12 13 2 3 4 25 26 27 24 23 22 19 20 21 16 17 18 7 6 9 8 5 28	92455.8
Tinit=50000	
Parametr ochlazování beta=0.99999	
počet iterací:10000	

Tabulka 9 Simulované žíhání výsledky

Pro povahu úlohy je parametr ochlazování beta vyšší, než bývá zvykem. Při nastavení parametru na doporučovanou hodnotu 0,95 se výsledek zhoršil.

Délka v km	6 077 km
Náklady	62 666,6 Kč
Doba jízdy	67 h

Tabulka 10 Simulované žíhání parametry optimalizované trasy č. 1

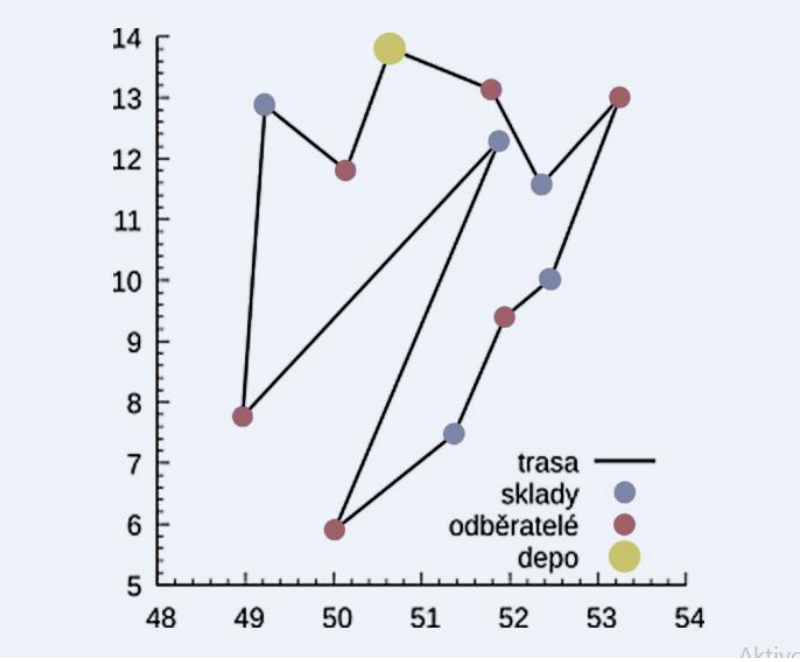


Graf 2 Simulované žíhání zoptimalizovaná trasa č. 1

Výsledky zoptimalizované trasy číslo dvě.

Vzdálenost v km	3 642 km
Náklady	32 849,9 Kč
Doba jízdy	41 h 8 min

Tabulka 11 Simulované žíhání výsledky trasy č.2

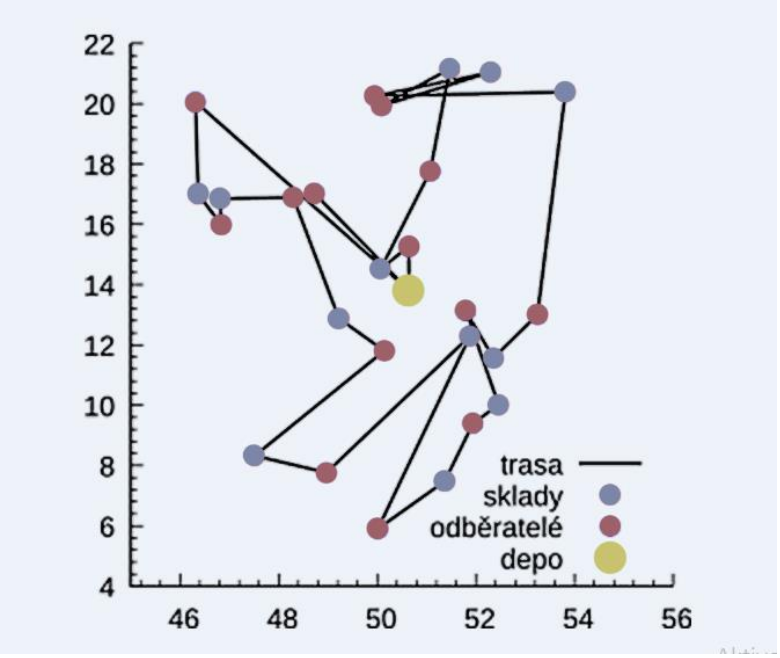


Obrázek 12 Simulované žíhání zoptimalizované trasa č.2

Výsledky pro sloučené trasy pomocí simulovaného žíhání je následující.

Vzdálenost v km	9 519 km
Náklady	92 455, 8 Kč
Doba jízdy	103 h

Tabulka 12 Simulované žíhání sloučené trasy



Graf 3 Simulované žíhání sloučené trasy

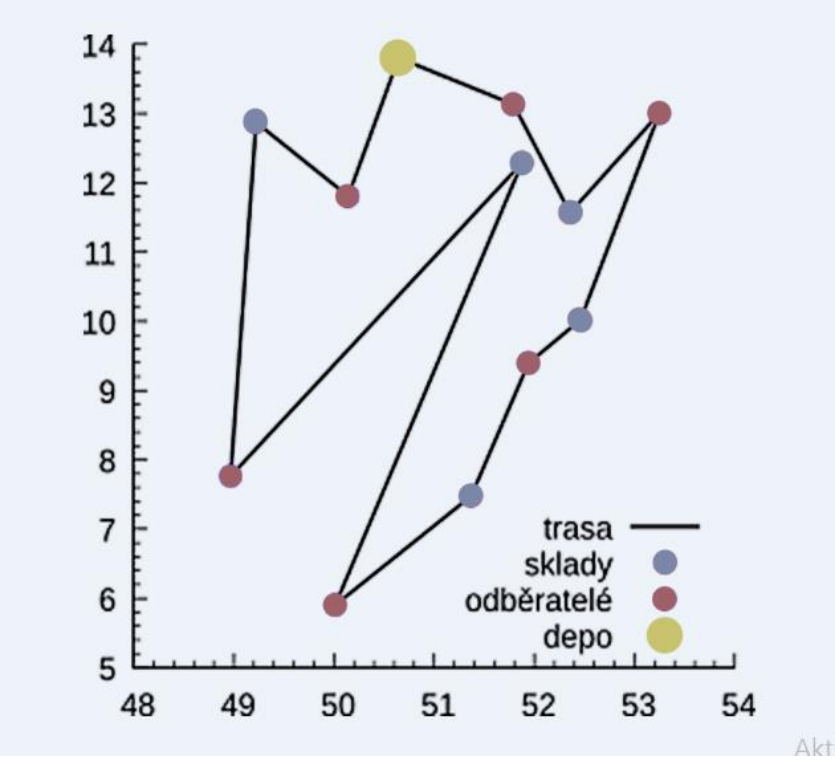
Nyní si představíme výsledky výpočtů pomocí mravenčí kolonie. Nejprve souhrnně pro všechny trasy.

