UNIVERZITA PALACKÉHO V OLOMOUCI
PŘÍRODOVĚDECKÁ FAKULTA
KATEDRA MATEMATICKÉ ANALÝZY A APLIKACÍ MATEMATIKY

DIPLOMOVÁ PRÁCE

Optimalizace postupu napojování komponent v rozpadlé silniční síti

Vedoucí diplomové práce:	**Rostislav Vodák, RNDr., Ph.D.**
Vypracoval:	**Michal Trněný, Bc.**
Studijní program:	N1101 Matematika
Studijní obor:	Matematika a její aplikace
Forma studia:	prezenční
Rok odevzdání:	2023

BIBLIOGRAFICKÁ IDENTIFIKACE

Autor: Bc. Michal Trněný
Název práce: Optimalizace postupu napojování komponent v rozpadlé silniční síti
Typ práce: Diplomová práce
Pracoviště: Katedra matematické analýzy a aplikací matematiky
Vedoucí práce: RNDr. Rostislav Vodák, Ph.D.
Rok obhajoby práce: 2023
Abstrakt: Optimalizace postupu napojování komponent v rozpadlé silniční síti patří do třídy NP-úplných úloh, pro které nejsou známy deterministické algoritmy, které by libovolnou instanci dokázaly vyřešit v čase, který je polynomiální funkcí velikosti úlohy. Proto používáme stochastické algoritmy. V práci je k řešení úlohy využit Max-Min Ant System, algoritmus ze třídy ACO metaheuristik. Ke konkrétní rovinné síti, rozmístění stanic, blokací a dob jejich oprav hledáme optimální kombinaci parametrů, pro niž Max-Min Ant System funguje nejlépe. Zjišťujeme, zda optimalita parametrů zůstává zachována i pro jiné situace na stejné síti.
Klíčová slova: silniční síť, komponenty souvislosti, napojování, optimalizace, mravenčí kolonie, Max-Min Ant System, kostra, složitost
Počet stran: 68
Počet příloh: 1
Jazyk: český

BIBLIOGRAPHICAL IDENTIFICATION

Author: Bc. Michal Trněný
Title: Optimization of the process of reconstruction of broken road networks
Type of thesis: Master's
Department: Department of Mathematical Analysis and Application of Mathematics
Supervisor: RNDr. Rostislav Vodák, Ph.D.
The year of presentation: 2023
Abstract: The optimization of the process of reconnection of broken road network belongs to the class of NP-complete problems for which there are no known deterministic algorithms that can solve any instance in time, which is a polynomial function of the problem size. Therefore, we use stochastic algorithms. In the thesis, the Max-Min Ant System, an algorithm from the ACO class of metaheuristics, is used to solve the problem. Given a specific planar network, a distribution of stations, blockages and their repair times, we search for the optimal combination of parameters for which Max-Min Ant System performs best. We check whether the optimality of the parameters is preserved for other situations on the same network.
Key words: road network, connected components, reconnection, optimization, ant colony, Max-Min Ant System, spanning tree, complexity
Number of pages: 68
Number of appendices: 1
Language: Czech

Děkuji svému vedoucímu za to, že mi za práci neplatil, protože kdyby mi platil, nedovolil bych si bavit se úvahami o Úloze tak moc a prostě bych ji vyřešil. To by však nebyla matematika, ale jen její aplikace. Poděkování vedoucímu patří také za příležitost řešit zajímavou úlohu, za pročtení textu této práce a za upozornění na chyby a na nesrozumitelné pasáže.

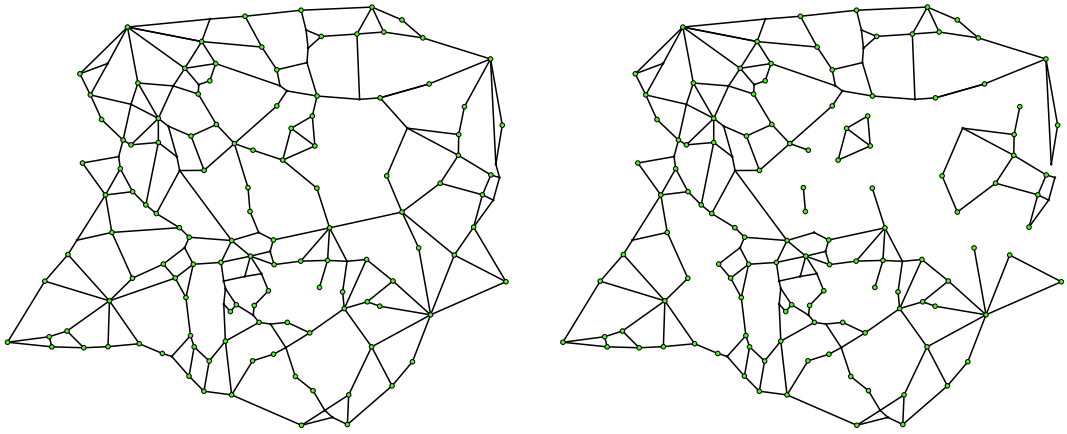
Prohlašuji, že jsem diplomovou práci zpracoval samostatně a všechny použité zdroje jsem uvedl v seznamu literatury.

V Olomouci dne 15.4.2023 Michal Trněný

Obsah

Úvod

Na svých cestách z jednoho místa do druhého často používáme síť silnic. Ať už je příčinou nehoda, kolona, opotřebení, údržba, útok nebo prostě jen špatné počasí, na silnici mezi sousedními obcemi vznikne blokace, kvůli níž silnici využít nelze. Většinou to nevadí, protože se do svého cíle dostaneme i po jiných cestách. Potíže vznikají, když se člověk ráno vzbudí a brzy na to zjišťuje, že se do práce nedostane po žádné cestě. Občas nastane rozsáhlá přírodní katastrofa, která způsobí, že je téměř současně zablokováno několik silnic. Výjimkou nejsou ani silnice v České Republice [1]. Jsou-li blokace rozmístěny nešťastným způsobem, rozpadá se síť na komponenty souvislosti, jako třeba na obrázku 1.



Obrázek 1: Vlevo silniční síť (Prostějov – Přerov – Kroměříž – Vyškov) a vpravo čtyři komponenty rozpadlé sítě

Komponenty souvislosti připomínají od sebe oddělené ostrovy. Přitom cestovat lze jen po ostrově, ale nelze jej opustit. Aby bylo možné cestovat v síti opět odkudkoliv kamkoliv, je potřeba komponenty spojit. To se provede odstraněním blokací. Aby došlo k napojení komponent co nejdříve, je vhodné hledat odpovědi na následující otázky:

1. Které blokace stačí odstranit, aby se síť stala souvislou?
2. V jakém pořadí blokace odstraňovat, aby při tom byla ujeta co nejkratší cesta?
3. Pokud máme k dispozici dvě nebo více cestářských čet, které čety poslat ke kterým blokacím?

Napojování komponent je jedním z typů úloh řešených v kombinatorické optimalizaci. Úlohy tohoto typu jsou řešitelné i snadno pochopitelnými algoritmy. Avšak jak víte, že řešení, které jste svým algoritmem našli, je nejlepší ze všech přípustných řešení? Abyste měli jistotu, stačí prověřit řešení všechna. Ale je nutné prověřovat řešení všechna u každé úlohy? Těch je totiž ve skoro každé úloze tohoto typu tolik,

že je všechna za přijatelně dlouhou dobu ověřit nestihnete. Proto jistotu, zda nalezené řešení je optimální, budete mít jen výjimečně. Máme-li odblokovat například 40 hran a jsou-li k dispozici 4 cestářské čety, které vyjedou do práce z jedné stanice (díky čemuž je nerozlišíme), existuje 478 způsobů[1], jakými lze zablokované hrany rozdělit mezi čety, a proto $478 \cdot 40! \doteq 3.9 \cdot 10^{50}$ různých řešení. Kdybychom měli měřit délku trvání opravy u každého řešení počítačem, který jich dokáže během sekundy vyhodnotit 10^{20}, tak by celková doba nalezení optimálního řešení trvala 10^{23} let. Ale aspoň bychom měli jistotu, že je to skutečně optimální řešení. Tak plýtvat nikdo nebude.

Řešit úlohy tohoto typu dokáže každý, ale z podstaty věci je prý vyřešit nedokáže nikdo [8] (s.3). Co když to není pravda? Co když každou z přijatelně velkých úloh tohoto typu lze vyřešit za přijatelně dlouhou dobu? Zodpovědět tuto otázku však není cílem této diplomové práce. Cílem je najít způsob řešení úlohy napojování komponent v rozpadlé silniční síti a napsat počítačový program, který na základě vhodných kritérií navrhne nejlepší postup pro opravu komunikací v libovolné silniční síti, kterou programu zadáte.

Požádá-li nás správce silnic o optimalizaci postupu napojování komponent v rozpadlé silniční síti, nebude chtít čekat ani 24 hodin na výpočet optimálního postupu, protože za tu dobu mohou čety cestářů mít práci hotovou, byť ne optimálně. Přijatelná doba čekání na výpočet proto bude výrazně kratší. Kdyby o dělení práce mezi čety cestářů rozhodoval autor této práce, neměl by trpělivost čekat na výsledek výpočtu déle, než jednu hodinu. I od toho se bude odvíjet velikost úloh, které je schopen řešit na svém počítači a kvalita jejich řešení.

V první kapitole uvedeme pojmy použité k zavedení modelu silniční sítě, k popisu úlohy, její složitosti a k popisu jejího řešení. Ve druhé kapitole popíšeme silniční síť a úlohu napojování komponent. Uvedeme, v jaké podobě a jaká očekáváme vstupní data, která určují zadání úlohy. Uvedeme podmínky kladené na řešení, míry kvality řešení a výstupní počítačový formát řešení úlohy. Popíšeme převody mezi příbuznými úlohami a tím i možnost použití metod vyvinutých k jejich řešení. Ve třetí kapitole představíme použité metody řešení úlohy – algoritmus úspor, mravenčí algoritmy a zlepšovací heuristiky 2-opt a 3-opt. U algoritmu úspor a u 2-opt si ukážeme i jejich přizpůsobení k řešení úlohy napojování komponent. Čtvrtá kapitola se zaměřuje na Max-Min Ant System, což je heuristika ze třídy mravenčích algoritmů, kterou používáme k řešení úlohy napojování komponent. Budeme se zabývat otázkou konvergence algoritmu a otázkou nastavení parametrů. Popíšeme naši počítačovou implementaci algoritmu. Pátá kapitola je věnována testům algoritmů a vlivu různých parametrů na fungování Max-Min Ant Systemu.

[1] Jazyk R, knihovna partitions, příkaz R(4, 40, include.zero = FALSE) – vrátí počet rozkladů č.40 na neuspořádaný součet 4 přirozených čísel. Používá Hindenburgovu (rekurzivní) metodu, viz [5].

1 Základní pojmy a značení

Nejpraktičtější věcí na světě je dobrá teorie. Teď lze tuto sekci přeskočit a vracet se k ní v případě potřeby. Později ji čtenář shledá užitečnou, zvláště pak teorii grafů.

1.1 Kombinatorika

Zde s občasnými úpravami přebíráme definice z [3] a [4].

Množiny pouze obsahují prvky bez jakéhokoliv uspořádání. Budeme potřebovat pracovat s uspořádanými k-ticemi objektů. Uspořádané k-tice $(a_1, \ldots, a_k)$ lze definovat různými způsoby a na volbě způsobu definice nám záležet nebude. Důležité však je, že to, co nesplňuje vlastnost 1.1, není uspořádaná k-tice.

$$((a_1, \ldots, a_k) = (b_1, \ldots, b_k)) \quad \Leftrightarrow \quad ((a_1 = b_1) \wedge \ldots \wedge (a_k = b_k)) \tag{1.1}$$

Pro ilustraci uveďme následující definici.

Definice 1 Uspořádanou dvojicí (a_1, a_2) rozumíme množinu $\{\{a_1\}, \{a_1, a_2\}\}$. Pro libovolné $k \in \mathbb{N}$ lze uspořádanou k-tici definovat rekurentním způsobem: $(a_1, \ldots, a_k) := ((a_1, \ldots, a_{k-1}), a_k)$. Pro $k = 1$, je uspořádaná k-tice $(a_1) = a_1$.

Definice 2 Množinu $\{i : i, a, b \in \mathbb{Z}, k \in \mathbb{N} \wedge (a \leq i) \wedge (i \leq b) \wedge (\forall i \exists n \in \mathbb{N}_0 : i = a + n \cdot k)\}$ značme symbolem $a(k)b$.

Příklad: Množinu $\{0, 4, 8, -2, 6, 2\}$ lze značit $-2(2)8$ nebo $-2(2)9$. Množinu prvních n přirozených čísel značíme $1(1)n$.

Definice 3 Nechť $|A|$ značí počet prvků množiny A.

Definice 4 Mějme množiny N, K, kde $K \subseteq N$. $|N| = n$, $|K| = k$. K se nazývá k-členná kombinace z prvků n-prvkové množiny N.

Poznamenejme, že kombinace je k-tice, která není uspořádaná.

Definice 5 Nechť $k, n \in \mathbb{N}$ a $k \leq n$. Mějme neprázdnou množinu N, $|N| = n$. $i, j \in 1(1)k$. $\forall i \in 1(1)k \;\; a_i \in N$. Dále nechť platí $(i \neq j) \Rightarrow (a_i \neq a_j)$. Uspořádaná k-tice $(a_1, \ldots, a_k)$ se nazývá k-členná variace z prvků n-prvkové množiny N.

Definice 6 n-členná variace z prvků n-prvkové množiny N se nazývá permutace z prvků n-prvkové množiny N.

Definice 7 Mějme množinu A neprázdnou a $|A| \geq m$, kde $m \in \mathbb{N}$. Dále nechť $\forall k, l \in 1(1)m$ platí:

a) $A_k \subseteq A \wedge A_k \neq \emptyset$

b) $(A_k \neq A_l) \Rightarrow (A_k \cap A_l \neq \emptyset)$

c) $A = \bigcup_{k=1}^{m} A_k$

Systém $\{A_k : k \in 1(1)m\}$ se nazývá rozklad množiny A na m tříd. (Jde o systém, ne o množinu, protože se připouští možnost, že pro $i \neq j$ je $A_i = A_j$.)

Definice 8 Nechť $\{A_k : k \in 1(1)m\}$ je rozklad množiny A na m tříd. Uspořádaná m-tice $(A_1, \ldots, A_m)$ se nazývá uspořádaný rozklad množiny A na m tříd.

Zkráceně budeme hovořit o rozkladu množiny nebo o uspořádaném rozkladu množiny.

1.2 Grafy a sítě

Zde s občasnými úpravami přebíráme definice, věty a postupy z knihy [6].

Definice 9 Mějme dvojici množin V, E, z nichž V je neprázdná. Dále mějme funkce $Pv : E \to V$ a $Kv : E \to V$. Čtveřice (V, E, Pv, Kv) se nazývá orientovaný graf.

Definice 10 Nechť čtveřice (V, E, Pv, Kv) je orientovaný graf. Množina V se nazývá množina vrcholů (resp. uzlů, bodů). Množina E se nazývá množina hran (resp. spojů, vazeb). $v_1 := Pv(e)$ se nazývá počáteční vrchol hrany e. $v_2 := Kv(e)$ se nazývá koncový vrchol hrany e. Funkce Pv, Kv se nazývají vztahy incidence. Říkáme, že v_1 a e jsou spolu incidentní, rovněž v_2 a e jsou spolu incidentní.

Definice 11 Nechť čtveřice (V, E, Pv, Kv) je orientovaný graf. Označme PK množinu $\{U \subset V; |U| = 2 \vee |U| = 1\}$. Zaveďme funkci $\varepsilon : E \to PK$ následujícím způsobem. $\forall e \in E\ \exists U \in PK;\ U = \{Pv(e), Kv(e)\}$. Trojice (V, E, ε) se nazývá neorientovaný graf.

Definice 12 Nechť trojice (V, E, ε) je neorientovaný graf. Množiny V, E se nazývají stejně, jako u orientovaného grafu. Položme $v_1 = Pv(e)$, $v_2 = Kv(e)$. Pak $\varepsilon(e) = \{v_1, v_2\}$ a říkáme, že vrcholy v_1, v_2 jsou incidentní s hranou e a nazývají se krajní vrcholy hrany e. Přitom nerozlišujeme, který vrchol je počáteční, a který koncový. Říkáme, že hrana e spojuje vrcholy v_1, v_2. Funkce ε se nazývá vztah incidence.

Je-li k dispozici graf G, bez ohledu na orientaci používejme značení $V(G)$ pro množinu vrcholů grafu G a značení $E(G)$ pro množinu jeho hran.

Definice 13 Nechť čtveřice (V, E, Pv, Kv) je orientovaný graf. $F := \{e \in E; \exists v_1, v_2 \in V\ Pv(e) = v_1 \wedge Kv(e) = v_2\}$. Počet prvků množiny F, t.j. $|F|$, se nazývá násobnost hrany. Je-li (V, E, ε) neorientovaný graf, násobností hrany e se myslí počet prvků množiny F, která se v tomto případě definuje jako $F := \{e \in E; \exists v_1, v_2 \in V\ \varepsilon(e) = \{v_1, v_2\}\}$.

Definice 14 Graf, jehož každá hrana má násobnost nejvýše jedna, se nazývá prostý graf. Graf, ve kterém existuje aspoň jedna hrana s násobností větší, než jedna, se nazývá multigraf.

Definice 15 Existuje-li v orientovaném grafu (V, E, Pv, Kv) hrana e a vrchol v takový, že $Pv(e) = v \wedge Kv(e) = v$, nazývá se hrana e smyčka. Existuje-li v neorientovaném grafu (V, E, ε) hrana e a vrchol v takový, že $\varepsilon(e) = \{v\}$, nazývá se hrana e smyčka.

Definice 16 Nechť $G = (V, E, \varepsilon)$ je neorientovaný graf, který je prostý, bez smyček a $(\forall v_1, v_2 \in V$, t.ž. $v_1 \neq v_2)$ $(\exists e \in E)(\varepsilon(e) = \{v_1, v_2\})$. Pak se G nazývá úplný graf. Má-li n vrcholů, značme jej K_n nebo K_V pro zdůraznění, že jde o úplný graf nad množinou vrcholů V.

Definice 17 Nechť G značí orientovaný graf (V, E, Pv, Kv). $W \subseteq V, F \subseteq E$. Zobrazení Qv budiž restrikcí zobrazení Pv na množinu F a Lv restrikcí zobrazení Kv také na množinu F. Nechť K_W je úplný graf nad množinou uzlů W (viz definice 16). Řekneme, že čtveřice $H := (W, F, Qv, Lv)$ je podgraf grafu G, je-li orientovaným grafem. Pokud $(W = V) \wedge (F \subseteq E)$, nazývá se graf H faktor grafu G. Pokud $(W \subseteq V) \wedge (F = E \cap K_W)$, nazývá se graf H podgraf grafu G indukovaný množinou vrcholů W.

Na množině všech podgrafů daného neorientovaného grafu G lze zavést relaci uspořádání pomocí relace „býti podgrafem“. Použijme značení $\leq$. Toto uspořádání obecně není úplné, protože v množině podgrafů grafu G mohou existovat aspoň dva, z nichž žádný není podgrafem toho druhého. Jako příklad takového grafu G můžeme vzít graf sestavený ze dvou vrcholů spojených hranou a jedním podgrafem je první uzel a druhým podgrafem je druhý uzel.

Definice 18 Řekneme, že grafy G_1 a G_2 jsou disjunktní, když platí
$(V(G_1) \cap V(G_2) = \emptyset) \wedge (E(G_1) \cap E(G_2) = \emptyset)$.

Definice 19 Nechť G je neorientovaný graf a $A \subseteq V(G)$. Označme $W_G(A)$ množinu hran, které jsou incidentní s právě jedním vrcholem z množiny A. $W_G(A)$ se nazývá řez určený množinou vrcholů A.

Definice 20 Nechť G je orientovaný (neorientovaný) graf a V jeho množina vrcholů a E jeho množina hran. Nechť I, J jsou množiny indexů. $\forall i \in I, j \in J$ nechť jsou množiny M_i, N_j neprázdné. Funkce $f_i : V \to M_i$, $g_j : E \to N_j$ se nazývají postupně ohodnocení vrcholů grafu G a ohodnocení hran grafu G. Graf spolu se svými ohodnoceními se nazývá orientovaná (neorientovaná) síť. Píšeme $(G, \{f_i; i \in I\}, \{g_j; j \in J\})$.

Pro naše potřeby se pojem sítě dal zavést i specifičtějším způsobem, například požadovat, aby množiny I, J byly konečné a aspoň jedna z nich neprázdná, abychom odlišili pojem graf od pojmu síť tím, že v síti existuje aspoň jedno ohodnocení narozdíl od grafu. Avšak zejména v aplikačně zaměřené literatuře se sítí často ve skutečnosti

míní graf. Termín graf se v kontextu matematické analýzy používá ve významu graf funkce a veřejnost si pod pojmem graf představuje právě tohle. Z těchto dvou důvodů chápejme graf G jako síť $(G, \{\}, \{\})$ s prázdnými množinami ohodnocení. Pak po úmluvě lze v teorii grafů a sítí používat pojem síť i pro označení grafů a pojmy definované pro grafy tímto způsobem přenést i na sítě.

V definici sítě se rovněž dalo po množinách hodnot M_i, N_j požadovat, aby každá z nich byla aspoň dvouprvková. Jinak se může stát že uzly nebo hrany budou ohodnoceny právě jednou hodnotou. To na první pohled vypadá zbytečně, protože mají-li všechny hrany stejnou hodnotu a všechny uzly stejnou hodnotu, lze takovou síť nahradit jednodušší strukturou – grafem. Ale zbytečné to není. Počítá-li se délka cesty v grafu, myslí se tím počet hran, z nichž se cesta skládá. Je-li umožněna jednoprvkovost množin M_i a N_j, lze zavést termín délky cesty v síti stejným způsobem jako termín délky cesty v grafu. Viz definice 25 a 26.

Definice 21 Nechť $G := (V, E, \varepsilon)$ je neorientovaný graf nebo $G := (V, E, Pv, Kv)$ je orientovaný graf. Nechť $N := (G, \{f_i; i \in I\}, \{g_j; j \in J\})$ je (neorientovaná resp. orientovaná) síť. Počet vrcholů $|V|$ grafu G (resp. sítě N) se nazývá řád grafu G (resp. sítě N) a značí se $|G|$ (resp. $|N|$). Počet hran $|E|$ grafu G (resp. sítě N) se nazývá velikost grafu G (resp. sítě N) a značí se $\|G\|$ (resp. $\|N\|$).

Definice 22 Nechť $G := (V, E, Pv, Kv)$ je orientovaný graf, $I := \{i \in \mathbb{N} : i \leq n, n \in \mathbb{N}\}$. Dále $(\forall i \in I)(v_i \in V, e_i \in E)$ a navíc $v_0 \in V$. Nechť $\forall i \in I$ v posloupnosti $(v_0, (e_i, v_i)_{i=1}^n)$ platí $Pv(e_i) = v_{i-1}$, $Kv(e_i) = v_i$. Pak se tato posloupnost nazývá orientovaný sled. Je-li G neorientovaný graf a platí-li v uvedené posloupnosti $\forall i \in I$, že hrana e_i spojuje vrcholy v_{i-1}, v_i, nazývá se posloupnost neorientovaný sled. V obou případech se v_0 nazývá počáteční (resp. startovní) vrchol sledu a v_n koncový (resp. cílový) vrchol sledu. O sledu říkáme, že spojuje vrchol v_0 s vrcholem v_n, nebo že vede z vrcholu v_0 do vrcholu v_n. O vrcholu v_n říkáme, že je dostupný z vrcholu v_0.

Definice 23 G je orientovaný (neorientovaný) graf. Posloupnost $(v_0, (e_i, v_i)_{i=1}^n)$ je orientovaný (neorientovaný) sled. Je-li každá hrana e_i ve sledu nejvýše jednou, nazývá se tato posloupnost orientovaný (neorientovaný) tah.

Definice 24 G je orientovaný (neorientovaný) graf. Posloupnost $(v_0, (e_i, v_i)_{i=1}^n)$ je orientovaný (neorientovaný) tah. Je-li každý vrchol v_i v tahu nejvýše jednou, nazývá se tato posloupnost orientovaná (neorientovaná) cesta.

Definice 25 V grafu G se délkou cesty $(v_0, (e_i, v_i)_{i=1}^n)$ rozumí počet n hran, které cesta obsahuje.

Definice 26 V síti $(G, \{f_i; i \in I\}, \{g_j; j \in J\})$ se délkou (cenou) cesty v ohodnocení g_j hran rozumí součet ohodnocení g_j hran, které cesta $(v_0, (e_i, v_i)_{i=1}^n)$ obsahuje, t.j.

$$\sum_{i=1}^{n} g_j(e_i)$$

Je-li v síti ohodnocení hran jedno, mluvíme jen o délce (ceně) cesty.

Definice 27 Nechť $G := (V, E, \varepsilon)$ je neorientovaný graf. Nechť pro každou dvojici vrcholů v_1, v_2 grafu G existuje neorientovaná cesta, která začíná ve v_1 a končí ve v_2. Takový graf G se nazývá souvislý graf.

Definice 28 Nechť G je neorientovaný graf a H jeho podgraf. Uspořádejme všechny podgrafy grafu G pomocí relace „býti podgrafem“. Nechť H je souvislý podgraf grafu G, t.j. $H \leq G$. Nechť dále neexistuje žádný graf $K < G$, který je souvislý a zároveň $H < K$. Potom se H nazývá komponenta souvislosti grafu G.

