Fakulta Matematiky, Fyziky a Informatiky, Univerzita Komenského v Bratislave

Ekonomická a finančná matematika

Hľadanie najkratšej cesty v grafoch pomocou riešení diferenciálnych rovníc

Diplomová práca

Diplomant: Michal Antol

Vedúci diplomovej práce: Doc RNDr. Daniel Ševčovič, Csc.

Bratislava 2010

Univerzita Komenského v Bratislave

Fakulta matematiky, fyziky a informatiky

HĽADANIE NAJKRATŠEJ CESTY V GRAFOCH POMOCOU RIEŠENÍ DIFERENCIÁLNYCH ROVNÍC

Diplomová práca

Študijný program: **9.1.9 Aplikovaná matematika**

Študijný odbor: **Ekonomická a finančná matematika**

Školiace pracovisko: **Katedra aplikovanej matematiky a štatistiky**

Školite3: **Doc RNDr. Daniel Ševčovič, Csc.**

Bratislava 2010

Michal Antol

Poďakovanie

Poďakovanie

Ďakujem svojmu vedúcemu Doc. RNDr. Danielovi Ševčovičovi, Csc. za ochotu, cenné rady a usmernenie pri písaní diplomovej práce.

Čestné prehlásenie

Prehlasujem, že som prácu vypracoval samostatne s využitím svojich poznatkov a s použitím uvedenej literatúry.

V Bratislave 25. apríla 2010

_____

podpis študenta

Abstrakt

ANTOL, Michal: ***Hľadanie najkratšej cesty v grafoch pomocou riešení diferenciálnych rovníc.*** [Diplomová práca] - Univerzita Komenského v Bratislava. Fakulta matematiky, fyziky a informatiky; Katedra aplikovanej matematiky a štatistiky. - Vedúci: Doc RNDr. Daniel Ševčovič, Csc.. Bratislava, 2010

Najpoužívanejsím a najobľúbenejším algoritmom na hľadanie najkratšej cesty v grafe je Dijkstrov algoritmus a jeho rôzne modifikácie. Nový spôsob na hľadanie najkratšej cesty v grafe je algoritmus založený na biologickom sa správaní jednoduchého organizmu - vápentky mnohohlavej a jeho rôzne obmeny. V práci sa snažíme zistiť efektívnosť týchto algoritmov a to z hľadiska výpočtovej zložitosti a časovej náročnosti.

Kľúčové slová: Dijktrov algoritmus, Vápenatka Mnohohlavá, najkratšia cesta v grafe.

Abstract

ANTOL, Michal: ***Hľadanie najkratšej cesty v grafoch pomocou riešení diferenciálnych rovníc.***

[Diplom theses] - Comenius University in Bratislava. Faculty of Mathematics, Physics and Informatics; Department of Aplied Mathematics and Statistics. - Tutor: Doc. RNDr. Daniel Ševčovič, Csc. Bratislava, 2010,

The most commonly used and most popular algorithm to solve the shortest path problem in the graph is Dijkstra's algorithm and its various modification. A new method for finding the shortest path in graph algorithm is based on the behavior of simple biological organism - Physarum Polycephalum and its various modifications. The work is to analyze the efficiency of these algorithms and in terms of computational complexity and time involved.

Key words: Dijkstra algorithm, Physarum Polycephalum, shortpath problem,

Obsah

Úvod **8**

1 Teória grafov a najkratšie cesty **9**

1.1 Optimálne sledy 9

1.2 Motivácia 9

1.3 Primov algoritmus 10

1.3.1 Algoritmus 11

1.4 Idea Dijkstrovho algoritmu 11

1.5 Dijkstrov algoritmus 13

1.5.1 algoritmus 14

1.6 Zložitosť algoritmu 15

2 Najkratšia cesta v grafoch a nové možnosti **16**

2.1 Význam vápenatky mnohohlavej 16

2.2 Vápenatka a bludisko 17

2.3 Physarum solver 18

2.4 Algoritmus 21

2.5 Analógia s elektrickým obvodom 21

2.6 Zložitosť algoritmu 22

3 Porovnanie algoritmov **23**

3.1 Cieľ práce 23

3.2 Úprava algoritmu vápenatky do matlabu 24

3.2.1 Gauss-Seidelova relaxačná SOR metóda 27

3.3 Problém bludiska 29

3.3.1 Algoritmus vápenatky 29

3.3.2 Dijkstrov algorimus 33

3.4 Tlakové pasmo - Koliba 34
3.4.1 Algoritmus vápenatky 35
3.4.2 Dijkstrov algoritmus 37
3.5 Simulácia poruchy vodovodného potrubia 38
3.6 Prvé tlakové pásmo Bratislavy 40
3.6.1 Dijsktrov algoritmus 40
3.6.2 Algoritmus vápenatky 40

4 Záver **43**

Úvod

Hľadanie najkratšej cesty v grafe vychádza z reálnych problémov, ktoré môžme nájsť všade okolo nás. Každý deň riešime jednoduché príklady, ako sa dostaneme z miesta A do miesta B. Pričom si potrebujeme vybrať z rôznych možných ciest. Cieľom je nájsť takú cestu, ktorá bude pre nás optimálna. Závisí to od viacerých faktorov, podľa ktorých jednotlivé cesty dostávajú svoje váhy. Optimálna je potom tá, ktorej súčet váh jednotlivých ciest je najmenší, je to najkratšia cesta. Existujú rôzne algoritmy, ktoré dokážu riešiť problém tohoto typu. Najpopularnejší a najpoužívanejší algoritmus na hľadanie najkratšej cesty v grafe je Dijkstrov algoritmus.

V prvej kapitole sa oboznámime práve s Dijkstrovým algoritmom, ako vznikol, a na akom jednoduchom systéme funguje.

V druhej kapitole bude načrtnutý nový spôsob hľadania najkratšej cesty v grafe a to pomocou algoritmu, ktorý je založený na biologickom sa správaní jednoduchého organizmu vápenatky mnohohlavej.

V tretej kapitole budeme porovnávať tieto dva algoritmy na jednotlivých príkladoch. Najprv na triviálnych a jednoduchých, neskôr na zložitejších príkladoch. Našou úlohou bude modifikovať matematický model navrhnutý na základe vápenatky tak, aby bol schopný konkurovať Dijkstrovmu algoritmu. Predpoklad je, že v jednoduchých príkladoch budú algoritmi porovnateľne avšak v zložitejších bude Dijkstrov algoritmus efektívnejší.

Dôvodom prečo je vhodné sa týmto zaoberať je že Dijkstrov algorimus je časovo náročný pri veľkých počtoch vrcholov, a preto je vhodné hľadať algoritmy, ktoré by ho mohli nahradiť.

1 Teória grafov a najkratšie cesty

Najdôležitejším pojmom v teórií grafov predstavuje samotný pojem graf G. Graf je vlastne formálne povedané konečná množina vrcholov V a množina (môže byť aj prázdna množina) dvojíc vrcholov (hrán) E. Vrcholy nám vlastne predstavujú jednotlivé stanovištia. A hrany E nám reprezentujú jednotlivé cesty medzi nimi. V aplikáciách teórie grafov sa často vyskytujú grafy sprevádzané ďalšou informáciou, ktorá nám opisuje realitu dokonalejšie, ako samotný graf. Podstatnejšie zväčšenie informácie prináša pribranie číselných údajov na hrany, resp. vrcholy.

1.1 Optimálne sledy

Mnoho reálnych optimalizačných úloh možno previesť na úlohu nájsť optimálny sled (obyčajne najkratšiu cestu). Sled S v grafe G nazveme striedavú postupnosť vrcholov a hrán $v_0, e_1, v_1, e_2,...,v_{p-1}, e_p, v_p$ takú, že $e_i = [v_{i-1}, v_i]$. Hovoríme, že sled začína vo v_0 a končí vo v_p. Tieto úlohy treba spresniť podľa toho, či je to úloha o grafe, alebo digrafe (orientovaný graf), či sú hrany ohodnotené a pod.

V nasledujúcich kapitolách sa zameriame prevažne na hľadanie práve najkratšej cesty. V teórií grafov hľadanie najkratšej cesty znamená nájsť takú cestu medzi dvoma vrcholmi (uzlami), aby súčet váh hrán, cez ktoré sa dostaneme od jedného vrcholu k druhému bol čo najmenší. Na hľadanie nakratšej cesty sa používa mnoho algoritmov. Z nich najpopulárnejší je dijsktrov algoritmus.

1.2 Motivácia

Teraz si uvedieme reálny problém, ktorý možno previesť na problém hľadania najkratšej cesty alebo resp. hľadania optimálnej cesty. A prečo by bolo vhodné sa týmto problémom najkratšej cesty zaoberať.

Uvažujme cestnú sieť nejakého okresu Slovenska. Úlohou je nájsť najrýchlejšiu

trasu z daného bodu s do daného bodu t. V tomto prípade, vrcholy grafu predstavujú mestá a každý úsek cesty i, j medzi vrcholom i a j má kladnú dĺžku $c_{i,j}$. Namiesto dĺžky by mohol byť aj čas potrebný na prejdenie daného úseku. Úlohou je potom nájsť najkratší $s - t$ sled v tomto vytvorenom grafe.

1.3 Primov algoritmus

Za priameho predchodcu Dijkstrovho algoritmu môžme pokladať primov algoritmus, ktorý objavil Vojtěch Jarnik (1930) a zároveň o veľa rokov neskôr nezávisle na Jarnikovi Robert Prim (1957). Algoritmus sa častejšie označuje ako Primov algoritmus. Týmto algoritmom sa inšpiroval holandský informatik Edsger Dijkstra, ktorý na jeho základe vytvoril Dijsktrov algoritmus na hľadanie najktratšej cesty v grafe. Vzhľadom na tieto historické okolnosti môžme naraziť aj na pomenovanie tohto algoritmu ako DJP algoritmus (podľa skratky mien Dikjstra, Jarnik, Prim).

Primov algoritmus sa používa na hľadanie najlacnejšej kostry grafu. Kostra v grafe G je podmnožina hrán pôvodného grafu taká, že zabezpečuje cesty medzi dvojicami vrcholov. V neorientovanom grafe je to podgraf G (strom T) taký, že existuje cesta medzi ľubovoľnými dvoma vrcholmi grafu G len po hranách stromu T. Najlacnejšia kostra je taká, spomedzi všetkých kostier G, ktorá má minimálny súčet ohodnotení hrán. Problémom nájdenia najlacnejšej kostry sa ako prvý zaoberal Otakar Borůvka (1926) inspirovaný úlohou navrhovania optimálnej elektrovodnej siete. Jeho postup vylepšili Jarnik s Primom.

Idea primovho algoritmu je nasledovná. Z grafu vyberieme ľubovoľný vrchol v. Tým nám vznikne podgraf G_1. Do minimálnej kostry vyberieme hranu s minimálnym ohodnotením, ktorá je z hrán vychádzajúcich z vrcholu v. Vznikne podgraf, ktorý obsahuje vrchol v, minimálnu hranu a s ňou ďalší incidentný vrchol. Tento podgraf nazveme G_2. Máme vytvorený podgraf G_i. Podgraf G_{i+1} vytvoríme tak, že vyberieme

hranu s najmenším ohodnotením vychádzajúcu z podgrafu G_i a končí mimo neho. Pri konečných grafoch tento postup skončí, pri n vrcholovom grafe po n krokoch. Výsledný podgraf je minimálna kostra grafu G. Tento algoritmus vychádza z myšlienky dynamického programovania. Rozmer úlohy sa zmenší na minimum, preto sa musí zanedbať jedna z podmienok platných pre výsledok. Potom sa postupne zväčšuje rozmer úlohy, až kým s nedostaneme k správnemu riešeniu.

1.3.1 Algoritmus

1. Zvolíme ľubovoľný vrchol u grafu G. Položíme $V(T) := \{u\}$, $H(T) := \emptyset$, $M := V(G) - \{u\}$. Pre každý vrchol v M položíme $V_v := u$ a $d_v := f(\{u, v\})$, ak $\{u, v\}$ je hrana v grafe G. V opačnom prípade V_v nedefinujeme a $d_v := \infty$.

2. Ak je $M = \emptyset$, výpočet končí.

3. Zvolíme vrchol $v \in M$, pre ktorý je číslo d_v minimálne. Pridáme v do $V(T)$, hranu $\{v, V_v\}$ do $H(T)$ a vyberieme vrchol z množiny M.

4. Pre každý vrchol $w \in M$, pre ktorý existuje hrana $\{v, w\}$ grafu G vychádzajúca z v a $f(\{v, w\}) < dw$, zmeníme V_w a d_w takto: $V_w := v$, $d_w := f(\{v, w\})$. Premenné V_w a d_w ostatných vrcholov ponecháme nezmenené.

5. Skok na bod 2.

1.4 Idea Dijkstrovho algoritmu

Z predchádzajúceho algoritmu vychádza Dijkstrov algoritmus. Dijsktrov algoritmus patrí medzi tzv. "hladné algoritmy"(greedy algorithm), to znamená že algoritmus nerozmýšľa dopredu, ale v každom kroku jednoducho zvolí najlepšiu z momentálnych možností. Nech máme n vrcholov, kde niektoré dvojice sú spojené hranami, kde dĺžka

každej hrany je daná. Predpokladajme že existuje aspon jedna cesta medzi každými dvoma uzlami. V originálnom článku [1] sa Dijkstrus zameral na 2 problémy:

- Skonštruovať minimálnu kostru grafu.
- Nájsť najkratšiu cestu medzi dvoma danými vrcholmi S a T.

Keďže sa budeme v nasledujúcich kapitolách venovať hľadaniu najkratšej cesty, tak sa zameriame na druhý problém. Uvažujme dva vrcholy S, T, pričom sa snažíme nájsť optimálnu (najkratšiu) cestu $s - t$. Využijeme skutočnosť, že ak vrchol X leží na najkratšej $s - t$ ceste, tak to vedie k predpokladu, že existuje najkratšia cesta $s - x$ medzi vrcholmi S a X. Takto sa postupne pripájajú vrcholy k uzlu S za účelom dosiahnutia vrcholu T [1]. Vrcholy sú rozdelené do 3 podmnožín:

A) V množine A sa nachádzajú vrcholy, ktorých najkratšia cesta z uzlu S je známa. Vrcholy sa pridávajú do tejto množiny za účelom dosiahnutia vrcholu T.

B) V množine B sa nachádzajú vrcholy, ktoré sú spojené najmenej s jedným vrcholom z množiny A. Ale ešte nepatria do množiny A. Z ktorých v ďalšom kroku vyberáme vrchol, ktorý sa má do množiny A pridať.

C) Zostávajúce vrcholy.

Hrany sú taktiež rozdelené do 3 podmnožín:

I) Hrany ktoré sú obsiahnuté v najkratšej ceste z vrcholu S do vrcholov v množine A.

II) Hrany z ktorých v ďalšom kroku vyberáme hranu, ktorá sa má pridať do množiny I. Ku každému vrcholu z množiny B bude viesť práve jedna hrana.

III) Zostávajúce hrany. Hrany, ktoré ešte neboli posúdené, alebo už nevhodné hrany.