Mravenčí kolonie Trasa	Cena
0 1 14 11 10 15 12 13 2 5 6 7 8 9 4 3 16	62666.6
0 1 2 9 10 11 8 7 6 5 4 3 12	32719.1
0 1 2 13 10 15 14 11 12 21 20 19 22 23 24 27 26 25 4 17 18 3 6 7 8 9 16 5 28	89349.5
počet mravenců:100	
počet iterací:10000	
parametr vypařování rho=0,1	

Když srovnáme výsledek trasy č. 1, tak zjistíme, že výsledek je totožný s předchozí metodou a není třeba se tímto dále zabývat. Přesuneme se tedy k trase číslo dvě, ta nám vyšla touto metodou, co se týče nákladů lépe než u předchozí metody.

Vzdálenost v km	3 604 km
Náklady	32 719,1 Kč
Doba jízdy	42 h 29 min

Tabulka 13 Mravenčí kolonie zoptimalizovaná trasa č.2



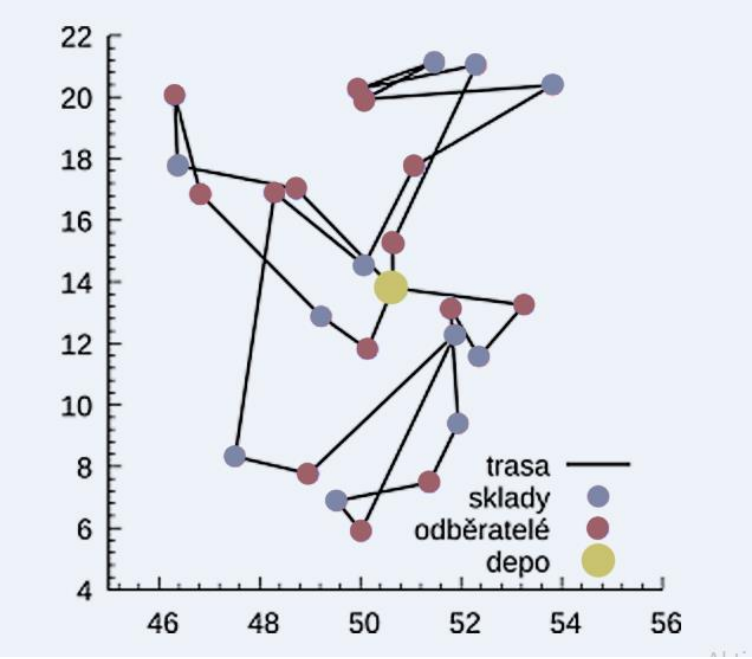
Graf 4 Mravenčí kolonie zoptimalizovaná trasa č.2

A jako poslední jsou představeny výsledky pro sloučené trasy.

Vzdálenost v km	8 920 km
Náklady	89 349,5 Kč
Doba jízdy	98 h 12 min

Tabulka 14 Mravenčí kolonie zoptimalizovaní sloučená trasa

Výsledek v grafu:



Graf 5 Mravenčí kolonie zoptimalizovaná sloučená trasa

8.3 Zhodnocení výsledků

Z hlediska úspor nákladů nám první trasa vyšla totožně u obou metod. Délka zajetých tras nepřesáhla původní hodnoty, můžeme tedy říct, že došlo skutečně k optimalizaci. Celkové úspory na tuto trasu jsou 3 551 Kč. Druhá trasa vyšla lépe pomocí metody mravenčí kolonie a to o 130 Kč, celkové úspory oproti původní trase jsou 4 432,9 Kč V případě tras sloučených byl opět lepší druhý algoritmus, a to o 3 105,5 Kč. Celkové úspory, které by mohly přinést zoptimalizované trasy při zachování systému dvou tras jsou ve výši 7 983 Kč/měsíc. Kdyby se změnila rajonizace tras a sloučily bychom je dohromady celkové měsíční úspory vůči separátním zoptimalizovaným trasám by činily 6 036,2 Kč/měsíc a 7 984,10 Kč/měsíc oproti původním trasám.

Závěr

V první části této práce byl popsán problém obchodního cestujícího, bylo představeno jeho historické pozadí, matematická formulace a modifikace úlohy, současně byly i krátce představeny aktuální světové rekordy ve výpočtech TSP. Dále byla popsána teorie grafů, která s tématem úzce souvisí. A v neposlední řadě pak optimalizační metody vedoucí k řešení tohoto problému.

Z metod popsaných v teoretické části byly vybrány dvě metody z kategorie metaheuristik, a to Simulované žíhání a Mravenčí kolonie. Tyto dva optimalizační algoritmy byly následně naprogramovány podle požadavků a podmínek příkladu v praktické části. Nejprve byly oba programy testovány na malých příkladech, které se daly snadno ověřit pomocí simplexové metody ručním výpočtem. Tímto testováním prošly úspěšně oba programy. Později už probíhaly výpočty skutečných tras vybrané společnosti.

Dále byla zkoumána kvalita poskytnutých výsledků u obou algoritmů, přičemž nejlepší výsledky poskytl algoritmus mravenčí kolonie. Až na jednu výjimku u trasy č. 1, kdy byl výsledek totožný s druhým algoritmem.

Polední část práce byla věnována zhodnocení zoptimalizovaných tras z hlediska nákladů. Kdy vznikly dvě varianty řešení, a to varianta zoptimalizovaných tras, které budou i nadále fungovat separátně, pro tento případ byla celková měsíční úspora ve výši 6 036,2 Kč. Druhou možností bylo sloučení původních dvou tras, tato sloučená trasa poskytla po optimalizaci úsporu ve výši 7 984,1 Kč.

Seznam použité literatury

1. DEVLIN, Keith J. *Problémy pro třetí tisíciletí: sedm největších nevyřešených otázek matematiky*. Přeložil Luboš PICK. Praha: Dokořán, 2005. Aliter (Argo: Dokořán). ISBN 80-7363-016-8.
2. COOK, William. *Po stopách obchodního cestujícího: matematika na hranicích možností*. Praha: Argo, 2012. Zip (Argo: Dokořán). ISBN 978-80-7363-412-4.
3. GRECO F., Traveling salesman problem. inTech, 2018. ISBN 978-953-7619-10-7.
4. APPLEGATE, David L. *The traveling salesman problem: a computational study*. Princeton: Princeton University Press, c2006. ISBN 9780691129938.
5. BONDY, J. A. a U. S. R. MURTY. *Graph theory*. New York: Springer, c2008. ISBN 978-1-84628-969-9.
6. KUČERA, Petr. *Modely teorie grafů I*. V Praze: Česká zemědělská univerzita, Provozně ekonomická fakulta, 2006. ISBN 80-213-1440-0.
7. PEARL, Judea. *Heuristics: intelligent search strategies for computer problem solving*. Reading, Mass.: Addison-Wesley Pub. Co., c1984. ISBN 0201055945.
8. REINELT, G. *The traveling salesman: computational solutions for TSP applications*. New York: Springer-Verlag, c1994. ISBN 3540583343.
9. SCHRIJVER, Alexander. *Theory of linear and integer programming*. Chichester: Wiley, c1986. ISBN 0471982326.
10. LUENBERGER, David G. a David G. LUENBERGER. *Linear and nonlinear programming*. 2nd ed. Reading, Mass.: Addison-Wesley, c1984. ISBN 0201157942.
11. File:Eulerův diagram pro P, NP, NP-úplný a NP-obtížný.png - Wikimedia Commons. [online]. Dostupné z: https://commons.wikimedia.org/wiki/File:Euler%C5%AFv_diagram_pro_P,_NP,_NP-%C3%BApln%C3%BD_a_NP-obt%C3%AD%C5%BEn%C3%BD.png