Definice 29 Nechť G je neorientovaný graf. Nechť je v tahu $(v_0, (e_i, v_i)_{i=1}^n)$ každý vrchol v_i, s výjimkou v_0 a v_n, nejvýše jednou. Nechť $v_0 = v_n$. Pak se tato posloupnost nazývá uzavřená cesta (resp. kružnice).

Definice 30 Nechť G je neorientovaný graf. $H \leq G$ a H je kružnice. Pokud žádný takový podgraf H grafu G neexistuje, nazývá se G les.

Definice 31 Nechť G je les a je souvislý, pak se G nazývá strom.

Definice 32 Faktor H grafu G, který je strom, nazýváme kostra grafu G (resp. napnutý strom grafu G).

Definice 33 Faktor H grafu G, který je les a má stejný počet komponent souvislosti jako graf G, nazýváme napnutý les grafu G.

Definice 34 Nechť $G := (V, E, \varepsilon)$ je souvislý graf a $c : E \to \mathbb{R}$ je ohodnocení jeho hran, které budeme nazývat cenami. Označme $\mathcal{K}(G)$ množinu všech koster grafu G. $H \in \mathcal{K}(G)$. Cenou $c(H)$ kostry H chápejme hodnotu $\sum c(e)$ pro $e \in E(H)$. Kostra L se nazývá nejlevnější kostra grafu G (vzhledem k ohodnocení c), pokud $\forall H \in \mathcal{K}(G)$: $c(L) \leq c(H)$.

Algoritmus 1 (Obecný postup hledání nejlevnější kostry)
Vstup: Souvislý graf G a ohodnocení hran cenami $c : E(G) \to \mathbb{R}$.
Úkol: Najít nejlevnější kostru grafu G.
Pomocná proměnná: Faktor L grafu G neobsahující kružnici (tj. L je les).
(1) Nechť $E(L) = \emptyset$ a $V(L) = V(G)$.
(2) Je-li les L stromem, výpočet končí. L je hledaná nejlevnější kostra.
(3) Zvolme libovolně hranu e s touto vlastností:
Hrana e spojuje dvě různé komponenty lesa L a alespoň pro jednu z těchto komponent (označme ji A) platí, že cena hrany e je nejmenší ze všech cen hran z množiny $W_G(A)$.
(4) Hranu e přidejme k lesu L (tím snížíme počet komponent) a pokračujeme krokem (2).

Pro tento postup existují různé varianty volby hran. Ukážeme si jen dvě:
Varianta 1.: Hrany grafu uspořádáme podle jejich cen do neklesajícího pořadí. Pak je v tomto pořadí probíráme a do postupně vytvářeného grafu L přidáváme ty z nich, které nezpůsobí vznik kružnice. Ačkoliv je algoritmus využívající tuto variantu volby hran znám spíše jako Kruskalův, tak s tímto způsobem přišel již dříve (v roce 1926) Otakar Borůvka.
Varianta 2.: Začneme z libovolného vrcholu a postupně k němu přidáváme vrcholy a hrany tak, že v každé fázi výpočtu máme strom. Přidávanou hranu přitom vždy vybíráme tak, aby měla nejmenší cenu z množiny $W_G(A)$, kde A je množina vrcholů stromu. Ačkoliv je algoritmus využívající tuto variantu volby hran znám spíše jako Primův, tak tento postup navrhl již dříve (v roce 1930) Vojtěch Jarník.

Definice 35 Nechť G je neorientovaný graf a C_1, C_2 jeho disjunktní podgrafy. Vrchol $u \in V(C_1)$ se nazývá hraniční vrchol grafu C_1, pokud existuje vrchol $v \in V(C_2)$ a pokud existuje cesta P, která je podgrafem grafu G taková, že u je její počáteční vrchol a v její koncový vrchol a $\forall e \in E(P) : e \notin E(C_1) \wedge e \notin E(C_2)$.

Definice 36 Nechť $G := (V, E, Pv, Kv)$ je orientovaný graf. Nechť ρ je rovina. Označme symbolem κ množinu všech spojitých jednoduchých křivek ležících v ρ. Nechť $\phi : V \to \rho$, $\psi : E \to \kappa$ jsou prostá zobrazení. Dále nechť $\forall e \in E$ platí $\phi(Pv(e))$ je počátečním bodem křivky $\psi(e)$, t.j. $\psi(e)(0) = \phi(Pv(e))$ a $\phi(Kv(e))$ je koncovým bodem křivky $\psi(e)$, t.j. $\psi(e)(1) = \phi(Kv(e))$ a zároveň nechť $\forall t \in (0, 1)\ \forall e \in E\ \forall v \in V\ \psi(e)(t) \neq \phi(v)$. Dvojice zobrazení (ϕ, ψ) se nazývá nakreslení grafu G do roviny ρ.

Definice 37 Existuje-li nakreslení (ϕ, ψ) grafu $G := (V, E, Pv, Kv)$ do roviny ρ takové, že pro libovolné dvě hrany $e_1, e_2 \in E$ platí, že průnik křivek $\psi(e_1), \psi(e_2)$ je právě jedna z následujících tří množin $\emptyset$, $\{\phi(Pv(e_1))\}$, $\{\phi(Kv(e_1))\}$, $\{\phi(Pv(e_1)), \phi(Kv(e_1))\}$, nazývá se graf G rovinný.

Obdobně se definuje nakreslení a rovinnost sítí, neorientovaných grafů a na nich založených sítí.

Věta 1 Nechť G je neorientovaný prostý souvislý rovinný graf s alespoň 3 vrcholy. Potom platí $\|G\| \leq 3|G| - 6$.

Věta 2 Nechť G je graf s n vrcholy, $n \geq 1$. Potom následující podmínky jsou vzájemně ekvivalentní:
1. G je strom.
2. G neobsahuje kružnici a má právě $n - 1$ hran.
3. G je souvislý a má právě $n - 1$ hran.
4. G je souvislý a odebráním kterékoliv hrany přestane být souvislý.
5. G neobsahuje kružnici a po přidání libovolné hrany bude obsahovat právě jednu kružnici.

Všechny kostry téhož grafu jsou jeho souvislé faktory a s těmito vlastnostmi jsou minimální vzhledem k relaci „býti podgrafem grafu“, nikoliv nejlevnější (jak je definováno výše) a nikoliv nejmenší. To jsou tři pojmy s odlišnými významy, které se občas zaměňují. Kostra je minimální, protože neexistuje menší souvislý faktor. Kostra je nejmenší souvislý faktor grafu jen tehdy, když všechny ostatní souvislé faktory jsou větší.

1.3 Optimalizace

Zde s občasnými úpravami čerpáme z [9], [10] a [2].

Označme D neprázdnou množinu a f funkci $f : D \to \mathbb{R}$. D je definiční obor funkce f.

Definice 38 $x^* \in D$. Řekneme, že funkce f má v bodě x^* lokální minimum $f(x^*)$, jestliže existuje otevřené okolí $U(x^*)$ bodu x^*, takové, že $\forall x \in U(x^*)$ je

$$f(x) \geq f(x^*) \tag{1.2}$$

Bod x^* se nazývá bod lokálního minima. Pokud nerovnost 1.2 platí ostře, říkáme, že funkce f má v bodě x^* ostré lokální minimum $f(x^*)$.

Definice 39 Jestliže $\forall x \in D$ platí nerovnost 1.2, řekneme, že funkce f má v bodě x^* globální minimum. Bod x^* se nazývá bod globálního minima. Pokud nerovnost 1.2 platí ostře, říkáme, že funkce f má v bodě x^* ostré globální minimum $f(x^*)$.

Definice 40 Úloha minimalizace spočívá v nalezení bodu $x^* \in D$, ve kterém funkce f má minimum. Toto řešení x^* se nazývá optimální řešení.

Úlohou optimalizace, zde konkrétně – úlohou minimalizace, je najít bod, ve kterém nabývá funkce f svého minima, ideálně globálního minima. Často se však spokojíme i s lokálním minimem. V praxi se o funkci f hovoří jako o cenové funkci nebo o účelové funkci. Je-li potřeba místo bodu minima hledat bod maxima funkce f, řešíme úlohu maximalizace. Ta se přímočaře dá převést na úlohu minimalizace, díky čemuž se můžeme zabývat jen jedním typem optimalizace. Bod maxima funkce f je totožný s bodem minima funkce $-f$, protože $\forall S \subseteq D$ platí $\max_{x \in S} f(x) = -\min_{x \in S}(-f(x))$.

Dosud byla řeč o nepodmíněné optimalizaci. Některá řešení, byť jsou optimální, dokonce globálně optimální, mohou být nepřípustná protože nesplňují podmínky, které na ně klademe. Řešení splňující podmínky, které jsou na ně kladeny, tvoří množinu S, $S \subset D$. Množina S značí množinu přípustných řešení. V podmíněné optimalizaci hledáme bod optima funkce f na množině S. To vede k následujícím definicím.

Definice 41 Nechť $x^* \in S$. Řekneme, že funkce f má v bodě x^* lokální minimum $f(x^*)$ vázané na množinu S, jestliže existuje otevřené okolí $U(x^*)$ bodu x^*, takové, že $\forall x \in U(x^*) \bigcap S$ platí nerovnost 1.2.

Definice 42 Jestliže $\forall x \in D \bigcap S$ platí nerovnost 1.2, řekneme, že funkce f má v bodě x^* globální minimum na množině S. Bod x^* se nazývá bod globálního minima na množině S.

Definice 43 Úloha podmíněné minimalizace spočívá v nalezení bodu $x^* \in D \bigcap S$, ve kterém funkce f má minimum vázané na množinu S.

Definice 44 Řešíme úlohu minimalizace funkce f na množině S. Našli jsme několik přípustných řešení $x_1, x_2, \ldots, x_N \in S$, o kterých nevíme, zda jsou optimální. Nechť po seřazení platí: $f(x_{i_1}) < f(x_{i_2}) \leq \ldots \leq f(x_{i_N})$. Body $x_{i_2}, \ldots, x_{i_N}$ se nazývají řešení suboptimální vzhledem k funkci f.

Dokud hledáme bod minima funkce $f : D \to \mathbb{R}$ s hodnotami v množině $\mathbb{R}$, využíváme uspořádání množiny reálných čísel. Jak ale mezi sebou porovnávat funkční hodnoty funkce $f : D \to \mathbb{R}^n$, jinými slovy: Jak mezi sebou porovnávat uspořádané n-tice reálných čísel? Která z dvojic je větší a proč? $(1, 4)$ nebo $(3, 2)$? Způsobů porovnání se dá vymyslet, kolik jich potřebujeme. My v této práci budeme používat lexikografické uspořádání.

Toto uspořádání nebudeme definovat pro množinu $\mathbb{R}^n$, ale omezíme se na množinu $\{(a_1, \ldots, a_n) : \forall i, j \in 1(1)n \ (a_i \in \mathbb{R}) \wedge ((i < j) \Rightarrow (a_i \geq a_j))\} =: \mathbf{R}_n$. Jde o množinu všech uspořádaných n-tic sestupně řazených reálných čísel. Tedy např. $(1, 3, 2)$ do $\mathbf{R}_3$ nepatří, zatímco $(0, 0, -7)$ ano. Budeme-li dále uvedeným způsobem chtít porovnat dvě n-tice $x, y \in \mathbb{R}^n \setminus \mathbf{R}_n$, je nutné nejdříve zobrazit je zobrazením $s : \mathbb{R}^n \to \mathbf{R}_n$ na obrazy $s(x)$, $s(y)$ a porovnat spolu teprve až tyto obrazy. Zobrazení s je sestupné seřazení.

Definice 45 Mějme dvě n-tice $(a_1, \ldots, a_n), (b_1, \ldots, b_n) \in \mathbf{R}_n$. Dále nechť $i \in 1(1)n$. Lexikografické uspořádání „<“ na $\mathbf{R}_n$ definujeme takto:
$((a_1, \ldots, a_n) < (b_1, \ldots, b_n)) \Leftrightarrow (\exists i \in 1(1)n \ \ \forall j < i \ \ a_j = b_j \wedge a_i < b_i)$.
$((a_1, \ldots, a_n) = (b_1, \ldots, b_n)) \Leftrightarrow (\forall i \in 1(1)n \ \ a_i = b_i)$.

1.4 Složitost

Zde čerpáme s občasnými úpravami z [8].

Úloha, kterou v této práci řešíme, se řadí mezi vyhledávací úlohy. Hledáme optimální řešení k libovolnému konkrétnímu zadání (tj. instanci) řešeného typu úlohy. Ve vyhledávacích úlohách chceme k dané instanci získat množinu řešení. V rozhodovacích úlohách chceme k dané instanci získat buď odpověď ano, nebo odpověď ne. Vyhledávací úlohy jsou zobecněním rozhodovacích úloh. Rozhodovací úlohy mají přirozený protipól v teorii jazyků a automatů, což je hlavní důvod volby kontextu úloh rozhodovacích. Úloha odpovídá formálnímu jazyku. Algoritmus (postup řešení) odpovídá Turingovu stroji. Algoritmus k dané instanci úlohy vrátí odpověď buď ano, nebo ne.

Analogicky k tomu Turingův stroj dané slovo jazyka buď přijme nebo zamítne.

Pojmy z teorie jazyků a automatů jako symbol, abeceda, slovo, jazyk, gramatika, automat, Turingův stroj,... zde definovat nebudeme a odkážeme čtenáře na [7]. Následující definice stačí zavést pro rozhodovací úlohy. Zůstávají v platnosti i pro vyhledávací úlohy.

Definice 46 Nechť se dá úloha Π napsat jako dvojice (D, Y), kde D značí množinu všech instancí úlohy Π a $Y \subseteq D$ značí množinu instancí úlohy Π, na které existuje odpověď ano. Π se nazývá rozhodovací úloha.

Definice 47 Nechť $I \in D$ je instance úlohy $\Pi = (D, Y)$. Instance I odpovídá slovu x v jazyce L příslušnému úloze Π. Mějme funkci len : $D \to \mathbb{Z}^+$, kde $\text{len}(I) = |x|$.
$\text{len}(I)$ se nazývá velikost instance úlohy.

Definice 48 Nechť L je jazyk reprezentující úlohu Π a $x \in L$ jeho slova reprezentující instance $I \in D$ úlohy $\Pi = (D, Y)$. Nechť A je algoritmus řešící úlohu Π a M je jeho deterministický Turingův stroj rozpoznávající slova jazyka L. Fakt, že x reprezentuje I značme $x(I)$. Fakt, že M reprezentuje A značme $M(A)$. Nechť $k(x)$ je počet kroků, které M vykoná před tím, než se zastaví a x přijme nebo nepřijme. Zaveďme $D_n \subset D$ následujícím způsobem: $D_n := \{I \in \Pi : \text{len}(I) = n\}$. Definujme časovou složitost algoritmu A jako funkci $T_A : \mathbb{Z}^+ \to \mathbb{Z}^+$ způsobem $T_A(n) := \max\{k(x) : x(I) \in L,\ I \in D_n\}$.

Definice 49 A se nazývá algoritmus s polynomiální časovou složitostí, pokud existuje polynom p takový, že $\forall n \in \mathbb{Z}^+ : T_A(n) \leq p(n)$.

Definice 50 Nechť A_P je algoritmus s polynomiální časovou složitostí, který řeší úlohu Π a $M = M(A_P)$ jeho deterministický Turingův stroj. Nechť L_M značí množinu slov, které stroj M rozpozná. Fakt, že jazyk L reprezentuje úlohu Π značme $L(\Pi)$. Třídu úloh Π složitosti P zavedeme jako: $\text{P} := \{\Pi : \exists A_P, L(\Pi) = L_M \text{ pro } M = M(A_P)\}$

Definice 51 Nechť A_P je algoritmus s polynomiální časovou složitostí, který řeší úlohu Π a $M = M(A_P)$ jeho nedeterministický Turingův stroj. Nechť L_M značí množinu slov, které stroj M rozpozná. Fakt, že jazyk L reprezentuje úlohu Π značme $L(\Pi)$. Třídu úloh Π složitosti NP zavedeme jako: $\text{NP} := \{\Pi : \exists A_P, L(\Pi) = L_M \text{ pro } M = M(A_P)\}$

Rozdíl mezi deterministickým (DTS) a nedeterministickým (NTS) Turingovým strojem, zjednodušeně řečeno, je v tom, že DTS je v každém kroku výpočtu v jednom stavu, zatímco NTS může být v několika různých stavech současně.

Věta 3 $\Pi \in$ NP. Pak existuje polynom p takový, že Π lze vyřešit deterministickým algoritmem se složitostí $O(2^{p(n)})$, kde $n = \text{len}(I)$, kde I je instance Π.

Úloha, která je řešitelná v polynomiálním čase na DTS, je v polynomiálním čase řešitelná také na NTS. Proto platí $(\Pi \in \mathrm{P}) \Rightarrow (\Pi \in \mathrm{NP})$. Tedy platí $\mathrm{P} \subseteq \mathrm{NP}$. Dosud se nepodařilo dokázat existenci Π takové, že $\Pi \in \mathrm{NP} \wedge \Pi \notin \mathrm{P}$, ačkoliv u většiny kombinatorických úloh z praxe neznáme algoritmus, který by je dokázal řešit v P čase.

Definice 52 Σ^* značí množinu všech konečných řetězců (slov) nad abecedou Σ.

Definice 53 Nechť $\Pi_1, \Pi_2 \in \mathrm{NP}$ a jazyk $L_1 \subseteq \Sigma_1^*$ reprezentuje Π_1 a $L_2 \subseteq \Sigma_2^*$ reprezentuje Π_2. Zobrazení $f : \Sigma_1^* \to \Sigma_2^*$ nechť je vyčíslitelné algoritmem s polynomiální časovou složitostí a nechť $\forall x \in \Sigma_1^* : (x \in L_1) \Leftrightarrow (f(x) \in L_2)$. Říkáme, že existuje polynomiálně složitá transformace úlohy Π_1 na úlohu Π_2 a píšeme $\Pi_1 \propto \Pi_2$.

Definice 54 Úloha $\Pi \in \mathrm{NP}$ se nazývá NP-úplná, jestliže $\forall \Pi' \in \mathrm{NP} : \Pi' \propto \Pi$.

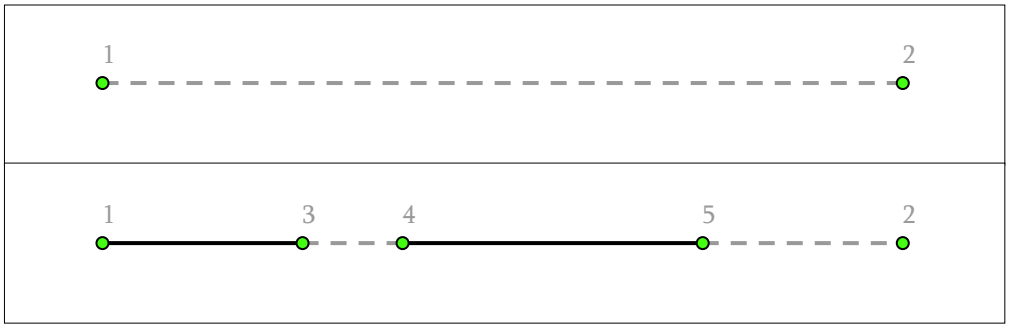
2 Úloha napojování komponent

2.1 Silniční síť

Na věci se nic nezmění, když města budeme považovat za obce. Ta část reality, která obsahuje obce, křižovatky mimo obec a silnice, je běžně označována za silniční síť. Dá se na ni nahlížet zjednodušeně pomocí matematického modelu, který se nazývá síť (definice 20). Síť občana se liší od sítě správce silnic například způsoby ohodnocení vrcholů a hran. Když nastane katastrofa, občanská síť se mění jiným způsobem, než správcovská síť. Vytvořme model silniční sítě, který budeme ve zbytku textu používat k optimalizaci postupu napojování komponent v rozpadlé silniční síti. Tento model nazývejme správcovská síť (resp. síť správce).

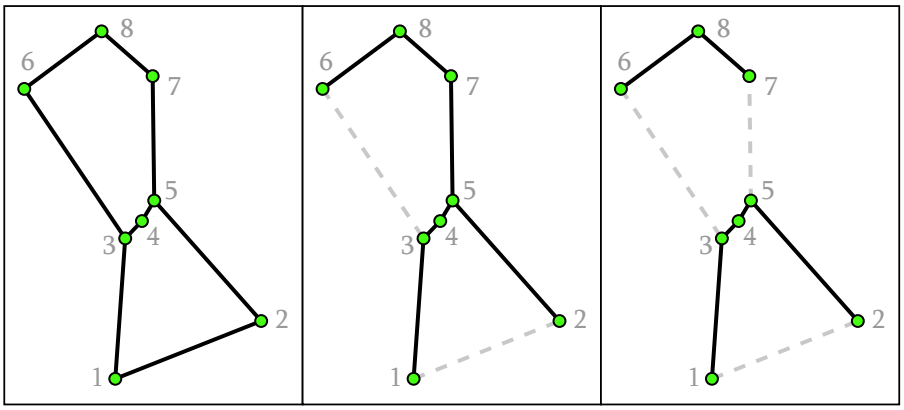
Uvažujme nejdříve neorientovaný graf $G = (V, E, \varepsilon)$ (definice 11). Vrcholy z množiny vrcholů V reprezentují obce, křižovatky a dělící body, které popíšeme vzápětí. Stane se katastrofa, která nánosem bahna, spadnutím stromů nebo jinak způsobí zablokování některých úseků několika silnic, a tak jejich neprůjezdnost pro osobní automobily. Dělící body (viz obrázek 2) jsou místa na silnici, která silničářům poslouží k vymezení zablokovaných úseků. Dělícím bodem může být i obec nebo křižovatka. Hrany z množiny hran E reprezentují silnice nebo úseky silnic. Po hranách se lze pohybovat v obou směrech, protože se zaměřujeme na transport mezi obcemi. Nechť je obec nebo křižovatka nebo dělící bod reprezentován vrcholem $v_i \in V$ a vede z něj silnice nebo úsek silnice reprezentován hranou $e \in E$ do jiné obce, křižovatky nebo dělícího bodu reprezentovaného vrcholem $v_j \in V$. Tento fakt je zaznamenán vztahem incidence $\varepsilon(e) = \{v_i, v_j\}$. To stačí k zavedení neorientovaného grafu G.

V této práci uvažujeme graf G souvislý. Množinu všech zablokovaných hran označme R, $R \subseteq E$. Obyvateli zbývají k použití jen hrany z množiny E', $E' = E \setminus R$. Původní $G = (V, E, \varepsilon)$ se tak pro občana mění na $G' = (V, E', \varepsilon')$. $\varepsilon'(e) = \varepsilon(e)$ pro $e \in E'$. G' je



Obrázek 2: Nahoře je celá hrana {1,2} mezi vrcholy 1 a 2 chápaná jako zablokovaná. Dole v přesnějším grafu přibyly dělící body 3, 4, 5, hrana {1,2} byla nahrazena hranami {1,3},{3,4},{4,5},{5,2}, z nichž zablokované jsou ve skutečnosti pouze {3,4} a {5,2}.

faktor grafu G (definice 17). Existují takové blokace R, že graf G' zůstává souvislý a existují i jiné blokace R, po nichž se graf G rozpadá na nesouvislý graf G'. Nesouvislý graf, a tak i nesouvislá (rozpadlá) občanská síť pak neumožňuje svým uživatelům cestovat z libovolného uzlu do libovolného jiného. V této práci budeme uvažovat graf G' nesouvislý.



Obrázek 3: Vlevo je graf G původní sítě. Vpravo je graf G' sítě rozpadlé na dvě komponenty, která vznikla blokací tří hran v původní síti. Uprostřed je graf G'' sítě, která vznikla spojením komponent grafu G' rozpadlé sítě. Ke spojení stačilo otevřít hranu {3,6} nebo hranu {5,7}.

Ty hrany z množiny R, jejichž zablokování způsobilo rozpad souvislého grafu G na nesouvislý graf G', tvoří množinu B, $B \subseteq R$. Množinu hran z B, které stačí otevřít, aby se nesouvislý graf G' změnil na souvislý graf G'', označme A, $A \subseteq B$. Obecně množina A není určena jednoznačně. $E' \cup A =: E''$. $G'' := (V, E'', \varepsilon'')$, kde ε'' je restrikce funkce ε na množinu E''. Otevřením všech hran z A tedy vznikne z nesouvislého grafu G' souvislý graf G''. Obrázek 3 znázorňuje všechny tři fáze sítě z pohledu občana — původní stav, napojené komponenty, rozpadlou síť. $G \geq G'' > G'$.