Pomocou tohto rozdelenia môžme Dijkstrov algoritmus zhrnúť do nasledujúcich 2 krokov [1]:

1. V prvom kroku musíme posúdiť všetky hrany v také, ktoré spájajú vrchol, ktorý bol práve presunutý do množiny A s uzlami X, ktoré sú v množinách B alebo C. Ak vrchol X patrí do množiny B, tak musíme vyšetriť, či hrana v nám dáva kratšiu cestu z S do X, ako dovtedy známa cesta využívajúca hrany z množiny II. Ak hrana dáva kratšiu cestu $S - X$, tak nahradí zodpovedajúce hrany z množiny II, inak je zamietnutá. Ak vrchol X patrí do množiny C, tak je pridaný do množiny B a hrana, ktorá spája X s nejakým vrcholom z množiny A je pridaná do množiny hrán II.

2. Každý vrchol z množiny B môže byť spojený s vrcholom S iba jednou hranou, tieto hrany majú rôzne dĺžky. Vrchol s minimálnou vzdialenosťou od vrcholu S je presunutý z B do A. Zodpovedajúca hrana je presunutá z II do I. Teraz sa vraciame na krok 1.

Celý proces sa opakuje, pokiaľ nie je vrchol T preradený do množiny A. To znamená, že algoritmus končí, lebo sa našlo riešenie. Zaujímavosťou je že celý článok [1] bol veľmi krátky (2 strany) bez obrázkov, alebo matematického vyjadrenia algoritmu.

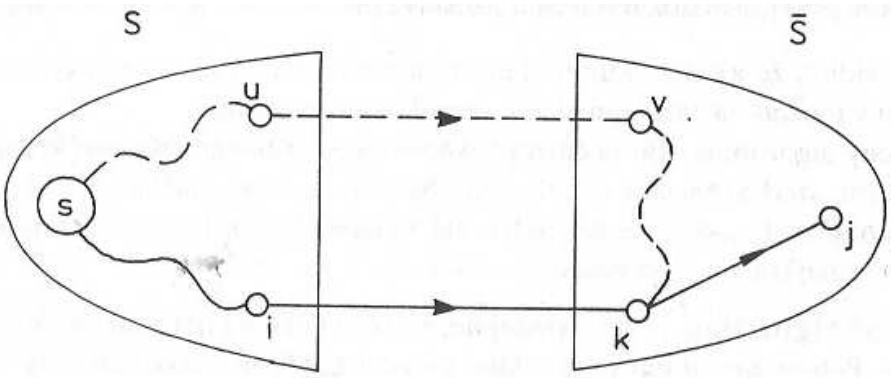
1.5 Dijkstrov algoritmus

V predchádzajúcej kapitole bola vysvetlená samotná idea Dijkstrovho algoritmu. Predpokladáme, že dĺžka každej hrany ij je $c_{i,j}$. A každý vrchol j dostáva značku (p_j,d_j). To znamená predchádzajúci vrchol a dĺžku medzi predchádzajúcim vrcholom a daným vrcholom. Značky sú dočasné, a postupne sa budú meniť na trvalé značky [3].

Na začiatku dostane vrchol S teda začiatočný vrchol označenie $(0, 0)$ a každý iný vrchol j dočasné označenie (s, ∞). Nech v algoritme S označuje množinu vrcholov,

ktoré majú už permanentnú značku a $S_1 = V_G - S$. V_G predstavujú všetky vrcholy grafu. Na začiatku $S = \{s\}$.

Všeobecný krok algoritmu potom vyzerá nasledovne. V množine S_1 nájdeme taký vrchol k, že $d_k = min\{d_j | j \in S_1\}$ a presunieme ho do S. Potom dočasné označenie pozmeníme. Pre každé j z novej množiny S_1 označenie ponecháme, ak $d_j \leq d_k + c_{kj}$ v opačnom prípade ju zmeníme na $(k, d_k + c_{k,j})$ [3].



Obr. 1: idea rozdelenia vrcholov [3]

1.5.1 algoritmus

Algoritmus sa dá potom podľa [3] zapísať nasledovne:

1. $S = \{s\}$, $S_1 := V_G - S$, $(p_s, p_d) := (0, 0)$ a $(p_j.d_j) := (s, c_{s,j})$ pre $j \in S_1$, kde $c_{s,j} = \infty$ pre sj "nepatrí"E_G.

2. Nájdeme vrchol $k \in S_1$ taký, že $d_k = min\{d_j | j \in S_1\}$. Ak $d_k = \infty$, STOP (žiadna $s - S_1$ cesta neexistuje). $S := S \cup \{k\}$, $S_1 := S_1 - \{k\}$. Ak $S_1 = \emptyset$, STOP (dĺžka najkratšej $s - t$ cesty je d_t). Inak ideme na krok 3.

3. Pre každé $j \in S_1 \cap V^+(k)$ ak $d_j > d_k + c_{k,j}$, tak urobíme $(p_j, d_j) := (k, d_k + c_{k,j})$ inak sa značka nemení. Vraciame sa na krok 2.

Je vhodné, aby pri minimalizácií v kroku 2 kandidoval vrchol t na k ako prvý. Pretože výpočet možno ukončiť hneď ako sa vrchol t dostane do S [3].

1.6 Zložitosť algoritmu

Krok 1 je zanedbateľný. Krok 2 si vyžaduje $O(n)$ operácií a robí sa $n-1$ krát, teda celkovo to je $O(n^2)$ operácií. Rovnako je to aj v kroku 3 a to znamená, že celková zložitosť algoritmu je $O(n^2)$ [3].

2 Najkratšia cesta v grafoch a nové možnosti

Ako nová možnosť riešenia problému najkratšej cesty sa nám može javiť najnovší objav japonských vedcov R. Kobayashiho a A. Tera o jednoduchom organizme, ktorý sa volá vápenatka mnohohlavá. Vedci nasimulovali jednoduchý matematický model, ktorý je založený na správaní sa vápenatky a ktorý dokáže efektívne vyriešiť problém najkratšej cesty v grafe.

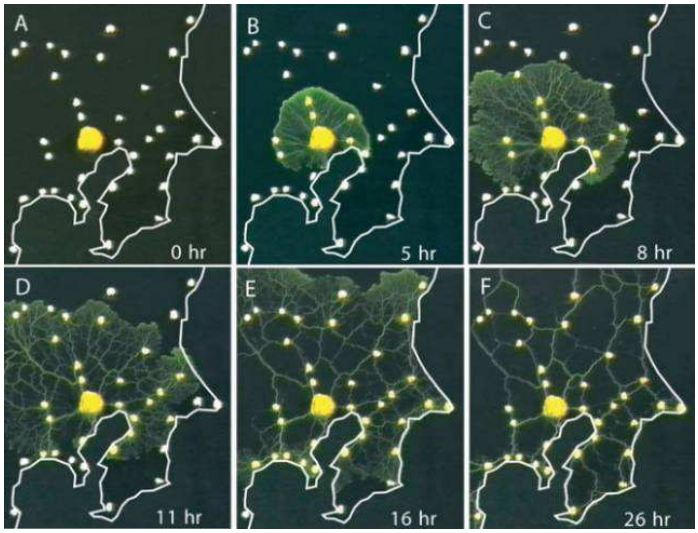
2.1 Význam vápenatky mnohohlavej

Vápenatka mnohohlavá (Physarum Polycephalum) je veľký améboidný organizmus s intracelulárnou štruktúrou, ktorá pozostáva zo siete rúr alebo tunelov (obr.2). Sieť týchto tunelov slúži na prepravu chemických a fyzikálnych signálov a na cirkuláciu živín a metabolitov v celom organizme. Taktiež slúži ako orgán pohybu. Vápenatka sa môže vyskytovať aj ako útvar jasne žltej farby, takzvané plazmódium, ktoré je vlastne jednou obrovskou mnohojadrovou bunkou. Toto plazmódium sa môže rozrastať najrôznejšími smermi a vytvoriť super organizmus o rozlohe až niekoľko štvorcových metrov. Tento multi-funkčný systém je teda fyziologicky významný pre vápenatku. Tvar siete tunelov sa dramaticky mení, keď vápenatka reaguje na rôzne podnety z vonkajšieho okolia a tiež na zmenu životného prostredia. Napríklad ak sa súčasne na dosku kde sa nachádza vápenatka umiestnia zdroje živín, tak sa ich snaží pospájať tak, aby tunely vo vzniknutej sieti mali čo najkratšiu vzdialenosť [6].

Skutočnosť ako dokáže vápenatka nájsť najkratšiu cestu medzi jednotlivými zdrojmi živín je predmetom výskumu, pretože používaním tohto jednoduchého systému prežíva vápentka už viac ako miliardu rokov.

Vedci pod vedením Atsushi Tera uskutočnili zaujímavý pokus. Ovsené vločky rozložili okolo centra vápenatky presne do podoby slepej mapy siete železníc Tokia a jeho okolia, resp. 36 miest, ktoré ho obklopujú. Na ich prekvapenie tunely vápenatky

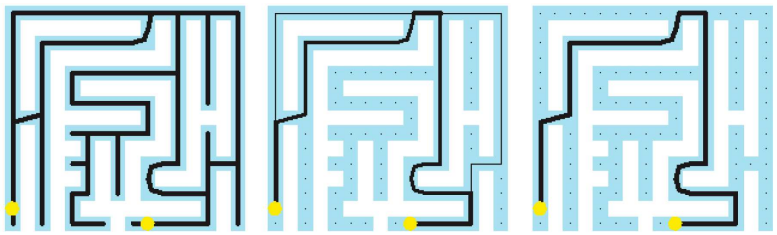
ktoré pospájali jednotlivé vločky vytvorili navlas rovnakú sieť, ako je tá, ktorá v skutočnosti už existuje. Sieť železníc, ktorá jednotlivé miesta prepojuje. Vapenatka sa vlastne snaží minimalizovať náklady a maximalizovať efektivitu.



Obr. 2: sieť železníc podľa vápenatky

2.2 Vápenatka a bludisko

Vápenatka dokáže vyriešiť aj problém bludiska. Ak sa zdroje živín nachádzajú na obidvoch východoch z bludiska, organizmus, ktorý sa rozširuje cez celé bludisko všetkými smermi je schopný nájsť spojenie medzi týmito východmi už v priebehu niekoľkých dní a to tak že vyberie najkratší tunel ktorý spája tieto zdroje z pomedzi mnohých ostatných možností. Táto schopnosť sa dá nasimulovať do jednoduchého matematického modelu, ktorý sa nazýva physarum solver [6]. Model bol založený na experimentálnom pozorovaní biologického správania sa vápenatky.

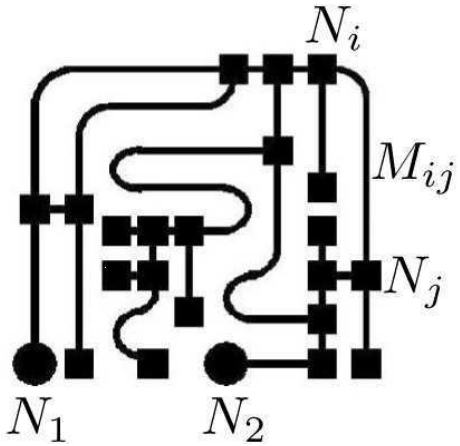


Obr. 3: vápenatka dokáže nájsť cestu z bludiska [6]

2.3 Physarum solver

Matematický model physarum solver pozostáva zo sady rovníc, ktoré reprezentujú protoplazmatický tok a schopnosť tunelov, alebo trubíc meniť svoju hrúbku. V tomto modeli tvar tela vápenatky reprezentuje graf (obr.4), pričom hrany grafu predstavujú jednotlivé trubice. A vrcholy sú jednotlivé uzly, kde sa trubice križujú.

Dva špeciálne uzly, ktoré predstavujú zdroje potravy sú označené ako $N1$ a $N2$ ostatné uzly sú označené $N3$, $N4$, $N5$ atď. Jeden z uzlov $N1, N2$ sa vždy javí ako "source", to znamená, že odtiaľ tok pochádza(kladné hodnoty) a druhý ako "sink", kde sa tok stráca (záporné hodnoty). Hrana medzi uzlami i a j je označená ako $M_{i,j}$.



Obr. 4: bludisko prerobené na sieť vrcholov a hrán [7]

Predpokladajme, že tlaky vo vrcholoch i a j sú p_i a p_j a že vrcholy i a j su spojené tunelom (valcom) dĺžky $L_{i,j}$ s polomerom $r_{i,j}$. Potom tok $Q_{i,j}$ ktorý preteká medzi vrcholmi i a j môžeme zapisať do rovnice:

$$Q_{i,j} = \frac{\pi r^4 (p_i - p_j)}{8\xi L_{i,j}},$$

Táto rovnica sa dá zapísať v tvare:

$$Q_{i,j} = \frac{D_{i,j}}{L_{i,j}}(p_i - p_j) \tag{1}$$

Pričom ξ predstavuje vlastne vodivosť kvapaliny a

$$D_{i,j} = \frac{\pi r^4}{8\xi}$$

V každom uzle musia platiť kirchoffove zákony a to znamená že v uzloch musí byť prítok a odtok vyvážený.

$$\sum Q_{i,j} = 0 \qquad (i \neq 1, 2) \tag{2}$$

V uzloch, ktoré predstavujú zdroje potravy je rovnica (2) modifikovaná, podľa toho, či ide o uzol $N1$ to znamená uzol, ktorý predstavuje zdroj (source, i=1), alebo o uzol $N2$, ktorý predstavuje uzol, kde sa tok stráca (sink, i=2).

$$\sum_j Q_{1j} - I_0 = 0 \tag{3}$$
$$\sum_j Q_{2j} + I_0 = 0$$

Kde I_0 je tok, ktorý vyteká z uzlu $N2$ a vteká do uzlu $N1$. I_0 je konštanta v tomto matematickom modeli [7].

Dĺžky jednotlivých hrán $L_{i,j}$ medzi uzlami sú behom procesu adaptácie vápenatky konštantné. To znamená, keď sa vápenatka snaží nájsť najkratšiu cestu, ktorá

spojí uzol $N1$ s uzlom $N2$ z pomedzi mnohých ďaľších možností. Prietoky medzi jednotlivými uzlami $Q_{i,j}$ pre danú konštantu I_0, parametre vodivosti $D_{i,j}$ a dĺžku hrán $L_{i,j}$ dostaneme potom pomocou rovníc (1),(2) a (3) nasledovne:

$$\sum_i \frac{D_{i,j}}{L_{i,j}}(p_i - p_j) = \begin{cases} -I_0 & \text{pre } j = 1; \\ I_0 & \text{pre } j = 2; \\ 0 & \text{inak ;} \end{cases} \tag{4}$$

Stanovením základného tlaku $p_2 = 0$, sa dajú ostatné tlaky p_i v jednotlivých uzloch vyrátať zo sústavy rovníc (4). Potom spätným dosadením do rovnice (1) dostaneme aj prietoky $Q_{i,j}$ medzi uzlami [7].