Seznam obrázků

Seznam tabulek

Seznam grafů

Příloha

	T(s)	Ž(z)	P(s)	R (z)	Š(s)	S(z)	M(s)	M(z)	M(s)	M(z)	P(s)	P(z)	P(s)	P(z)	P(s)	P(z)	T(s)
T(s)	0	1466,952	1000000	4691,48	1000000	4090,63	1000000	7073,44	1000000	8129,34	1000000	6801,42	1000000	5459,636	1000000	6687,5	1000000
Ž(z)	1466,952	0	996,29	1000000	8320,48	1000000	4030,48	1000000	5780	1000000	5469,69	1000000	5474,04	1000000	5862,55	1000000	1900,42
P(s)	1000000	996,29	0	3799,22	1000000	3026,77	1000000	5803,58	1000000	7196,89	1000000	6135,94	1000000	3459,29	100000	5767,36	1000000
R (z)	4691,48	1000000	3799,22	0	9567,24	1000000	2540,85	1000000	3409,52	1000000	6278,54	1000000	9505,7	1000000	7422,8	1000000	4699,29
Š(s)	1000000	8320,48	1000000	9567,24	0	9593,86	1000000	7143,41	1000000	12998,48	1000000	14124,22	1000000	11737,96	1000000	13454,44	1000000
S(z)	4090,63	1000000	3026,77	1000000	9593,86	0	2836,56	1000000	3981,27	1000000	6287,88	1000000	9226,73	1000000	7027,9	1000000	4180,84
M(s)	1000000	4030,48	1000000	2540,85	1000000	2836,56	0	10,15	1000000	4235,52	1000000	7624,5	1000000	8145,87	1000000	7490,34	1000000
M(z)	7073,44	1000000	5803,58	1000000	7143,41	1000000	10,15	0	1396,21	1000000	9588,69	1000000	12415,54	1000000	10167,75	1000000	7361,84
M(s)	1000000	5780	1000000	3409,52	1000000	3981,27	1000000	1396,21	0	2565,9	1000000	6645,2	1000000	9039,26	1000000	8672,13	1000000
M(z)	8129,34	1000000	7196,89	1000000	12998,48	1000000	4235,52	1000000	2565,9	0	9401,49	1000000	13602,85	1000000	12228,07	1000000	8968,96
P(s)	1000000	5474,04	1000000	6278,54	1000000	6287,88	1000000	9588,69	1000000	9401,49	0	2636,21	1000000	2491,78	1000000	1671,6	1000000
P(z)	6801,428	1000000	6135,94	1000000	14124,22	1000000	7624,5	1000000	6645,2	1000000	2636,21	0	6160	1000000	3300,16	1000000	6801,42
P(s)	1000000	5862,55	1000000	9505,7	1000000	9226,73	1000000	12,415,54	1000000	13602,85	1000000	6160	0	5309,3	1000000	5822,11	1000000
P(z)	5459,636	1000000	3459,29	1000000	11737,96	1000000	8145,87	1000000	9039,26	1000000	2491,78	1000000	5309,3	0	3688,7	1000000	4550
P(s)	1000000	1900,42	1000000	7422,8	1000000	7027,9	1000000	10167,75	1000000	12228,07	1000000	3300,16	1000000	3688,7	0	3381	1000000
P(z)	6687,5	1000000	5767,36	1000000	13454,44	1000000	7490,34	1000000	8672,13	1000000	1671,6	1000000	5822,11	1000000	3381	0	6535,8
T(s)	100000	1900,42	100000	4699,29	100000	4180,84	100000	7361,84	100000	8968,96	100000	6801,42	100000	4550	100000	6535,8	0

Matice trasy č. 1

	T(s)	N(z)	N(s)	N(z)	N(s)	N(z)	N(s)	N(z)	N(s)	N(z)	N(s)	L(z)	T(s)
T(s)	0	1950,153	1000000	2091,443	1000000	4078,164	1000000	4889,505	1000000	7487,992	1000000	8178,559	1000000
N(z)	1950,153	0	2009,811	1000000	3113,602	1000000	4203,144	1000000	5225,99	1000000	4694,224	1000000	1997,45
N(s)	1000000	2009,11	0	4825,469	1000000	6510,727	1000000	5622,217	1000000	5637,154	1000000	7137,946	1000000
N(z)	2091,443	1000000	4825,469	0	1902,513	1000000	3243,733	1000000	5845,912	1000000	7152709	1000000	2008,695
N(s)	1000000	3113,602	1000000	1902,513	0	2371,961	1000000	2326,382	1000000	6603,346	1000000	6855,882	1000000
N(z)	4078,164	1000000	6510,272	1000000	2371,961	0	3788,289	1000000	6449,697	1000000	9161,921	1000000	3790
N(s)	1000000	4203,144	1000000	3243,733	1000000	3788,289	0	871,105	1000000	5561,873	1000000	5367,022	1000000
N(z)	4889,505	1000000	5622,217	1000000	2326,382	1000000	871,105	0	2890,855	1000000	5156,118	1000000	4984,27
N(s)	1000000	5225,99	1000000	5845,912	1000000	6449,697	1000000	2890,855	0	3765,715	1000000	3531,378	1000000
N(z)	7487,992	1000000	5637,156	1000000	6603,346	1000000	5561,873	1000000	3765,715	0	2664,329	1000000	7101,119
N(s)	1000000	4694,224	1000000	7152,709	1000000	9161,921	1000000	5156,118	1000000	2664,329	0	3093,601	1000000
L(z)	8178,559	1000000	7137,946	1000000	6855,882	1000000	5367,022	1000000	3531,378	1000000	3093,601	0	8114,352
T(s)	1000000	1997,45	1000000	2008,695	1000000	3790,091	1000000	4984,27	1000000	7101,119	1000000	8114,352	0