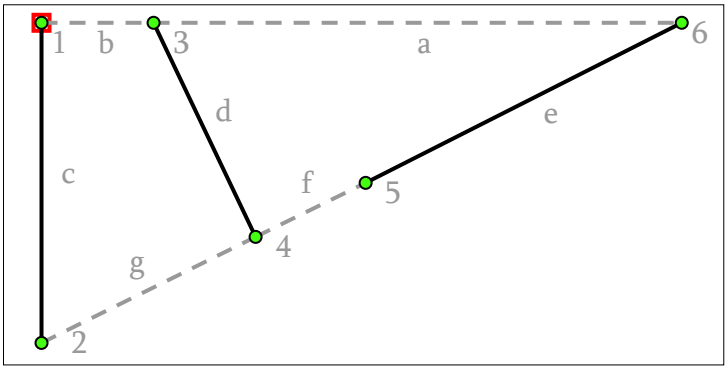
K definici správcovské sítě $S := (G, \{o\}, \{c, r\})$ potřebujeme zavést funkce o, c, r. Funkce $o : V \to \mathbb{N}_0$ ohodnocuje vrcholy počtem obyvatel, kteří v objektu zastoupeném příslušným vrcholem žijí. Jestliže vrchol v zastupuje osídlenou obec, tak $o(v) > 0$, jinak $o(v) = 0$. Předpokládáme, že čety silničářů se narozdíl od osobních automobilů mohou přesouvat přes zablokovanou silnici nebo podél ní a navíc je to ani nezdrží [2] (s.1094). Dále předpokládáme, že těžká technika se pohybuje předem zvolenou konstantní rychlostí *vel*. Z délek silnic pak lze dopočítat množství času v sekundách, které vyžaduje přesun těžké techniky po té které hraně. Tím se dostáváme k definici ohodnocení c. Funkce $c : E \to \mathbb{R}^+$ přiřadí každé hraně dobu v sekundách přesunu těžké techniky po této hraně předem zvolenou rychlostí. Funkce $r : E \to \mathbb{R}_0^+$ ohodnocuje hrany dobou v sekundách potřebnou k opravě (odblokování) příslušné silnice. Pro $e \in R$ je $r(e) > 0$, jinak je $r(e) = 0$.

2.2 Zadání úlohy

Jde o zorganizování silničářských prací tak, aby rekonstrukce sítě proběhla co nejrychleji a současně byl nalezen v jistém smyslu co nejmenší souvislý podgraf G'' grafu G, jehož podgrafem je nesouvislý graf G'.

Úkolem Π_0 správce silnic je navrátit silniční síť do původního stavu, t.j. graf G' změnit na původní souvislý graf G, pokud je to možné. To někdy možné být nemusí, například když se propadne silnice nebo se při zemětřesení zřítí most a postavět nový trvá léta. Naštěstí zemětřesení u nás nejsou tak silná. Pak je úkolem Π_1 odstranit všechny odstranitelné blokace, což v kontextu střední Evropy jsou všechny z R a udělat to za co nejkratší dobu. Je-li síť rozpadlá na komponenty souvislosti, může dostat prioritu úkol Π_2. V něm jde o to odstranit za co nejkratší dobu ty blokace, po jejichž odstranění se síť spojí do jediné komponenty souvislosti, t.j. graf G' změnit na souvislý graf G'', kde $G' < G'' \leq G$.

Komponenty rozpadlé sítě cestáři napojí na sebe, a tím vznikne souvislá síť, ve které stále ještě mohou některé hrany být zablokované, ale obyvatelé se v ní už dostanou ze kteréhokoliv vrcholu do kteréhokoliv jiného bez ohledu na délku cesty. Vyřešením úlohy Π_2 práce správce sítě nekončí, protože je potřeba odblokovat všechny zablokované hrany. Naše práce však uměním řešit úlohu Π_2 končí. Označme řešení úlohy Π_2 symbolem σ_2 a její optimální řešení symbolem σ_2^*. Označme řešení úlohy Π_1 symbolem σ_1 a její optimální řešení symbolem σ_1^*. Řekněme, že jsme našli σ_2^* a pak k tomuto řešení přidávali hrany v nejlepším možném pořadí a nejvhodnějším možným četám silničářů tak, abychom jej rozšířili na řešení úlohy Π_1. Může se stát, že toto nové řešení σ_1 nebude optimální právě proto, že σ_2^* optimální je (viz obrázek 4). Proto doporučujeme uživateli rozmyslet si, jak moc a kdy je řešení které úlohy důležité.



Obrázek 4: Uzel 1 je stanice s jedinou četou. a,b,f,g jsou zablokované hrany. $\sigma_2^* = ((b, f))$ pro sled (1,b,3,d,4,f,5). $\sigma_1^* = ((b, a, f, g))$ pro sled (1,b,3,a,6,e,5,f,4,g,2). Prodloužíme-li σ_2^* na σ_1 nejlepším možným způsobem, dostaneme $\sigma_1 = ((b, f, a, g))$ pro sled (1,b,3,d,4,f,5,e,6,a,3,d,4,g,2).

Dále upřesníme formulaci úlohy Π_2. Předpokládejme, že máme k dispozici m jednotek silničářů, $m \in \mathbb{N}$ a všechny je využijeme. V každém okamžiku každá z jednotek bude pracovat na odstranění nejvýše jedné blokace nebo se bude přesouvat nebo bude ve stanici. V každém okamžiku na jedné blokaci bude pracovat nejvýše jedna jednotka silničářů. Můžeme obecně předpokládat, že každá z jednotek silničářů se v konkrétním čase (například hned na začátku) nachází v jiném vrcholu. Pokud se četa nachází mimo vrchol grafu G, na některé z hran, lze tento graf změnit na takový, v němž každá z čet bude v některém z vrcholů tím, že do něj přidáme vrcholy tam, kde se aktuálně vyskytují čety a každý z vrcholů připojíme novými hranami ke krajním vrcholům hrany, na niž jsme nový vrchol přidali. Jde nám o zachycení okamžitého rozmístění čet.

Také lze uvažovat i případy, kdy čety silničářů vyjíždějí k blokacím z několika stanic, z nichž každá se nachází v jiné obci. Množinu stanic označme ST. $ST \subset V$. $|ST| = z$. V těchto z stanicích je rozmístěno m čet způsobem, který určuje uspořádaná z-tice $h := (h_1, \ldots, h_z)$, takže platí $\sum_{j=1}^{z} h_j = m$. Nechť je tedy na počátku m jednotek silničářů rozmístěno ne nutně v různých vrcholech $v_{j_1}, v_{j_2}, \ldots, v_{j_m} \in ST$. Připomeňme si předpoklad, že každá z čet může přes zablokovanou hranu projet bez zdržení, aniž by ji opravovala.

Každá množina A hran postačujících ke spojení komponent sítě do jediné komponenty se dá napsat jako sjednocení $A = \bigcup_{k=1}^{m} A_k$ po dvou disjunktních neprázdných množin A_k, kterých je m. A_k značí množinu hran, které má za úkol odblokovat k-tá cestářská četa. Některé z čet se staly rozlišitelnými od jiných díky tomu, že vyjíždějí z odlišných vrcholů. Proto záleží na tom, které z čet přiřadíme který díl práce. Toto rozdělení práce vyjadřuje uspořádaný rozklad $\mathcal{A} := (A_1, \ldots, A_m)$ množiny A (viz definice 8). k-tá četa otevírá hrany množiny A_k v pořadí určeném permutací π_k z prvků této množiny. Označme $a_k = |A_k|$ a požadujme, ať $\forall k \in 1(1)m \;\; a_k \geq 1$.

Doba, kterou k-tá četa potřebuje k opravení všech hran z množiny A_k nezávisí jen na součtu $\sum_{e \in A_k} r(e)$, ale i na pořadí π_k, v němž hrany četa navštíví. Celkovou dobu práce k-té čety označme T_k. Spočítá se následovně:

$$T_k = T(\pi_k) = \sum_{e \in A_k} r(e) + \sum_{e \in W_k} c(e) \tag{2.3}$$

kde W_k je sled k-té čety, který začíná ve vrcholu v_{j_k} a obsahuje nejen všechny hrany z množiny A_k, ale i hrany nejkratší cesty, přes něž četa přejede na své cestě od jedné zablokované hrany ke druhé. Tento sled končí ve stejném vrcholu, ve kterém začal.

Cílem je mezi všemi podmnožinami množiny blokací B najít takovou A spolu s takovým jejím rozkladem $\mathcal{A}$ a spolu s takovou m-ticí permutací $(\pi_1, \ldots, \pi_m)$, že veličina $q(T_1, \ldots, T_m)$ bude nabývat svého minima. $q(T_1, \ldots, T_m)$ zavedeme v další sekci.

2.3 Kritéria optimality řešení

Zde uvádíme zdaleka ne úplný výčet tvarů, jak může veličina q vypadat:

1. $q := q_1(T_1, \ldots, T_m) = \sum_{k=1}^{m} T_k$. Nevýhodou je, že i při minimálním q stále ještě může práce jedné z čet trvat výrazně déle, než práce ostatních čet a to jen z toho důvodu, že dostala více práce, než ostatní čety. Tomuto jevu se snaží předejít následující zavedení veličiny q.
2. $q := q_2(T_1, \ldots, T_m) = \max\{T_k : k \in 1(1)m\}$. Nevýhodou této funkce je, že považuje za stejně dobrá řešení například (9,8,10) a (1,10,2). Přitom v druhém případě je zřejmé, že silničáři dokončili práci za celkově kratší dobu a spotřebovali tak menší množství energie. Rozlišit tyto případy dokáže následující definice q.
3. $q := q_3(T_1, \ldots, T_m) = (t_1, \ldots, t_m)$, kde $(t_1, \ldots, t_m)$ je m-tice časů $T_1, \ldots, T_m$ seřazených sestupně od největšího po nejmeší.
   Speciálně $t_1 = \max\{T_k : k \in 1(1)m\}$ a $t_m = \min\{T_k : k \in 1(1)m\}$. m-tice $(t_1, \ldots, t_m)$ pak mezi sebou porovnáváme pomocí lexikografického uspořádání (definice 45), např. $(10, 9, 8) > (10, 2, 1)$. Nápad na využití tohoto způsobu porovnávání různých řešení pochází od [2] (s.1095).

V této práci používáme funkci q_3. I ona má nedostatky. Uvažujme předchozí příklad $(10, 9, 8) > (10, 2, 1)$ a změňme v něm $(10, 2, 1)$ na $(11, 2, 1)$. Pak bude platit $(10, 9, 8) < (11, 2, 1)$ a zároveň $10 + 9 + 8 > 11 + 2 + 1$. V případě $(10, 9, 8)$ dojde k napojení komponent za dobu T=10 a celková doba práce je 27. V případě $(11, 2, 1)$ dojde k napojení komponent sice později – za dobu T=11, ale celková doba práce je 14. To je důvod k zamyšlení se nad kritérii optimality řešení. Tedy nejenže optimalita nalezeného řešení je závislá na typu řešené úlohy (Π_2 či Π_1), ale je závislá i na volbě funkce q časů $T_1, \ldots, T_m$.

Definice 55 Značíme-li $\sigma := (\pi_1, \ldots, \pi_m)$ a $T_k = T(\pi_k)$, viz vzorec 2.3 a q je funkce dob prací čet, značme dále $\overline{T}(\sigma) := (T(\pi_1), \ldots, T(\pi_m))$. Cenovou funkci p řešení σ zavedeme jako $p(\sigma) := q(\overline{T}(\sigma))$. Pro $i \in \{1, 2, 3\}$ můžeme psát $p_i = q_i \circ \overline{T}$.

Nyní lze úlohu Π_2 formulovat tímto způsobem. Odstraňte za co nejkratší dobu ty blokace, po jejichž odstranění se síť spojí do jediné komponenty souvislosti. V množině všech m-tic permutací (v sekci 2.2 popsaných) najděte takovou, pro niž funkce p_3 nabývá svého minima.

Pokud správci sítě řešič úlohy poskytne několik nejlepších p_3-suboptimálních (definice 44) řešení, bude správce moci zvolit to, které mu vyhovuje nejvíce. Bude-li například kromě doby napojení komponent optimalizovat i množství spotřebované energie, lze z množiny poskytnutých řešení vybrat to, které má nejmenší hodnotu p_1. Nebo z ní může vybírat podle jiných kritérií, která vymyslí lépe, než bychom to dokázali my.

2.4 Další modely rozpadlé sítě

Až na výjimečná mimoúrovňová křížení realizovaná různými kombinacemi mostů a nebo tunelů lze graf silniční sítě aproximovat rovinným grafem (def. 37). Rovinné grafy se vyznačují tím, že počet jejich hran nebývá větší, než trojnásobek počtu jejich vrcholů (věta 1). Silniční sítě, s nimiž pracujeme, mají počet vrcholů v rozmezí desítek (mikroregiony) až desetitisíců (státy). V případě rozpadu velkých sítí je na místě otázka, zda je nutné pracovat s celou sítí, nebo zda stačí pracovat s těmi částmi sítě, které jsou relevantní pro řešenou úlohu.

Na ni článek [2] (s. 1095) odpovídá vytvořením nové sítě na základě původní následujícím způsobem. Stačí pracovat nad hraničními (def. 35) vrcholy komponent (včetně stanice silničářů) a pohybovat se mezi nimi po nejkratších cestách. Nechť $n \in \mathbb{N}$ značí počet komponent souvislosti grafu G', který vznikne z původního grafu G odebráním zablokovaných hran (t.j. hran z množiny B). Označme $C := \{C_i : i \in 1(1)n\}$ množinu komponent souvislosti C_i grafu G'. Dále označme BC_i množinu hraničních vrcholů komponenty C_i. Nechť ST značí množinu vrcholů, ve kterých se nachází stanice silničářů. $BC := \bigcup_{i=1}^{n} BC_i \cup ST$ je množina hraničních vrcholů všech komponent spolu se stanicemi. Nechť $u, v \in BC \wedge u \neq v$. V síti S existuje nejkratší cesta $P(u, v)$ začínající ve vrcholu u a končící ve vrcholu v s délkou $l(u, v) := \sum c(e)$ pro $e \in E(P(u, v))$. Nechť $SP := \{P(u, v) : u, v \in BC \wedge u \neq v\}$ značí množinu nejkratších cest mezi každou dvojicí hraničních nebo staničních vrcholů. Cesta $P(u, v)$ jako prvek množiny SP hraje roli jedné hrany. Vztah incidence ε mezi hranami z SP a vrcholy z BC je přímočarý: $\varepsilon(P(u, v)) = \{u, v\}$. Vznikne tak úplný graf $CG := (BC, SP, \varepsilon)$ nad hraničními vrcholy a stanicemi. S jeho pomocí definujeme úplnou síť $(CG, \{\}, \{l\})$.

Definice 56 Mějme síť $S := (G, \{o\}, \{c, r\})$ a množinu zablokovaných hran B. Síť $(CG, \{\}, \{l\})$ definovanou výše uvedeným způsobem nazývejme CG-síť nebo úplná síť (nad hraničními vrcholy a stanicemi).

Prostá rovinná síť S řádu 1000 má velikost nejvýše 2994 (věta 1). Pokud se S rozpadne na n komponent s celkem 78 hraničními vrcholy, její CG-síť bude mít řád 78 a velikost 3003, tedy větší, než je velikost sítě S. V tom případě se na první pohled zdá, že jsme ze sítě S vytvořili složitější síť CG. Vytvořili jsme síť, která má sice větší velikost, ale je jednodušší. Je jednodušší v tom smyslu, že je úplná – každý vrchol v ní sousedí s každým jiným a k ohodnocení hran se použijí hodnoty uspořádané ve čtvercové matici. Jakmile se jednou vypočítá čtvercová matice $l(u, v)$ a seznam SP nejkratších cest mezi vrcholy u, v, tak se těmito výpočty řešič úlohy nemusí zabývat stále znova a v případě potřeby pouze vyhledá údaje v paměti.

Před tím, než začneme hledat řešení úlohy napojování komponent, zamysleme se nad tím, jaké bude mít vlastnosti. Původní síť spolu s blokacemi lze vnímat abstraktněji.

Definice 57 Nechť C je množina komponent souvislosti grafu G' a $C_i \in C$ pro $i \in 1(1)n$. Nechť $B \subseteq E(G)$ je množina zablokovaných hran grafu G a nechť $b \in B$. Definujme komponentový multigraf (KMG) jako (C, B, ζ), kde $\zeta(b) = \{C_i, C_j\}$ kdykoliv $\exists u \in BC_i, v \in BC_j$ takové, že $\varepsilon(b) = \{u, v\}$ pro $G = (V, E, \varepsilon)$. KMG spolu s ohodnocením hran $r : B \to \mathbb{R}_0^+$, které přiřazuje zablokovaným hranám doby oprav, tvoří KMG-síť $(KMG, \{\}, \{r\})$.

V grafu KMG vrcholy reprezentují komponenty souvislosti grafu G'. Dva vrcholy v KMG jsou spolu spojeny hranou kdykoliv se jedná o sousední komponenty G', tj. o takové, k nimž když přidáme vhodnou hranu z B, spojíme je tím do nové, větší komponenty. KMG je multigraf (definice 14), protože mezi dvojicí vrcholů mívá běžně více různých hran, což v silniční síti odpovídá většímu množství zablokovaných silnic mezi dvěmi komponentami. KMG je souvislý graf.

Ke spojení dvou sousedních komponent do jedné není nutné odstranit všechny blokace, které je od sebe oddělují, ale stačí odstranit jednu, například tu, jejíž oprava zabere nejméně času. Předpokládejme, že G má konečný počet vrcholů. Existuje k libovolné dané dvojici G, B nějaká hodnota, která odpovídá minimálnímu počtu hran, které stačí přidat (množina A), aby se z nesouvislého grafu G' stal souvislý graf G''? Ano, existuje, protože je konečný počet způsobů, jakým lze napojení komponent provést a my jen z nich vybereme ten, který ke G' přidává nejméně hran. Jaký je nejmenší počet hran, které stačí ke G' přidat, aby se nesouvislý graf G' propojil do souvislého grafu G''? A existuje na to vzorec?

Jinými slovy: Uvažujme faktor L_0 (definice 17) grafu KMG, takový, že $L_0 := (C, \emptyset, \zeta)$, tj. bez hran. Jaký nejmenší počet hran z grafu KMG stačí přidat k faktoru L_0, aby výsledný graf byl souvislý? Jak se na to přijde?

Má-li nesouvislý graf G' n komponent souvislosti, tak KMG má n vrcholů. Uvažujme kostru $L \in \mathcal{K}(KMG)$ grafu KMG. Z definice kostry (definice 32) plyne, že je to strom, a že obsahuje všechny vrcholy KMG. Tedy L má n vrcholů. Z věty 2 plyne, že vzhledem k relaci „býti podgrafem grafu“ je (libovolná) kostra grafu KMG minimálním souvislým podgrafem KMG obsahující všechny jeho vrcholy a má právě $n - 1$ hran.

Ke grafu L_0 stačí přidat $n - 1$ hran tvořících kostru grafu KMG, aby výsledný graf byl souvislý. B určuje, na kolik komponent souvislosti se G rozpadne. Značí-li n počet komponent a A minimální množinu hran postačujících k napojení komponent, pak z výše uvedeného plyne, že $|A| = n - 1$. Potom řešení σ úlohy Π_2, které obsahuje (opravuje) nejvýše $n - 2$ blokací, není přípustné, protože nenapojuje všechny komponenty.

Předpokládejme naopak, že se řešení σ skládá aspoň z n blokací. Dále předpokládejme, že je mezi nimi $n - 1$ blokací, které tvoří kostru L grafu KMG, která, jak již víme, zajišťuje napojení komponent, a tím i přípustnost řešení σ. Celkovou cenu hran kostry

L označme $c_L = \sum r(b)$ pro $b \in E(L)$. Množinu všech blokací z řešení σ označme $B(\sigma)$ a jejich celkovou cenu $c_\sigma = \sum r(b)$ pro $b \in B(\sigma)$. Z nezápornosti funkce r plyne $c_\sigma \geq c_L$. Protože cenu řešení σ lze snížit (tím, že některé z jeho blokací neopravíme), není toto řešení optimální. Tím je dokázána následující věta.

Věta 4 Nechť síť má n komponent souvislosti. Pak řešení úlohy Π_2, které má jiný počet, než $n - 1$, opravených blokací, není přípustné nebo není optimální.

Proto nutnou podmínkou optimality řešení úlohy Π_2 je, že počet odstraněných blokací je o jedna menší, než počet komponent. Tuhle podmínku lze užít jako kontrolu již existujících řešení. V této práci z ní vycházíme při implementaci té části řešiče úlohy, která má na starosti konstrukci tras čet cestářů.

2.5 Složitost úlohy

Na třídě složitosti úlohy závisí způsob, který zvolíme k jejímu řešení, protože pokud úloha patří do třídy NP (definice 51), nechceme strávit věčnost nebo dobu neurčitou čekáním na výsledek. Před tím, než odhadneme složitost úlohy napojování komponent (Π_2), ukážeme si dvě s ní příbuzné (a slavné) úlohy – problém obchodního cestujícího (TSP) a rozvozní problém (VRP). Podaří-li se převést zadání úlohy Π_i na zadánní úlohy Π_j, pak není nutné řešit Π_i, ale stačí Π_i převést na Π_j, vyřešit Π_j a řešení úlohy Π_j převést na řešení úlohy Π_i.

Problém obchodního cestujícího

Traveling Salesman Problem (TSP). Je dáno n měst a matice vzdáleností $l(u, v)$ mezi nimi. $u, v \in 1(1)n$. Najděte kružnici (definice 29), která:
(1) prochází všemi městy
(2) její délka je menší nebo rovna délce libovolné jiné kružnice splňující p.1.
Existuje celkem $(n - 1)!$ různých kružnic. Podle [8] TSP patří mezi NP-úplné úlohy (definice 54), takže se nevěří na existenci algoritmu s polynomiální časovou složitostí (definice 49), který by ke každé instanci úlohy dokázal najít globální minimum. TSP se řeší nejen v logistice, ale i v kontrole kvality výrobků, v astronomii nebo v analýze dat. Pro některé instance byla nalezena globálně optimální řešení pomocí řešiče založeného na deterministických algoritmech vycházejících z lineárního programování (viz např. [11]). Bližší informace ke zmíněnému řešiči čtenář najde v knize [12].

Rozvozní problém

Vehicle Routing Problem (VRP). Je dáno jedno depo, n zákazníků, m vozidel, která z depa vyjíždějí k zákazníkům a matice vzdáleností $l(u, v)$ mezi nimi. $u, v \in 0(1)n$. Vrchol 0 zde značí depo. Najděte m-tici tras (kružnic) takovou, že:
(1) všechny začínají a končí v depu

(2) každý ze zákazníků se nachází na trase právě jednoho vozidla

(3) součet délek tras je menší nebo roven součtu délek tras libovolné jiné přípustné m-tice. Přípustná je ta m-tice, která splňuje p.1 a p.2.

VRP lze převést na TSP tím, že depo nahradíme m-tící dep, která:

(1) se nacházejí na tomtéž místě, díky čemuž se nezmění ta část matice $l(u, v)$, která uchovává vzdálenosti mezi depem a zákazníky

(2) jsou-li vrcholy u, v depa, tak položíme $l(u, v) = \infty$.

Tato transformace VRP na TSP má polynomiální složitost (definice 53). TSP lze převést na VRP také v polynomiálním čase. Proto i VRP patří do třídy NP-úplných úloh. Provede se to tak, že v TSP zvolíme jedno město za depo a vozidlo pak navštíví všech $n-1$ zbývajících měst každé právě jednou. Umíme-li vyřešit jednu z úloh, umíme vyřešit i druhou. Rozvozní problém má mnoho variant, aplikací a metod řešení. Jako přehled může čtenáři posloužit [13].

Varianta VRP, která se více podobá úloze řešené v této práci, ve které uvažujeme obecně $z \geq 1$ stanic, se nazývá Multi Depot Vehicle Routing Problem (MDVRP) a řeší ji např. zde [14].

Odstranění všech blokací (Π_1)

Úloha je popsána v sekci 2.2. Podle toho, zda uvažujeme jednu nebo více stanic cestářů, ji na VRP (resp. MDVRP) převedeme následujícím způsobem. Pro jednoduchost uvažujme jen případ jedné stanice a m čet. Představme si, že stojíme v nějakém vrcholu a konstruujeme trasu tím, že volíme, přes kterou hranu se vydáme dál. Úkolem je opravit všechny blokace. Vrchol, který jsme již jednou navštívili, podruhé už navštívit nesmíme, protože naše předchozí návštěva ho vyřadila ze seznamu dostupných, dosud nenavštívených, vrcholů. Proto jakmile se ocitneme v krajním vrcholu některé blokace, nemáme na výběr a musíme přes tuhle blokaci přejít. Kdyby spolu žádné dvě blokace nesousedily (tj. kdyby neměly ani jeden společný vrchol) mohli bychom teď pokračovat postupem napsaným o dva odstavce níže. Avšak v některých instancích úlohy se vyskytuje aspoň jeden vrchol incidentní s $p > 1$ zablokovanými hranami. Každý takový vrchol nahradíme úplným grafem nad p-tící vrcholů a hrany tohoto úplného grafu ohodnotíme cenou 0. Pokud však byla p-tice blokací incidentní se stanicí, nahradíme ji úplným grafem nad $p + 1$ vrcholy, z nichž jeden je stanice.

Takto modifikovaná instance úlohy vypadá tak, že krajní vrcholy libovolných různých blokací jsou od sebe odděleny neblokovanou hranou. V Π_1 je pevně daný počet b blokací množiny R, které je potřeba opravit, abychom získali přípustné řešení. Silnice je opravena (blokace odstraněna), když se četa přesune z jednoho krajního vrcholu zablokované hrany do druhého krajního vrcholu stejné hrany. V modifikované CG-síti je $2b + 1$ vrcholů.

Úlohu lze řešit dvoufázově. V lichém kroku se vybere jeden z vrcholů modifikované CG-sítě dosud nepřítomný v žádné z m tras nebo se přesuneme zpět do stanice. A v sudém kroku se prostě přesuneme po zablokované hraně z jednoho jejího krajního vrcholu do druhého, protože jinou možnost nemáme. Přítomné blokace nám přímo určují, které hrany modifikované CG-sítě musí řešení úlohy VRP obsahovat. Lichý krok konstrukce tras (kromě návratu do stanice) proběhne b-krát, což odpovídá řešení úlohy VRP nad b vrcholy. Jak je vidět, úlohu Π_1 lze převést na VRP v čase polynomiálně závislém na velikosti úlohy.