Dostali sme teda počiatočné prietoky $Q_{i,j}$. Aby bolo možné opísať zmeny polomerov jednotlivých tunelov. To znamená polomery jednotlivých tunelov sa budú zmenšovať vtedy, ak nevedú k najkratšej ceste. A to bude viesť až k ich zaniknutiu. A naopak zväčšovať sa budú vtedy ak sa jedná o najkratšiu cestu. Predpokladáme, že parametre vodivosti $D_{i,j}$ sa budú meniť s časom v závislosti od prietokov $Q_{i,j}$. A ich zmeny vieme vyjadriť pomocou diferenciálnej rovnice:

$$\frac{d}{dt}D_{i,j} = f(|Q_{i,j}|) - \alpha D_{i,j}, \tag{5}$$

Prvý člen pravej strany rovnice predstavuje zmenu polomeru tunelu v závislosti od prietoku. Funkcia f je monotónna rastúca funkcia, ktorá spĺňa podmienku $f(0) = 0$. Druhý člen reprezentuje konštantnú rýchlosť zmeny polomeru tunelu. Kde α je kladná konštanta, ktorá zabezpeči to,že tunely sa pri absencií prietoku vytratia [6].

Tunely sú medzi sebou vzájomné prepojené, to znamená, že ak sa zmení prietok v jednej hrane, ovplyvní to prietoky v ostatných tuneloch. Celkové množstvo toku v sieti musí byť zachované. To znamená, že keď na začiatku vojde do siete tok I_0 v uzle $N1$, tak v uzle $N2$ musíme dostať tok $-I_0$.

Z diferenciálnej rovnice (5) dostaneme nové $D_{i,j}$. Pomocou nových $D_{i,j}$ vieme

využitím rovníc (1), (2), (3) dostať nové tlaky p_i v jednotlivých vrcholoch a nové prietoky $Q_{i,j}$ v jednotlivých hranách. Opakovaním tohto postupu sa nám menia jednotlivé prietoky v tuneloch a po konečnom počte iterácií sa uzatvoria tunely, ktoré vedú do slepej uličky, alebo nepredstavujú najkratšiu cestu k východu z bludiska. Za optimálnu cestu (najkratšiu cestu) budeme pokladať takú, cez ktorú bude prechádzať 100 percent daného toku. To znamená v našom prípade I_0.

2.4 Algoritmus

Nech máme daný graf G, konečný počet vrcholov N_i kde N_1 a N_2 sú začiatočný a koncový vrchol a konečný počet hrán $M_{i,j}$. Nech $L_{i,j}$ sú dané dĺžky hrán medzi uzlom i a j. Nech je daný základný tlak $P_k = 0$ v ľubovoľnom vrchole $K \in G$. Nech sú ďalej dané konštanty I_0 ako začiatočný tok, r ako polomer tunelu, ktorý reprezentuje hranu (na začiatku má každá hrana rovnaký polomer) a konštanta ξ. Algoritmus sa dá zapísať do niekoľkých krokov nasledovne:

1. V prvom kroku dostaneme tlaky p_i spojením (1),(2) a (3) do sústavy rovníc (4).

2. Dosadením tlakov do rovnice (1) dostaneme prietoky $Q_{i,j}$ medzi jednotlivými vrcholmi. Ak sa dá jednoznačne určiť, že I_0 preteká jedinou cestou, tak končíme. Inak krok 3.

3. Pomocou rovnice (6) dostaneme nové $D_{i,j}$, ktoré použijeme v rovnici (1) namiesto starých $D_{i,j}$. Nasleduje krok 1.

2.5 Analógia s elektrickým obvodom

Matematický model physarum solver sa dá prirovnať k elektrickému obvodu [6]. Pričom každú hranu v našej sieti môžme považovať za rezistor s odporom ktorý je úmerný $L_{i,j}$ a $r_{i,j}^{-4}$. Telo vápenatky mnohohlavej reprezentuje potom sieť rezistorov.

Toky $Q_{i,j}$, ktoré prúdia medzi jednotlivými vrcholmi môžme prirovnať k elektrickému prúdu, ktorý preteká medzi rezistormi. A vrcholy $N1$ a $N2$, ktorými do siete preniká vstupný a výstupný tok, môžme prirovnať k vstupnému a výstupnému elektrickému prúdu. Tlaky $p_{i,j}$ potom môžme prirovnať k elektrickému napätiu v jednotlivých častiach elektrického obvodu.

Ak potom elektrický prúd, ktorý prechádza rezistorom je dostatočne veľký, odpor rezistora začne klesať a následne sa zvyšuje elektrický prúd prechádzajúci rezistorom. Naopak, ak je prúd malý, tak sa odpor rezistora zvyšuje a nakoniec sa odpor rezistora približuje k nekonečnu, čo môžme prirovnať ku uzatváraniu sa ciest, ktoré nevedú k najkratšej ceste v grafe.

2.6 Zložitosť algoritmu

Krok 1 si vyžaduje $O(n^2)$ operácií a robí sa n krát, čo si vyžaduje celkovo $O(n^3)$ operácií, krok 3 sa robí rovnako n krát čo dáva teda zložitosť celého algoritmu $O(n^3)$.

3 Porovnanie algoritmov

V nasledujúcej praktickej časti budeme porovnávať efektivitu algoritmu, ktorý je navrhnutý na základe biologického sa správania vápenatky s Dijkstrovým algoritmom.

3.1 Cieľ práce

V prvom rade budeme sledovať časovú náročnosť algoritmov. A to na jednotlivých príkladoch, pričom bude dôležité porovnanie obidvoch algoritmov pri malom počte vrcholov a následne budeme zvyšovať počet vrcholov v grafe, aby sme si vedeli urobiť celkový obraz o efektivite algoritmu. Predpoklad je, že pri grafoch s malým počtom vrcholov bude časová náročnosť porovnateľná. To znamená, že by bol v reálnych problémoch algoritmus navrhnutý podľa vápenatky schopný nahradiť Dijsktrov algoritmus. Avšak pri postupnom zvyšovaní počtu vrcholov v grafe bude podľa nášho predpokladu časovo efektívnejší Dijkstrov algoritmus.

V druhom rade budeme postupným menením funkcie $f(|Q_{i,j}|)$ v diferenciálnej rovnici:

$$\frac{d}{dt}D_{i,j} = f(|Q_{i,j}|) - \alpha D_{i,j},$$

pozorovať urýchľovanie algoritmu. Našou snahou bude nájsť takú funkciu $f(|Q_{i,j}|)$, aby bol algoritmus založený na vápenatke schopný konkurovať dijkstrovmu algoritmu aj v grafoch s väčším počtom vrcholov.

Ako funkciu f v diferenciálnej rovnici budeme na začiatku voliť

$$f(|Q|) = |Q|^{\mu}(\mu = 1)$$

Pre model s dvoma zdrojmi potravy je najvhodnejšia práve táto voľba funkcie f [6]. Neskôr budeme pozorovať rôzne obmeny funkcie f a následné spomaľovanie alebo urýchľovanie algoritmu, ktorý je založený na matematickom modeli physarum solver.

Pokúsime sa tiež modifikovať algoritmus, aby sme dosiahli urýchlenie algoritmu podľa vápenatky, tak že zmeníme spôsob riešenia systému lineárnych rovníc $Ap = b$. Pretože nie je potrebné presné riešenie daného systému ale postačuje približné riešenie. A to vzhľadom na to že je súčasťou celého cyklu numerických riešení. Na testovanie algoritmov budeme používať matlab.

3.2 Úprava algoritmu vápenatky do matlabu

V obidvoch algoritmoch sa vychádza z matice susednosti (adjacency matrix) vrcholov grafu. Graf si vieme pretransformovať do tejto matice nasledovne. V matici susednoti typu $N \times N$ prvok matice i, j obsahuje 1, ak hrana $(i, j) \in E$, inak prvok i, j obsahuje 0. My budeme používať ohodnotenú maticu susednosti. To znamená že číslo 1 nahradia $L_{i.j}$, ktoré budú predstavovať dĺžku hrany medzi vrcholmi i a j. Matica je v neorientovanom grafe symetrická.

Aby sme vedeli použiť sústavu rovníc (4) v matlabe.

$$\sum_i \frac{D_{i,j}}{L_{i,j}}(p_i - p_j) = \begin{cases} -I_0 & \text{pre } j = 1; \\ I_0 & \text{pre } j = 2; \\ 0 & \text{inak ;} \end{cases}$$

bude nutné si ju pretransformovať do vhodnej podoby. Dĺžky $L_{i,j}$ máme zachytené v ohodnotenej matici susednosti, ktorá je typu $N \times N$. Tak isto máme dané parametre $D_{i,j}$ ako $D_{i,j} = \frac{\pi r^4}{8\xi}$. Na ľavej strane ostali ešte tlaky p_i, ktoré sú v tomto prípade hľadanou neznámou. Sústavu rovníc pretransformujeme do podoby:

$$Ap = b$$

Pričom p bude vektor $p = (p_1, p_2, ..., p_n)$ jednotlivých tlakov. b bude vektor $b = (I_0, -I_0, 0, 0...0)$. Matica A bude typu $N \times N$ a jej prvky dostaneme nasledovne. Na diagonále matice A:

$$\sum_j \frac{D_{i,j}}{L_{i,j}}$$

Ostatné prvky matice A budú 0 ak matica susednosti má na tom istom prvku 0, inak:

$$-\frac{D_{i,j}}{L_{i,j}}$$

Aby sme si zjednodušili zápis, tak prvky matice A, ktoré nie sú na diagonálach nám bude reprezentovať premenná y. Konečná podoba sústavy rovníc bude teda:

$$\begin{pmatrix} \sum_j \frac{D_{1,j}}{L_{1,j}} & y & y & y & y \\ y & . & y & y & y \\ y & y & \sum_j \frac{D_{i,j}}{L_{i,j}} & y & y \\ y & y & y & . & y \\ y & y & y & y & \sum_j \frac{D_{n,j}}{L_{n,j}} \end{pmatrix} \times \begin{pmatrix} p_1 \\ p_2 \\ . \\ . \\ p_n \end{pmatrix} = \begin{pmatrix} I_0 \\ -I_0 \\ 0 \\ . \\ 0 \end{pmatrix}$$

Aby sme dostali hľadané tlaky p_i, potrebujeme teda vypočítať sústavu rovníc $Ap = b$ čo dosiahneme v matlabe nasledovne:

$$p = A/b$$

Tu nastáva ale problém singularity matice. To znamená že determinant matice A je nulový $Det(A) = 0$. Aby sme sa vyhli problému zo singularitou matice, musíme si stanoviť základný tlak $p_k = 0$, to nám zmení sústavu rovníc na:

$$\begin{pmatrix} \sum_j \frac{D_{1,j}}{L_{1,j}} & y & y & y & y \\ y & . & y & y & y \\ y & y & \sum_j \frac{D_{i,j}}{L_{i,j}} & y & y \\ 0 & 0 & 0 & 1 & 0 \\ y & y & y & y & \sum_j \frac{D_{n,j}}{L_{n,j}} \end{pmatrix} \times \begin{pmatrix} p_1 \\ p_2 \\ . \\ p_k \\ p_n \end{pmatrix} = \begin{pmatrix} I_0 \\ -I_0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Ďalšou možnosťou ako sa vyhnúť singularite je že jeden riadok matice zameníme, a to tak že súčet všetkých tlakov $p_i = 1$. A to nasledovne:

$$\begin{pmatrix} \sum_j \frac{D_{1,j}}{L_{1,j}} & y & y & y & y \\ y & . & y & y & y \\ y & y & \sum_j \frac{D_{i,j}}{L_{i,j}} & y & y \\ 1 & 1 & 1 & 1 & 1 \\ y & y & y & y & \sum_j \frac{D_{n,j}}{L_{n,j}} \end{pmatrix} \times \begin{pmatrix} p_1 \\ p_2 \\ . \\ p_k \\ p_n \end{pmatrix} = \begin{pmatrix} I_0 \\ -I_0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \tag{6}$$

Teraz sme dostali tlaky p_i vo vrcholoch. Prietoky $Q_{i,j}$ medzi jednotlivými vrcholmi dostaneme spätným dosadením do rovnice:

$$Q_{i,j} = \frac{D_{i,j}}{L_{i,j}}(p_i - p_j)$$

Dostali sme základne prietoky, ktoré by vďaka tlakom vo vrcholoch už s časti mali preukazovať že ktoré cesty medzi vrcholmi $N1$ a $N2$ to znamená začiatočným a koncovým vrcholom budú súperiť o to, ktorý vrchol bude patriť do najkratšej optimálnej cesty. To uvidíme v nasledujúcich príkladoch. Tak isto budeme môcť z prvých prietokov predpovedať ktoré cesty vedú do slepých uličiek.

Z diferenciálnej rovnice (5) dostaneme nové $D_{i,j}$, ktoré reprezentujú zmeny polomerov jednotlivých tunelov, ktorými prechádzajú prietoky. Aby sme vedeli použiť túto diferenciálnu rovnicu, musíme ju upraviť do formy, ktorá je vhodná do matlabu a to nasledovne:

$$\frac{d}{dt}D_{i,j} = f(|Q_{i,j}|) - \alpha D_{i,j}$$

$$\frac{D_{i,j}^{nov} - D_{i,j}^{star}}{\tau} = f(Q_{i,j}^{nov}) - aD_{i,j}^{nov}$$

$$(1 + a\tau)D_{i,j}^{nov} = D_{i,j}^{star} + \tau f(Q_{i,j}^{nov})$$

Kde τ predstavuje časový krok. Úpravami sa nakoniec dostaneme ku vzťahu, ktorý

nám dáva nove parametre $D_{i,j}$:

$$D_{i,j}^{nov} = \frac{D_{i,j}^{star} + \tau f(Q_{i,j})}{(1 + a\tau)} \tag{7}$$

Kde τ predstavuje časový krok. Pomocou nových $D_{i,j}$ vieme využitím rovníc (1), (2), (3) dostať nové tlaky p_i v jednotlivých vrcholoch a nové prietoky $Q_{i,j}$ v jednotlivých hranách. Tento postup opakujeme dovtedy, pokiaľ sa nám niektoré cesty neuzavrú a z prietokov $Q_{i,j}$ vieme vyčítať najkratšiu cestu z N1 do N2.

3.2.1 Gauss-Seidelova relaxačná SOR metóda

V tejto časti sa zameriame na problém numerického riešenia sústavy lineárnych rovníc $Ap = b$. Pri výpočte nie je potrebné presné riešenie sústavy lineárnych rovníc, ale postačuje aj približné riešenie a to z dôvodu toho, že sústava rovníc je súčasťou celého cyklu numerických riešení. Populárna metóda na riešenie lineárnych sústav rovníc je Gauss-Seidelova relaxačná SOR metóda (Successive Over Relaxation). Jej podstata spočíva v hľadaní riešenia sústavy lineárnych rovníc pomocou iteratívne skonštruovanej postupnosti približných riešení [5].