Matice trasy č. 2

	T(s)	Ž(z)	P(s)	R (z)	Š(s)	S(z)	M(s)	M(z)	M(s)	M(z)	P(s)	P(z)	P(s)	P(z)	P(s)	P(z)	T(s)	N
T(s)	0	1466,952	1000000	4691,48	1000000	4090,63	1000000	7073,44	1000000	8129,34	1000000	6801,42	1000000	5459,636	1000000	6687,5	0	
Ž(z)	1466,952	0	996,29	1000000	8320,48	1000000	4030,48	1000000	5780	1000000	5469,69	1000000	5474,04	1000000	5862,55	1000000	1466,952	
P(s)	1000000	996,29	0	3799,22	1000000	3026,77	1000000	5803,58	1000000	7196,89	1000000	6135,94	1000000	3459,29	100000	5767,36	1000000	
R (z)	4691,48	1000000	3799,22	0	9567,24	1000000	2540,85	1000000	3409,52	1000000	6278,54	1000000	9505,7	1000000	7422,8	1000000	4691,48	
Š(s)	1000000	8320,48	1000000	9567,24	0	9593,86	1000000	7143,41	1000000	12998,48	1000000	14124,22	1000000	11737,96	1000000	13454,44	1000000	
S(z)	4090,63	1000000	3026,77	1000000	9593,86	0	2836,56	1000000	3981,27	1000000	6287,88	1000000	9226,73	1000000	7027,9	1000000	4090,63	
M(s)	1000000	4030,48	1000000	2540,85	1000000	2836,56	0	10,15	1000000	4235,52	1000000	7624,5	1000000	8145,87	1000000	7490,34	1000000	
M(z)	7073,44	1000000	5803,58	1000000	7143,41	1000000	10,15	0	1396,21	1000000	9588,69	1000000	12415,54	1000000	10167,75	1000000	7073,44	
M(s)	1000000	5780	1000000	3409,52	1000000	3981,27	1000000	1396,21	0	2565,9	1000000	6645,2	1000000	9039,26	1000000	8672,13	1000000	
M(z)	8129,34	1000000	7196,89	1000000	12998,48	1000000	4235,52	1000000	2565,9	0	9401,49	1000000	13602,85	1000000	12228,07	1000000	8129,34	
P(s)	1000000	5474,04	1000000	6278,54	1000000	6287,88	1000000	9588,69	1000000	9401,49	0	2636,21	1000000	2491,78	1000000	1671,6	1000000	
P(z)	6801,428	1000000	6135,94	1000000	14124,22	1000000	7624,5	1000000	6645,2	1000000	2636,21	0	6160	1000000	3300,16	1000000	6801,42	
P(s)	1000000	5862,55	1000000	9505,7	1000000	9226,73	1000000	12,415,54	1000000	13602,85	1000000	6160	0	5309,3	1000000	5822,11	1000000	
P(z)	5459,636	1000000	3459,29	1000000	11737,96	1000000	8145,87	1000000	9039,26	1000000	2491,78	1000000	5309,3	0	3688,7	1000000	5459,636	
P(s)	1000000	1900,42	1000000	7422,8	1000000	7027,9	1000000	10167,75	1000000	12228,07	1000000	3300,16	1000000	3688,7	0	3381	1000000	
P(z)	6687,5	1000000	5767,36	1000000	13454,44	1000000	7490,34	1000000	8672,13	1000000	1671,6	1000000	5822,11	1000000	3381	0	6687,5	
T(s)	1000000	1466,952	1000000	4691,48	1000000	4090,63	1000000	7073,44	1000000	8129,34	1000000	6801,42	1000000	5459,636	1000000	6687,5	0	
N(z)	1950,153	1000000	2444,012	1000000	5502,364	1000000	8175,295	1000000	6717,221	1000000	9495,395	1000000	10160,65	1000000	9353,333	1000000	1950,153	
N(s)	1000000	2977,478	1000000	4882,514	1000000	4961,087	1000000	5832,277	1000000	8739,454	1000000	8381,056	1000000	5353,03	1000000	8004,736	1000000	
N(z)	2091,443	1000000	2677,89	1000000	8560,071	1000000	9046,485	1000000	9915,473	1000000	7422,102	1000000	7333,055	1000000	7200,908	1000000	2091,443	
N(s)	1000000	4662,879	1000000	8085,88	1000000	7579,88	1000000	10433,59	1000000	11508,33	1000000	6134,868	1000000	8171,803	1000000	7861,833	1000000	
N(z)	4078,164	1000000	4882,514	1000000	10103,67	1000000	10737,3	1000000	11693,5	1000000	7764,146	1000000	6434,663	1000000	7068,755	1000000	4078,164	
N(s)	1000000	5792,3	1000000	9242,595	1000000	8392,595	1000000	11175,65	1000000	12789,83	1000000	9970,421	1000000	7666,091	1000000	9515,16	1000000	
N(z)	4889,505	1000000	5603,54	1000000	7674,363	1000000	11497,3	1000000	12657,32	1000000	9960,401	1000000	10147,97	1000000	9624,61	1000000	4889,505	
N(s)	1000000	7451,966	1000000	10871,94	1000000	10724,44	1000000	11695,35	1000000	14768,39	1000000	11985,74	1000000	9681,405	1000000	11609,41	1000000	
N(z)	7487,992	1000000	7233,343	1000000	4200,635	1000000	11601,6	1000000	12502,66	1000000	14388,76	1000000	13671,58	1000000	14500,61	1000000	7487,992	
N(s)	1000000	7570,976	1000000	9754,922	1000000	9752,166	1000000	10497	1000000	13355	1000000	13012,69	1000000	10740,53	1000000	12833,95	1000000	
L(z)	8178,559	1000000	9228,353	1000000	5533,701	1000000	12774,67	1000000	14149,92	1000000	15256,46	1000000	15289,92	1000000	14549,22	1000000	8178,559	
T(s)	1000000	1466,952	1000000	4691,48	1000000	4090,63	1000000	7073,44	1000000	8129,34	1000000	6801,42	1000000	5459,636	1000000	6687,5	1000000	