VRP se řeší nad úplnou sítí. Nechť má tato síť lichý počet vrcholů $n \geq 3$, z nichž jeden je depo. Když v této síti označíme $\frac{1}{2}(n-1)$ spolu (i s depem) neincidentních hran za povinně přítomné v řešení (tj. zablokované), dostaneme z VRP úlohu Π_1. Z toho plyne, že i úloha Π_1 je NP-úplná.

Spojení komponent (Π_2)

Úloha je popsána v sekci 2.2. Pro jednoduchost uvažujme jen případ jedné stanice a m čet. Konkrétní zvolená kostra $L \in \mathcal{K}(KMG)$ má b hran. Ty dohromady tvoří nějakou množinu A hran postačujících k napojení komponent do souvislého grafu G''. Když položíme $R = A$, řešíme NP-úplnou úlohu Π_1 (opravu všech silnic) nad množinou blokací A.

Kterou z koster zvolit? Tu nejlevnější? Nechť cena kostry L_1 je menší, než cena kostry L_2. Snadno najdete příklad, kdy celková délka tras čet obsahujících mj. i prvky z $E(L_1)$ je větší, než celková délka tras čet obsahujících mj. i prvky z $E(L_2)$. Tedy k vyřešení úlohy Π_2 nám nepomůže nalézt nejlevnější kostru L grafu KMG a nad ní vyřešit úlohu Π_1. Proto se úloha Π_2 nedá rozložit na dvojici úloh „najít nejlevnější kostru KMG“ a Π_1. Protože silniční sítě jsou (téměř) rovinné, tím spíše na nich založené KMG jsou rovinné grafy. Podle [15] pro každý rovinný graf s n vrcholy platí, že nemá více, než $O(5.2852^n)$ koster. Může mít třeba jen jednu, ale mnohem častěji jich má mnohem více a nechceme pro každou z nich řešit Π_1.

Pokud je zadání úlohy Π_2 převoditelné v polynomiálním čase na zadání úlohy Π_1, tak si tím jen potvrdíme, že Π_1 je NP-úplná. Ale to nepotřebujeme. To už víme. A autorovi se během psaní práce ani nepodařilo najít zobrazení Π_2 na Π_1. (To ale neznamená, že to nejde.) Více nás však zajímá, zda je NP-úplná i úloha Π_2. Pokud se podaří najít zobrazení Π_1 na Π_2 vyčíslitelné v polynomiálním čase, tak ano.

Jak již víme, protože v Π_2 jde o napojení komponent, k řešení úlohy není nutné odstranit všechny blokace z R, ale stačí odstranit jen některé – přesněji ty, které jsou hranami libovolné z koster grafu KMG příslušného grafu G' rozpadlé sítě. Řešíme-li Π_1, musí řešení této úlohy obsahovat všechny hrany z R. Převádíme zadání úlohy Π_1 na zadání úlohy Π_2. Obsahuje-li řešení úlohy Π_1 všechny hrany z R, musí všechny

tyto hrany být i součástí řešení právě teď konstruovaného zadání úlohy Π_2. Jenže v Π_2 musí navíc platit, aby celá R byla minimální množinou hran napojující všech $|R| + 1$ nějakých komponent souvislosti. Kdyby nebyla minimální (s vlastností že napojuje všechny komponenty), šlo by ji zmenšit, takže řešení s těmito blokacemi by bylo suboptimální (viz věta 4). Takovou minimální množinou je množina hran $E(L)$ kostry L komponentového multigrafu $KMG(\Pi_2)$ úlohy Π_2. Pokud kostra L bude ještě navíc jediná nejlevnější kostra $KMG(\Pi_2)$, tak postup řešení vhodně zadané úlohy Π_2, ať už nás dovede k libovolnému řešení, každé řešení bude obsahovat právě všechny blokace z množiny R a žádné jiné. Takže takovým způsobem bychom rádi měli zadanou úlohu Π_2.

Převod Π_1 na Π_2 lze provést následovně. K množině R všech blokací z Π_1 vytvoříme $KMG(\Pi_2)$ takový, že v něm všechny hrany z R budou hranami nejlevnější možné kostry. Jak? Zbavíme se kružnic v $KMG(\Pi_1)$ tím, že některé vrcholy rozdělíme na více vrcholů přidáním nových blokací (které nejsou v R). Vznikne $KMG(\Pi_2)$. Například nechť 1–2–3–1 je kružnice v $KMG(\Pi_1)$, které se zbavíme rozdělením vrcholu 1 na vrcholy 1_5 a 1_6, díky čemuž vznikne 1_5–2–3–1_6 v $KMG(\Pi_2)$. Přidáme tam hranu 1_5–1_6, která bude novou (pomocnou) blokací. Nové blokace ohodnotíme nekonečnou dobou opravy, díky čemuž jedinou nejlevnější kostru $KMG(\Pi_2)$ bude ta, která bude obsahovat právě všechny hrany z R. To samo by nestačilo. Důležité je, že pomocné blokace jsou ohodnoceny nekonečnou dobou opravy. Díky tomu se nestanou součástí řešení úlohy Π_2. Matice dob jízd mezi vrcholy CG-sítě z Π_1 zachováme stejnou i pro Π_2. Toto zobrazení Π_1 na Π_2 má polynomiální složitost. Proto i úloha Π_2 je NP-úplná.

Závěry:

(1) Známe-li nějaký způsob řešení Π_2, lze ho použít k řešení Π_1, VRP nebo TSP.

(2) Pokud se nám podaří pomocí deterministického algoritmu vyřešit v polynomiálním čase každou instanci úlohy Π_2, tak se v polynomiálním čase podaří vyřešit také každou instanci TSP a tím dokážeme, že P = NP.

(3) Sázíme na to, že se nám to nepodaří. Proto místo deterministických algoritmů použijeme stochastické. Jim je věnována další sekce.

(4) Nalezení nejlevnější kostry v úplném ohodnoceném grafu je jedna z úloh, u kterých se na první pohled zdá, že vyřešit je bude těžší, než nad tímtéž grafem vyřešit TSP, protože koster má úplný graf více, než kružnic. Ale na druhý pohled existuje polynomiálně složitý algoritmus, který úlohu řeší a známe ho už téměř sto let (viz 1.2). Takže má-li úloha speciální strukturu, možná k ní existuje i algoritmus s polynomiální časovou složitostí. Π_2 taktéž na první pohled vypadá složitěji, než VRP, takže beznadějně. Možná právě díky tomu množství podmínek navíc, které řešení musí splňovat, nakonec bude úloha Π_2 řešitelná deterministickým algoritmem s polynomiální časovou složitostí. Ale je to jen spekulace. Možná půjde o jinou úlohu, než Π_2. Dále se této problematice nevěnujeme. Cíl práce je jiný (viz Úvod).

2.6 Počítačová reprezentace zadání úlohy

V této sekci popíšeme, jakým způsobem lze zadat úlohu našemu řešiči. Úloha Π_2 je popsaná grafem G, ohodnocením c jeho hran dobami transportu v sekundách, množinou B hran, jejíž odebrání (zablokování) z G způsobilo rozpad G na G', ohodnocením $r : B \to \mathbb{R}_0^+$ blokací dobami oprav v sekundách, množinou dep ST – vrcholů, ve kterých se vyskytují stanice cestářů a počty čet h v jednotlivých stanicích.

Síť $(G, \{o\}, \{c\})$ zadáváme pomocí tří souborů: `uzly.txt`, `hrany.txt`, `xy.txt`, uložených v adresáři pojmenovaném `sitXXX`, například `sit001`, `sit002` atd.

`uzly.txt` obsahuje nezáporná celá čísla ve dvou sloupcích pojmenovaných *id, obyvatel*. První sloupec, *id*, obsahuje vzestupně seřazená čísla množiny $0(1)(n-1)$, která slouží jako indexy vrcholů sítě. Každému vrcholu odpovídá počet *obyvatel* o, kteří tam žijí (viz 2.1). Například řádek s hodnotami [209, 1877] znamená, že ve vrcholu 209 žije 1877 obyvatel.

`hrany.txt` obsahuje matici, která má tři sloupce pojmenované *uzel1, uzel2, sekund*. Každý řádek reprezentuje jednu hranu grafu pomocí dvojice proměnných [*uzel1, uzel2*], které nabývají hodnot z množiny $0(1)(n-1)$ podle toho, které vrcholy hrana spojuje. Proměnná *sekund* nabývá kladných celočíselných hodnot a reprezentuje ohodnocení c. Například řádek s hodnotami [3200, 1171, 59] znamená, že po hraně spojující vrchol 3200 s vrcholem 1171 se četa cestářů dostane při obvyklé rychlosti za 59 sekund.

`xy.txt` obsahuje dva sloupce *x, y* a má stejný počet řádků jako `uzly.txt`, aby bylo možné vrcholu na i-tém řádku souboru `uzly.txt` přiřadit hodnoty i-tého řádku souboru `xy.txt`, které mají význam kartézských souřadnic bodu v rovině. Hodnoty *x, y* jsou reálná čísla, v počítači reprezentovaná typem float. S těmito údaji řešič úlohy nepracuje, ale hodí se k nakreslení výsledku.

Nad takto zadanou sítí lze uvažovat různé situace. V adresáři `sitXXX` pro každou situaci zvlášť vytvoříme nový adresář `kYYdZZ`. Ten naplníme soubory `blokace.txt`, `doby_oprav.txt`, `starty.txt` popisujícími konkrétní situaci. `kYYdZZ` je jen naše konvence. YY značí počet komponent, na které se síť rozpadla a ZZ značí počet dep. Název adresáře pro situaci není tak důležitý jako to, že má být uvnitř adresáře pro síť, pro niž danou situaci zadáváme, např. `sit002\k41d07` nebo `sit002\katastrofa`.

`blokace.txt` reprezentuje množinu B. Obsahuje jeden sloupec čísel, která slouží jako indexy do matice hran grafu G. Tyto indexy ukazují, které hrany byly zablokovány. Kdybychom například chtěli zadat, že byly zablokovány hrany na 1., 5. a 9. řádku matice ze souboru `hrany.txt`, měli bychom do souboru `blokace.txt` do sloupce pod sebe zapsat hodnoty 0, 4, 8. (Hodnoty jsou zde o 1 menší, protože používáme pythonovské indexování.) Nutno podotknout, že do souboru nepatří indexy těch hran,

jejichž odebrání nezpůsobí rozpad sítě na nesouvislou síť.

doby_oprav.txt reprezentuje zobrazení r. Místo toho, abychom přiřazovali doby oprav všem hranám grafu G, přiřadíme je jen zablokovaným hranám. Proto doby_oprav.txt má stejný počet řádků jako blokace.txt a má jen jeden sloupec, jehož hodnoty jsou celá nezáporná čísla. Příklad: 4. řádek souboru doby_oprav.txt obsahuje 3205, na 4. řádku souboru blokace.txt je číslo 9 a na (9+1). řádku matice hrany (tj. (9+2). řádku souboru hrany.txt) je trojice [2523, 550, 109]. To znamená, že hrana spojující vrchol 2523 s vrcholem 550 je zablokovaná a odhadem ji potrvá opravit 3205 sekund.

starty.txt obsahuje do sloupce uspořádané indexy těch vrcholů grafu G, ve kterých se vyskytují stanice silničářů. Reprezentuje množinu ST (sekce 2.4).

Když to shrneme, vstupní data od uživatele, bez kterých náš program spočítá nic, jsou data sítě uzly.txt, hrany.txt, xy.txt uložená v adresáři sitXXX a data situace na síti blokace.txt, doby_oprav.txt, starty.txt uložena v adresáři sitXXX\kYYdZZ. Lze je najít v příloze v adresáři Ulohy.

Řešič úlohy ke svému fungování potřebuje ještě datové soubory uKompId.txt, uzlyCG.txt a casyCG.txt, které reprezentují CG-síť $(CG, \{\}, \{l\})$, kde $CG = (BC, SP, \varepsilon)$, viz 2.4. Tyto soubory na základě dat sítě a situace na ní lze vygenerovat spuštěním skriptu vytvorCGdata.py poté, co na jeho začátku uživatel vyplní hodnoty proměnných sit a situace.

uKompId.txt obsahuje čísla množiny $0(1)(k-1)$, kde k je počet komponent v rozpadlé síti. Tato čísla jsou seřazena v jednom sloupci a je jich tam tolik, kolik je vrcholů v grafu G, resp. kolik řádků (bez prvního řádku se jmény sloupců) má soubor uzly.txt. Vrcholu na řádku i souboru uzly.txt patří číslo komponenty, kde se nachází a toto číslo je v souboru uKompId.txt na řádku $i-1$.

uzlyCG.txt obsahuje jeden sloupec pojmenovaný *id*. V něm jsou uloženy indexy vrcholů grafu G, které jsou hraničními vrcholy (definice 35) komponent souvislosti rozpadlého grafu G' nebo ve kterých se nacházejí depa cestářů. Jde tedy o prvky množiny BC indexované čísly vrcholů původní sítě S (nad G).

casyCG.txt uchovává hodnoty matice $l(u, v)$ zavedené v sekci 2.4.

Přípravu posledních tří uvedených souborů si lze zkusit pomocí skriptu vytvorCGdata.py na situaci Ulohy\sit003\k06d02_test. Všechna potřebná data a zdrojové kódy najde čtenář přiloženy na CD.

Poslední vstupní hodnoty, které uživatel musí zadat, jsou počty čet h v jednotlivých

depech. To udělá tak, že změní hodnotu proměnné `poctyTymuVeStanicich` nahoře ve skriptu `hlavni.py`. Pokud například soubor `starty.txt` obsahuje hodnoty 123, 234 a 345, tak proměnná `poctyTymuVeStanicich` musí být seznam tří celých nezáporných čísel (typ int). Konkrétně třeba `poctyTymuVeStanicich = [1,0,4]` znamená, že z vrcholu 123 do práce vyrazí 1 četa, z vrcholu 234 žádná a z vrcholu 345 vystartují 4 čety.

2.7 Počítačová reprezentace řešení úlohy

Když uživatel zadal vše, co zadat měl, stačí v adresáři `Kody` najít a pustit skript `hlavni.py` a počkat na výsledek výpočtu. O způsobech výpočtu a fungování programu budou následující sekce. V této sekci se podíváme na to, jak číst výstupy programu a co jsou jejich teoretické protějšky popsané v sekcích 2.2 a 2.3.

V adresáři `Kody` vedle souborů se zdrojovými kódy se nachází adresář `vysledky`, do kterého skript `hlavni.py` zapisuje výstupy programu. Jedná se o tyto soubory: `reseniCGnn.csv`, `reseniGnn.csv`, `reseniPBnn.csv`, `reseniPB_vsechna.csv`, `ceny_vsechny.csv`, `dobyPraci_vsechny.csv`. Tamtéž jsou zapsány také obrázky `komponenty.svg` a `reseniGnn.svg`. (Viz obrázek 5.)

Uvažujme situaci, kdy m čet má za úkol propojit spolu k komponent souvislosti. Program hledá m-tici kružnic, přípustnou a optimální vzhledem k cenové funkci $p(\sigma) = q_3(\overline{T}(\sigma))$ (definice 55) v CG-síti. Nejlepší nalezené řešení je uloženo do souboru `reseniCGnn.csv`. Máme-li `poctyTymuVeStanicich = [1, 1, 1]` pro `startyCG = [172, 173, 174]`, může obsah souboru `reseniCGnn.csv` vypadat například takto:

```
"[[172, 126, 37, 57, 137, 20, 24, 115, 18, 172], [173, 48, 133,
158, 87, 163, 93, 173], [174, 122, 33, 148, 73, 150, 77, 174]]"
"[[0, 1, 0, 1, 0, 1, 0, 1, 0], [0, 1, 0, 1, 0, 1, 0], [0, 1, 0,
1, 0, 1, 0]]"
```

Jsou to v jednom sloupci uložené dva řetězce. Řetězec na prvním řádku obsahuje trojici seznamů vrcholů CG-sítě. Např. třetí seznam začíná a končí ve vrcholu číslo 174 a obsahuje informace pro třetí četu o tom, přes které vrcholy CG-sítě má projet. Řetězec na druhém řádku obsahuje trojici seznamů oprav. 0 znamená pouze projet, 1 znamená opravit hranu, odstranit blokaci. Všimněte si, že i-tý seznam z druhého řádku má vždy o jedno méně čísel, než i-tý seznam z prvního řádku. To proto, že z definice cesty plyne, že cesta má vždy o jedna menší počet hran, než jaký má počet vrcholů. Třetí trasa začíná [174,122,33,148,...] a jí odpovídající třetí seznam oprav začíná [0,1,0,...] a znamená to, že mezi vrcholy 174 a 122 CG-sítě se 3. četa pouze přesouvá po nejkratší cestě grafu G a případné blokace na té cestě ignoruje, vrcholy 122 a 33 jsou krajní

vrcholy zablokované hrany, kterou má tato četa opravit a hraně [33,148] v seznamu oprav odpovídá 0, což znamená, že jde opět jen o transport po nejkratší cestě mezi těmito vrcholy. Z uvedeného příkladu se může zdát, že seznamy oprav jsou zbytečnou součástí řešení, protože se v nich stále střídají nuly s jedničkami, ale není tomu tak. Jsou případy, kdy některý ze seznamů oprav začíná například takto: [0,1,1,0,1,0,...]. Jedná se o řešení úlohy, ve které se síť rozpadla do jedenácti komponent souvislosti. Podle nutné podmínky optimality (věta 4) víme, že kdyby součet jedniček na druhém řádku souboru `reseniCGnn.csv` byl jiný, než deset, určitě by se nejednalo o řešení optimální.

`reseniGnn.csv` obsahuje totéž řešení, jako předchozí soubor, v tomtéž způsobu zápisu, ale v podrobnější podobě, protože nalezená m-tice kružnic v CG-síti je zde nahrazena m-tící sledů W_k hran v původní síti S (resp. G) a jim odpovídají nové seznamy oprav. To se hodí přímo cestářům, ať ví, kudy jet. Řešení v podrobnější podobě zabírá půl stránky (23 řádků), tak nechť zájemce nahlédne do adresáře `vysledky` a v něm do souboru `reseniGnn.csv`. Upozorňujeme, že tentýž vrchol má v CG jiné číslo, než jaké má v G.

`reseniPBnn.csv` obsahuje nejlepší nalezené řešení, totéž co je obsaženo v předchozích dvou souborech, tentokrát však ve formě pořadí indexů opravených blokací. Obsah souboru je tento:

```
"[45, 71, 28, 20]"
"[60, 107, 115]"
"[37, 91, 95]"
```

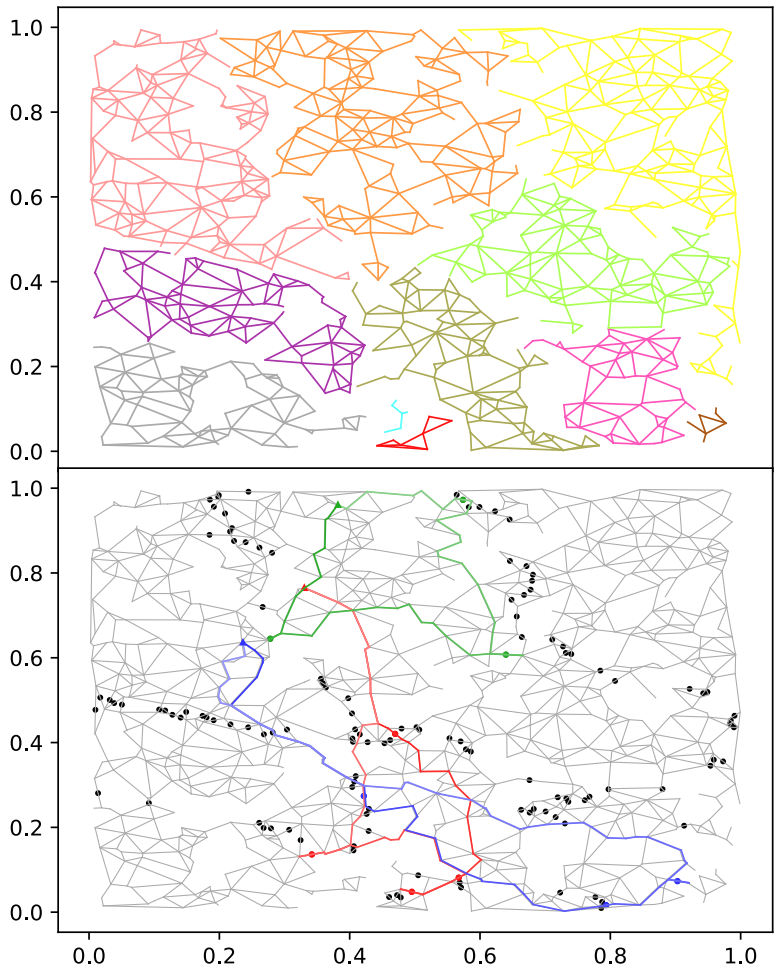
Jsou to indexy řádků do vstupního souboru `blokace.txt`, přičemž indexování začíná nulou. Na prvním řádku souboru `reseniPBnn.csv` je, že první četa má nejdříve opravit blokaci, na niž ukazuje index 45, poté blokaci 71, pak 28 a 20. Kudy se mezi nimi přepraví, to umí dopočítat funkce `resenizPoradiBlokaci()` ze skriptu `pomocne.py`. Druhá četa dostala za úkol opravit blokace z druhého řádku a třetí ze třetího. Tato reprezentace řešení má svůj protějšek v m-tici permutací $(\pi_1, \ldots \pi_m)$ prvků množin $A_1, \ldots, A_m$ tvořících rozklad některé množiny A blokací postačujících k napojení komponent (viz sekce 2.2). V tomto případě je $A_1 = \{45, 71, 28, 20\}$, $A_2 = \{60, 107, 115\}$, $A_3 = \{37, 91, 95\}$ a A_k lze různě permutovat a tím získávat různá pořadí návštěv zablokovaných hran.

`reseniPB_vsechna.csv` obsahuje všechna řešení získaná programem. Na každém řádku je jedno řešení. Řešení σ v právě uvedeném formátu pořadí blokací jsou seřazena vzestupně podle ceny $p_3(\sigma) = q_3(\overline{T}(\sigma))$. Na prvním řádku je to nejlepší nalezené – "[[45, 71, 28, 20], [60, 107, 115], [37, 91, 95]]"a na posledním to nejhorší nalezené.

`dobyPraci_vsechny.csv` obsahuje na i-tém řádku m-tici dob prací jednotlivých čet, která patří řešení z i-tého řádku souboru `reseniPB_vsechna.csv`.

Například na prvním řádku jsou tyto doby prací: `13279,9046,13255`, což znamená, že četa, která opravovala blokace [37, 91, 95] v tomto pořadí, strávila prací (včetně transportu) 13255 sekund.

$\overline{T}(\sigma) = (13279, 9046, 13255)$. $q_3(\overline{T}(\sigma)) = (13279, 13255, 9046)$. $q_1(\overline{T}(\sigma)) = 35580$. Napíšeme-li za sebe p_3 a p_1, dostaneme `13279,13255,9046,35580` – první řádek souboru `ceny_vsechny.csv`, který obsahuje v prvních m sloupcích ceny p_3 všech řešení řazené lexikograficky (definice 45) vzestupně spolu s cenami p_1 v $(m + 1)$. sloupci. Takže podle tabulky v `ceny_vsechny.csv` je možné vybrat si několik řešení s nízkou cenou p_3 takových, která mají i nízkou cenu p_1.



Obrázek 5: SVG výstupy. V horní části barevně odlišené komponenty, v dolní reseniGnn. Černé kroužky značí blokace, barevné kroužky odstraněné blokace, trojúhelníky depa.

3 Metody řešení úlohy

Naše úloha Π_2 se ze známých úloh podobá nejvíce úloze VRP. Pro VRP existuje mnoho metod řešení, které lze přizpůsobit k řešení úlohy Π_2. Zde si ukážeme metody, které jsme si zvolili k řešení úlohy. Volba metod k řešení naší úlohy se odvíjí od metod použitých v [2] pro možnost srovnání různých implementací.

Volíme heuristické metody. Heuristiky jsou algoritmy, jejichž cílem je získat rychle dobré řešení, bez ambice získat řešení optimální. Co považuje autor za rychlé, uvedl již v úvodu. Za dobré řešení autor této práce považuje řešení, jehož cena je lepší, než očekávaná cena náhodného řešení.

3.1 Algoritmus úspor

Algoritmus úspor (AU) pochází od [16] z roku 1964. Od té doby vzniklo několik modifikací. Je to heuristika, kterou používáme k získání přípustného řešení VRP. S její pomocí získáme přímo optimální řešení jen ve výjimečných případech. Některé důmyslnější algoritmy vyžadují jako jeden ze vstupů počáteční aproximaci řešení a tu lze získat s pomocí AU.

Mějme úlohu VRP s jedním depem (značeným 0), n zákazníky a m vozidly, která povezou zboží z depa k zákazníkům. Matice $C := (c(i, j))_{i,j=0}^{n}$ uchovává délky (nebo ceny) nejkratších cest mezi dvojicemi vrcholů i, j.