Maticu A možme rozpísať ako súčet jej poddiagonálnej, diagonálnej a naddiagonálnej časti:

$$A = L + D + U \tag{8}$$

kde

$$\begin{aligned} L_{i,j} &= A_{i,j} \quad \text{pre } j < i \quad \text{pre } L_{i,j} = 0, \\ D_{i,j} &= A_{i,j} \quad \text{pre } j = i \quad \text{pre } D_{i,j} = 0, \\ U_{i,j} &= A_{i,j} \quad \text{pre } j > i \quad \text{pre } U_{i,j} = 0, \end{aligned}$$

O diagonálnej matici sa predpokladá, že jej prvky sú nenulové. Nech $\omega > 0$ je zvolený parameter, potom riešenie úlohy $Ap = b$ je rovnaké ako riešenie úlohy:

$$Dp = Dp + \omega(b - Ap)$$

Využízím vzťahu (8) a toho že matica $D + \omega L$ je invertovateľná [5] dostaneme po krátkych úpravách že p je riešením lineárnej úlohy:

$$p = T_\omega p + c_\omega$$

kde

$$T_\omega = (D + \omega L)^{-1}((1 - \omega)Dp - \omega U)$$
$$c_\omega = \omega(D + \omega L)^{-1}b$$

Pomocou operátora T_ω definujeme rekurentnú postupnosť aproximatívnych riešení úlohy $Ap = b$:

$$p^{k+1} = T_\omega p^k + c_\omega \quad \text{pre } k = 1, 2, ...$$

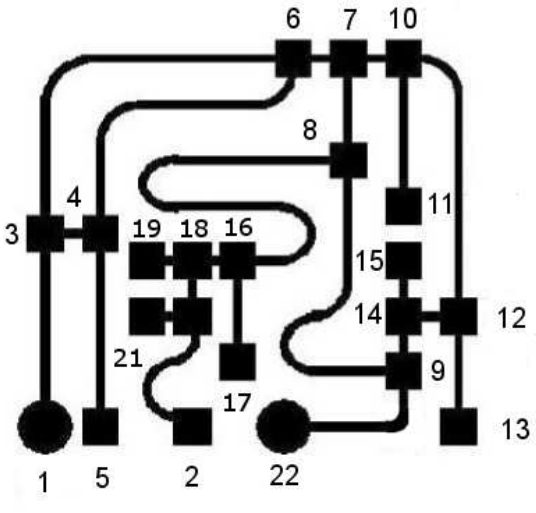
Ako počiatočnú podmienku je v našom prípade vhodné voliť začiatočné tlaky p_i^0, ktoré dostaneme v prvom kroku. A vždy v každom ďalšom kroku pri hľadaní nových tlakov p_i, použiť ako počiatočnú podmienku tie predchádzajúce.

Algoritmus vápenatky sme modifikovali teraz tak, že sme nahradili pôvodnú metódu na vyrátanie lineárnej úlohy $Ap = b$, ktorou sme dostali presné riešenie, Gauss-Seidelovou relaxačnou SOR metódou, ktorou dostaneme iba približné riešenie danej úlohy, avšak pre naše potreby to bude postačujúce.

Predpoklad je že modifikovaný algoritmus bude časovo efektívnejší ako pôvodný algoritmus. V príkladoch sme testovali obidve metódy na riešenie lineárnej sústavy rovníc $Ap = b$.

3.3 Problém bludiska

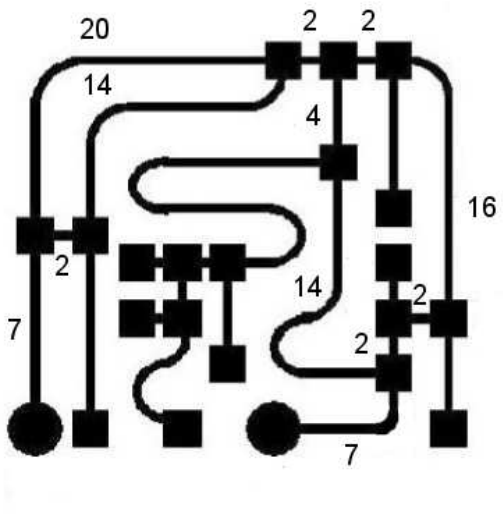
Prvý príklad, ktorý nám reprezentuje graf ktorý bol prevzatý z článku [6] simuluje problém bludiska (obr.5). V prvom rade si označíme vrcholy grafu, ktoré tvoria jednotlivé križovatky medzi hranami.



Obr. 5: označenie vrcholov

3.3.1 Algoritmus vápenatky

Vrcholy zo zdrojmi živín sú 1 a 22. Budeme hľadať najkratšiu cestu medzi vrcholom 1 a 22. Spolu máme 22 vrcholov, to znamená ohodnotenú maticu susednosti typu 22×22, ktorá je symetrická.



Obr. 6: dlžky medzi vrcholmi

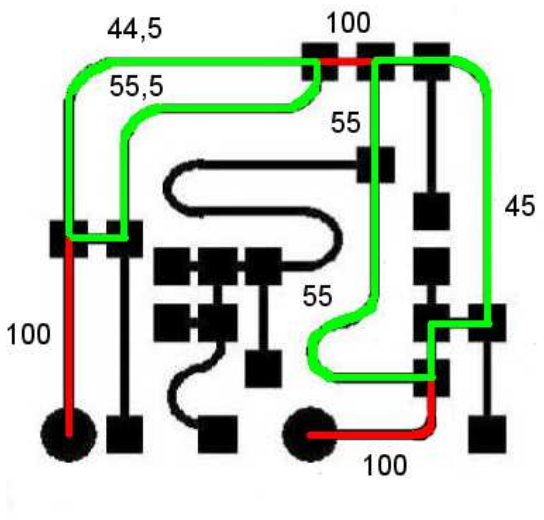
Na obr.6 sú znázornené vzdialenosti medzi niektorými vrcholmi. Ostatne vzdialenosti (ktoré v matici sú) neuvádzame preto ,aby zostal obrázok prehľadný. Nech sú dané parametre:

- Začiatočný tok $I_0 = 100$,
- Časový krok $\tau = 0.01$,
- Začiatočný rovnaký polomer všetkých hrán $r = 1$,
- Parameter vodivosti $\xi = 1$

Z ohodnotenej matice susednosti, kde sú dané dĺžky $L_{i,j}$ si spravíme pomocnú maticu A, podľa predošlej kapitoly. Pomocou sústavy rovníc (6) získame počiatočné tlaky:

$p_1 = 4903.4$, $p_2 - 212.16$ $p_3 = 3120.9$, $p_4 = 2838$, $p_5 = 2838$, $p_6 = 857.36$ $p_7 = 348.06$, $p_8 = -212.16$, $p_9 = -2173$, $p_{10} = 118.88$, $p_{11} = 118.88$, $p_{12} = -1714.6$, $p_{13} = -1714.6$, $p_{14} = -1943.8$, $p_{15} = -1943.8$, $p_{16} = -212.16$, $p_{17} = -212.16$, $p_{18} = -212.16$, $p_{19} = -212.16$, $p_{20} = -212.16$, $p_{21} = -212.16$, $p_{22} = -3955.5$.

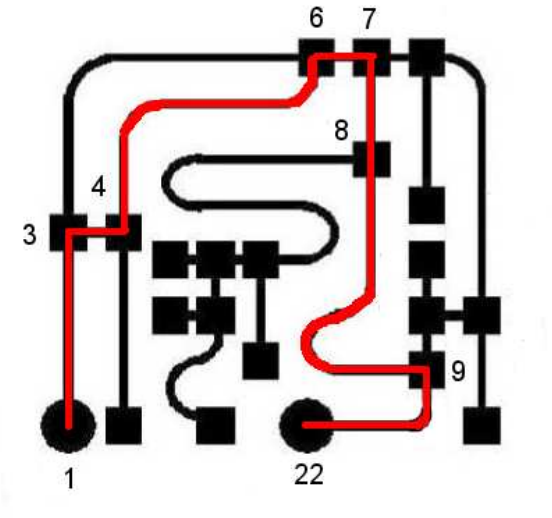
Zo získaných tlakov ešte nemožno určiť prakticky nič. Teraz spätným dosadením do rovnice (1) získame prietoky v jednotlivých hranách. Kvôli prehľadnosti sú v obr.7 zaznačené iba niektoré prietoky.



Obr. 7: problémové hrany sú znázornené zelenou farbou

Už pri prvých prietokoch vidieť, ktoré hrany vedú do slepej ulice a preto nie sú prietoky pri nich uvedené. Sú to cesty medzi vrcholmi $4-5$, $8-2$, $10-11$, $14-15$, $12-13$, prietoky tu boli rádovo 10^{-14}, čo je prakticky 0. Zaujímavé ale sú cesty medzi vrcholmi $3-6$, kde sa ponúka priama cesta $3-6$, alebo cesta cez vrchol 4. To znamená cesta cez vrcholy $3, 4, 6$. V tomto prípade vidíme že cez kratšiu cestu prechádza $55, 6$ to znamená cesta cez vrcholy $3, 4, 6$, ako cez dlhšiu cestu, cez ktorú prechádza zvyšok toku. Takýto istý próblem sa vyskytuje pri cestách medzi vrcholmi 7 a 9.

Už z týchto prvých výsledkov, by sme vedeli určiť najkratšiu cestu, Podmienka ale je aby optimálnou (najkratšou) cestou prechádzalo 100 percent daného toku. Pomocou rovnice (7) získame nové $D_{i,j}$. S ktorými opakujeme postup. Za funkciu $f(|Q|)$ budeme na začiatku voliť $|Q|$.



Obr. 8: optimálna cesta je znazornená červenou.

Po 211 iteráciách môžme na obr.8 pozorovať už jasný výsledok. Problematickými cestami prechádza už taktiež nulový tok. A najkratšiu optimálnu cestu $1 - 22$ tvoria vrcholy $1, 3, 4, 6, 7, 8, 9, 22$. Najkratšia cesta je dĺžky 50. Čas potrebný v matlabe na uskutočnenie týchto 211 iterácií bol relatívne malý, a to $0,371$ sekundy.

Teraz zmenou funkcie $f(|Q|)$ sa pokúsime urýchliť algoritmus:

- $f(|Q|) = |Q|^2$, 20 iterácií, čas trvania $0,218$ s.
- $f(|Q|) = |Q|^3$, 10 iterácií, čas trvania $0,016$ s.
- $f(|Q|) = |Q|^4$, 6 iterácií, čas trvania $0,016$ s.

Ako vidíme, počet iterácií sa dá značne vylepšiť vhodným volením funkcie $f(|Q|)$ v diferenciálnej rovnici. Aj keď čas je relatívne krátky vo všetkých prípadoch. Predpokladá sa, že v grafoch s väčším počtom vrcholov bude čas hrať významnejšiu úlohu. V modifikovanom algoritme vápenatky, kde bola použitá SOR metóda sme nezaznamenali žiadne zrýchlenie, algoritmus pri rovnakých parametroch prebehol v krátkom

čase $0,328$ sekundy.

3.3.2 Dijkstrov algorimus

V rovnakom príklade nájdeme riešenie pomocou Dijkstrovho algoritmu. Ako vstup použijeme znovu ohodnotenú maticu susednosti.

Na začiatku si vytvoríme 2 množiny a to množinu S a S_1. Pri čom množina S bude obsahovať vrcholy, ktoré už majú permanentné označenie a množina S_1 bude obsahovať vrcholy, ktoré majú zatiaľ dočasné označenie. V množine S je iba začiatočný vrchol 1 a značíme pri ňom $(0,0)$. V množine S_1 sa nachádzajú všetky ostatne vrcholy a majú označenie $(1,\infty)$. Prvé číslo znamená predchádzajúci vrchol a druhé číslo znamená, najkratšiu dĺžku, akou sa do tohto vrcholu dalo dostať. Na začiatku majú všetky vrcholy (okrem prvého) predchodcu začiatočný vrchol a dĺžku ∞.

Na obr.9 môžme vidieť nultý krok algoritmu a zároveň aj trvalé značky vrcholov. To znamená algoritmus je u konca, našla sa najkratšia cesta. Spätným odčítaním z tabuľky dokážeme zistiť vrcholy, cez ktoré vedie hľadaná optimálna cesta.

A to vrcholy $22,9,8,7,6,4,3,1$. Rovnako je na obr.9 aj dĺžka najkratšej cesty a to 50. Čas potrebný na zbehnutie algoritmu bol zanedbateľný $0,015$ s.

V probléme bludisko môžme pozorovať, že algoritmus vápenatky zaostáva v porovnaní s Dijsktrovým algoritmom pri volení funkcií $f(|Q|) = |Q|$ iba niekoľko desatín sekundy. A pri vhodnom volení funkcie $f(|Q|) = |Q|^2$, $f(|Q|) = |Q|^3$, $f(|Q|) = |Q|^4$ postupne dobieha Dijsktrov algorimus. Pri poslednej volenej funkcií je časová náročnosť obidvoch algoritmov rovnaká. To znamená z pohľadu efektívnosti je jedno ktorý algoritmus použijeme.

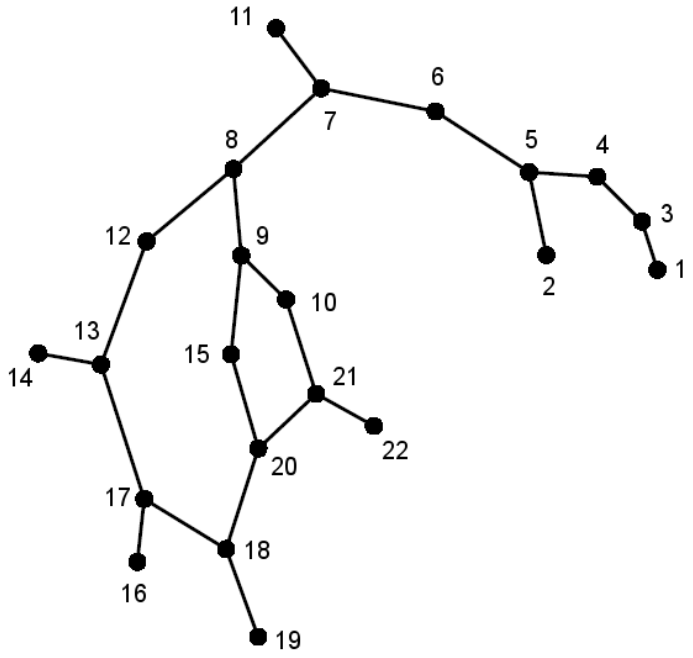
1	0	0	1	0	0
2	1	Inf	2	20	58
3	1	Inf	3	1	7
4	1	Inf	4	3	9
5	1	Inf	5	4	16
6	1	Inf	6	4	23
7	1	Inf	7	6	25
8	1	Inf	8	7	29
9	1	Inf	9	8	43
10	1	Inf	10	7	27
11	1	Inf	11	10	33
12	1	Inf	12	10	43
13	1	Inf	13	12	47
14	1	Inf	14	12	45
15	1	Inf	15	14	47
16	1	Inf	16	8	47
17	1	Inf	17	16	51
18	1	Inf	18	16	49
19	1	Inf	19	18	50
20	1	Inf	20	18	51
21	1	Inf	21	20	52
22	1	Inf	22	9	50

Obr. 9: inicializácia a výsledky po zbehnutí algoritmu

3.4 Tlakové pasmo - Koliba

V prvom reálnom probléme sa budeme zaoberať tlakovým vodárenským pásmom Bratislava - Koliba. Je to sieť vodovodných potrubí. Na rozdiel od predošlého príkladu, je tento príklad s reálnymi skutočnými dátami. Keďže je to opäť príklad s malým počtom vrcholov konkrétne 22. Budeme hlavne skúmať, či dajú obidva algoritmy exaktne rovnaké riešenie. Pretože časovú náročnosť sme už testovali na predošlom prípade, predpokladá sa rovnaká efektívnosť obidvoch algoritmov. Tlakové pásmo -

Koliba môžme vidieť v prilohe č.2 mapa 1. Zjednodušili sme ho v obrázku 10 a 11:



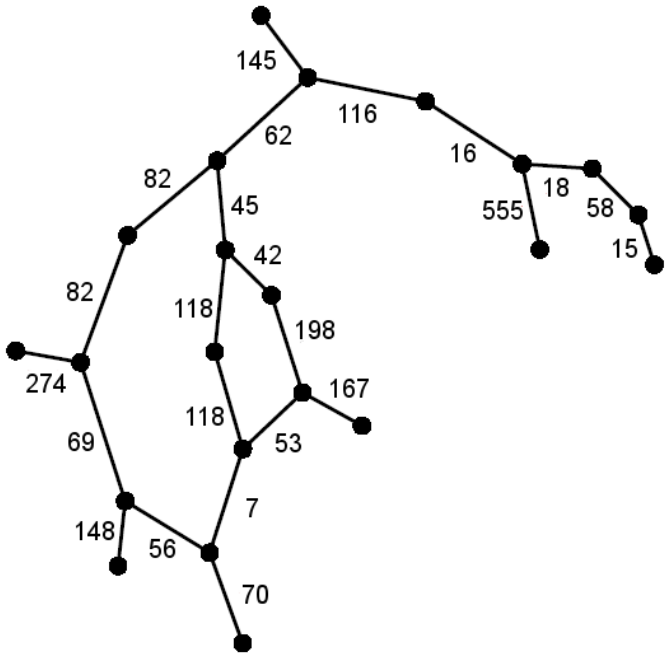
Obr. 10: očíslované vrcholy

Budeme hľadať najkratšiu cestu medzi vrcholmi $1 - 16$.