Matice trasy č. 3 společných tras 1. část

N(z)	N(s)	N(z)	N(s)	N(z)	N(s)	N(z)	N(s)	N(z)	N(s)	L(z)	T(s)
1950,153	1000000	2091,443	1000000	4078,164	1000000	4889,505	1000000	7487,992	1000000	8178,559	1000000
1000000	2977,478	1000000	4662,879	1000000	5792,3	1000000	7451,966	1000000	7570,976	1000000	1466,952
2444,012	1000000	2677,89	1000000	4882,514	1000000	5603,54	1000000	7233,343	1000000	9228,353	1000000
1000000	4882,514	1000000	8085,88	1000000	9242,595	1000000	10871,94	1000000	9754,922	1000000	4691,48
5502,364	1000000	8560,071	1000000	10103,67	1000000	7674,363	1000000	4200,635	1000000	5533,701	1000000
1000000	4961,087	1000000	7579,88	1000000	8392,595	1000000	10724,44	1000000	9752,166	1000000	4090,63
8175,295	1000000	9046,485	1000000	10737,3	1000000	11497,3	1000000	11601,6	1000000	12774,67	1000000
1000000	5832,277	1000000	10433,59	1000000	11175,65	1000000	11695,35	1000000	10497	1000000	7073,44
6717,221	1000000	9915,473	1000000	11693,5	1000000	12657,32	1000000	12502,66	1000000	14149,92	1000000
1000000	8739,454	1000000	11508,33	1000000	12789,83	1000000	14768,39	1000000	13355	1000000	8129,34
9495,395	1000000	7422,102	1000000	7764,146	1000000	9960,401	1000000	14388,76	1000000	15256,46	1000000
1000000	8381,056	1000000	6134,868	1000000	9970,421	1000000	11985,74	1000000	13012,69	1000000	6801,42
10160,65	1000000	7333,055	1000000	6434,663	1000000	10147,97	1000000	13671,58	1000000	15289,92	1000000
1000000	5353,03	1000000	8171,803	1000000	7666,091	1000000	9681,405	1000000	10740,53	1000000	5459,636
9353,333	1000000	7200,908	1000000	7068,755	1000000	9624,61	1000000	14500,61	1000000	14549,22	1000000
1000000	8004,736	1000000	7861,833	1000000	9515,16	1000000	11609,41	1000000	12833,95	1000000	6687,5
1950,153	1000000	2091,443	1000000	4078,164	1000000	4889,505	1000000	7487,992	1000000	8178,559	1000000
0	2009,811	1000000	3113,602	1000000	4203,144	1000000	5225,99	1000000	4694,224	1000000	1997,45
2009,11	0	4825,469	1000000	6510,727	1000000	5622,217	1000000	5637,154	1000000	7137,946	1000000
1000000	4825,469	0	1902,513	1000000	3243,733	1000000	5845,912	1000000	7152709	1000000	2008,695
3113,602	1000000	1902,513	0	2371,961	1000000	2326,382	1000000	6603,346	1000000	6855,882	1000000
1000000	6510,272	1000000	2371,961	0	3788,289	1000000	6449,697	1000000	9161,921	1000000	3790
4203,144	1000000	3243,733	1000000	3788,289	0	871,105	1000000	5561,873	1000000	5367,022	1000000
1000000	5622,217	1000000	2326,382	1000000	871,105	0	2890,855	1000000	5156,118	1000000	4984,27
5225,99	1000000	5845,912	1000000	6449,697	1000000	2890,855	0	3765,715	1000000	3531,378	1000000
1000000	5637,156	1000000	6603,346	1000000	5561,873	1000000	3765,715	0	2664,329	1000000	7101,119
4694,224	1000000	7152,709	1000000	9161,921	1000000	5156,118	1000000	2664,329	0	3093,601	1000000
1000000	7137,946	1000000	6855,882	1000000	5367,022	1000000	3531,378	1000000	3093,601	0	8114,352
1997,45	1000000	2008,695	1000000	3790,091	1000000	4984,27	1000000	7101,119	1000000	8114,352	0