Myšlenka AU spočívá ve slučování tras, kterých je na začátku n a na konci m. Má-li být sloučení dvou tras do jedné provedeno, mělo by při tom dojít k úspoře, jinak to nemá cenu dělat. Každá z počátečních n tras má tvar 0–i–0, kde $i \in 1(1)n$. Nechť 0–j–0 je druhá trasa, pro $j \in 1(1)n$, $j \neq i$. Trasa třetí, která vznikne jejich sloučením, vypadá tak: 0–i–j–0. Cena první trasy je cena(i) = c(0,i) + c(i,0) a cena druhé je cena(j) = c(0,j) + c(j,0). Celková cena je cena(i) + cena(j). Kolik ušetříme, když první dvě nahradíme třetí? cena(ij) = c(0,i) + c(i,j) + c(j,0).
Úspora(ij) = cena(i) + cena(j) - cena(ij) = c(i,0) + c(0,j) - c(i,j).

Algoritmus 2 (Algoritmus úspor)
Dokud platí, že $n > m$, opakujeme kroky (1 až 4):

(1) Máme n tras Tr a pro každou dvojici i, $j \in 1(1)n$ spočítáme úspora(ij).

(2) Vybereme tu dvojici i, j, pro niž je úspora(ij) největší a dvojici tras Tr(i), Tr(j) sloučíme do jedné trasy Tr(ij).

(3) Ze seznamu dostupných tras vymažeme trasy Tr(i), Tr(j) a přidáme do něj trasu Tr(ij).

(4) Položíme n rovno počtu tras v seznamu.

Každou trasu reprezentujeme posloupností vrcholů, zde posloupností čísel z množiny $0(1)n$. Každá taková posloupnost začíná a končí 0. Například Tr(1) = [0, 1, 2, 7, 5, 0] a Tr(2) = [0, 4, 3, 0]. Sloučení dvou tras lze provést čtyřmi způsoby, přičemž vybíráme ten, který s sebou nese největší úsporu. Podle toho, jak Tr(1) sloučíme s Tr(2), vznikne [0, 1, 2, 7, 5, 4, 3, 0], nebo [0, 1, 2, 7, 5, 3, 4, 0], nebo [0, 5, 7, 2, 1, 4, 3, 0], nebo [0, 5, 7, 2, 1, 3, 4, 0].

Tak je tomu pro VRP. V úloze Π_2 bylo potřeba zařídit, aby algoritmem úspor nalezené řešení bylo přípustné a zároveň bylo potřeba provést modifikaci tohoto algoritmu pro situace s více depy. Kód je v souboru `reseniA.py` v adresáři `Kody`.

Přípustnost řešení
Ze sekce 2.4 víme, že řešení sestavené z tras, mezi jejichž hranami nejsou blokace KMG, které stačí k vytvoření kostry grafu KMG, není přípustné. Aby bylo přípustné, stačí, když na jeho trasách budou ležet zablokované hrany (a tyto budou opraveny), které tvoří některou z koster KMG. Aby AU vracel řešení přípustná v Π_2, pracujeme nad krajními vrcholy (více jich není třeba) hran nejlevnější kostry KMG s následující úpravou AU. Je-li do trasy zvolen jeden krajní vrchol blokace, pak je do stejné trasy zvolen i druhý krajní vrchol.

Více dep
Vrcholy pro depa označme D1 až Dp. Vrcholy CG-sítě značme i nadále hodnotami z $1(1)n$. Kdybychom nepotřebovali trvat na tom, že oba krajní vrcholy blokace musí ležet ve stejné trase, postupovali bychom následovně. Pro každý vrchol $i \in 1(1)n$ bychom spočítali vektor $d_i := (c(i, D1), \ldots, c(i, Dp))$. Zjistili bychom, na které pozici má d_i nejmenší hodnotu – řekněme, že na pozici j. Vrchol i bychom přiřadili k depu Dj. Takhle bychom každý z vrcholů přiřadili k některému z dep podle toho, ke kterému to má nejblíže. Připouštíme možnost, že některá z dep nebudou mít ve svém rajonu žádný vrchol z $1(1)n$. Množinu vrcholů přiřazených depu Dj nazýváme rajon depa Dj. A na konec bychom pro každý z rajonů použili AU pro jedno depo popsaný výše.

Ale protože potřebujeme zachovat přípustnost řešení, tak v případě více dep zajistíme, aby oba krajní vrcholy blokace ležely ve stejném rajonu. Ve fázi přiřazování vrcholů rajonům dep zde také vypočítáme vektory d_i, ale jiným způsobem. Vrchol i_1 totiž leží na blokaci spolu s druhým vrcholem i_2. Potom pro $i = i_1$ i pro $i = i_2$ počítáme $d_i := (c(i_1, D1) + c(i_2, D1), \ldots, c(i_1, Dp) + c(i_2, Dp))$.

3.2 Zlepšovací heuristiky

Pokud jsme již nějakým způsobem získali přípustné řešení úlohy, můžeme se pomocí některé zlepšovací heuristiky podívat, zda v okolí tohoto řešení existuje nějaké lepší řešení. Pokud ano, získáme lokálně optimální řešení. Optimalita závisí na definici okolí řešení.

První, snad nejjednodušší, zlepšovací metoda pochází z roku 1958 od [17]. Je založena na opakování operace tehdy nazývané inverze tak dlouho, dokud nějakou inverzi v některé části řešení provést lze při současném snížení ceny řešení. Jako příklad uvažujme řetězec vrcholů $[1, 2, 3, 4, 5, 6, 7, 8, 9]$, který tvoří část cesty CG-sítí jedné z čet. To znamená, že v řešení jsou přítomny hrany $[1, 2]$, $[2, 3]$, $[3, 4]$, $[4, 5]$, $[5, 6]$, $[6, 7]$, $[7, 8]$, $[8, 9]$. Příkladem inverze v uvedeném řetězci je $[1, 2, 3, 6, 5, 4, 7, 8, 9]$. Vznikla vynecháním hran $[3, 4], [6, 7]$, inverzí řetězce $[4, 5, 6]$ v řetězec $[6, 5, 4]$ a opětovným napojením tohoto řetězce ke zbytku pomocí nových hran – $[3, 6], [4, 7]$. Na tuhle operaci inverze části řetězce se můžeme dívat i jiným způsobem, a sice jako na výměnu dvou hran za jiné dvě hrany, v řešení nepřítomné. Hrany $[3, 4], [6, 7]$ byly nahrazeny hranami $[3, 6], [4, 7]$.

Z této dříve ruční práce se později stal 2-opt algoritmus, který slouží ke zlepšení daného řešení $\sigma = (\pi_1)$ úlohy TSP nebo každé trasy $\pi_1, \ldots, \pi_m$ úlohy VRP zvlášť. S menšími úpravami, které popíšeme později, slouží 2-opt také ke zlepšení jednotlivých tras čet cestářů v úloze Π_2. 2-opt používáme na kružnice (definice 29). 2-opt však funguje i v případě cest (definice 24) a dokonce i v případě nesouvislého grafu, který vznikne rozpadem cesty na komponenty souvislosti, z nichž každá má aspoň dva vrcholy, jak jsme později zjistili při implementaci 2-opt pro úlohu Π_2. U každého řešení, které máme v úmyslu zlepšit pomocí 2-opt, předpokládáme, že každý z vrcholů tohoto řešení je incidentní aspoň s jednou hranou a nejvýše se dvěmi hranami.

Pro jednoduchost popisu algoritmu zde budeme řešení uvažovat jako posloupnost navzájem různých vrcholů $v_1, \ldots, v_n$ zapsanou v řetězci $\pi = [v_1, \ldots, v_n]$, jehož význam spočívá v tom, že $\forall i \in 1(1)(n-1)$ $[v_i, v_{i+1}]$ značí hranu přítomnou v řešení. Cenu řešení spočítáme jako cena$(\pi) = \sum_{i=1}^{n-1} l(v_i, v_{i+1})$, kde l je ohodnocení hran CG-sítě dobami jízd (sekce 2.4).

Algoritmus 3 (**2-opt**)
Nastavíme proměnnou zkraceno = True.
Dokud platí, že zkraceno = True, opakujeme kroky (1 až 5):
(1) Nastavíme zkraceno = False.
(2) $\forall i \in 1(1)(n-2)$ $\forall j \in (i+2)(1)n$ spočítáme hodnotu zkrácení způsobenou případnou výměnou hran: zkraceni(i,j):= $l(v_i, v_{i+1}) + l(v_j, v_{j+1}) - l(v_i, v_j) - l(v_{i+1}, v_{j+1})$.
(3) Vybereme tu dvojici $[i, j]$, pro niž je zkraceni(i,j) největší a uložíme
do proměnné kde.
(4) Je-li největší zkrácení větší, než 0, tak ze vstupní trasy
odstraníme hrany $[v_i, v_{i+1}]$ a $[v_j, v_{j+1}]$ a nahradíme je hranami $[v_i, v_j]$ a $[v_{i+1}, v_{j+1}]$.
(5) Je-li největší zkrácení větší, než 0,
tak položíme zkraceno = True, jinak zkraceno = False.

Na krok (4) algoritmu se lze dívat jednodušeji, totiž jako na inverzi trasy $[v_1, \ldots, v_i, v_{i+1}, \ldots, v_j, v_{j+1}, \ldots, v_n]$ v trasu $[v_1, \ldots, v_i, v_j, \ldots, v_{i+1}, v_{j+1}, \ldots, v_n]$. 2-opt má polynomiální složitost, konkrétně $O(n^2)$. Následují definice a věty z [18].

Definice 58 Nechť Π je některá z úloh TSP, VRP, Π_1 nebo Π_2 a p je cenová funkce (definice 55) řešení této úlohy, kterou se snažíme minimalizovat. Řekneme, že přípustné řešení σ úlohy Π je λ-optimální (nebo λ-opt), pokud z něj nelze získat žádné jiné přípustné řešení s menší cenou pomocí náhrady λ hran jinými λ hranami.

Věta 5 Mějme úlohu Π. Každé její řešení je 1-optimální.

Věta 6 Nechť n je počet hran řešení úlohy Π. Řešení je optimální právě tehdy, když je n-optimální.

Věta 7 Označme C_λ množinu λ-optimálních řešení úlohy Π. Pak platí $C_1 \supseteq C_2 \supseteq \ldots \supseteq C_{n-1} \supseteq C_n$.

Řešení získané algoritmem 2-opt je 2-optimální. Není však vyloučeno, že výměnou 3 hran v σ získáme σ', které má menší cenu, než σ. Takže σ nemusí být 3-optimální nebo obecně λ-optimální pro $\lambda > 2$.

Abychom zajistili 3-optimalitu řešení σ, necháme jej projít 3-opt algoritmem. Pokud zůstane stejné jako na začátku, tak je 3-optimální a pokud získáme jiné řešení, tak to nové bude 3-optimální. Algoritmus 3-opt funguje stejně jako 2-opt s tím rozdílem, že vybírá trojice hran a nahrazuje je jinými trojicemi hran tak, aby byla zachována přípustnost zlepšeného řešení. Výpočet zkrácení má ve 3-opt více členů, než ve 2-opt. Algoritmus je složitější a lepší, než popisovat ho zde, bude, když si čtenář v příloze práce vyhledá přímo jeho implementaci. Nachází se v souboru `reseniZ.py` ve složce `Kody`. Ve 2-opt algoritmu je jen jedna možnost, jak přípustným způsobem provést náhradu dvojice hran, ve 3-opt algoritmu je těch možností přípustné náhrady trojice hran novými sedm, z nichž jedna odpovídá výměně pouze dvojice hran za dvě nové.

3-opt algoritmus má polynomiální složitost, konkrétně $O(n^3)$. Stále to není exponenciální složitost, ale i tak takto složité výpočty zaberou mnoho času. Pro naši úlohu Π_2 a sítě, nad kterými pracujeme a situace, které řešíme, se to čekání stále ještě dá vydržet. O tom později. Z počátečního řešení σ získáme pomocí 3-opt řešení σ_3 a pomocí 2-opt řešení σ_2. Pak platí $p(\sigma_3) \leq p(\sigma_2)$.

Pro dvě různá řešení $\sigma_1, \sigma_2 \in C_3$ může platit $p(\sigma_1) > p(\sigma_2)$. Jde o to, že výměnou 3 hran ze σ_1 nedostaneme σ_2, protože σ_1 je již 3-optimální. Když tedy pomocí 3-opt algoritmu získáme σ_1, které ještě není optimální (n-optimální), co můžeme udělat pro to, abychom získali 3-optimální σ_2, které má menší cenu, než σ_1? Musíme se v prostoru všech přípustných řešení úlohy Π dostat dost daleko od σ_1, aby nás tam vzápětí algoritmus 3-opt nevrátil. To lze dělat různě. Například vygenerovat náhodné přípustné řešení. My k tomu využijeme algoritmy uvedené v následujících sekcích.

3.3 Mravenčí algoritmy

Když řešíme úlohu najít nejkratší uzavřenou cestu (kružnici) v úplném grafu procházející všemi jeho vrcholy, můžeme volit náhodné cesty a ty pak zlepšit pomocí některého λ-opt algoritmu a vybrat z nich tu nejlepší. Lepší by možná bylo, kdyby ty náhodné cesty nebyly náhodné úplně, ale spíše kdyby očekávaná hodnota jejich délky byla vychýlena více směrem k nule, tedy kdyby generátor náhodných cest generoval spíše kratší cesty a teprve na ně bychom pustili některý λ-opt algoritmus. Takže vzniká otázka: „Jak zařídit, aby generátor náhodných cest poskytoval častěji kratší cesty?“

Výše popsaná úloha je problém obchodního cestujícího. Když se obchodník nachází ve městě i, má na výběr z $n - i$ následujících měst, která ještě nenavštívil. Mohl by si vybrat prostě to nenavštívené, které je nejblíže, ale takto by nezískal řešení náhodným způsobem. Nebo by mohl vybrat se stejnou pravděpodobností libovolné z nenavštívených. Nebo by z nenavštívených měst mohl vybrat některé náhodně, při čemž pravděpodobnost výběru města by byla nepřímo úměrná jeho vzdálenosti. ... nebo třeba nepřímo úměrná vzdálenosti umocněné na $\beta \in \mathbb{R}$. Ochromen tolika možnostmi, rezignuje na práci, povalí se na trávník a jen tak hledí, do vzduchu, do země a do zorného pole mu vlezou mravenci.

Chování mravenců

Mravenci musí na své cestě za potravou občas překonávat vzdálenosti od jednoho do sta metrů. Zrak jim při orientaci mezi trsy trávy v trávníku příliš nepomůže. Do sta metrů se mravenec vejde tolikrát, kolikrát člověk do deseti kilometrů. Představte si chodit z domu do práce deset kilometrů temným lesem bez GPS a bez mapy. Jak mravenec dokáže neztratit se? Odpovědí je, že zanechává feromonové stopy, které se odpařují dostatečně pomalu na to, aby je ostatní mravenci mohli cítit a orientovat se podle nich.

Goss a spol. [19] v roce 1989 experimentovali s mravenci argentinskými, kteří vylučují feromony nejen cestou od potravy k mraveništi, ale i opačným směrem. Nechali chodit mravence z jedné části arény do druhé přes most, který se uprostřed větvil na delší a kratší cestu a kousek dál tyto dvě větve opět splývaly v jednu. Sledovali průměrný tok mravenců oběma větvemi. Po čase se stalo to, že tok v kratší větvi byl větší, než tok v delší větvi. Toto globální chování, jak si ověřili, nemá nic společné s viděním ani s pamětí, ale pouze s orientací každého jednotlivého mravence podle lokální informace – tedy koncentrace feromonů, podle jejíž velikosti mravenec upřednostní jednu z cest, kdykoliv má na výběr. Na delší cestě nestihne vzniknout takový tok mravenců, jako na kratší. Na kratší cestu je díky většímu toku tedy vyloučeno více feromonu. Tím se kratší cesta stane lákavější pro více mravenců. A na konec po ní chodí skoro všichni.

Za pozornost stojí, že tento jev – častější vznik krátkých koridorů – není nijak zapsán v mravenci samotném, aby se každý individuálně mohl rozhodnout, po které trase bude cestovat, ale vzniká jako nezamýšlený důsledek interakce mezi jednotlivci. Tohle je jeden z příkladů samo-organizace a emergentních jevů, kterých je příroda plná. Když nepochopíme jejich vznik, nepochopíme ani, co je to vědomí a co je to duše.

Další provedený experiment, o němž [19] referuje, zkoumá, zda mravenci začnou chodit krátkou cestou poté, co vznikl koridor na delší cestě. Experimentátoři odstranili krátkou větev mostu, takže mravencům zůstala jen ta dlouhá, kterou začali používat. Po nějaké době přidali i krátkou cestu, aby mravenci měli na výběr. Sice tu a tam některý z mravenců zabloudil do kratší cesty, ale feromonová stopa, kterou zanechal byla ve srovnání se stopou na již ustaveném koridoru na delší cestě tak malá, že skoro všichni mravenci, ačkoliv měli na výběr, upřednostnili tu, na které je větší koncentrace feromonu – tedy tu delší.

Ant Colony Optimization

Mnoho jednoduchých jednotlivců, zanechávání (v čase slábnoucích) stop v prostředí, sčítání stop od více jednotlivců, volba dalšího kroku jednotlivce podle nejintenzivnější stopy. Tyto principy chování mravenčích kolonií inspirovaly informatiky ke tvorbě celé třídy algoritmů, které stručně nazýváme mravenčí algoritmy. Přišel s nimi Dorigo a spol. (viz [21] a [22]) v roce 1991. Algoritmy této třídy se dají popsat obecně pomocí metaheuristiky Ant Colony Optimization (ACO), optimalizace algoritmem mravenčí kolonie. Zde si uvedememe algoritmus ACO ([20], s. 38). Tato metaheuristika poslouží jako šablona k návrhu dalších heuristik, které patří do třídy mravenčích algoritmů.

Algoritmus 4 (Metaheuristika ACO)
(1) Rozvrhneme následující aktivity (2,3,4) a pak je opakovaně vykonáváme.
(2) Každý mravenec sestrojí své řešení.
(3) Aktualizují se feromony na součástech řešení.
(4) Proběhnou dodatečné akce.

Poznámky k jednotlivým krokům:

(1) Rozvrh aktivit. Sem patří například, kolik mravenců bude konstruovat své řešení, kdy bude probíhat změna hodnot feromonů na součástech řešení, kolikrát celkem se kroky 2,3,4 provedou, atd.
(2) V tomto kroku mravenci současně konstruují svá řešení nad grafem, který reprezentuje řešenou úlohu. Dělají to tak, že se na základě informace lokalizované ve vrcholech grafu rozhodují, do kterého sousedního vrcholu se přesunou, a tedy která hrana se stane novou součástí řešení. Informace dostupná ve vrcholu, kde se právě mravenec nachází, se skládá z množství feromonů na hranách incidentních s tímto vrcholem a z dodatečné (tzv. heuristické) informace na těchto hranách.

(3) V tomto kroku proběhnou změny hodnot feromonů na hranách grafu, který reprezentuje úlohu. Na některé hrany feromon přibude podle toho, kolik mravenců přes ně prošlo. Čím více feromonu na hraně bude, tím větší je pravděpodobnost, že si tuhle hranu příště některý z mravenců zvolí. Hrany, které jsou součástí lepších řešení dostanou více feromonů. Část feromonů se odpaří, což umožňuje algoritmu prozkoumat větší část prostoru řešení před tím, než zkonverguje nebo skončí.
(4) Mezi dodatečné akce se řadí akce globálního charakteru, které mravenec nedokáže udělat sám. Příkladem je výběr mravence, který našel nejlepší řešení. Tomu pak bude dovoleno přidat feromon na hrany svého řešení. Jinou takovou akcí je aplikace zlepšovací heuristiky na nejlepší v iteraci nalezené řešení.

To, jak konkrétně metaheuristika ACO vypadá, závisí také na úloze, kterou chceme řešit. Pro úlohu Π_2 se ACO příliš neliší od ACO pro úlohu TSP, pro niž má následující tvar, a pro niž ukážeme několik heuristik vycházejících z ACO.

Algoritmus 5 (Metaheuristika ACO pro statické úlohy)
(1) Nastavíme parametry, připravíme matici feromonů, případně další data, jako např. heuristickou informaci nebo počáteční řešení.
(2) Dokud není splněna ukončovací podmínka, opakujeme kroky 3,4,5.
(3) Každý mravenec sestrojí své řešení.
(4) Zlepšíme mravenčí řešení zlepšovací heuristikou.
(5) Aktualizujeme matici feromonů.

Aby letadlo mohlo létat, nemusí mávat křídly jako pták. Už zde v algoritmu 5 lze vidět přístup, kdy se při návrhu algoritmů nedržíme striktně toho, co se dá pozorovat v přírodě na mravenčích koloniích, ale napodobujeme jen to, co se nám hodí. U řešení TSP pomocí ACO se zjistilo, že algoritmus vrací lepší výsledky, když aktualizuje hodnoty feromonů až po konstrukci celého řešení, nikoliv během konstrukce s volbou každé hrany, která tvoří řešení ([20], s. 70). Naproti tomu mravenci argentinští zanechávají feromonové stopy stále. Ti ale neřeší TSP.

I nadále čerpáme z knihy [20]. Uvedeme si několik konkrétních heuristik, které patří do třídy ACO. Pro ten účel předpokládejme, že řešíme úlohu TSP nad úplným grafem K_n s n vrcholy a s hranami ohodnocenými délkami $D = (d_{ij})_{i,j=1}^{n}$, kde $d_{ij} = d(i, j)$ pro $i, j \in V(K_n)$. Řešení k-tého mravence zde značíme T^k a jeho cenu C^k. Označme $\mathcal{N}_i^k$ množinu vrcholů dosud nenavštívených mravencem k, který stojí ve vrcholu i. Počet mravenců zde uvažujeme m. Nechť je každá z hran $[i, j] \in E(K_n)$ označena množstvím τ_{ij} feromonu. $\Delta\tau_{ij}^k$ značí množství feromonu, které na hranu $[i, j]$ přidá k-tý mravenec. ρ udává část feromonu, která se odpaří. Platí $0 < \rho < 1$. η_{ij} nese tzv. heuristickou informaci. $\alpha, \beta > 0$ jsou koeficienty, s jejichž pomocí lze stanovit, zda bude větší důraz při konstrukci řešení kladen na feromonovou stopu nebo na heuristickou informaci.

Ant System (AS)

Konstrukce trasy. V AS mravenci konstruují své trasy současně. k-tý mravenec stojící ve vrcholu i vybere jeden ze sousedních vrcholů $j \in \mathcal{N}_i^k$ s pravděpodobností p_{ij}^k definovanou následujícím způsobem:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta} \tag{3.4}$$

V případě AS klademe $\eta_{ij} = 1/d_{ij}$.
Update feromonu proběhne poté, co každý mravenec dokončil konstrukci své trasy, ve dvou fázích. Nejprve necháme část feromonu z každé hrany odpařit.

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij}, \quad \forall [i, j] \in E(K_n) \tag{3.5}$$

Po odpaření feromonu každý mravenec vyloučí určité množství feromonu na ty hrany, přes které prošel, na každou stejné množství, avšak každý mravenec jiné.

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k, \quad \forall [i, j] \in E(K_n) \tag{3.6}$$

přičemž

$$\Delta\tau_{ij}^k = \begin{cases} 1/C^k & \text{pro} [i, j] \in T^k \\ 0 & \text{jinak} \end{cases} \tag{3.7}$$

Hrana, kterou použilo více mravenců a která je částí více tras, má větší šanci, že si ji příště zvolí některý z mravenců. Pro malé instance TSP funguje AS dobře, ale pro větší hůře. Proto vznikly i jiné heuristiky z třídy ACO.

Elitist Ant System (EAS)

Označme T^{bs} dosud nejlepší nalezenou trasu a C^{bs} jeho cenu. Myšlenkou elitářského mravenčího systému (EAS) je značit intenzivněji hrany, které patří do T^{bs}. Konstrukce tras probíhá stejně, jako v případě AS (3.4). Totéž platí pro odpar feromonů (3.5). Označení hran feromony od mravenců proběhne podobně jako v (3.6), ale s jedním členem navíc:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k + e\Delta\tau_{ij}^{bs}, \quad \forall [i, j] \in E(K_n) \tag{3.8}$$

přičemž $e > 0$ je váha kladená na ohodnocení hran, které se vyskytují v řešení T^{bs} a přírůstek feromonu na hranách je:

$$\Delta\tau_{ij}^{bs} = \begin{cases} 1/C^{bs} & \text{pro } [i, j] \in T^{bs} \\ 0 & \text{jinak} \end{cases} \tag{3.9}$$

Je-li parametr e nastaven vhodně, dává EAS lepší výsledky, než AS, a během menšího počtu iterací.

Rank-Based Ant System (RAS)

RAS zobecňuje EAS. Konstrukce tras probíhá stejně, jako v případě AS (3.4). Totéž platí pro odpar feromonů (3.5). Poté, co mravenci zkonstruují svá řešení, jsou seřazeni vzestupně podle ceny řešení a je vybráno pouze $w-1$ prvních, kterým bude dovoleno označit hrany vyskytující se na jejich trasách. Označení hran feromony od mravenců proběhne následovně:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{r=1}^{w-1}(w-r)\Delta\tau_{ij}^{r} + w\Delta\tau_{ij}^{bs} \tag{3.10}$$

Dosud nejlepší mravenec označí hrany ze své trasy v každé iteraci. Vůbec se nemusí jednat o mravence z aktuální iterace. RAS vrací trochu lepší výsledky, než EAS.