3.4.1 Algoritmus vápenatky

Použijeme rovnaké parametre ako v prvom príklade:

- Začiatočný tok $I_0 = 100$,
- Časový krok $\tau = 0.01$,
- Začiatočný rovnaký polomer všetkých hrán $r = 1$,
- Parameter vodivosti $\xi = 1$

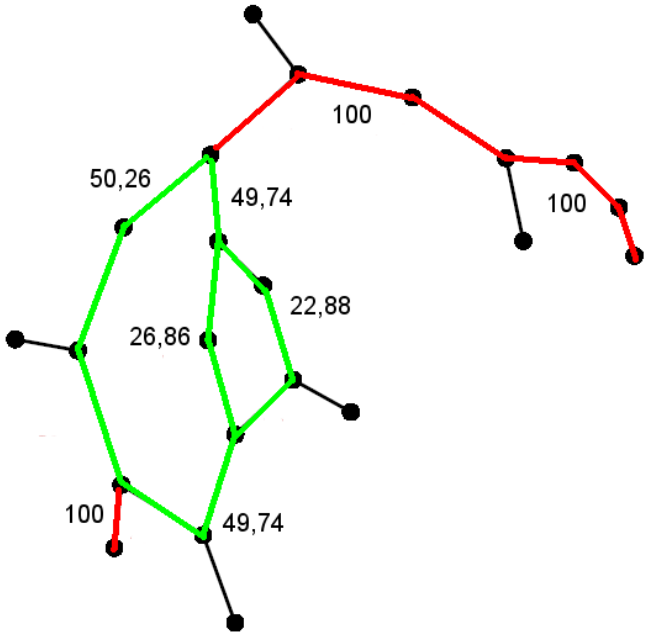


Obr. 11: dĺžky medzi jednotlivými vrcholmi

Zopakovaním postupu z prvého príkladu dostaneme tlaky p_i v jednotlivých uzloch a následne prvé prietoky $Q_{i,j}$ v jednotlivých vodovodných potrubiach, ktoré vyzerajú nasledovne:

Na obr.12 vidíme stav riešenia po prvej iterácií. Cez hrany, ktoré vedú do slepej ulice už po prvej iterácií prechádza nulový tok. Sú to cesty medzi vrcholmi $2-5$, $7-11$, $21-22$, $13-14$, $18-19$. Niektorými vodovodnými potrubiami prechádza už teraz 100 percent daného toku. Algoritmus musí teraz posúdiť problematické cesty, ktoré sú na obr.12 označené zelenou farbou. Sú to 3 možné cesty medzi vrcholmi $8-17$.

Na obr.13 vidíme konečný stav, ktorý nastal po 117 iteráciách. Algoritmus založený na vápenatke našiel najkratšiu cestu za $0,11$ sekundy. Je to cesta zložená z vrcholov $1, 3, 4, 5, 6, 7, 8, 12, 13, 17, 16$. Najkratšia cesta je dĺžky 666.



Obr. 12: prvý krok, inicializácia

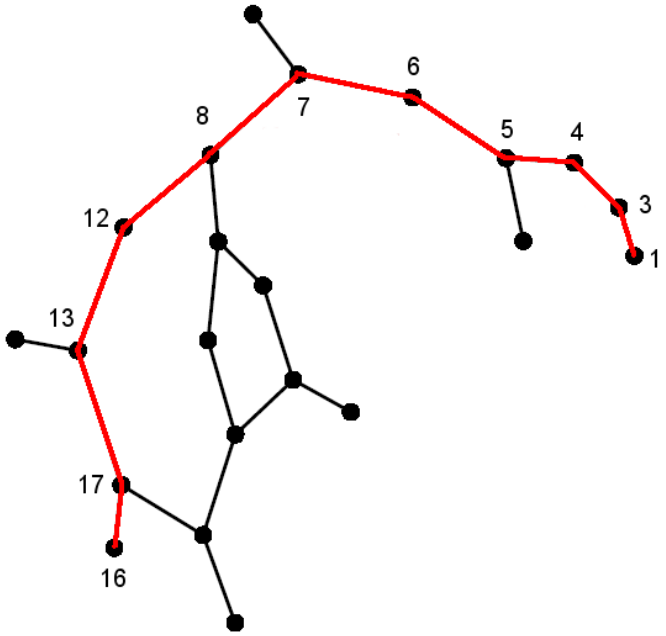
Teraz zmenou funkcie $f(|Q|)$ sa pokúsime urýchliť algoritmus:

- $f(|Q|) = |Q|^2$, 12 iterácií, čas trvania $0,047$ s.
- $f(|Q|) = |Q|^3$, 10 iterácií, čas trvania $0,016$ s.
- $f(|Q|) = |Q|^4$, 5 iterácií, čas trvania $0,015$ s.

V modifikovanom algoritme vápenatky, kde bola použitá SOR metóda sme nezaznamenali žiadne zrýchlenie, algoritmus pri rovnakých parametroch prebehol v krátkom čase $0,10$ sekundy.

3.4.2 Dijkstrov algoritmus

Dijkstrov algoritmus funguje na tomto reálnom probléme podobne ako v probléme bludisko. Preto len stručne zhodnotíme výsledky, ktoré sme dostali z Dijkstrovho



Obr. 13: po 117 iteráciách

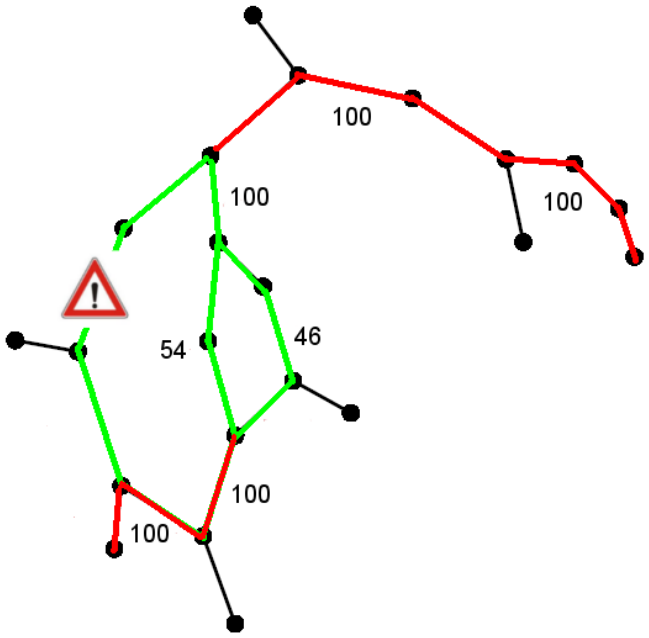
algoritmu. Po zbehnutí sme odčítali z tabuľky, kde sú už konečné označenia vrcholov nasledovnú najkratšiu cestu: 16, 17, 13, 12, 8, 7, 6, 5, 4, 3, 1. Táto cesta sa zhoduje s optimálnou najkratšou cestou podľa algoritmu vápenatky. Tak isto dĺžka 666 na tejto ceste musí byť rovnaká. Obidva algoritmy dospeli k rovnakému riešeniu.

Časovo Dijkstrov algoritmus riešil úlohu $0,015$ s, čo je porovnateľné z algoritmom vápenatky, keď sa algoritmus urýchli vhodným volením funkcie $f(|Q|)$ v diferenciálnej rovnici.

3.5 Simulácia poruchy vodovodného potrubia

Predpokladajme že sa v reálnom probléme tlakového vodárenského pásma Bratislava - Koliba stane porucha a to že sa pretrhne potrubie na ceste medzi vrcholmi $12 - 13$ (obr.14). Teraz overíme či bude algoritmus podľa vápenatky schopný sa vysporiadať

so vzniknutou poruchou a nájde alternatívnu najkratšiu cestu medzi vrcholmi $1 - 16$.



Obr. 14: pretrhnuté potrubie $12 - 13$

Problematické ostávajú cesty medzi vrcholmi $9 - 20$, kde sa algoritmus bude rozhodovať medzi $9 - 15 - 20$ a $9 - 10 - 21 - 20$. Po 300 iteráciách už dostávame požadovaný výsledok. Najkratšia cesta je $1, 3, 4, 5, 6, 7, 8, 9, 15, 20, 18, 17, 16$ dosiahnutá v krátkom čase $0,188$ sekundy pri volení rovnakých parametrov ako v predchádzajúcich príkladoch. Najkratšia cesta je dĺžky 777.

Dijkstrov algoritmus dáva pri simulácii poruchy medzi vrcholmi $12 - 13$ a hľadaní najkratšej cesty medzi vrcholmi $1 - 16$ po spätnom odčítaní z výstupu totožnú najkratšiu cestu $1, 3, 4, 5, 6, 7, 8, 9, 15, 20, 18, 17, 16$ s rovnakou dĺžkou 777 avšak v kratšom čase $0,015$ sekundy.

3.6 Prvé tlakové pásmo Bratislavy

Ďalší reálny problém na ktorom budeme testovať obidva algoritmy je tlakové vodárenské pásmo celej Bratislavy. To znamená rozvody vodovodných potrubí na celom území Bratislavy. Vyberieme si niektoré 2 vrcholy a budeme hľadať najkratšiu cestu medzi nimi. Ako vstup nám slúži matica susednoti vrcholov 237×237, pretože sa v sieti nachádza 237 jednotlivých uzlov. A medzi nimi je 353 vodovodných potrubí.

3.6.1 Dijsktrov algoritmus

Zvolíme 2 okrajové vrcholy siete a to ČS Petržalka - uzol 333 (príloha č.2, mapa č. 8 a 9) a vodojem Vtáčnik - uzol 10 (príloha č.2, mapa č.5). Pomocou Dijsktrovho algoritmu nájdeme najkratšiu cestu v grafe. Dijsktrov algoritmus našiel najkratšiu cestu dĺžky 7917, ktorá vedie cez uzly $333, 233, 243, 280, 267, 268, 12, 48, 31, 18, 290, 43, 22, 9, 10$. Algoritmus prebehol za krátky čas $0,234$ sekundy.

3.6.2 Algoritmus vápenatky

Použiieme rovnaké parametre ako v prvom príklade:

- Začiatočný tok $I_0 = 100$,
- Časový krok $\tau = 0.01$,
- Začiatočný rovnaký polomer všetkých hrán $r = 1$,
- Parameter vodivosti $\xi = 1$

Algoritmus založený na vápenatke našiel najkratšiu cestu v grafe dĺžky 7917, ktorá vedie cez uzly $333, 233, 243, 280, 267, 268, 12, 48, 31, 18, 290, 43, 22, 9, 10$. Obidva algoritmy dali rovnaký výsledok. Rôznym volením funkcie $f(|Q|)$ v diferenciálnej rovnici sa dá algoritmus vápenatky urýchliť, čo možno pozorovať aj v nasledujúcich meraniach.

Voľba funkcie $f(|Q|) = Q$:

- 25 iterácií, čas trvania $4,234$ s , 65 percent prietoku Q prechádza najkratšou cestou.
- 50 iterácií, čas trvania $8,5$ s , 85 percent prietoku Q prechádza najkratšou cestou.
- 100 iterácií, čas trvania $17,8$ s , 98 percent prietoku Q prechádza najkratšou cestou.

Voľba funkcie $f(|Q|) = |Q|^2$:

- 5 iterácií, čas trvania $0,641$ s , 70 percent prietoku Q prechádza najkratšou cestou.
- 10 iterácií, čas trvania $1,313$ s , 99 percent prietoku Q prechádza najkratšou cestou.
- 20 iterácií, čas trvania $3,3$ s , 99,9 percent prietoku Q prechádza najkratšou cestou.

Ako možno pozorovať z výsledkov, tak algoritmus vápenatky značne zaostáva za Dijsktrovím algoritmom pri volení funkcie $f(|Q|) = Q$. Vhodnou voľbou funkcie $f(|Q|) = |Q|^2$,alebo následne $f(|Q|) = |Q|^3$ kde bol algoritmus vápenatky schopný vyriešiť daný problém za krátky čas $1,02$ s ,kde po 10 iteráciách prechádzalo najkratšou cestou už $99,9$ daného prietoku $I_0 = 100$ sa dá čas potrebný na nájdenie najkratšej cesty v grafe značne skrátiť.

Ako jediný z volených parametrov dokáže zásadne ovplyvniť algoritmus iba časový krok τ. Čím je väčší časový krok, tým menej iterácií je potrebných na to, aby najkratšou cestou prechádzal požadovaný objem prietoku $I_0 = 100$.

V modifikovanom algoritme vápenatky, kde bola použitá SOR metóda sme podobne ako v príklade 1 a 2 nezaznamenali žiadne zrýchlenie:

- 50 iterácií, čas trvania $12,44$ s , 86 percent prietoku Q prechádza najkratšou cestou.
- 100 iterácií, čas trvania $23,6$ s , 98 percent prietoku Q prechádza najkratšou cestou.
- 150 iterácií, čas trvania 35 s , 99 percent prietoku Q prechádza najkratšou cestou.

Mohla by sa rozprúdiť diskusia o tom, koľko percent prietoku je vlastne potrebné na to, aby mohla byť hrana, cez ktorú tento prietok preteká posudzovaná ako hrana, ktorá patrí do najkratšej cesty v grafe. Už pri prvých prietokoch Q, ktoré dostaneme v jednotlivých uzloch dokáže pozorovateľ v jednoduchých príkladoch vyzistiť hľadanú najkratšiu cestu.