Matice trasy č. 4společných tras 2. část

Skript mravenčí kolonie

```
#include <iostream>
#include <fstream>
#include <vector>
#include <stdlib.h>
#include <math.h>
#include <boost/bind/bind.hpp>
#include <algorithm>
#include <time.h>
#include <boost/algorithm/string.hpp>
#include <boost/lexical_cast.hpp>
#include <boost/algorithm/string/trim.hpp>

using namespace std;

struct Ant { //definice struktury mravenec.
  vector<int> path; //kazdy mravenec zna trasu, kterou prosel
  int node; //a vi, ve kterem meste se prave nachazi
} ;

struct Anthill { //definice struktury maveniste
  vector<vector<double> > feromones; //mraveniste obsahuje
informaci o intenzite feromonu na vsech cestach (matice)
  vector<vector<double> > distances; //informace o vsech
vzdalenostech (hodnotach cest mezi mesty -> matice)
  vector<double> nodeWeight; //informace o vahach vsech mest, vaha
se meni podle toho, kolikrat bylo mesto mravenci navstiveno
  vector<vector<double> > changeOfFeromones; // informace o tom, o
jakou hodnotu se feromonova stopa v dane iteraci metody zmeni
} ;

vector<vector<double> > generateNodes(int nodeNumber){ //generuj
nahodne souradnice mest
    vector<vector<double> > nodes(nodeNumber,vector<double>(2));
    for (int i=0;i<nodeNumber;i++){
        nodes[i][0]=rand()/double(RAND_MAX);
        nodes[i][1]=rand()/double(RAND_MAX);
    }
    return nodes;
}

vector<vector<double> > loadNodes(const char *filename,int
nodeNumber){ //nacti mesta jako matici ze souboru filename o poctu
mest "nodenumber"
    vector<vector<double> >
nodes(nodeNumber,vector<double>(nodeNumber));

    ifstream in(filename);
    int i=0;
    vector<string> strs;
    std::string line,s;
    getline( in, line );
    for( line; getline( in, line ); )
    {
        boost::split(strs,line,boost::is_any_of(";"));
```

```
        for(int j=1;j<nodeNumber+1;j++){
            s=strs[j];boost::trim(s);
            nodes[i][j-1]=boost::lexical_cast<double>(s);
        }
        i++;
    }

    return nodes;
}

Anthill generateAnthill(int nodeNumber,vector<vector<double> >
nodes,double alpha, double beta){ //vygeneruj mraveniste
    Anthill anthill; //vytvoreni noveho mraveniste
    anthill.feromones=vector<vector<double>
>(nodeNumber,vector<double>(nodeNumber)); //vytvoreni prazdnych
matic s vlastnostmi mraveniste
    anthill.distances=vector<vector<double>
>(nodeNumber,vector<double>(nodeNumber));
    anthill.changeOfFeromones=vector<vector<double>
>(nodeNumber,vector<double>(nodeNumber));
    anthill.nodeWeight=vector<double>(nodeNumber);
    double value;//pomocna promenna
    for (int i=0;i<nodeNumber;i++){ //prochazej v cyklu pres
vsechny dvojice mest
        for (int j=i+1;j<nodeNumber;j++){
            value=rand()/double(RAND_MAX); //vygeneruj nahodnou
hodnotu feromonu na ceste z i do j a z j do i (stejna trasa)
            anthill.feromones[i][j]=value;
            anthill.feromones[j][i]=value; //a uloz hodnotu do
mraveniste
            /*rij[0]=nodes[j][0]-nodes[i][0];
            rij[1]=nodes[j][1]-nodes[i][1];
            value=sqrt(rij[0]*rij[0]+rij[1]*rij[1]);*/
            value=nodes[i][j]; //do hodnoty value nastav cenu
trasy z i do j
            anthill.distances[i][j]=1.0/value; //hodnota prechodu
z i do j je obracena hodnota teto ceny
            anthill.distances[j][i]=1.0/value;
            anthill.changeOfFeromones[i][j]=0; //na zacatku
nedochazi ke zmenam feromonu
            anthill.changeOfFeromones[j][i]=0;
        }
    }

    return anthill;
}


vector<Ant> placaAnts(int antsNumber,int nodeNumber){ //umisti
zadany pocet mravencu na start (mesto s indexem 0)
    vector<Ant> ants(antsNumber);
    for(int i=0;i<antsNumber;i++)
ants[i].node=0;//rand()%nodeNumber;
    return ants;
}
```

```
double pathLength(vector<vector<double> > &nodes,vector<int>
&path){ //vypocet euklidovske vzdalenosti
    double rij[2];
    double length=0;
    for(int i=0;i<nodes.size();i++){
        if(i==0){
            rij[0]=nodes[path[i]][0]-nodes[path[path.size()-
1]][0];
            rij[1]=nodes[path[i]][1]-nodes[path[path.size()-
1]][1];
            length+=sqrt(rij[0]*rij[0]+rij[1]*rij[1]);
        }else {
            rij[0]=nodes[path[i]][0]-nodes[path[i-1]][0];
            rij[1]=nodes[path[i]][1]-nodes[path[i-1]][1];
            length+=sqrt(rij[0]*rij[0]+rij[1]*rij[1]);
        }
    }
    return length;
}

double pathWeight(vector<vector<double> > &nodes,vector<int>
&path){ //vypocet ceny nalezene trasy
    double length=0;
    for(int i=1;i<nodes.size();i++){
        length+=nodes[path[i]][path[i-1]];
    }
    return length;
}


void move(vector<Ant> &ants, Anthill
&anthill,vector<vector<double> > &nodes,int steps,double alpha,
double beta){ //pohyb mravencu mezi mesty
    double g;
    double criterion;
    double length;
    double nodeWeight;
    for(int i=0;i<ants.size();i++){ //kazdemu mravenci
        ants[i].path=vector<int>(steps); //nastav novou prazdnou
trasu
        for(int j=0;j<steps;j++)ants[i].path[j]=0; //nastav ji na
nulu
        ants[i].path[0]=ants[i].node; //pocetecni mesto nastav na
start (index 0)
        ants[i].path[steps-1]=nodes.size()-1; //cilove mesto
nastav na index cile
        for(int s=1;s<steps-1;s++){ //projdi vsemi mesty mimo
startu a cile
            nodeWeight=0; //vaha aktualniho mesta je nula
            for(int j=0;j<nodes.size();j++){ //spocitej vahu dle
vzorce podle feromonovych stop a hodnot cest do vsech ostatnich
mest
                if(find(ants[i].path.begin(),ants[i].path.end(),
                                                    j) ==
ants[i].path.end()){//pokud je mesto s indexem "j" na trase
mravence
```

```
nodeWeight+=pow(anthill.feromones[ants[i].node][j],alpha)*

pow(anthill.distances[ants[i].node][j],beta); //spocitej vahu
                }
            }
            g=rand()/double(RAND_MAX); //vygeneruj nahodne cistlo
            criterion=0;
            for(int j=0;j<nodes.size();j++){ //spocti kriterium
prechodu do noveho mesta
                if(find(ants[i].path.begin(),ants[i].path.end(),
                                j) != ants[i].path.end())continue;
//pokud toto mesto jeste nebylo mravencem navstiveno

criterion+=((pow(anthill.feromones[ants[i].node][j],alpha)*pow(ant
hill.distances[ants[i].node][j],beta))/
                    nodeWeight); //vypocet kriteria prechodu
                if(g<=criterion){ //pokud je nahodne cislo mensi
nez toto kriterium
                    ants[i].path[s]=j; // pridej mesto do trasy
mravence
                    ants[i].node=j; //premisti mravence do tohoto
mesta
                    break; //konec cyklu
                }
            }
        }
        length=pathWeight(nodes,ants[i].path); //spocitej cenu
mravencovi aktualni trasy

        for(int s=0;s<steps-1;s++){ //spocitej zmenu feromonove