Max-Min Ant System (MMAS)

Konstrukce tras probíhá stejně, jako v případě AS (3.4). Totéž platí pro odpar feromonů (3.5). Po zkonstruování řešení MMAS podobně jako EAS a RAS využívá T^{bs} – dosud nejlepší nalezené řešení a spolu s tím využívá i T^{ib} – nejlepší řešení nalezené v aktuální iteraci m mravenci. Během aktualizace matice feromonu se využívá (náhodně) střídavě T^{bs} a T^{ib}. Aby nedošlo rychle ke stagnaci algoritmu v jedné oblasti prostoru řešení, zavádí MMAS omezení na maximální τ_{max} a minimální τ_{min} množství feromonu, které se na hranách může vyskytovat. Překročí-li během aktualizace feromon tyto meze, je jeho množství korigováno tak, aby $\forall[i,j] \in E(K^n)\ \ \tau_{min} \leq \tau_{ij} \leq \tau_{max}$. Na začátku jsou všechny hodnoty τ_{ij} nastaveny na τ_{max}. Pokud dojde ke stagnaci, provede se restart matice $(\tau_{ij})_{i,j=1}^{n}$. Aktualizace feromonu proběhne následovně:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{best}, \quad \forall[i,j] \in E(T^{best}) \tag{3.11}$$

kde s pravděpodobností g je $T^{best} = T^{ib}$ a $\Delta\tau_{ij}^{best} = 1/C^{ib}$
a s pravděpodobností $1-g$ je $T^{best} = T^{bs}$ a $\Delta\tau_{ij}^{best} = 1/C^{bs}$,

kde g je volný parametr. Korekce hodnot feromonu proběhne podle vzorce:

$$\tau_{ij} \leftarrow \min\{\max\{\tau_{min}, \tau_{ij}\}, \tau_{max}\} \tag{3.12}$$

Ant Colony System (ACS)

Konstrukce tras probíhá následujícím způsobem. Když mravenec k stojí na vrcholu i, vybírá následující vrchol podle pravidla:

$$j = \begin{cases} \operatorname{argmax}_{l \in \mathcal{N}_i^k}\{\tau_{il}[\eta_{il}]^{\beta}\}, & \text{když } \ q \leq q_0; \\ J, & \text{jinak;} \end{cases} \tag{3.13}$$

q je číslo náhodně vybrané z rovnoměrného rozdělení na intervalu $\langle 0,\ 1\rangle$ a q_0 je předem stanovená hodnota, $0 \leq q_0 \leq 1$. $J \in \mathcal{N}_i^k$ je vrchol náhodně zvolený podle vzorce 3.4 z AS, kde položíme $\alpha = 1$. V ACS každý mravenec již během konstrukce své trasy odstraní nějaké množství feromonu z hrany, přes kterou přejde. To vyjadřuje vzorec:

$$\tau_{ij} \leftarrow (1 - \xi)\tau_{ij} + \xi\tau_0 \tag{3.14}$$

$0 < \xi < 1$ a τ_0 jsou dva parametry. Použité hrany se díky tomu stávají méně lákavější pro mravence, což podpoří průzkum dosud nenavštívených oblastí prostoru řešení. Jinak odpaření feromonu a přidání nového feromonu probíhá pouze na hranách z dosud nejlepšího řešení T^{bs} podle vzorce:

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \rho\Delta\tau_{ij}^{bs}, \quad \forall [i, j] \in E(T^{bs}) \tag{3.15}$$

$\Delta\tau_{ij}^{bs}$ je stejné jako u EAS (3.9).

4 Max-Min Ant System

4.1 Max-Min Ant System

Z algoritmů AS, EAS, RAS, MMAS, ACS, uvedených v sekci 3.3, nejlepších výsledků podle [20] (s. 92) pro TSP dosahují MMAS a ACS. My k řešení úlohy napojování komponent v rozpadlé silniční síti volíme MMAS. Máme v úmyslu jej otestovat i na jiných sítích a v jiných situacích, než byly použity v článku [2].

Naše verze MMAS pro napojování komponent (algoritmus 6) se od metaheuristiky ACO pro statické úlohy (algoritmus 5) liší v kroku (2), ve kterém jako ukončovací podmínku volíme dosažení předem zadaného počtu iterací. Další odlišnost je v kroku (3) – v algoritmu 5 jedno řešení konstruoval jeden mravenec, v algoritmu 6 řešení konstruuje m-tice mravenců, tedy celá kolonie. Poslední odlišnost je v kroku (5). V 5 se aktualizovala pouze matice feromonů. V 6, protože jde o MMAS, přibyly meze τ_{max} a τ_{min}, jejichž hodnoty příležitostně také aktualizujeme.

Algoritmus 6 (MMAS pro napojování komponent)
(1) Nastavíme parametry.
(2) Dokud nebylo dosaženo zadaného počtu iterací, opakujeme kroky 3,4,5,6.
(3) Každá kolonie mravenců sestrojí své řešení.
(4) Zlepšíme mravenčí řešení zlepšovací heuristikou.
(5) Aktualizujeme meze feromonů.
(6) Aktualizujeme matici feromonů.

Tento algoritmus se nachází v adresáři `Kody` v modulu `reseniM.py`. Lze ho najít jako funkci s názvem `MMAS()`. Krok (6) probíhá podle vzorce 3.11 a hodnoty τ_{ij} jsou následně korigovány vzorcem 3.12. Oba tyto úkony provede funkce `aktualizaceFeromonu()`.

Podle [23] (s. 899) kdyby to šlo, tak τ_{max} by se nastavilo na $1/(\rho C^{opt})$, kde C^{opt} je cena optimálního řešení. Optimální řešení ale skoro nikdy neznáme, a proto průběžně nahrazujeme C^{opt} cenou C^{bs} nejlepšího dosud nalezeného řešení a s tím průběžně aktualizujeme i τ_{max} a na ní přímo úměrně závislou mez τ_{min}. Z toho důvodu se provádí krok (5) algoritmu 6. V kódu je tento krok realizován pomocí funkce `aktualizaceMeziFeromonu()`.

Krok (4) provedeme podle nastavení některým k-opt algoritmem ze sekce 3.2. Kroku (1) budou věnovány sekce 4.3 a kroku (3) sekce 4.4. Před tím se podívejme na chování Max-Min Ant Systému.

4.2 Konvergence MMAS

Nacházíme se v prostoru Σ řešení σ úlohy Π_2 a chceme se umět přesouvat od řešení k řešení, ale ne tak, aby posloupnost našich možných kroků obsahovala přípustná řešení jen z malé části $O \subset \Sigma$ prostoru a my se z této části téměř neměli šanci přesunout jinam. To je případ, kdy zkonverguje matice feromonu $\boldsymbol{\tau}$ k matici $\boldsymbol{\tau}_O$. O takovou konvergenci nestojíme ani kdyby se v O mělo nacházet globálně optimální řešení σ^*, protože nevíme, kde v Σ se σ^* nachází a potřebujeme volnost, ne vězení v O.

Kdyby v MMAS nebyla stanovena dolní mez $\tau_{min} > 0$ pro množství feromonu na hranách, feromon by nějaký zůstával ve větším množství jen na hranách dosud nejlevnějšího nalezeného řešení. Všude jinde by odpar feromonu (3.5) způsobem $\tau_{ij} \leftarrow (1-\rho)\tau_{ij}$ pokračoval až k nule. Díky tomu by k nule klesala i pravděpodobnost, že libovolný z mravenců zkonstruuje jiné řešení, než nejlepší dosud nalezené. V tomto případě by past O byla tvořena sice dobrým, ale asi ne optimálním řešením.

Ze vzorce 3.11 plyne, že maximální možné množství feromonu přidané na libovolnou hranu v každé iteraci je $D = 1/p_1(\sigma^*)$, protože (z definice globálního optima) nikdy nenajdeme řešení σ lepší, než σ^*. Množství feromonu $\tau_{ij}(n)$ po n iteracích na hraně $[i, j]$ není větší, než $\tau_{ij}^{max}(n) = \tau_0(1-\rho)^n + D(1-\rho)^{n-1} + \ldots + D$. Proto

$$\lim_{n\to\infty} \tau_{ij}^{max}(n) = D/\rho =: \tau_{max}$$

V praxi toho asymptotického τ_{max} nikdy nedosáhneme (ani neznáme jeho hodnotu). Volíme odhad hodnoty τ_{max} pomocí $\tau_{max}(n) := 1/(\rho p_1(\sigma^{bs}))$, kde σ^{bs} je dosud nejlepší nalezené řešení. Tím je odůvodněn krok (5) algoritmu 6. A od toho se odvíjí volba $\tau_{min} = \tau_{max}/k$, kde k je vhodná konstanta větší, než 1, aby zůstalo v platnosti $\forall n \in \mathbb{N} : \ 0 < \tau_{min}(n) < \tau_{max}(n)$.

Díky platnosti podmínky $\tau_{min} > 0$, se odvodí, že během konstrukce řešení v libovolné iteraci pro každou hranu $[i, j]$ existuje nenulová pravděpodobnost p_{ij}, že si ji mravenec zvolí, a proto každé přípustné řešení $\sigma \in \Sigma$ má šanci vzniknout v některé iteraci algoritmu MMAS. Na tom stojí důkaz obecnější verze následující věty uvedený v [20] na stranách 128-130.

Věta 8 Nechť $\alpha, \beta, \tau_{min} > 0$ a $P^*(n)$ značí pravděpodobnost, že se během prvních n iterací mezi řešeními získanými algoritmem MMAS vyskytne optimální řešení σ^* aspoň jednou. Potom $\lim_{n\to\infty} P^*(n) = 1$.

Věta neříká nic jiného, než že nedojde k uvíznutí MMAS v nějaké vlastní podmnožině O prostoru Σ, byť některé části prostoru bude MMAS navštěvovat častěji, než jiné části. To, co zde konverguje, je míra pravděpodobnosti jevu, že během některé z dosud vykonaných iterací vzniklo optimální řešení. Tedy máme-li dostatek času, dříve či později získáme optimální řešení. To jsme mohli i systematickým způsobem hrubou

silou. V tom případě bychom ale asi čekali neprakticky dlouho. To jsme mohli také generováním náhodných přípustných řešení. I v tomto případě bychom mohli čekat dlouho, navíc bychom při nedostatku paměti neměli ponětí, která všechna řešení jsme už náhodně vygenerovali. To nemusíme mít ani u MMAS. S MMAS alespoň častěji získáme více lepších řešení.

V praxi se hledají způsoby, jak zabránit uvíznutí MMAS v nějaké O ještě efektivněji, než jak to dělají τ_{min}, τ_{max} spolu s náhodným střídáním přírůstků feromonu na hranách řešení σ^{ib}, σ^{bs}. Jednou z dalších možností je nastavit pro začátek matici feromonů tak, aby $\forall[i, j] : \ \tau_{ij} = \tau_{max}$. Tím se zařídí, že v počátečních iteracích bude MMAS generovat řešení rozptýlená po větší části prostoru Σ a s postupným odpařováním feromonu MMAS bude vracet stále častěji řešení koncentrovaná v menší části prostoru Σ.

Další možností je čas od času provést restart matice feromonu - začít znova. To zařídíme tím, že místo toho, abychom provedli například jedenkrát 10000 iterací MMAS, provedeme jen 1000 iterací MMAS a tohle zopakujeme 10-krát. Celkový počet iterací bude stejný, ale během procesu došlo devětkrát k restartu.

Není nutno rozdělit iterace tak na pevno, jak jsme právě uvedli, a jak používáme v našich testech algoritmu. Dá se měřit míra stagnace algoritmu a restart feromonové matice provést až tehdy, kdy tato míra dosáhne určité hodnoty. Jednou z možných měr stagnace je například průměrná entropie $\overline{\mathcal{E}}$ pravděpodobnosti p_{ij} volby následujícího vrcholu, spočítané podle vzorce 3.4. Zde N značí počet vrcholů CG-sítě. Předpokládáme $\forall i \in 1(1)N \ p_{ii} = 0$.

$$\overline{\mathcal{E}} = -\frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{j \in 1(1)N \setminus \{i\}} p_{ij} \ln(p_{ij})$$

$\overline{\mathcal{E}}$ nabývá nezáporných hodnot. Maxima $\ln(N-1)/(N-1)$ míra $\overline{\mathcal{E}}$ nabývá, pokud platí $\forall i \in 1(1)N \ \forall j \in 1(1)N : \ \ (j \neq i) \ \Rightarrow \ (p_{ij} = 1/(N-1))$, což znamená, že mravenec má stejnou šanci zvolit si kterýkoliv ze sousedních dostupných vrcholů. To odpovídá situaci, kdy MMAS má stejnou šanci navštívit libovolnou část prostoru Σ. Čím blíže je hodnota $\overline{\mathcal{E}}$ nule, tím více MMAS stagnuje v nějaké části prostoru řešení.

4.3 Nastavení parametrů

Poté, co uživatel připravil všechna potřebná vstupní data (viz sekce 2.6), před vykonáváním příkazů skriptu `hlavni.py`, je možné nastavit hodnoty parametrů pro funkci `MMAS` ze skriptu `reseniM.py` přípravou slovníku s názvem `vstupniParametry` a voláním funkce `pripravParametry()`.

```
parametry = reseniM.pripravParametry(vstupniParametry, ...)
reseni_bs = reseniM.MMAS(..., parametry, ...)
```

Je možné nastavit tyto parametry.

`parametry["odpar"]` v rovnici 3.5 odpovídá členu ρ. Nabývá hodnot z intervalu $(0, 1)$. Není-li hodnota odparu nastavena uživatelem, nastaví se automaticky na 0.2, ať je to stejně jako v [2] (s.1099), kde se používá $r = 0.8$, přičemž platí $r = (1 - \rho)$.

`parametry["iterLimit"]` určuje počet iterací pro krok (2) algoritmu 6. Nabývá přirozených čísel. Není-li hodnota počtu iterací nastavena uživatelem, nastaví se automaticky na 1000, což pro naše sítě na běžném (sekce 5) počítači za cenu řádově 1000Kč proběhne v řádu jednotek minut.

`parametry["pomerFmaxFmin"]` je hodnota τ_{max}/τ_{min}, která umožňuje ze zadané hodnoty τ_{max} automaticky spočítat hodnotu τ_{min} kdykoliv se během iterací algoritmu změní hodnoty τ_{max} a τ_{min}. Požadujeme, aby platilo $\tau_{max}/\tau_{min} > 1$. Pokud uživatel tento poměr nestanoví, je nastaven automaticky na $2N_{CG}$, dvojnásobek počtu vrcholů CG-sítě. Tuto hodnotu přebíráme z článku [23], s.905.

`parametry["feroMax"]` je τ_{max}, tedy maximální hodnota, kterou feromony na hranách nepřekročí. Pokud uživatel nestanoví jinou hodnotu, je nastaveno $\tau_{max} = 1$ stejně jako to mají [2] na s.1099.

`parametry["feroMin"]` je τ_{min}, tedy minimální hodnota, pod kterou se feromony na hranách nedostanou. Pokud uživatel nestanoví jinou hodnotu, je nastaveno $\tau_{min} = \tau_{max}/$`pomerFmaxFmin`.

`parametry["alfa"]` je mocnina α ve vzorci 3.4, která může nabývat libovolných reálných čísel. Pokud uživatel nestanoví jinou hodnotu, je nastaveno $\alpha = 1$.

`parametry["beta"]` je mocnina β ve vzorci 3.4, která může nabývat libovolných reálných čísel. Pokud uživatel nestanoví jinou hodnotu, je nastaveno $\beta = 1$.

`parametry["frekvencePruzkumu"]` je funkce fp čísla iterace x, jejímž oborem hodnot je množina $\langle 0, 1\rangle$. $g = \text{fp}(x)$. V iteraci číslo x jsou k aktualizaci feromonu zvoleny s pravděpodobností g hrany z nejlepšího řešení nalezeného v této iteraci a s pravděpodobností $1 - g$ hrany z řešení nejlepšího od počátku až do iterace x včetně. Pokud uživatel nestanoví jinak, volíme konstantní funkci $\text{fp}(x) = 0.5$, tedy `parametry["frekvencePruzkumu"] = lambda x: (0.5)`. Frekvence průzkumu se parametr jmenuje z toho důvodu, že updatujeme-li feromon stále jen na hranách z toho nejlepšího nalezeného řešení, tak MMAS vrací častěji výsledky z menšího okolí nejlepšího řešení, takže prostor všech řešení je prohledáván málo. Pokud však častěji updatujeme feromon i na hranách ne nejlepšího řešení, je prohledávána větší část prostoru všech řešení. Zpočátku bychom mohli chtít více prozkoumávat celý prostor a až ke konci se zaměřit více na jednu jeho malou část ([20], s.75). To se dá zařídit nějakou

klesající funkcí, například $\mathrm{fp}(x) = 1 - \tanh((4x - 2\texttt{iterLimit})/\texttt{iterLimit})/2$. Vliv tohoto parametru na kvalitu řešení jsme nezkoumali.

`parametry["zlepsovaciAlgoritmus"]` může nabývat některé z těchto třech hodnot: 'opt2', 'opt3', 'nic'. Vyjadřuje, zda má být být mravenci nalezené řešení zlepšeno a pokud ano, která zlepšovací heuristika z modulu `reseniZ.py` se použije. Není-li hodnota pro zlepšovací algoritmus nastavena uživatelem, nastaví se na 'opt2'.

`parametry["iterLimitKopt"]` Zlepšovací heuristiky 2-opt nebo 3-opt sice skončí po konečném počtu kroků, ale někdy těch kroků může být příliš mnoho. Pro tento parametr je nastavena hodnota `math.inf`. To znamená, že k-opt, pokud je použit, doběhne až do konce. Kdyby uživatel chtěl výpočet urychlit (výměnou za horší kvalitu zlepšení), může tomuto parametru nastavit malé přirozené číslo n. Potom kroky 1-5 v k-opt algoritmu (algoritmus 3) proběhnou n-krát nebo pokud by skončil k-opt bez omezení počtu iterací dříve, tak proběhnou méně než n-krát.

`parametry["uzitPocatecniReseni"]` může nabývat hodnoty buď True nebo False a je nastaven na True. V tom případě před tím, než začnou své trasy konstruovat mravenci, je první řešení zkonstruováno pomocí algoritmu úspor (algoritmus 2) a během aktualizace feromonu přibude určité množství feromonu na hrany tohoto řešení.

`parametry["typCeny"]` Tímto parametrem lze nastavit cenovou funkci $q_i(\overline{T}(\sigma))$, podle které se bude hledat optimální řešení σ (sekce 2.3). q_i lze vybírat ze třech možností: q_1 hodnotou 'sum', q_2 hodnotou 'max', q_3 hodnotou 'vse'. Pokud uživatel hodnotu tohoto parametru nezmění, optimalizace probíhá podle q_3.

`parametry["zpusobVyberuMravence"]` Během kroku (3) algoritmu 6, kdy mravenci konstruují svá řešení, se mravenci střídají v tom, kdo je „na tahu“. Způsobů, jakými lze vybírat mravence, by se dalo vymyslet více. Zde má uživatel na výběr ze dvou způsobů – 'vm1' a 'vm2'. Pokud uživatel tento parametr nenastaví, je jeho hodnota nastavena na 'vm2'. Podrobnosti jsou v sekci 4.4.

`parametry["pocetKolonii"]` m mravenců, reprezentujících m čet, tvoří dohromady jednu kolonii, která konstruuje jedno řešení. Počet kolonií určuje, kolik kolonií v každé jedné iteraci bude nezávisle na sobě hledat své řešení vycházejíce při tom z téže matice feromonů. Kvůli úspoře času je počet nastaven na 1. V [2] (s. 1100), používají 30.

`parametry["aktualizovatFeromony"]` Nastavením hodnoty tohoto parametru na False lze vypnout krok (6) algoritmu 6 a tím z naší implementace MMAS udělat generátor náhodných přípustných řešení (pro α=0, β=0). Používáme optimalizaci algoritmem mravenčí kolonie, a proto je hodnota tohoto parametru nastavena na True.

Matice feromonu $\boldsymbol{\tau}$ má na počátku nastaveny hodnoty na: $\tau_{ij} = \tau_{max}\ \forall i, j \in 1(1)N_{CG}$. Tím se podpoří rovnoměrnější výběr hran a tím větší průzkum prostoru řešení v počátečních iteracích.

Matici heuristik $\boldsymbol{\eta}$ je možné spočítat jednou na začátku a pak ji mít celou k dispozici v každé iteraci. Místo toho v každé iteraci počítáme pokaždé znova jen ty hodnoty η_{ij}, které potřebujeme. Počítáme je podle vzorce $\eta_{ij} = 1/d_{ij}$, kde $d_{uv} = l(u, v)$ je ohodnocení hran CG-sítě (sekce 2.4).

4.4 Konstrukce řešení

Řešení je m-tice kružnic (tras) v CG-síti. Některé z hran těchto tras mohou být opravené blokace. Pokud hrany množiny opravených blokací propojují všechny komponenty do jedné, jedná se o přípustné řešení.

Každá četa k opravuje blokace na své trase, kterou pro ni připraví mravenec k tím, že postupně vytvoří kružnici v CG-síti. V jedné iteraci pro m-tici čet jedno řešení zkonstruuje kolonie sestavená z m mravenců.

Mravenci se střídají v tom, kdo je „na tahu“ – který z nich si vybírá nový vrchol, kterým prodlouží svou dosavadní cestu. Když je na řadě k-tý mravenec, který stojí na vrcholu i, vybírá si některý z dostupných sousedních vrcholů j z množiny $\mathcal{N}_i^k$ náhodným způsobem podle pravidla 3.4. Podle čeho však určíme, který z mravenců je na řadě?

První způsob konstrukce tras

V článku [2] se řeší napojování komponent v situacích, kdy cestáři vyjíždějí z jedné stanice, kterou označíme `start`. V symbolice zavedené v sekci 2.4 je zde množina stanic $ST = \{\texttt{start}\}$. Pracujeme s vrcholy $v \in BC$, kde $BC = V(CG)$. Do množiny X zakázaných vrcholů postupně přidáváme vrcholy v navštívené některým mravencem. Uvažujme, že v je z k-té komponenty BC_k. V okamžiku, kdy zařadíme v do X, tak spolu s ním do X zařadíme také všechny ostatní vrcholy z BC_k, protože tato komponenta již byla navštívena, a tak skrz opravy na trase některého z mravenců propojena s komponentou, ve které se nachází `start`. Označme Y množinu (seznam) dostupných vrcholů, ze kterých mravenci mohou vybírat. $Y = BC \setminus X$. Nechť a značí a-tého mravence z kolonie m mravenců. Označme s_{ai} celkovou odpracovanou dobu mravence a, kterou mu práce zabrala, než se dostal ze stanice přes opravy blokací na cestě až do jeho aktuálního vrcholu i. Dobu cesty z i do j označme t_{ij} a r_{ij} součet dob oprav blokací na nejkratší cestě z i do j, které mravenec opraví. Nechť BD značí množinu dostupných (neopravených) blokací. Následuje algoritmus z [2], s.1097-1098.

Algoritmus 7 (První způsob konstrukce tras)

(1) Umístíme všechny mravence na `start`, provedeme změnu $Y \leftarrow Y \setminus \{\texttt{start}\}$ a položíme $DB = B$.

(2) Dokud $Y \neq \emptyset$, opakujeme kroky 3 až 10.

(3) $\forall a \in 1(1)m$ provedeme kroky 4 až 7.

(4) `char[a]` = 0

(5) Podle vzorce 3.4 v sekci 3.3 vybereme vrchol j.

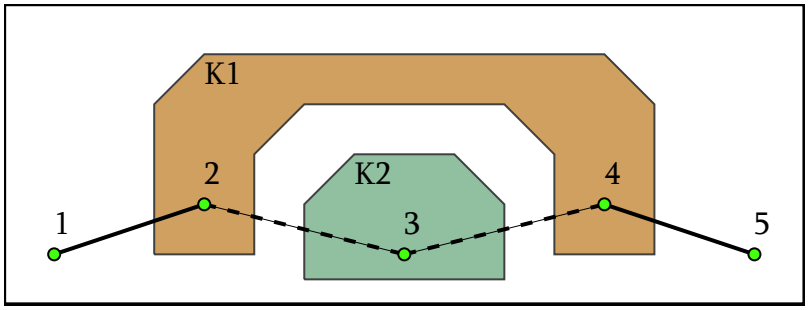
(6) Pokud $j \notin Y$, mravenec a uzavře svou trasu návratem do stanice.

(7) Jinak se spočítá `char[a]` $= 1/(s_{ai} + t_{ij} + r_{ij})$.

(8) Na základě pravděpodobností rovných normalizovaným hodnotám `char[1]` až `char[m]` se náhodně vybere mravenec číslo b, $b \in 1(1)m$ a jeho volba j.

(9) Mravenec b prodlouží svou cestu o vrchol j, opraví všechny dostupné blokace z BD na své trase z $i(b)$ do j, tvořící množinu BT a aktualizuje svou celkovou odpracovanou dobu z $s_{bi(b)}$ na s_{bj}.

(10) $j \in BC_j$. Provedeme změny $Y \leftarrow Y \setminus BC_j$ a $BD \leftarrow BD \setminus BT$.



Obrázek 6: Dvě komponenty K1 a K2 a mezi nimi blokace hran [2,3] a [3,4] na hraně [1,5] v CG-síti.