Predpokladajme že máme uzol z ktorého budú vychádzať 3 hrany, pričom jednou priteká 100 percent prítoku Q. Prietoky ktoré potom vychádzajú z uzlu do ďalších 2 hrán, nám dávajú jasnú predstavu o tom, ktorá cesta je kratšia. Hrana ktorou bude prechádzať nadpolovičná väčšina je cesta kratšia. Ak by sa prietok rozdelil presne na polovicu, znamenalo by to že obidvoma hranami sa dostaneme do cieľa o rovnakej dĺžke. Potom sa následne vyberieme po hrane ktorou prechádza nadpolovičná väčšina prietoku a opäť zopakujeme tento postup. Takto sa postupne dostaneme do cieľa, hrany po ktorých sme išli budú tvoriť najkratšiu cestu.

Avšak pri komplikovanejších grafoch, kde máme veľa uzlov aj hrán, by bolo náročné ak nie nemožné použiť tento postup, preto je jednoduchšie zvoliť si požadovaný prietok.

4 Záver

Dijkstrov algoritmus v porovnaní s algoritmom vápenatky je globálny algoritmus, to znamená, že dáva celkový obraz v každom kroku o všetkých vrcholoch a hranách. Ktorý vrchol má už trvalé označenie, ktorý vrchol má zatiaľ dočasné značenie. Cez ktoré hrany už prechádzal, a patria do najkratšej cesty, a ktoré sa ešte len budú posudzovať. Naopak algoritmus podľa vápenatky je lokálny. V každom vrchole algoritmus vie, že sa súčet prítoku a odtoku musí rovnať 0, ale nedáva to obraz o celej sieti vrcholov a hrán.

V jednoduchých príkladoch, ako je problém bludiska a tlakové pásmo Koliba, kde bolo relatívne málo vrcholov a hrán sa algoritmus po vhodných modifikáciách dokázal vyrovnať Dijsktrovmu algoritmu a mohol by ho nahradiť pri niektorých reálnych problémoch. A to vhodnou úpravou diferenciálnej rovnice. Na rozdiel od toho očakávané zrýchlenie použitím Gauss-Seidelovej SOR metódy sa nepotvrdilo.

Tak isto bol schopný algoritmus vápenatky reagovať na prekážku, ktorá sa vyskytla práve na najkratšej ceste z miesta A do miesta B a vedel nájsť alternatívnu cestu, ktorá sa zhodovala s tou, ktorú našiel Dijkstrov algoritmus.

V zložitejších príkladoch už algorimus založený na vápenatke ani po vhodných úpravách diferenciálnej rovnice sa nedokázal vyrovnať Dijkstrovmu algoritmu aj keď ho v značnej miere dobiehal. Rovnako modifikovanie Gauss-Seidelovou SOR metódou neviedlo k očakávaným výsledkom. Pri jednoduchých príkladoch, kde sa v grafoch nachádza málo vrcholov nebol problém zo vstupnou maticou susednosti vrcholov. Avšak keď vrcholov pribúdalo algoritmus založený na vápenatke často kolaboval pretože sa v matici nachádzalo mnoho núl. Týka sa to hlavne prvého kroku kde sa matica susednosti využíva na vypočítanie základných prietokov pre jednotlivé vrcholy. Naopak Dijkstrov algoritmus si s tým poradil, a to napriek to mu že pracoval s rovnakým vstupom.

Pre obrovské matice je algoritmus založený na vápenatke práve z tohto dôvodu nepoužiteľný. Dijkstrov algorimus je efektívnejší a hodí sa na reálne problémy veľkých rozmerov. Využiteľnosť algoritmu vápenatky by sa mohla nájsť práve v tom, že je založený na prietokoch, a že sa dá zvoliť požadovaný prietok. Tak isto vhodnou modifikáciou algoritmu tak aby hneď po vypočítaní prvých prietokov sa dala určiť najkratšia cesta v grafe bez toho aby bolo potrebné zvyšovať a znižovať prietoky v jednotlivých hranách.

Literatúra

[1] DIJKSTRA E.W (1959) *A Note on Two Problems in Connexion with Graphs*, Numerische Mathematik 1, 269 – 271

[2] AKAGAKI T.,KOBAYASHI R.,UEDA T.,NISHIUR Y.(2006) *Obtaining multiple separate food sources: behavioural intelligence in the Physarum plasmodium* Physica A: Statistical Mechanics and its Applications, volume 363, issue 1, 115-119

[3] PLESNÍK J. (1983) *Grafové Algoritmy*, Vydavateľstvo slovenskej akadémie vied, Bratislava

[4] SNIEDOVICH M.(2006) *Dijkstra's algorithm revisited: the dynamic programming connexion*, Control and Cybernetics, volume 35,

[5] ŠEVČOVIČ D., STEHLÍKOVÁ B., MIKULA K.(2009) *Analytické a numerické metódy oceňovania finančných derivátov*, Slovenská technická univerzita v Bratislave

[6] TERO A, YUMIKI K, KOBAYASHI R, SAIGUSA T, NAKAGAKI T (2008) *Flow-network adaptation in Physarum amoebae*, Theory in Biosciences, volume 127, 89-94

[7] TERO A, KOBAYASHI R., NAKAGAKI T. (2008) *Physarum solver: a biologically inspired method of road-network navigation*, Physica A: Statistical Mechanics and its Applications, volume 363, Issue 1, 115-119

[8] MILKOVÁ E. (2009) *Constructing Knowledge in Graph Theory and Combinatorial Optimization within the multimedia application 'Graphs'*,Transactions on Mathematics, volume 8, issue 8, 424-434

[9] ZHANG F.,QIU A., LI Q. *Improve on dijkstra shortest path algorithm for huge data* Chinese Academy of Surveying and Mapping, Beijing, China

Internet

http://www.me.utexas.edu/~jensen/network_02/topic_pages/Bhaskaran/page7.html

Príloha č.1

Program

Program je nastavený na príklad č.1 problém bludiska, na jeho pretransformovanie na ostatné príklady stačia jednoduché úpravy. Na spustenie je potrebný program matlab.

```
function [tempmin]=DijkstrovAlgoritmus
 predchodca=zeros(22,1);
vzdialenost=inf(22,1);vzdialenost(1)=0; vis=zeros(22,1); start=1;
koniec=22; vis(start)=1; cur=start; tic while cur~=koniec
           tempmin=inf;
           vis(cur)=1;
       for i=1:n
             if (T(cur,i)>0) & (vzdialenost(i)>vzdialenost(cur)+T(cur,i))
                   vzdialenost(i)=vzdialenost(cur) + T(cur,i);
                   predchodca(i)=cur;
             end
           end
for i=1:n
     if (vis(i)==0) &(vzdialenost(i)<tempmin)
               cur=i;
               tempmin=vzdialenost(i);
     end
 end
end toc t=toc;

% T je vstupna matica
```

```
n=22; m=22; r=1; I0=100; viscosity=1; for i=22:n b(i,1)=0; end
b(1,1)= I0; b(22,1)=-I0; a=1; c=2; Tau=0.01;
       for i=1:n
        for j=1:m
              if T(i,j)>0
                  D(i,j)=(pi*r^4)/(8*viscosity);
              end
        end
       end
tic [Q]=iteracie(k,D,T,A,b,p,Tau,c,Q); toc t=toc;

function [Q]=iteracie(k,D,T,A,b,p,Tau,c,Q)
z=0;
while z<10
    [A] = maticaA(D,T);
    [p] = SORmetoda(A, b, N,p);
   %[p] = LinarnaMetoda(A,b);
    [Q] = prietoky(p,D,T);
    [D] = DiferencialnaRovnica(Q,D,Tau,c,k);
    z=z+1;
end
function [A] = maticaA(D,T) for i=1:n
    for j=1:m
          if T(i,j)>0
              A(i,i)= A(i,i)+D(i,j)/T(i,j);
              A(i,j)=-D(i,j)/T(i,j);
          end
```

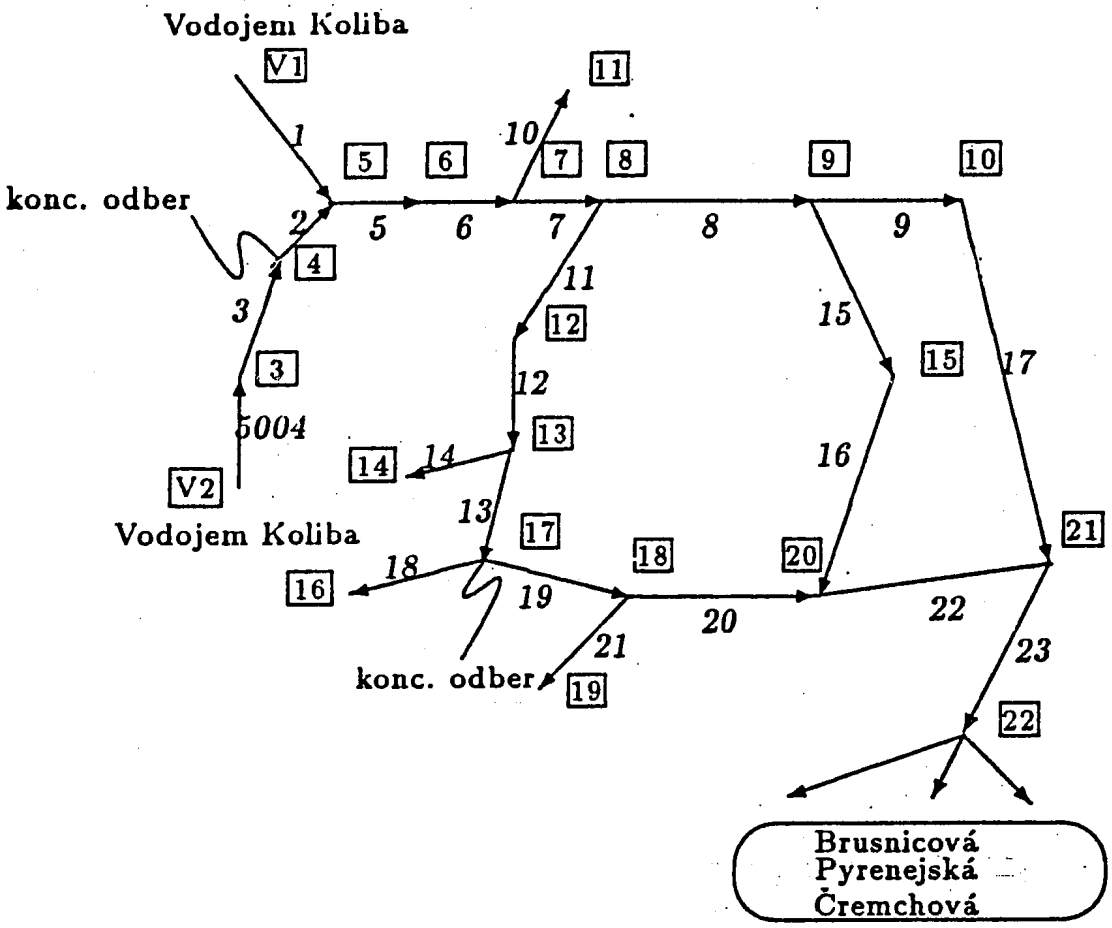
```
            A(2,j)=1;
    end
end

function [p]=SORmetoda(A,b,p)
Diag = diag(diag(A));
L=tril(-A,-1);
U = triu(-A,1);
Tomega = inv(Diag-oL)*U;
comega=inv(Diag-L)*b;
 k = 1;
while k <= 100
   p(:,1) = Tomega*p(:,1) + comega;
   k = k+1;
end

function [p] = LinearnaMetoda(A,b)
p = A\b

function [Q] = Prietoky(p,D,T) for i=1:n
   for j=1:m
        if T(i,j)>0
            Q(i,j)=(D(i,j)/T(i,j))*(p(i)-p(j));
        end
    end
end
```

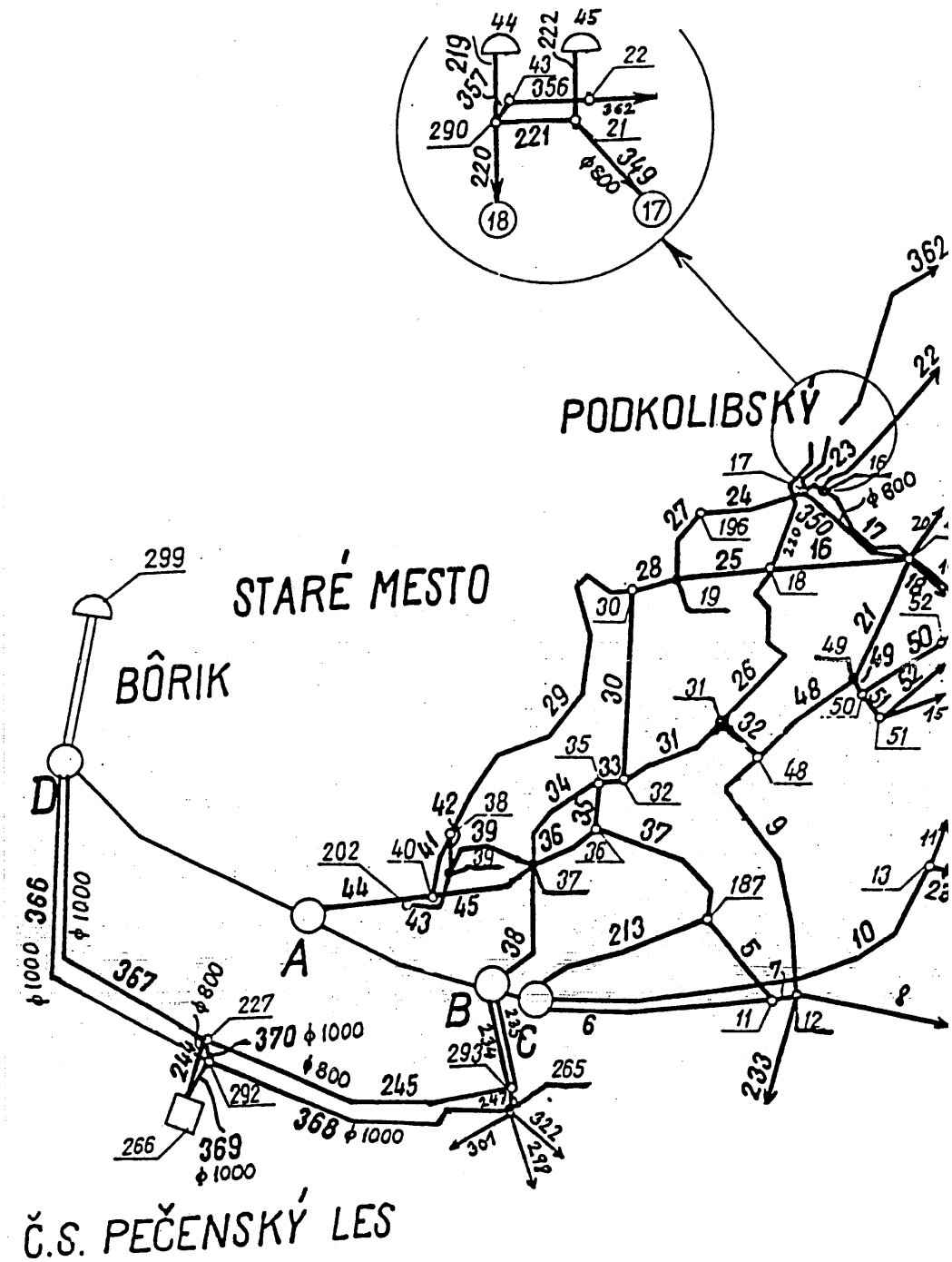
```
function [D] = DiferencialnaRovnica(Q,D,Tau,c,a)
    for i=1:n
        for j=1:m
              if D(i,j)>0
                     D(i,j)= (D(i,j)+Tau*((abs(Q(i,j))^c)))/(1+a*Tau);
                   (i,j)= (D(i,j)+Tau*abs(Q(i,j))/(1+abs(Q(i,j))))/(1+a*Tau);
              end
        end
    end
```