stopy

anthill.changeOfFeromones[ants[i].path[s]][ants[i].path[s+1]]+=1.0
/length;

anthill.changeOfFeromones[ants[i].path[s+1]][ants[i].path[s]]+=1.0
/length;
        }


anthill.changeOfFeromones[ants[i].path[ants[i].path.size()-
1]][0]+=1.0/length;

anthill.changeOfFeromones[0][ants[i].path[ants[i].path.size()-
1]]+=1.0/length;
    }
}


void elitism(vector<Ant> &ants, Anthill
&anthill,vector<vector<double> > &nodes,int steps,double alpha,
double beta){
    //mame-li elitniho mravence (mravenec s nejlepsi trasou z
minule iterace), nastavime feromonove stopy, abychom tuto cestu
zvyhodnili pro dalsi generaci mravencu
```

```
    double length;
    for(int i=0;i<ants.size();i++){
        length=pathWeight(nodes,ants[i].path);
        for(int s=0;s<steps-1;s++){

anthill.changeOfFeromones[ants[i].path[s]][ants[i].path[s+1]]+=1.0
/length;

anthill.changeOfFeromones[ants[i].path[s+1]][ants[i].path[s]]+=1.0
/length;
        }


anthill.changeOfFeromones[ants[i].path[ants[i].path.size()-
1]][0]+=1.0/length;

anthill.changeOfFeromones[0][ants[i].path[ants[i].path.size()-
1]]+=1.0/length;
    }
}


void checkAnthill(Anthill &anthill){ //tato funkce aktualizuje
feromonove stopy
    for(int i=0;i<anthill.feromones.size();i++){
        for(int j=i+1;j<anthill.feromones.size();j++){

anthill.feromones[i][j]+=anthill.changeOfFeromones[i][j];

anthill.feromones[j][i]+=anthill.changeOfFeromones[j][i];
            anthill.changeOfFeromones[j][i]=0;
            anthill.changeOfFeromones[i][j]=0;
        }
    }
}
void evaporatePheromone(Anthill &anthill,double rho){ //tato
funkce snizuje intenzitu feromonove stopy podle parametru
vyparovani rho
    for(int i=0;i<anthill.feromones.size();i++){
        for(int j=i+1;j<anthill.feromones.size();j++){
            anthill.feromones[i][j]=(1.0-
rho)*anthill.feromones[i][j];
            anthill.feromones[j][i]=anthill.feromones[i][j];
        }
    }
}

void writePath(vector<vector<double> > &nodes,vector<int> &path){
    ofstream fout("cesta.dat");
    for(int i=0;i<path.size();i++){
        fout<<nodes[path[i]][0]<<" "<<nodes[path[i]][1]<<endl;
    }
    fout<<nodes[path[0]][0]<<" "<<nodes[path[0]][1]<<endl;
    fout.close();
}
```

```
Ant findBest(vector<vector<double> > &nodes,vector<Ant>ants){
//nalezeni nejlepĹˇĂ-ho mravence z celeho mraveniĹˇtÄ› (hledani
vyvoleneho)
    double minPath=MAXFLOAT; //minimalni trasa je maximalni cislo
    int theChosenOneIndex; //priprav si index na nalezeni
vyvoleneho
    double value;
    for(int i=0;i<ants.size();i++){ //projdi pres vsechny mravence
        value=pathWeight(nodes,ants[i].path); //ohodnot jejich
trasy
        if(value<minPath){ //je-li mravencova cesta lepsi nez
aktualni rekord (minimum)
            minPath=value; // uloz ji jako rekord (nove minimum)
            theChosenOneIndex=i; // a oznac tohoto mravence jako
vyvoleneho, dokud se nenajde nekdo lepsi
        }
    }
    return ants[theChosenOneIndex]; //vrat vyvoleneho
}


int main()
{
    srand( time( NULL )); //nahodna cisla se pocitaji od
aktualniho casu systemu, aby dva ruzne behy programu nebyly
totozne
    int nodeNumber=29; //pocet mest
    int antNumber=100; //pocet mravencu
    int eliteNumber=1; //pocet vyvolenych v kazde iteraci ->
VYVOLENY JE JEN JEDEN!!!  a nebo nikdo ...
    int iter=0;
    double alpha=1; //parametry mraveniste pro vypocet vahy mest
    double beta=1; //parametry mraveniste pro vypocet vahy mest
    double rho=0.1; //parametry mraveniste pro vyparovani feromonu

    //vector<vector<double> > nodes=generateNodes(nodeNumber);
    vector<vector<double> >
nodes=loadNodes("./trasa_1+2.csv",nodeNumber); //nacteni matice
mest
    Anthill anthill=generateAnthill(nodeNumber,nodes,alpha,beta);
//generovani mraveniste
    vector<Ant> ants,eliteAnts(eliteNumber); //priprava mravencu a
vyvoleneho mravence
    ofstream fout("cena.dat"); //otevreni souboru pro zapis ceny
aktualni trasy
    Ant theChosenOne;

    while(iter<10000){ //dokud neprobehne 10 000 iteraci
        cout<<"iterace: "<<iter<<endl;
        ants=placaAnts(antNumber-eliteNumber,nodeNumber); //umisti
mravence
        move(ants,anthill,nodes,nodeNumber,alpha,beta); //nech je
chodit mezi mesty
        if(iter>1 && eliteNumber>0){ // hledame-li vyvoleneho a
nejsem v prvni iteraci
```

```
            eliteAnts[0]=theChosenOne; //oznac vyvoleneho z minule
iterace

elitism(eliteAnts,anthill,nodes,nodeNumber,alpha,beta); //uprav
mraveniste dle jeho potreb
            for(int i=0;i<eliteNumber;i++)ants[antNumber-
eliteNumber-1+i]=eliteAnts[i]; //a pridej ho mezi obycejne
mravence z nove iterace
        }
        checkAnthill(anthill); //uprav mraveniste
        evaporatePheromone(anthill,rho); //vypareni feromonu
       // if(iter%100==0){
            theChosenOne=findBest(nodes,ants); //nalezeni
vyvoleneho
            fout<<iter<<"
"<<pathWeight(nodes,theChosenOne.path)<<endl; //vypis do souboru
aktualni cenu trasy
        //}
        iter++;
    }
    theChosenOne=findBest(nodes,ants); //nalezeni finalniho
vyvoleneho
    cout<<"cena trasy:
"<<pathWeight(nodes,theChosenOne.path)<<endl; //vypis ceny jeho
trasy na obrazovku
    for(int i=0;i<nodeNumber;i++) //vypis seznam mest na nejlepsi
trase
        cout<<theChosenOne.path[i]<<" ";
    cout<<endl;
    fout.close();

    return 0;
}
```

```
Skript Simulované žíhání
#include <iostream>  //naÄŤtenÄ- potĹ™ebnÄ´ch c++ knihoven
#include <fstream>
#include <vector>
#include <stdlib.h>
#include <math.h>
#include <boost/algorithm/string.hpp>
#include <boost/lexical_cast.hpp>
#include <boost/algorithm/string/trim.hpp>

using namespace std;

vector<vector<double> > loadNodes(const char *filename,int
nodeNumber){
    //nacteni matice vstupnich dat ze souboru "filename". Matice
ma rozmer nodeNumber x nodeNumber
    vector<vector<double> >
nodes(nodeNumber,vector<double>(nodeNumber)); //definice prazdne
matice (vektor vektorĹŻ)

    ifstream in(filename); //otevreni- souboru
    int i=0; //citac poctu radku
    vector<string> strs; //vektor retezcu pro docasne ulozeni
prvku v radku
    std::string line,s; //definice promennych pro cely radek a
pomocna promenna pro jednotlive retezce
    getline( in, line ); //nacteni prvniho radku (nazvy mest)
    for( line; getline( in, line ); ) //v cyklu nacitej radky az
do konce souboru
    {
        boost::split(strs,line,boost::is_any_of(";")); //nacteny
radek rozdel na jednotlive retezce. Jako separator pouzij ";"
        for(int j=1;j<nodeNumber+1;j++){ // v cyklu prochazej
jednotlive retezce vznikle z predchoziho prikazu. Prvni (j=0) se
vynechava, je to totiz nazev mesta
            s=strs[j];boost::trim(s); //do pomocne promenne "s" se
uklada j-ty retezec z radku, prikazem "trim" se pak odstrani bile
znaky (mezery a zalomeni radku)
            nodes[i][j-1]=boost::lexical_cast<double>(s); //