Uvažujme situci na obrázku 6, kdy mravenec stojící ve vrcholu 1 si v CG-síti zvolí jít do souseda 5. Nejkratší cesta v G, která do něj vede, je přes hraniční vrcholy, které jsou v CG-síti číslovány 2, 3, 4. Vrchol 2 je hraničním vrcholem komponenty K1, vrchol 3 komponenty K2, vrchol 4 komponenty K1. Hrany [2,3] a [3,4] jsou blokace. V této situaci krok (9) algoritmu 7 opraví obě blokace [2,3] a [3,4] a zajistí propojení komponent K1 a K2. K propojení K1 s K2 stačí opravit jednu z blokací. Kdybychom opravili obě (a kdyby $r_{2,3} > 0 < r_{3,4}$), byly by součástí řešení, které určitě není optimální (věta 4), když můžeme některou z blokací objet bez zdržení (sekce 2.2). To je jeden z důvodů, proč budeme trasy konstruovat jinak. A dalším důvodem je to, že v Π_2 uvažujeme množinu stanic ST, kde $|ST| = z \geq 1$.

Kdyby v případě dvou stanic mravenci měli společnou množinu dostupných vrcholů Y, mravenec a_1 z první stanice by svou volbou vrcholu zamezil mravenci a_2 ze druhé stanice zvolit si některý vrchol z mravencem a_1 již navštívené komponenty, čímž by mohl zamezit napojení komponent, kdyby obě stanice byly v jiných komponentách.

Proto v následujícím způsobu konstrukce tras bude mít k-tá stanice svůj seznam dostupných vrcholů Y^k.

Druhý způsob konstrukce tras

Algoritmus 8 (Druhý způsob konstrukce tras)

(1) Dokud je počet komponent > 1, opakujeme kroky 2 až 9.

(2) Vybereme mravence a jedním z dále uvedených způsobů.

(3) Zvolený mravenec stojící ve vrcholu i podle vzorce 3.4 v sekci 3.3 náhodně vybere nový vrchol j z množiny dostupných vrcholů $Y^{k(a)}$ příslušných stanici $k(a)$, ze které pochází mravenec a. `novy` = j.

(4) Mravenec na nejkratší cestě CG-sítí z vrcholu i do vrcholu j najde všechny dostupné blokace tvořící množinu BT.

(5) Je-li $|BT| \geq 1$, mravenec opraví pouze první blokaci vyskytující se na cestě z i do j. Cesta se zkrátí tak, aby končila vzdálenějším vrcholem v první opravené hrany $e \in BT$. `novy` = v.

(6) Trasa mravence a se prodlouží o vrchol `novy`. Zvětší se odpracovaná doba mravence a. Mravencův seznam oprav se aktualizuje.

(7) Zjistíme množinu hraničních vrcholů komponenty, kam patří vrchol `novy` a označíme ji BC_{jv}. Provedeme změnu $Y^{k(a)} \leftarrow Y^{k(a)} \setminus BC_{jv}$.

(8) Výchozí vrchol i leží v komponentě C_i a `novy` v C_{jv}. Množinu blokací, které dělí C_i od C_{jv}, označme BB. Změníme množinu dostupných blokací $BD \leftarrow BD \setminus BB$.

(9) Spojíme C_{jv} s C_i. Jsou-li různé, zmenší se počet komponent.

Tohle dělá funkce `konstrukceTras()` ze skriptu `reseniM.py`. Pro ni lze nastavit `parametry["zpusobVyberuMravence"]` na `'vm1'`, má-li tato funkce použít první způsob výběru mravence, nebo na `'vm2'`, má-li použít druhý způsob výběru mravence.

Algoritmus 9 (První způsob výběru mravence)

(1) $\forall a \in 1(1)m$ provedeme kroky 2 až 5.

(2) `char[a]` = 0

(3) Podle vzorce 3.4 v sekci 3.3 vybereme vrchol $j \in Y^k$.

(4) Pokud takový j neexistuje, pokračujeme krokem (1) pro další a.

(5) Jinak se spočítá `char[a]` = $1/(s_{ai} + t_{ij} + r_{ij})$.

(6) Na základě pravděpodobností rovných normalizovaným hodnotám `char[1]` až `char[m]` se náhodně vybere mravenec číslo b, $b \in 1(1)m$ a jím zvolený vrchol j.

Algoritmus 10 (Druhý způsob výběru mravence)

(1) Na základě odpracovaných dob s_{ai} mravenců vybereme prvního mravence s nejmenší odpracovanou dobou.

5 Testy

Testy provádíme na počítači s parametry: 2.16GHz, 4GB RAM, x64 OS Windows.

5.1 Přehled sítí a situací

Zde si představíme ty sítě a situace, pro které řešíme napojování komponent v následujících testech MMAS algoritmu. Jedná se o sítě `sit001`, `sit002` a `sit006`. První dvě jsou umělé rovinné sítě. Poslední je silniční síť zlínského kraje, kterou jsme převzali z práce [2] v podobě uvedené v adresáři `sit006\ZlinKraj`. Umělé rovinné sítě vznikly následujícím způsobem. Pomocí funkcí ze skriptu `kreator.R` z adresáře `Kody\vznikSiti` se ve čtverci v rovině náhodně vygeneruje zadaný počet bodů. Na těchto bodech proběhne triangulace a další ne příliš důležité estetické operace jako například smazání nejdelší hrany každého úzkého trojúhelníka. Triangulace spolu s těmito operacemi nám určí, který vrchol sousedí se kterým, čímž získáme hrany budoucího grafu G. Ty ohodnotíme dobami jízd v sekundách přímo úměrnými jejich délce. Tyto doby ještě přenásobíme náhodnými čísly z rovnoměrného rozdělení z intervalu $\langle 1, 1.8 \rangle$.

Pro sítě `sit001` a `sit002` ještě zvolíme náhodně některé vrcholy za stanice. Výsledky katastrof a rozpad sítí na komponenty souvislosti simulujeme funkcemi ze skriptu `destruktor.R`. Funguje to tak, že zvolíme počet n_C komponent, na které chceme, aby se síť rozpadla. Dále náhodně zvolíme n_C různých vrcholů grafu G. Tyto vrcholy si lze představit jako centra, ze kterých se současně přes s nimi incidentní hrany rozlévá do sousedních bezbarvých vrcholů barva. Každé centrum má jinou barvu. Proces probíhá do okamžiku obarvení všech vrcholů grafu. Vrcholy se stejnou barvou leží ve stejné komponentě souvislosti. Hrana, jejíž krajní vrcholy mají odlišné barvy se stane zablokovanou hranou.

Zbývá ohodnotit zablokované hrany dobami oprav. To proběhne ve skriptu `zdroje.R` následovně. Z ohodnocení hran grafu G se vybere náhodný vzorek o velikosti stejné, jako je počet blokací. Hodnoty v tomto vzorku zdesetinásobíme a získáme tak ohodnocení blokací dobami oprav.

`sit001` má 815 vrcholů a 1591 hran. Hodnoty dob jízd po těchto hranách leží v rozmezí 3 až 792 sekund. Počítáme v ní se třemi situacemi.

(1) Situace `k21d02` má tento popis. 2 stanice, jedna ve vrcholu 516 a druhá ve vrcholu 109. Jsou od sebe vzdáleny 4218 sekund. Je zde 234 blokací. Ty dělí G na 21 komponent. Doby oprav blokací se pohybují v rozmezí 60 až 5190 sekund. Doby jízd mezi vrcholy CG-sítě se pohybují v rozmezí 7 až 5676 sekund.

(2) Situace k21d02_blokaceZk21d03 má tento popis. 2 stanice, jedna ve vrcholu 516 a druhá ve vrcholu 109. Jsou od sebe vzdáleny 4218 sekund. Je zde 238 blokací. Ty dělí G na 21 komponent. Doby oprav blokací se pohybují v rozmezí 160 až 5580 sekund. Doby jízd mezi vrcholy CG-sítě se pohybují v rozmezí 6 až 6270 sekund.

(3) Situace k21d02_blokaceDeleno100 má tento popis. 2 stanice, jedna ve vrcholu 516 a druhá ve vrcholu 109. Jsou od sebe vzdáleny 4218 sekund. Je zde 234 blokací. Ty dělí G na 21 komponent. Doby oprav blokací se pohybují v rozmezí 1 až 52 sekund. Doby jízd mezi vrcholy CG-sítě se pohybují v rozmezí 7 až 5676 sekund.

sit002 má 797 vrcholů a 1574 hran. Hodnoty dob jízd po těchto hranách leží v rozmezí 4 až 2871 sekund, přičemž druhá největší hodnota je 1200 sekund a třetí je 1043 sekund. Počítáme v ní s jednou situací.

(1) Situace k41d07 má tento popis. 7 stanic postupně ve vrcholech 403, 317, 133, 669, 166, 632, 2. Nejbližší stanice jsou vzdáleny 455 sekund a nejvzdálenější 3935 sekund. Je zde 314 blokací. Ty dělí G na 41 komponent. Doby oprav blokací se pohybují v rozmezí 40 až 28710 sekund. Druhá největší doba opravy blokace je 9170 sekund a třetí největší 3960 sekund. Doby jízd mezi vrcholy CG-sítě se pohybují v rozmezí 8 až 6007 sekund.

sit006 má 734 vrcholů a 968 hran. Hodnoty dob jízd po těchto hranách leží v rozmezí 1 až 2069 sekund. Počítáme v ní se dvěma situacemi, přičemž v obou případech je jediná stanice a nachází se ve vrchlu 461.

(1) Situace k16d01 má tento popis. Je zde 46 blokací. Ty dělí G na 16 komponent. Doby oprav blokací se pohybují v rozmezí 9000 až 67000 sekund. Doby jízd mezi vrcholy CG-sítě se pohybují v rozmezí 3 až 5443 sekund.

(2) Situace k16d01_blokaceDeleno10 má tento popis. Je zde 46 blokací. Ty dělí G na 16 komponent. Doby oprav blokací se pohybují v rozmezí 900 až 6700 sekund. Doby jízd mezi vrcholy CG-sítě se pohybují v rozmezí 3 až 5443 sekund.

5.2 ACO versus Náhoda

„Každý algoritmus z třídy ACO vrací v průměru lepší výsledky, než generátor náhodných přípustných rovnoměrně rozdělených řešení.“ Je toto tvrzení pravdivé? To autor nedokáže říct. Nezůstal čas promyslet to. Různým experimentům (uvedeným např. zde [20] a zde [23]) starým i přes třicet let se nepodařilo najít protipříklad, který by uvedené tvrzení vyvracel. To, že hypotéza odolává testům tak dlouho, z ní nedělá pravdu, ale dělá z ní zajímavý předmět výzkumu. Jak čtenář sám uvidí, odolala i testům

v následujících experimentech.

Experimentujeme s MMAS nad úlohou: `Ulohy\sit001\k21d02`, $h = [2, 2]$ a nepoužíváme při tom počáteční řešení získané algoritmem úspor.

K výrobě generátoru (RAND) náhodných přípustných řešení použijeme MMAS, v němž vypneme aktualizaci feromonu a budeme ignorovat heuristickou informaci (délky hran) tím, že nastavíme $\beta = 0$. Dále nechť $\alpha = 1$ a nepoužíváme žádnou zlepšovací heuristiku. Takto má během konstrukce řešení každý dostupný soused aktuálního vrcholu k-tého mravence stejnou šanci být zvolen k-tým mravencem. Protože hodnoty feromonu zůstanou neměnné, vzorec 3.4 se změní na

$$p_{ij}^k = \frac{\tau_{ij}}{\sum_{l \in \mathcal{N}_i^k} \tau_{il}} = \frac{1}{|\mathcal{N}_i^k|}$$

Během konstrukce přípustného řešení se v naší implementaci MMAS střídá m mravenců způsobem, který nastaví uživatel. V základním (a zde použitém) nastavení je mravenec vybírán deterministicky podle toho, jak velká je jeho odpracovaná doba. Další na řadě je ten, který má nejmenší odpracovanou dobu. Výsledkem je, že přípustné řešení je sestaveno z m tras. Tohle by generátor naprosto náhodných řešení splňovat nemusel. Pakliže doby oprav blokací jsou (přibližně) stejné, dalším výsledkem tohoto způsobu výběru mravence je, že s rostoucím počtem komponent souvislosti (roste minimální počet blokací, které je potřeba odstranit, a tím) se vyrovnávají odpracované doby mravenců. Generátor naprosto náhodných řešení by tohle nesplňoval a odpracované doby jednotlivých mravenců by se navzájem lišily více.

Použijeme-li zlepšovací algoritmus 2-opt, získáme z RAND algoritmus, který nazveme RANDopt2. S těmito dvěma algoritmy budeme srovnávat MMAS. Ten má nastaveno $\alpha = 1$, $\beta = 1$ a aktualizace feromonu v něm probíhá. Zlepšovací algoritmus nepoužijeme. Pokud zlepšovací algoritmus použijeme, získáme z MMAS už čtvrtý algoritmus – MMASopt2.

Algoritmy (AX) generují tolik řešení σ, kolik iterací jim zadáme vykonat, což je zde 1000, a my si ke každému z AX uložíme ceny $p_3(\sigma)$ (sekce 2.3 a 2.7) první tří nejlepších řešení, která AX najde. V této úloze má cena p_3 řešení čtyři složky. Výsledky lze najít v příloze v adresáři `Testy\ACOvsNahoda` v souborech `RAND_H.csv`, `RANDopt2_H.csv`, `MMAS_H.csv`, `MMASopt2_H.csv`, kde H je 0, 1 nebo 2 podle toho, kolikátá nejlepší řešení soubor obsahuje. Výsledky shrnuje tabulka 1. Všechna měření byla provedena desetkrát a na každém řádku tabulky 1 je aritmetický průměr těchto deseti měření zaokrouhlený na jednotky.

Tabulka 1:

V pořadí 1. nejlepší ceny.

Algoritmus	t1 [s]	t2 [s]	t3 [s]	t4 [s]
RAND	15896	15107	14530	13818
RANDopt2	14726	14277	13574	12550
MMAS	13654	13204	12765	12024
MMASopt2	14156	13730	13192	12045

V pořadí 2. nejlepší ceny.

Algoritmus	t1 [s]	t2 [s]	t3 [s]	t4 [s]
RAND	16128	15584	15171	13558
RANDopt2	15257	14887	14121	13172
MMAS	13721	13324	12880	11887
MMASopt2	14382	13954	13582	12657

V pořadí 3. nejlepší ceny.

Algoritmus	t1 [s]	t2 [s]	t3 [s]	t4 [s]
RAND	16221	15651	14804	13352
RANDopt2	15469	14919	14188	12731
MMAS	13771	13329	12675	12004
MMASopt2	14618	14241	13480	12345

Ať už použijeme nebo nepoužijeme 2-opt heuristiku, lze z tabulky 1 vyčíst, že pro $k \in \{0, 1, 2\}$ k-té nejlepší řešení MMAS je levnější (v lexikografickém uspořádání), než k-té nejlepší řešení RAND. To jsme čekali, a i proto používáme algoritmy z třídy ACO. Překvapivé je, že MMAS řešení v každé iteraci zlevněná 2-opt algoritmem vrací dražší výsledky, než když použijeme MMAS bez 2-optu. Tato měření sice uvádíme na začátku sekce testů, ale provedena byla až jako poslední, jinak bychom více prostudovali interakci mezi MMAS a 2-opt a podle toho volili nastavení do dalších testů.

5.3 Algoritmus úspor a MMAS

Jak moc nám pomůže, když poskytneme algoritmu MMAS počáteční řešení získané algoritmem úspor (AU)? Na rozdíl od základního nastavení uvedeného v sekci 4.3, v následujících testech použijeme toto nastavení MMAS: 1000 iterací, první způsob výběru mravence, $h = [1, 1, 1, 1, 1, 1, 1]$ pro úlohu `sit002\k41d07`. V prvním měření použijeme počáteční řešení získané pomocí AU a v druhém ne. Data z měření jsou v příloze ve složce `Testy\AUaVyber` v souborech `True_vm1.csv`, resp. `False_vm1.csv`.

Tabulka 2: Aritmetický průměr a směrodatná odchylka cen $p_3()$ řešení ze souboru True_vm1.csv získaných MMAS s AU a prvním způsobem výběru mravence.

Statistika	t1 [s]	t2 [s]	t3 [s]	t4 [s]	t5 [s]	t6 [s]	t7 [s]
průměr	15374	15081	14514	14017	13319	11847	10592
odchylka	318	358	580	677	711	1332	1716

Tabulka 3: Aritmetický průměr a směrodatná odchylka cen $p_3()$ řešení ze souboru False_vm1.csv získaných MMAS bez AU s prvním způsobem výběru mravence.

Statistika	t1 [s]	t2 [s]	t3 [s]	t4 [s]	t5 [s]	t6 [s]	t7 [s]
průměr	15679	15377	14985	14217	13689	12770	11137
odchylka	466	458	536	859	1081	1267	1140

Spočítat 1000 iterací, které využívají první způsob výběru mravence při konstrukci řešení (algoritmus 9) a výpočet počátečního řešení pomocí algoritmu úspor (algoritmus 2), trvalo 10822 sekund. Spočítat 1000 iterací tímto způsobem, ale bez použití algoritmu úspor, trvalo 11683 sekund. Použitím AU jsme dosáhli napojení komponent za dobu, která je průměrně 0.98-násobek doby napojení komponent bez použití AU a výpočet trval 0.926-násobek doby výpočtu, ve kterém jsme nepoužili AU. Je to divné, ale je to tak. Jeden by čekal, že bez AU výpočty skončí dříve, protože jich bude méně, protože nepoběží AU. Jedno možné vysvětlení je, že Python si ukládá výledky některých výpočtů bez vědomí autora a to, co mu vyjde jednou (např. řešení získané pomocí AU) už znova nepočítá a tím si šetří čas.

5.4 Způsob výběru mravence

V tomto měření používáme spolu s MMAS algoritmus úspor a zajímá nás, jaké výsledky a za jak dlouho získáme, když použijeme druhý způsob výběru mravence (viz sekce 4.4) během konstrukce řešení. Řešíme úlohu `sit002\k41d07` s těmito počty čet ve stanicích $h = [1, 1, 1, 1, 1, 1, 1]$. Cena p_3 nejlepšího nalezeného řešení je uložena v příloze ve složce `Testy\AUaVyber` na řádku souboru `True_vm2.csv`. Měření je zopakováno desetkrát, takže soubor obsahuje deset řádků. Z těch se spočítají aritmetické průměry a směrodatné odchylky, které shrnuje tabulka 4.

Tabulka 4: Aritmetický průměr a směrodatná odchylka cen $p_3()$ řešení ze souboru True_vm2.csv získaných MMAS s AU a druhým způsobem výběru mravence.

Statistika	t1 [s]	t2 [s]	t3 [s]	t4 [s]	t5 [s]	t6 [s]	t7 [s]
průměr	15025	14488	14174	13709	12974	12214	11154
odchylka	427	460	399	370	496	832	1772

Spočítat všech 10 měření, v nichž proběhlo 1000 iterací MMAS, které využívají druhý způsob výběru mravence při konstrukci řešení (algoritmus 10) a výpočet počátečního řešení pomocí algoritmu úspor (algoritmus 2), trvalo 7298 sekund. Spočítat $10 \cdot 1000$ iterací s použitím AU a prvního způsobu výběru mravence trvalo 10822 sekund. S použitím druhého způsobu výběru mravence bylo v průměru dosaženo napojení komponent za dobu, která je 0.98-násobkem průměrné doby napojení komponent při použití prvního způsobu výběru (viz tabulka 2) a výpočet trval 0.674 doby výpočtu při použití prvního způsobu výběru. Pro úsporu času dále využíváme druhý způsob výběru mravence.

Výpočty řešení ze sekcí 5.2, 5.3, 5.4 byly v takovém nastavení provedeny desetkrát a pak z nich byly spočítány aritmetické průměry a směrodatné odchylky. Sekce 5.3 ilustruje naše pozorování, že vynecháním algoritmu úspor o moc nepřijdeme a spíše se zjednoduší kód. AU si však necháváme pro možnost srovnat výsledky této práce s výsledky práce [2]. Sekce 5.4 ilustruje, že použitím druhého způsobu výběru mravence se výsledky příliš nezhorší, spíše se zjednoduší kód a trochu se zkrátí doba výpočtu řešení. Tolik motivace k uvedenému nastavení (4.3) hodnot parametrů `"uzitPocatecniReseni"` a `"zpusobVyberuMravence"`.

5.5 Hledání optimálních (α, β)

Zadání instance úlohy Π_2 závisí na zvolené síti S, tj. na G a ohodnocení c jeho hran, na rozmístění stanic ST ve vrcholech sítě, na počtech čet h ve stanicích, na množině B zablokovaných hran a na dobách r jejich oprav. $\Pi_2 = \Pi_2(G, c, ST, h, B, r)$. To, jak dobrá řešení MMAS algoritmus k zadané Π_2 vrací, závisí na nastavení jeho parametrů. Používáme základní nastavení parametrů uvedené v sekci 4.3. Měníme jen hodnoty parametrů α a β a podle toho získáme algoritmus MMAS(α, β), pro který opakovaně měříme cenu p_3 nejlepšího nalezeného řešení během IL iterací.

Označme P množinu $\{p_3(\sigma_i) : i \in 1(1)IL\}$ cen $p_3()$ všech přípustných řešení úlohy Π_2, která nám vygeneruje MMAS(α, β) v prvních IL iteracích. V našem případě $IL = 1000$, takže $|P| = 1000$. $\mathrm{p}_{min}(\alpha, \beta) := \min(P)$. Protože MMAS je stochastický algoritmus, $\mathrm{p}_{min}(\alpha, \beta)$ je náhodná veličina a nás zajímá její střední hodnota $\langle \mathrm{p}_{min}(\alpha, \beta) \rangle$, tj. očekávaná minimální cena řešení získaných MMAS(α, β) během IL iterací. Hledáme bod $(\alpha^*, \beta^*) \in \mathbb{R}^2$ takový, že $\forall (\alpha, \beta) \in \mathbb{R}^2 \ \langle \mathrm{p}_{min}(\alpha^*, \beta^*) \rangle \leq \langle \mathrm{p}_{min}(\alpha, \beta) \rangle$

Kdyby existoval bod (α^*, β^*) takový, že pro libovolnou instanci úlohy Π_2 by MMAS(α^*, β^*) generoval řešení s nejlepšími očekávanými cenami, pro zákazníka by to bylo krásné, protože by měl jeden hotový nástroj pro všechny situace, které kdy může řešit (a matematici by byli bez práce). Ukazuje se ale (např. v [2], s. 1000, 1001 a 1003), že aspoň různé typy úloh mají obecně různé (α^*, β^*). V této sekci se snažíme zjistit, zda bude nutno hledat (α^*, β^*) pro každou instanci úlohy typu Π_2 pokaždé znova. Pokud ano, bylo by to špatné, protože zjistit aspoň přibližně optimální hodnoty (α, β) k dané instanci úlohy je časově náročnější, než samotné řešení úlohy s libovolně zvolenými (α, β). Avšak toto zjištění by bylo zároveň dobré, protože příště bychom hned mohli vzdát hledání optimálních (α, β) a použít nastavení např. $(\alpha, \beta) = (1, 1)$.

Řešíme Π_2. Správce silnic spravuje svou jednu síť S, takže tím je dáno G, c. Rovněž je dané rozmístění dep ST a můžeme uvažovat, že i rozmístění čet cestářů h. Správci sítě by se hodilo, kdyby dvojice (α^*, β^*) nezávisela na přítomných blokacích B a dobách jejich oprav r, protože jen jednou by spočítal (α, β) optimální pro svou síť a pak by při každé katastrofě mohl používat řešič MMAS(α^*, β^*). Proto zde máme hypotézu, kterou budeme testovat, a která zní následovně. Optimální (α, β) pro MMAS řešící úlohu Π_2 na dané síti s daným rozmístěním dep a čet nezávisí na rozmístění blokací a nezávisí na době oprav blokací.

Nahrajeme úlohu. Není-li psáno jinak, použijeme výše uvedené parametry. Nastavíme α, β na konkrétní hodnoty, pustíme MMAS(α, β), počkáme, zaznamenáme cenu $p_3()$ a cenu $p_1()$ řešení nejlepšího (podle $p_3()$) z počtu IL iterací, které MMAS(α, β) vykonal. Tento výpočet desetkrát zopakujeme. Získanou desetiřádkovou tabulku uložíme do souboru `A_B.csv` do adresáře `Testy\alfabeta\P`. V této konvenci A je celočíselný index do sedmice hodnot (0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5), kterých bude v našem měření nabývat α a B je celočíselný index do devítice hodnot (0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5), kterých bude nabývat β. P je proměnná pro prvky množiny písmen $\{Z, A, B, C, D, E, F\}$, která od sebe odlišují různé úlohy. Pro každou ze sedmi úloh získáme tabulku M_{PAB} deseti „dvojcen“ (p_3, p_1) ke každé ze $7 \cdot 9$ dvojic (α, β). Pro každou tabulku M_{PAB} spočítáme aritmetický průměr a směrodatnou odchylku hodnot ve sloupcích, čímž získáme vektory m_{PAB} (průměry) a d_{PAB} (odchylky), oba délky $m + 1$, kde m je počet čet. Prvních m členů vektoru m_{PAB} je odhadem hodnoty $\langle \mathrm{p}_{min}(\alpha, \beta) \rangle$ v úloze P pro hodnoty (α, β) odpovídající indexům (A, B). Například soubor `6_3.csv` obsahuje tabulku $M_{PAB} = M_{P,6,3}$ pro $\alpha = 3.5$, $\beta = 2.0$. Dále popíšeme výsledky měření pro všech sedm úloh $Z, \ldots, F$, které jsou podrobně uloženy v `Testy\alfabeta` v souborech `P_zpracovani.txt`.