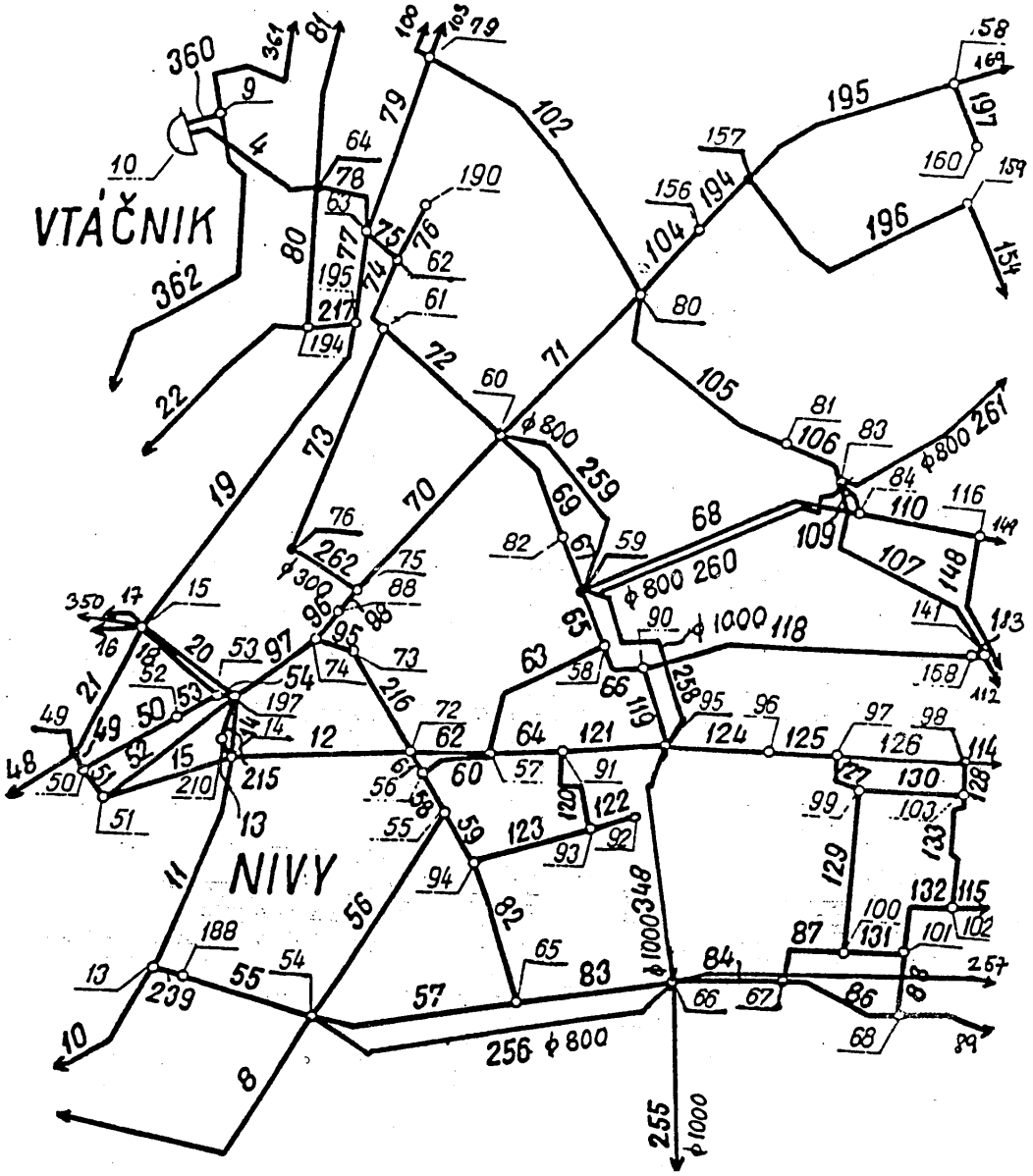
Príloha č.2

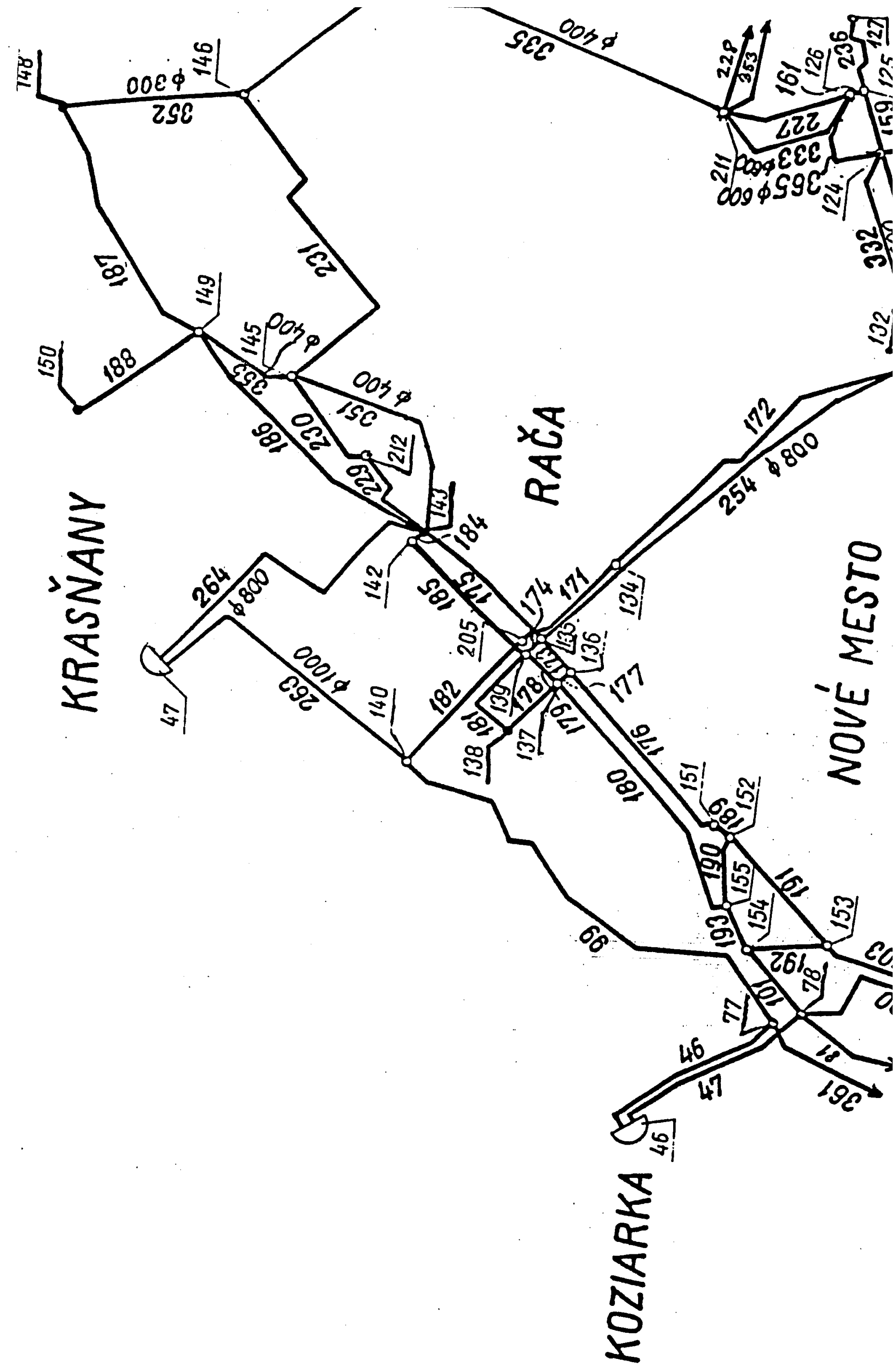
Prvé tlakové pásmo Bratislavy

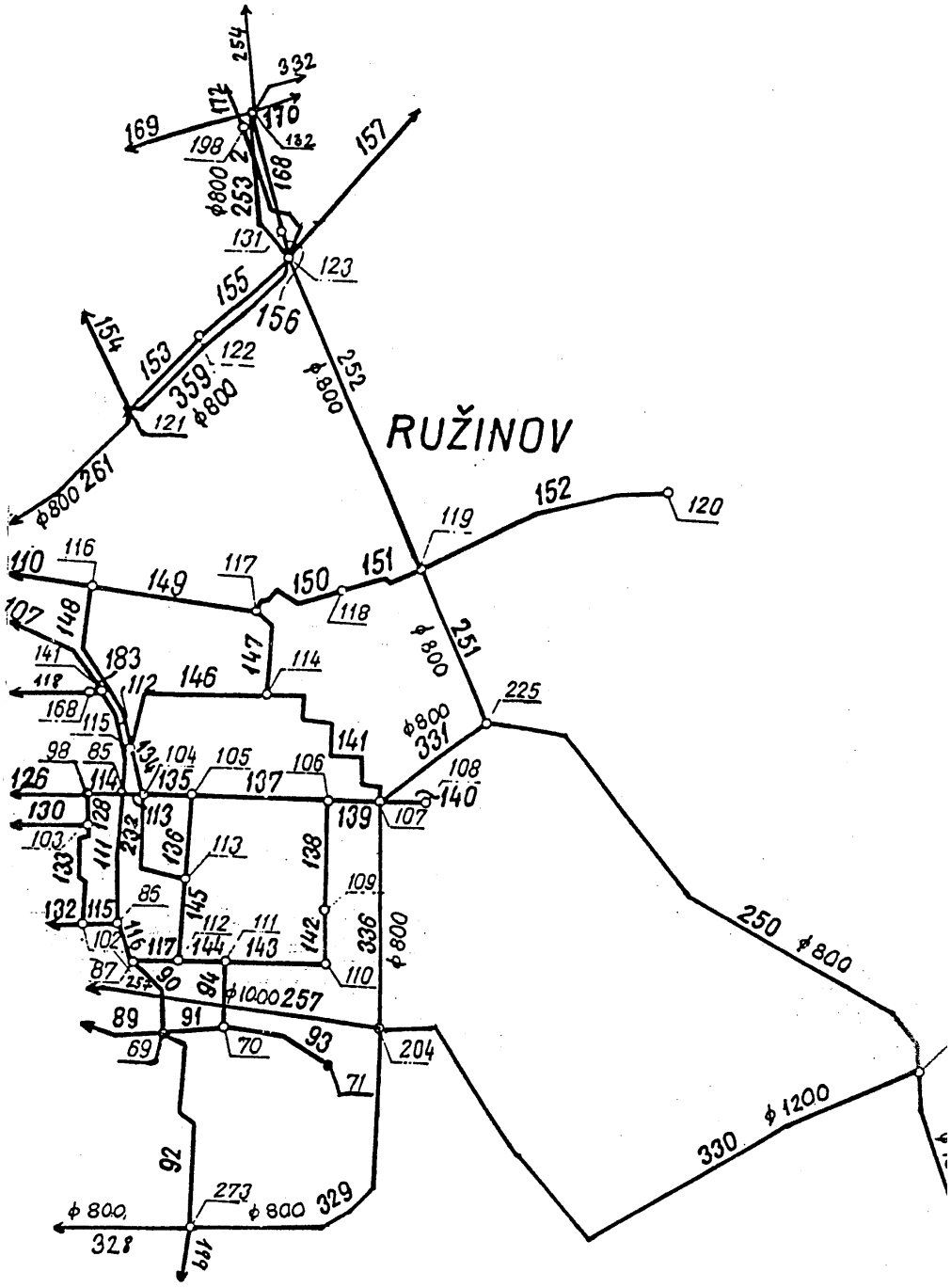
- 1 mapa - lokalita: 5. tlakové pásmo - Koliba
- 2 mapa - lokalita: Bôrik, ČS Pečniansky les, Staré mesto, podkolibské vodojemy
- 3 mapa - lokalita: Bôrik (detail), rozdelovací objekt, tunel
- 4 mapa - lokalita: ČS Karlova Ves, vyrovnávací objem
- 5 mapa - lokalita: Nivy, vodojem Vtáčnik
- 6 mapa - lokalita: Nové mesto, Rača, vodojemy Koziarka, Krasňany
- 7 mapa - lokalita: Ružinov
- 8 mapa - lokalita: Petržalka
- 9 mapa - lokalita: ČS Petržlka (detail)
- 10 mapa - lokalita: ČS Podunajské Biskupice, Vajnory

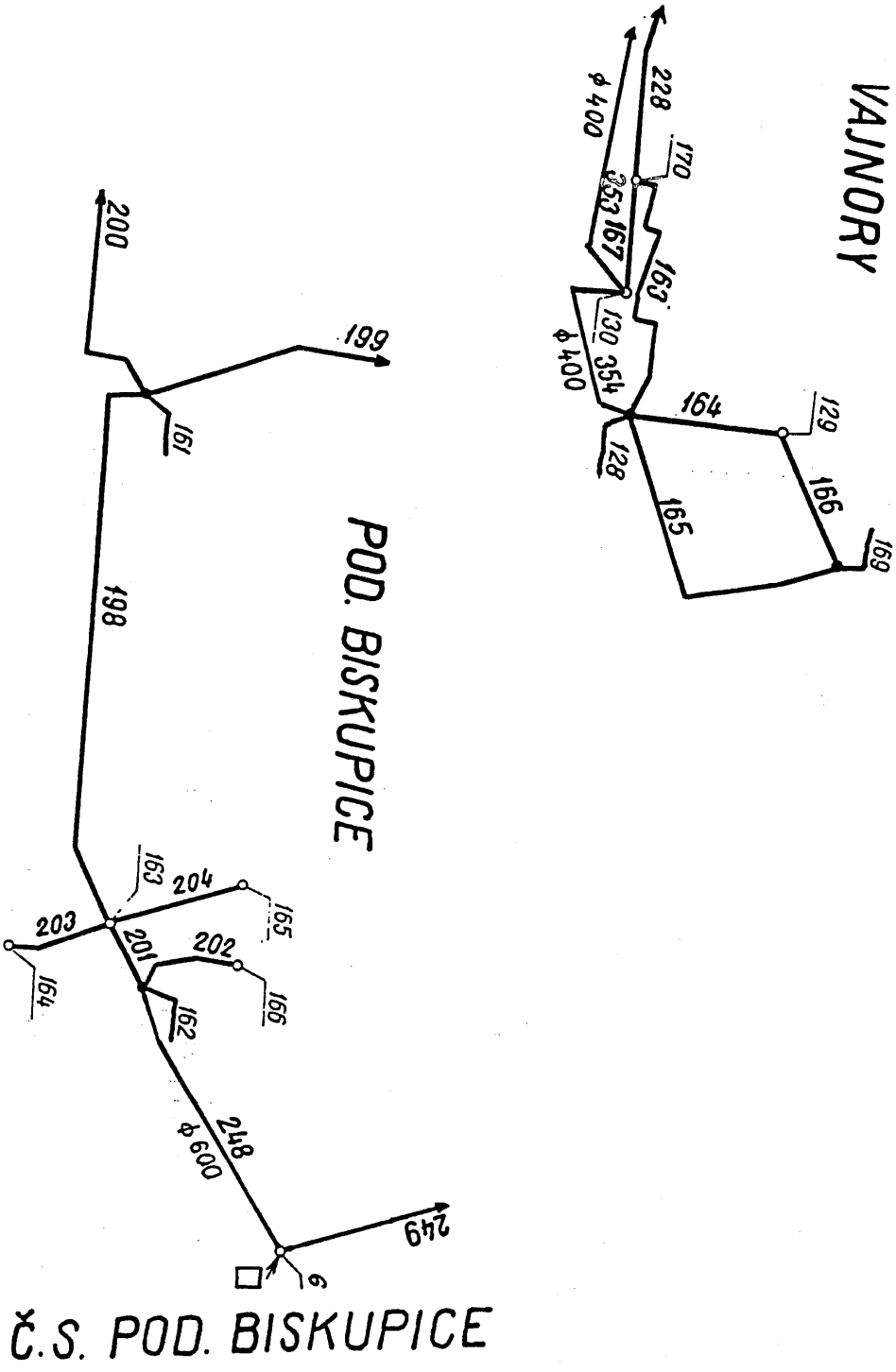

Vodojem Koliba
V1
1
11
10
5
6
7
8
9
10
konc. odber
2
5
6
7
8
9
4
11
3
15
12
3
15
17
12
5004
13
14
14
16
V2
13
17
21
Vodojem Koliba
18
18
20
16
19
20
22
21
23
konc. odber
19
22
Brusnicová
Pyrenejská
Čremchová