pomoci lexical cast se retezec prevede na desetinne cislo (double)
a ulozi se do matice
        }
```

```
        i++; //citac radku se zvysuje o jednicku
    }

    return nodes; //fukce loadNodes vraci- matici "nodes"
}


vector<vector<double> > generateNodes(int nodeNumber){ //nahodne
generovani matice mest jako vektor 2D souradnic
    vector<vector<double> > nodes(nodeNumber,vector<double>(2));
    for (int i=0;i<nodeNumber;i++){
        nodes[i][0]=rand()/double(RAND_MAX);
        nodes[i][1]=rand()/double(RAND_MAX);
    }
    return nodes;
}
vector<int> generatePath(int nodeNumber){
    vector<int> path(nodeNumber);

    for(int i=0;i<nodeNumber;i++)path[i]=i; //pocatecni cesta se
nastavi na: 0,1,2,3,...,nodeNumber-1

    return path; //vytvorena cesta se vraci do hlavniho programu
}

double pathLength(vector<vector<double> > &nodes,vector<int>
&path){ //vypocet euklidovske vzdalenosti mezi mesti pri zadane
ceste
    double rij[2];
    double length=0;
    for(int i=0;i<nodes.size();i++){
        if(i==0){
            rij[0]=nodes[path[i]][0]-nodes[path[path.size()-
1]][0];
            rij[1]=nodes[path[i]][1]-nodes[path[path.size()-
1]][1];
            length+=sqrt(rij[0]*rij[0]+rij[1]*rij[1]);
        }else {
            rij[0]=nodes[path[i]][0]-nodes[path[i-1]][0];
            rij[1]=nodes[path[i]][1]-nodes[path[i-1]][1];
            length+=sqrt(rij[0]*rij[0]+rij[1]*rij[1]);
        }
    }
    return length;
}

double pathWeight(vector<vector<double> > &nodes,vector<int>
&path){ //vypocet ceny cesty pri zadane ceste mezi mesty
    double length=0;
    for(int i=1;i<nodes.size();i++){ //v cyklu pricitej cenu mezi
jednotlivymi mesty na zadane ceste
        length+=nodes[path[i]][path[i-1]];
    }
    return length;
}
```

```
vector<int> changePath(vector<int> &path){ //nahodna zmena
aktualni trasy
    int index1=rand()%path.size()/2; //vygeneruj nahodny index
mesta z prvni poloviny cesty (udava, kolik mest se bude
prohazovat)
    int index2=rand()%path.size(); //vygeneruj druhy nahodny index
mesta z cele cesty (index prvniho mesta k prohozeni na ceste)
    int index3=index2+index1-1; //treti index je soucet
predchozich -1 (index druheho mesta k prohozeni na ceste)
    int index,indexi; //pomocne promenne
    int tmp;
    vector<int> path2=path; //vytvor novou cestu, nastav ji na
stejne hodnoty, jake ma ta puvodni
    for (int i=0;i<(index1)/2;i++){ //v cyklu prochazej prvni
        if(index3>=path2.size()) //podminka -> pokud je index3
vetsi nebo roven poctu prvku v ceste, je zmensen o tuto velikost
            index=index3-path2.size();
        else
            index=index3; //jinak nech index3 tak jak je a uloz ho
do promenne index
        if(index3<0)index=index3+path2.size(); //pokud je index
zaporny, pricty pocet mest na ceste, aby byl index v rozumnem
rozmezi hodnot
        if(index2>=path2.size()) //tataz kontrola pro druhy index
            indexi=index2-path2.size();
        else
            indexi=index2;
        tmp=path2[indexi]; //nasledujici tri radky prohazuji mesta
na indexech "index" a "indexi"
        path2[indexi]=path2[index];
        path2[index]=tmp;
        index2+=1; //pro pristi iteraci cyklu zmen hodnoty index2
a index3
        index3-=1;
    }
    return path2;
}

vector<int> changePath2(vector<int> &path){ //Zjednodusena verze
predchoziho. V teto funkci se prohazuji pouze dve mesta
    int index1=rand()%path.size()/2; //nahodne vygeneruj indexy
dvou mest
    int index2=rand()%path.size()/2; //kolikate mesto od mesta s
index1 se bude zamenovat
    int index3=index2+index1;
    if(index3>=path.size()-1)index3=path.size()-2; //kontrola zda
indexy nejsou nejsou mimo cestu a zaroven neukazuji na start a
cil(ty musi zustat nezmeneny)
    if(index3==0)index3=1;
    if(index1==0) index1=1;
    int tmp;
    vector<int> path2=path;//zkopiruj cestu
    tmp=path2[index3]; //prohod mesta
    path2[index3]=path2[index1];//prohod mesta
    path2[index1]=tmp;//prohod mesta
```

```
    return path2; //vrat novou cestu
}


void writePath(vector<vector<double> > &nodes,vector<int> &path){
//zapis souradnic cesty
    ofstream fout("cesta.dat");
    for(int i=0;i<path.size();i++){
        fout<<nodes[path[i]][0]<<" "<<nodes[path[i]][1]<<endl;
    }
    fout<<nodes[path[0]][0]<<" "<<nodes[path[0]][1]<<endl;
    fout.close();
}

int main()
{
    int nodeNumber=29; //pocet mest (velikost matice)
    int iter=0; //citar iteraci
    double length1,length2;
    double Tnew,Told,beta,deltaE,g;
    Tnew=50000; //pocatecni teplota zihani
    beta=0.99999; //parametr ochlazovani

    //vector<vector<double> > nodes=generateNodes(nodeNumber);
    vector<vector<double> >
nodes=loadNodes("./trasa_1+2.csv",nodeNumber); //nacti mesta
    vector<int> path=generatePath(nodeNumber); //nastav pocatecni
trasu
    vector<int> path2; //priprava nove cesty
    length1=pathWeight(nodes,path);//spocitej cenu teto trasy

    ofstream fout("cena.dat"); //otevri soubor delka, do kterĂ©
budes zapisovat aktualni cenu trasy

    cout<<"pocatecni - cesta s delkou: "<<length1<<endl; //vypis
na obrazovku -> pocatecni cena
    for(int i=0;i<nodeNumber;i++)cout<<path[i]<<" "; //vypis
indexu mest na pocatecni trase
    cout<<endl; //zalomeni radku u vypisu na obrazovku
    fout<<iter<<" "<<length1<<endl; //zapis ceny od souboru

    iter=1;
    while(Tnew>0.001 && iter<10000){ //dokud teplota neklesne pod
T=0.001 nebo pocet iteraci nepĹ™ekona 10 000
        path2=changePath2(path); //nahodne zmen trasu
        //length2=pathLength(nodes,path2);
        length2=pathWeight(nodes,path2); //ohodnot novou trasu
        deltaE=length2-length1; //spocitej zmenu hodnoty trasy
        g=rand()/double(RAND_MAX); //vygeneruj nahodne cisto
        if(g<exp(-deltaE/Tnew)){ //je-li nahodne cislo mensi nez
pravdepodobnost udana boltzmanovym faktorem
            path=path2; //prijmi cestu
            Told=Tnew; //uloz teplotu
            Tnew=beta*Told; //zmen teplotu
            length1=length2; //uloz cenu trasy
        }
```

```
        iter++; //zvys pocet iteraci o jednicku
        if(iter%100==0){ //jednou za sto iteraci zapis
            //writePath(nodes,path);
            fout<<iter<<" "<<Told<<" "<<length1<<endl; //cenu
trasy
        }
    }
    cout<<"vysledny cesta s delkou: "<<length2<<endl; //na
obrazovku vypis cenu nalezene trasy
    for(int i=0;i<nodeNumber;i++)cout<<path[i]<<" "; //a seznam
mest na trase
    cout<<endl;
    fout.close();

    return 0;
```

Evidence výpůjček

Prohlášení:

Dávám svolení k půjčování této bakalářské práce. Uživatel potvrzuje svým podpisem, že bude tuto práci řádně citovat v seznamu použité literatury.

Jméno a příjmení: Adéla Hojdarová

V Praze dne: 16. 05. 2019 Podpis:

Jméno	**Oddělení/ Pracoviště**	**Datum**	**Podpis**