V tabulkách 5 až 11 je každé dvojici (α, β) přiřazen první člen $m_{PAB}[0]$ vektoru m_{PAB}. Tento člen odpovídá prvnímu (tudíž největšímu) členu ceny p_3 a tedy ceně p_2 – době napojení komponent. Poslední člen $m_{PAB}[m]$ odpovídá součtu všech členů ceny p_3 a tedy ceně p_1 – celkové odpracované době všech čet. Prvních m členů vektoru m_{PAB} odpovídá odhadu očekávané hodnoty ceny p_3 pro nejlepší řešení získané s MMAS(α, β).

(Z) - základní úloha: `Ulohy\sit001\k21d02`, $h = [2, 2]$.
Měření trvalo celkem asi 37 hodin 43 minut 30 sekund.

Tabulka 5: $p_2()$

$\beta \setminus \alpha$	0.5	1.0	1.5	2.0	2.5	3.0	3.5
0.5	14584.1	14273.8	14186.6	14171.2	14030.4	13923.1	14009.1
1.0	14406.0	13979.6	13846.6	13597.0	13888.3	13910.5	13780.0
1.5	13818.6	13801.4	13556.5	13520.6	13689.6	13582.9	13287.5
2.0	13853.0	13467.9	13592.3	13376.0	13431.3	13359.5	13627.3
2.5	13538.9	13278.2	13303.9	13082.3	13193.8	13464.8	13238.1
3.0	13414.0	13395.4	13147.3	13295.5	13244.3	13428.9	13461.4
3.5	13365.7	13207.7	13047.6	13363.6	13157.1	13294.7	13392.3
4.0	13437.1	13407.9	13212.1	13114.3	13257.2	13121.5	13370.7
4.5	13127.1	13485.5	13251.6	13257.8	13306.2	13326.6	13259.6

První nejmenší p_2 je pro $(\alpha, \beta) = (1.5, 3.5)$.
$m_{PAB}[0] = 13047.6$, $d_{PAB}[0] = 464.298$, $m_{PAB}[m] = 49522.5$, $d_{PAB}[m] = 2272.03$.
Druhá nejmenší p_2 je pro $(\alpha, \beta) = (2.0, 2.5)$.
$m_{PAB}[0] = 13082.3$, $d_{PAB}[0] = 418.113$, $m_{PAB}[m] = 50332.3$, $d_{PAB}[m] = 1601.82$.

(A) Úloha: `Ulohy\sit001\k21d02_blokaceZk21d03`, $h = [2, 2]$.

Tabulka 6: $p_2()$

$\beta \setminus \alpha$	0.5	1.0	1.5	2.0	2.5	3.0	3.5
0.5	14342.5	14027.1	14183.7	14148.2	13671.0	13817.6	14096.1
1.0	14135.4	14190.9	13914.2	13782.4	13771.5	13875.2	13864.2
1.5	13965.6	13864.5	13721.8	13571.3	13776.1	13547.8	13766.7
2.0	13517.2	13524.4	13311.8	13660.0	13653.6	13639.5	13523.0
2.5	13436.8	13370.8	13333.5	13395.2	13542.2	13175.7	13205.0
3.0	13249.4	13236.5	13186.5	13301.4	13368.3	13363.5	13321.8
3.5	13284.7	13254.1	13354.7	13202.3	13323.7	13203.8	13053.2
4.0	13342.9	13377.4	13247.9	13001.5	13294.2	13153.4	13309.9
4.5	13396.2	13461.1	13236.7	12947.0	13187.2	13028.0	13104.6

První nejmenší p_2 je pro $(\alpha, \beta) = (2.0, 4.5)$.
$m_{PAB}[0] = 12947.0$, $d_{PAB}[0] = 259.98$, $m_{PAB}[m] = 48901.2$, $d_{PAB}[m] = 1713.91$.
Druhá nejmenší p_2 je pro $(\alpha, \beta) = (2.0, 4.0)$.
$m_{PAB}[0] = 13001.5$, $d_{PAB}[0] = 511.87$, $m_{PAB}[m] = 49251.1$, $d_{PAB}[m] = 1798.21$.

(B) Úloha: `Ulohy\sit001\k21d02`, $h = [4, 0]$.

Tabulka 7: $p_2()$

$\beta \setminus \alpha$	0.5	1.0	1.5	2.0	2.5	3.0	3.5
0.5	15039.1	15208.1	15173.8	14747.8	14842.5	14797.7	14887.9
1.0	15009.7	14987.6	14791.1	15049.2	15064.9	14913.0	14758.6
1.5	14757.4	14701.8	14607.8	14706.4	14923.9	14660.7	14753.3
2.0	14534.5	14532.4	14639.4	14640.9	14569.1	14538.5	14728.0
2.5	14347.7	14576.7	14414.0	14624.8	14452.2	14372.4	14691.8
3.0	14459.5	14315.3	14307.0	14503.8	14287.7	14453.4	14462.2
3.5	14484.1	14236.9	14252.1	14443.2	14534.6	14450.1	14494.7
4.0	14575.2	14359.0	14424.9	14496.6	14481.5	14473.0	14367.5
4.5	14202.9	14363.2	14595.9	14339.3	14390.0	14480.7	14321.8

První nejmenší p_2 je pro $(\alpha, \beta) = (0.5, 4.5)$.
$m_{PAB}[0] = 14202.3$, $d_{PAB}[0] = 359.382$, $m_{PAB}[m] = 54476.3$, $d_{PAB}[m] = 1378.351$.
Druhá nejmenší p_2 je pro $(\alpha, \beta) = (1.0, 3.5)$.
$m_{PAB}[0] = 14236.9$, $d_{PAB}[0] = 401.779$, $m_{PAB}[m] = 54400.8$, $d_{PAB}[m] = 2141.02$.

(C) Úloha: `Ulohy\sit001\k21d02_blokaceDeleno100`, $h = [2, 2]$.

Tabulka 8: $p_2()$

$\beta \setminus \alpha$	0.5	1.0	1.5	2.0	2.5	3.0	3.5
0.5	7087.6	6845.3	6689.9	6657.7	6414.7	6623.6	6693.7
1.0	6963.5	6738.1	6616.4	6440.9	6552.7	6325.5	6486.3
1.5	6579.6	6607.9	6433.5	6342.8	6345.6	6331.6	6349.8
2.0	6395.5	6305.2	6331.8	6142.6	6205.2	6299.3	6231.8
2.5	6225.8	6082.5	6197.0	6137.4	6169.3	6122.1	6102.5
3.0	6152.2	6055.8	6052.4	6131.9	6171.6	6051.5	6112.9
3.5	6105.0	6040.4	6065.4	6092.3	5994.1	6050.0	6013.2
4.0	5974.0	6045.0	5982.5	5971.5	6017.8	6072.0	6096.2
4.5	5995.8	6014.1	5927.4	6029.2	5968.2	5951.4	5997.7

První nejmenší p_2 je pro $(\alpha, \beta) = (1.5, 4.5)$.
$m_{PAB}[0] = 5927.4$, $d_{PAB}[0] = 125.405$, $m_{PAB}[m] = 22110.3$, $d_{PAB}[m] = 491.671$.
Druhá nejmenší p_2 je pro $(\alpha, \beta) = (3.0, 4.5)$.
$m_{PAB}[0] = 5951.4$, $d_{PAB}[0] = 161.014$, $m_{PAB}[m] = 22467.8$, $d_{PAB}[m] = 586.915$.

(D) Úloha: `Ulohy\sit006\k16d01`, $h = [4]$.

Tabulka 9: $p_2()$

$\beta \setminus \alpha$	0.5	1.0	1.5	2.0	2.5	3.0	3.5
0.5	101961.0	102598.0	99867.1	100992.0	103122.0	102876.0	99802.5
1.0	102205.0	103572.0	104901.0	101356.0	102232.0	102471.0	102812.0
1.5	104126.0	101439.0	101562.0	104888.0	102749.0	104249.0	101924.0
2.0	104045.0	104243.0	103434.0	102254.0	105173.0	102006.0	106734.0
2.5	105197.0	103085.0	105048.0	104730.0	102489.0	105500.0	101118.0
3.0	106386.0	105580.0	105828.0	104440.0	101058.0	102799.0	102399.0
3.5	108591.0	105926.0	106740.0	104606.0	104481.0	105986.0	103018.0
4.0	110691.0	107597.0	105882.0	105129.0	103894.0	104217.0	103748.0
4.5	110263.0	109492.0	106312.0	107666.0	108158.0	105592.0	104152.0

První nejmenší p_2 je pro $(\alpha, \beta) = (3.5, 0.5)$.
$m_{PAB}[0] = 99802.5$, $d_{PAB}[0] = 5397.15$, $m_{PAB}[m] = 377247$, $d_{PAB}[m] = 21067$.
Druhá nejmenší p_2 je pro $(\alpha, \beta) = (1.5, 0.5)$.
$m_{PAB}[0] = 99867.1$, $d_{PAB}[0] = 3499$, $m_{PAB}[m] = 376327$, $d_{PAB}[m] = 15064.5$.

(E) Úloha: `Ulohy\sit006\k16d01_blokaceDeleno10`, $h = [4]$.

Tabulka 10: $p_2()$

$\beta \setminus \alpha$	0.5	1.0	1.5	2.0	2.5	3.0	3.5
0.5	19278.2	19300.3	19376.0	19429.0	19391.2	19660.3	19241.9
1.0	18974.9	19148.3	19423.4	19040.4	19680.9	19226.1	19448.9
1.5	19615.2	19080.4	19401.8	19125.9	19461.8	19234.8	19346.2
2.0	19642.9	19277.4	19244.3	19541.0	19160.2	19163.6	19312.2
2.5	19785.6	19443.0	19461.3	19205.8	19149.1	19504.0	19124.7
3.0	19783.8	19586.6	19791.9	19348.8	19253.4	19116.2	19584.8
3.5	19980.5	19595.2	19129.0	19419.0	19543.3	19125.5	19311.2
4.0	19853.7	19791.5	19790.9	19667.0	19535.9	19422.7	19609.5
4.5	20263.0	19958.7	19838.8	19704.5	19803.5	19285.0	19565.2

První nejmenší p_2 je pro $(\alpha, \beta) = (0.5, 1.0)$.
$m_{PAB}[0] = 18974.9$, $d_{PAB}[0] = 443.472$, $m_{PAB}[m] = 72221.6$, $d_{PAB}[m] = 1583.43$.
Druhá nejmenší p_2 je pro $(\alpha, \beta) = (2.0, 1.0)$.
$m_{PAB}[0] = 19040.4$, $d_{PAB}[0] = 610.727$, $m_{PAB}[m] = 73010.1$, $d_{PAB}[m] = 3584.26$.
Výpočet zabral asi 8 hodin 45 minut 00 sekund.

(F) Úloha: `Ulohy\sit001\k21d02`, $h = [2, 2]$.
Podobá se úloze (Z) s tím rozdílem, že
`parametry["zpusobVyberuMravence"] = 'vm1'`.

Tabulka 11: $p_2()$

$\beta \setminus \alpha$	0.5	1.0	1.5	2.0	2.5	3.0	3.5
0.5	14619.7	14745.1	14339.1	14396.6	14144.3	14413.1	14507.8
1.0	14329.7	14187.3	14184.4	14203.1	14357.4	14157.5	14309.2
1.5	14332.2	14046.1	14118.6	14104.4	13961.5	13796.4	14140.1
2.0	13738.3	13878.0	13733.9	13753.9	13674.4	13848.2	13748.4
2.5	13668.1	13433.8	13610.2	13659.9	13825.1	13820.2	13762.4
3.0	13888.5	13768.6	13604.6	13417.2	13698.4	13591.1	13966.0
3.5	13743.3	13812.5	13564.5	13610.4	13810.6	13818.1	13557.0
4.0	13717.3	13974.1	13466.1	13487.9	13760.6	13549.6	13590.3
4.5	13672.1	13816.2	13755.8	13547.8	13437.2	13436.9	13564.8

První nejmenší p_2 je pro $(\alpha, \beta) = (2.0, 3.0)$.
$m_{PAB}[0] = 13417.2$, $d_{PAB}[0] = 646.51$, $m_{PAB}[m] = 49740.8$, $d_{PAB}[m] = 2295.92$.
Druhá nejmenší p_2 je pro $(\alpha, \beta) = (1.0, 2.5)$.
$m_{PAB}[0] = 13433.8$, $d_{PAB}[0] = 581.527$, $m_{PAB}[m] = 51304.4$, $d_{PAB}[m] = 1981.79$.
Výpočet zabral asi 52 hodin 9 minut 30 sekund.

Shrnutí:

Tabulka 12: První dvě nejlepší dvojice (α_1, β_1) a (α_2, β_2) ke každé úloze P.

$\alpha\beta \setminus$ P	Z	A	B	C	D	E	F
α_1	1.5	2.0	0.5	1.5	3.5	0.5	2.0
β_1	3.5	4.5	4.5	4.5	0.5	1.0	3.0
α_2	2.0	2.0	1.0	3.0	1.5	2.0	1.0
β_2	2.5	4.0	3.5	4.5	0.5	1.0	2.5

(1) Testujeme hypotézu: *Optimální* (α, β) *pro MMAS řešící úlohu* Π_2 *na dané síti s daným rozmístěním dep a čet nezávisí na rozmístění blokací a nezávisí na době oprav blokací.* K testu používáme měření cen řešení pro různé dvojice parametrů (α, β) v úlohách A a Z. Obě úlohy jsou řešeny nad stejnou sítí, se stejně rozmístěnými stanicemi a stejnými počty čet ve stanicích. Úloha A se liší od úlohy Z nejvíce rozmístěním a ohodnocením blokací. Doby oprav blokací jsou srovnatelné s ohodnocením hran v CG-síti a v obou případech pochází ze stejného rozdělení pravděpodobnosti.

Trochu se liší i počtem blokací (A 238b., Z 234b.). Podle výsledků v tabulce 12 se zdá, že optimální (α,β) vychází pro každou úlohu jinak. Jinými slovy: K dané síti, rozmístění stanic a počtu čet ve stanicích neexistuje (α,β), se kterým MMAS funguje optimálně při každé katastrofě. Protože správce silniční sítě dopředu neví, jaká nastane situace, může použít MMAS(1,1) a před rozdělením úkolů četám provést tolik iterací, kolik mu čas dovolí a z nich vybrat nejlepší řešení.

(2) Testujeme hypotézu: *Rozmístění čet ve stanicích nemá vliv na optimalitu* (α,β). Použijeme k tomu srovnání výsledků pro úlohy B a Z. B se liší od Z pouze rozmístěním čet ve stanicích. Pro Z je $h = [2,2]$, pro B je $h = [4,0]$. Tabulka 12 ukazuje, že (α^*,β^*) pro B, se liší výrazně od (α^*,β^*) pro Z. Proto pravděpodobně i rozmístění čet ve stanicích má vliv na optimalitu parametrů (α,β).

(3) *Jak se změní* (α^*,β^*) *se změnou dob oprav blokací?* Porovnáme výsledky řešení úlohy D s výsledky řešení úlohy E. Doby oprav r_E blokací v E jsou srovnatelné s ohodnocením hran v CG-síti. Jediné, v čem se úloha E liší od úlohy D, jsou hodnoty funkce r. Platí $\forall b \in B : r_D(b) = 10\, r_E(b)$. Z tabulky 12 vidíme, že β^* pro D je blízká β^* pro E, avšak α^* pro D se od α^* pro E liší výrazně. Zdá se, jako kdyby MMAS dokázal nacházet dobrá řešení až poté, co mu v úloze D bylo umožněno klást velký důraz na feromonovou stopu. Kdyby na ni kladl stejný důraz jako na heuristickou informaci (zde převrácenou hodnotu doby jízdy po hraně), nevyužíval by dostatečně informaci o tom, které trasy obsahují blokace s nízkými dobami oprav. Srovnáním výsledků úloh D, E jsme zjistili vliv zvětšení dob oprav blokací. Jaký vliv na (α^*,β^*) bude mít snížení až zanedbání dob oprav blokací? K tomuto porovnáme výsledky pro úlohu C s výsledky pro úlohu Z. Liší se jen v tom, že pro každou zablokovanou hranu b platí $r_C(b) = r_Z(b)/100$. Z tabulky 12 vidíme, že β^* pro C je blízká β^* pro Z a zároveň α^* pro C je stejné jako α^* pro Z. To je rozdíl proti předchozí situaci dvojice D a E. Zdá se, že zvýšení dob oprav má na (α^*,β^*) větší vliv, než jejich snížení.

(4) *Jak se změní* (α^*,β^*)*, když tutéž úlohu budeme řešit s MMAS, který během konstrukce využívá jiný způsob výběru mravence?* Úloha F je stejná, jako úloha Z. Liší se od sebe jen tím, že k řešení Z byl použit MMAS s výběrem mravence způsobem `'vm2'`, zatímco k řešení F byl použit MMAS s výběrem mravence způsobem `'vm1'`. Srovnáním sloupců F, Z v tabulce 12 vidíme, že se (α^*,β^*) pro F příliš neliší od (α^*,β^*) pro Z, a tedy způsob výběru mravence na optimální hodnoty α,β nemá velký vliv.

Závěr

Cílem práce bylo najít způsob řešení úlohy napojování komponent v rozpadlé silniční síti a napsat počítačový program, který navrhne nejlepší postup pro opravu komunikací. Dají se uvažovat dvě varianty této úlohy. (1) Zablokované silnice jsou neprůjezdné pro osobní přepravu, ale cestáři blokace dokáží překonat bez ztráty času. (2) Jediný způsob, jak projet přes zablokovanou silnici, je odstranit blokaci.

Autor diplomové práce se zaměřil na řešení první varianty úlohy v podobě zobecněné dvěma způsoby: (a) Cestáři sice mohli objet kteroukoliv blokaci, aniž by ji opravovali, ale pokud dostali zadáno některou z blokací odstranit, doba práce zabrala nenulový čas. (b) V původní práci vedoucí řešil situace s jednou stanicí cestářů. Autor diplomové práce řešil situace s obecným počtem stanic a volitelným počtem čet ve stanicích.

Výsledkem je sedm skriptů napsaných v jazyce Python, data několika testovacích rovinných sítí a situací v nich a zjištění, co má jaký vliv na volbu optimálních hodnot parametrů hlavního algoritmu, který úlohu řeší. Úlohu řeší Max-Min Ant System (MMAS) přizpůsobený úloze napojování komponent. Kódy jsou psány s co nejmenším použitím knihoven třetích stran, takže pro toho, kdo by potřeboval rychlejší výpočty, by nemělo být složité přepsat obsah skriptů do jazyka C.

Je-li dána síť, rozmístění stanic a čet v nich, bylo by dobré získat pro tohle zadání hodnoty parametrů, se kterými MMAS funguje nejlépe pro libovolnou katastrofu, která v síti nastane. Zjistili jsme, že takové nastavení neexistuje. Pro každou katastrofu v téže síti je vhodné jiné nastavení α^*, β^*, pro něž MMAS funguje nejlépe. Zjistit optimální hodnoty parametrů α, β řešiče pro danou situaci je časově náročnější, než samotný výpočet řešení a to nelze dělat pro každou novou situaci. Takže nastavíme α, β na pevné hodnoty s vědomím toho, že v některých situacích MMAS bude pracovat hůře, než v jiných.

Dalším přínosem je komponentový multigraf rozpadlé sítě. V tomto modelu komponenta rozpadlé sítě je jedním vrcholem a množina zablokovaných hran, které dělí od sebe sousední komponenty, je multihranou. Díky tomu lze uvažovat o kostrách multigrafu jako o grafech, jejichž množina hran je minimální (ne nutně nejlevnější) množinou blokací, které stačí odstranit, aby došlo k napojení komponent rozpadlé sítě. Tím získáváme nutnou podmínku optimality řešení. Ta je zakódována do našeho způsobu konstrukce přípustných řešení.

Za zmínku také stojí popis převodu rozvozního problému (VRP) na úlohu napojování komponent (CR). Uměním řešit CR umíme novým způsobem řešit VRP a tím i problém obchodního cestujícího. Převést CR na VRP se autorovi nepodařilo, což neznamená, že to nejde. Komu se to podaří, bude moci bez modifikací použít metody řešení VRP k řešení CR. V těchto převodech nachází uplatnění komponentový multigraf.

Bylo by pěkné, kdyby někdo zpracoval druhou variantu úlohy napojování komponent. Blokací, které nelze objet, může být někdy tolik a tak rozmístěných, že metoda řešení z této práce nebude použitelná. Druhá varianta úlohy se zdá být ještě složitější, než první varianta, protože s tím, jak četa cestářů odstraní jednu blokaci, se rozšíří ostatním četám z téže stanice možnosti, kam se dostanou, což s sebou nese potřebu s každou odstraněnou blokací přepočítat nejkratší cesty v aktuální CG-síti, protože ty se budou měnit v závislosti na nově průjezdných hranách. Zdání klame. Možná dokážete, že P = NP.

Seznam literatury

[1] Bíl Michal, Vodák Rostislav, Kubeček Jan, Bílová Martina, Sedoník Jiří. *Evaluating road network damage caused by natural disasters in the Czech Republic between 1997 and 2010.* Transportation Research Part A: Policy and Practice. 2015; 80: 90-103.

[2] Vodák Rostislav, Bíl Michal, Křivánková Zuzana. *A modified ant colony optimization algorithm to increase the speed of road network recovery process after disasters.* International Journal of Disaster Risk Reduction. 2018; 31: 1092-1106.

[3] Balcar Bohuslav, Štěpánek Petr. *Teorie množin.* 2nd ed. Praha: Academia, nakladatelství AVČR; 2001; 462 p.

[4] Švrček Jaroslav. *Úvod do kombinatoriky.* 3rd ed. Olomouc: Vydavatelství Univerzity Palackého; 2019. 83 p.

[5] Hankin Robin K.S. *Additive integer partitions in R.* Journal of Statistical Software, Code Snippets. 2006; 16: 1.

[6] Demel Jiří. *Grafy.* 1st ed. Praha: SNTL; 1988. 184 p.

[7] Chytil Michal. *Automaty a gramatiky.* 1st ed. Praha: SNTL; 1984. 336 p.

[8] Garey Michael R., Johnson David S. *COMPUTERS AND INTRACTABILITY, A Guide to the Theory of NP-Completeness.* New York: W. H. Freeman and Company; 1979. 340 p.

[9] Machalová Jitka, Netuka Horymír. *Numerické metody nepodmíněné optimalizace.* 1st ed. Olomouc: Vydavatelství Univerzity Palackého; 2013. 142 p.

[10] Machalová Jitka, Netuka Horymír. *Nelineární programování: Teorie a metody.* 1st ed. Olomouc: Vydavatelství Univerzity Palackého; 2013. 165 p.

[11] Ženčák Pavel. *Lineární programování.* 1st ed. Olomouc: Vydavatelství Univerzity Palackého; 2013. 195 p.

[12] Applegate David L., Bixby Robert E., Chvátal Vašek, Cook William J. *The Traveling Salesman Problem, A Computational Study.* Princeton: Princeton University Press; 2006. 593 p.

[13] Toth Paolo, Vigo Daniele. *Vehicle Routing Problems, Methods and Applications.* Philadelphia: MOS-SIAM; 2014. 463 p.

[14] Renaud J., Laporte G., Boctor F. F. *A tabu search heuristic for the multi-depot vehicle routing problem.* Computers & Operations Research. 1996; 23: 229–235.

[15] Buchin Kevin, Schulz André. *On the Number of Spanning Trees a Planar Graph Can Have.* 2018; arXiv:0912.0712v2 [math.CO] 5 Sep 2010.

[16] Clarke G., Wright J.W. *Scheduling of vehicles from a central depot to a number of delivery points.* Operations Research. 1964; 12: 568–581.

[17] Croes G. A. *A Method for Solving Traveling Salesman Problems.* Oper. Res. 1958; 5: 791-812.

[18] Lin Shen. *Computer Solutions of the Traveling Salesman Problem.* Bell System Technical Journal. 1965; 44: 2245-2269.

[19] Goss S., Aron S., Deneubourg J.L., Pasteels J.M. *Self-organized Shortcuts in the Argentine Ant.* The Science of Nature. 1989; 76: 579-581.

[20] Dorigo Marco, Stützle Thomas. *Ant Colony Optimization.* Cambridge, Massachusetts: The MIT Press; 2004. 305 p.

[21] Dorigo, M., Maniezzo, V., Colorni, A. *Positive feedback as a search strategy.* Technical report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milan. 1991.

[22] Dorigo, M., Maniezzo, V., Colorni, A. *The Ant System: An autocatalytic optimizing process.* Technical report 91-016 revised, Dipartimento di Elettronica, Politecnico di Milano, Milan. 1991.

[23] Stützle Thomas, Hoos Holger H. *MAX-MIN Ant System.* Future Generation Computer Systems. 2000; 16: 889–914.

Seznam souborů přiložených na CD

```
Prilohy\..
  Ulohy\..
    sit001\..
    sit002\..
    sit003\..
    sit004\..
    sit005\..
    sit006\..
  Kody\..
    reseniA.py
    reseniM.py
    reseniZ.py
    hlavni.py
    uloha.py
    pomocne.py
    vytvorCGdata.py
    vysledky\..
    vznikSiti\..
      kreator.R
      destruktor.R
      zdroje.R
  Testy\..
    ACOvsNahoda\..
    AUaVyber\..
    alfabeta\..
```