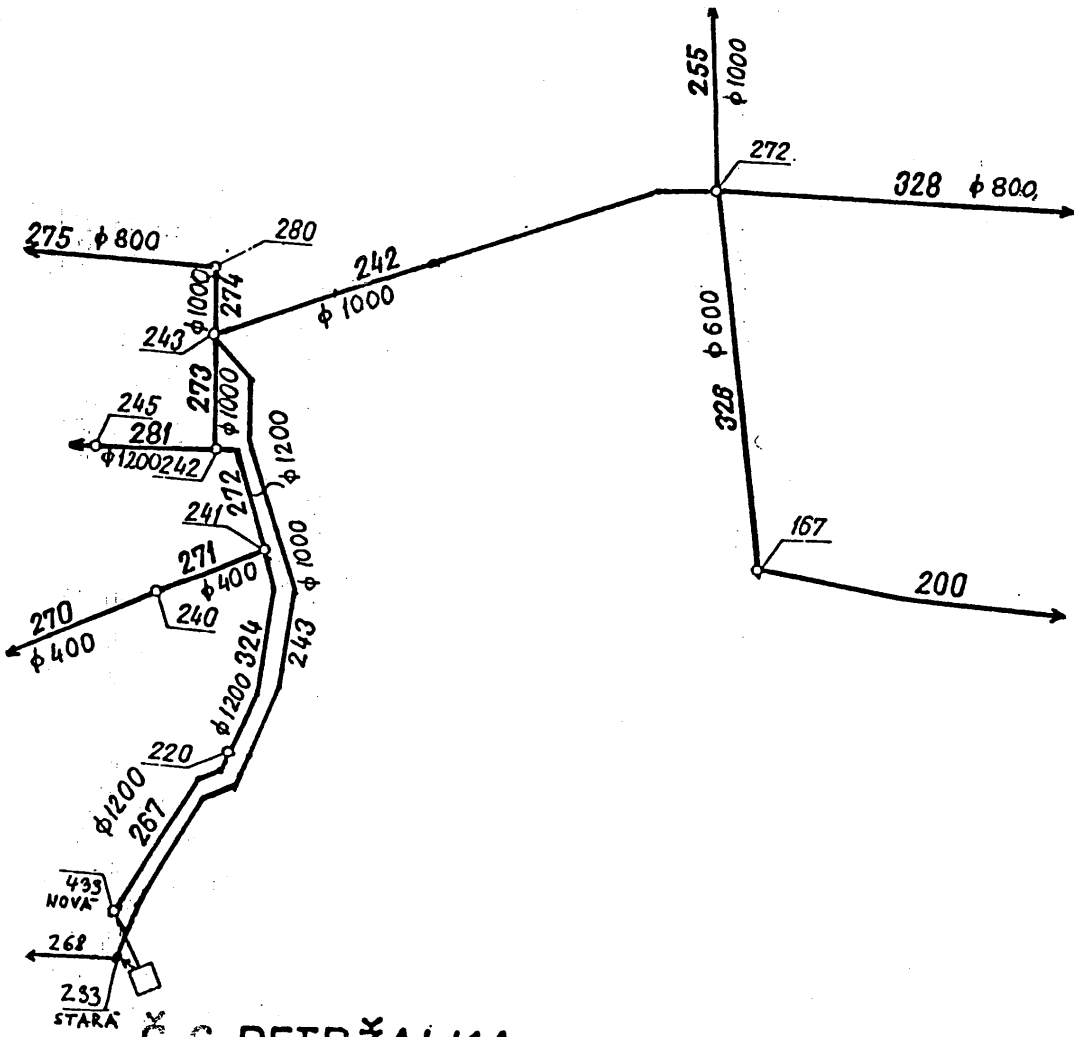

PODKOLIBSKÝ
STARÉ MESTO
BÔRIK
Č.S. PEČENSKÝ LES
A
B
C
D
ϕ800
ϕ1000
370 ϕ1000
368 ϕ1000
369 ϕ1000

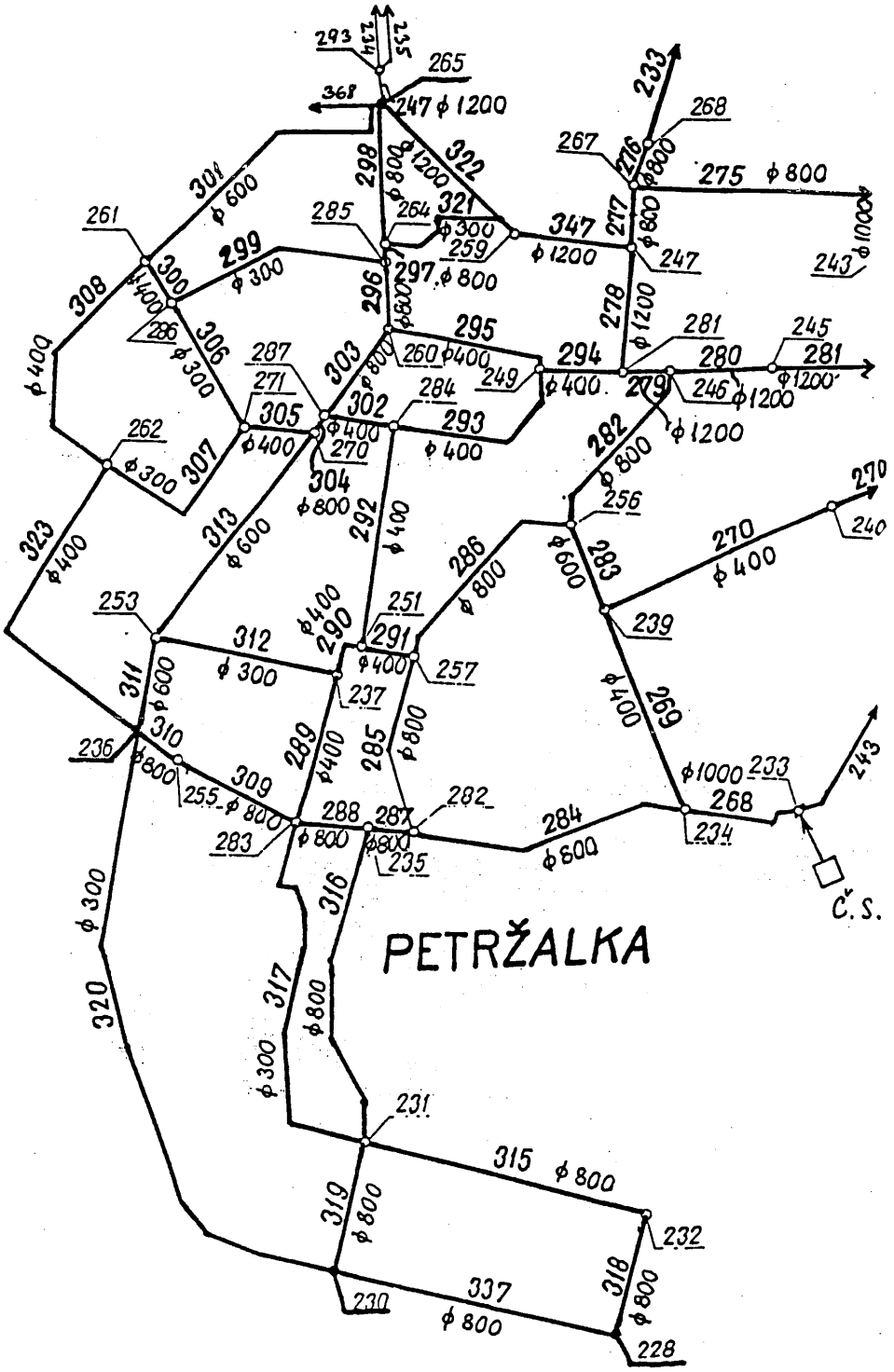

VTÁČNIK
NIVY
φ800
φ300
φ1000
256 φ800
255 φ1000
348 φ1000

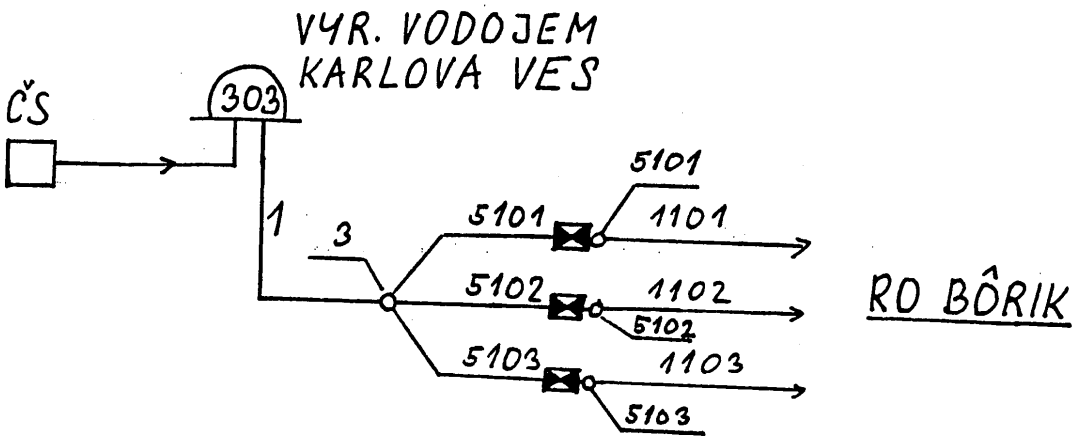

KRASŇANY
RAČA
NOVÉ MESTO
KOZIARKA

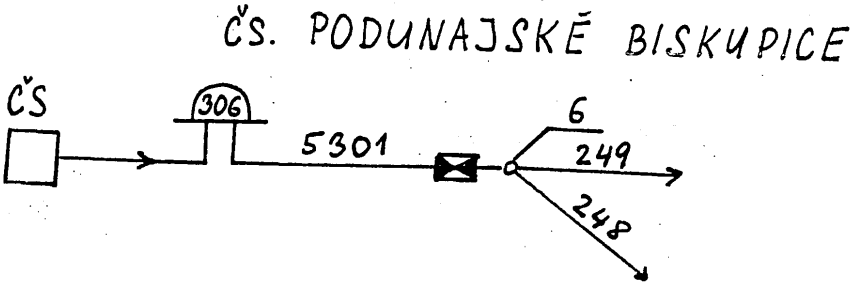

RUŽINOV
254
332
169
172
170
198
132
157
168
253
ϕ800
131
123
155
156
154
153
122
359
ϕ800
121
252
ϕ800
261
ϕ800
152
120
119
110
116
149
117
150
151
118
107
148
141
183
112
146
147
114
118
168
115
ϕ800
251
225
98
85
104
105
106
141
331
108
126
114
135
137
140
130
128
113
139
107
103
111
232
136
113
138
133
109
86
145
132
115
112
111
142
336
ϕ800
102
116
117
144
143
110
87
257
90
94
ϕ1000
257
250
ϕ800
89
91
93
204
69
70
71
92
330
ϕ1200
273
ϕ800
329
ϕ800
328
179

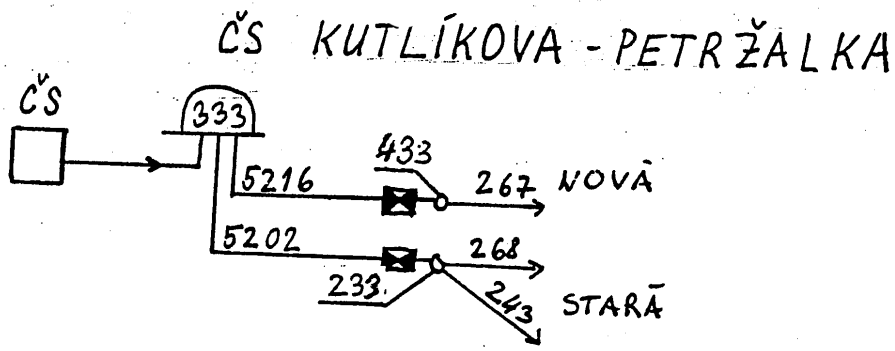

VAJNORY
228
170
ϕ 400
353
167
163
130
354
ϕ 400
164
129
128
165
166
169
POD. BISKUPICE
200
199
161
198
163
204
165
203
201
202
164
166
162
ϕ 600
248
249
6
Č.S. POD. BISKUPICE

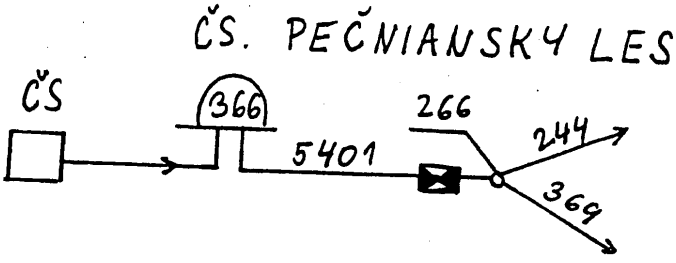

255
ϕ1000
272
328 ϕ800
275 ϕ800
280
242
ϕ1000
274
ϕ1000
243
273
ϕ1000
245
281
ϕ1200
242
ϕ1200
272
241
271
ϕ400
240
270
ϕ400
ϕ1000
243
324
ϕ1200
220
ϕ1200
267
433
NOVÁ
268
233
STARÁ
ϕ600
328
167
200
Č.S. PETRŽALKA

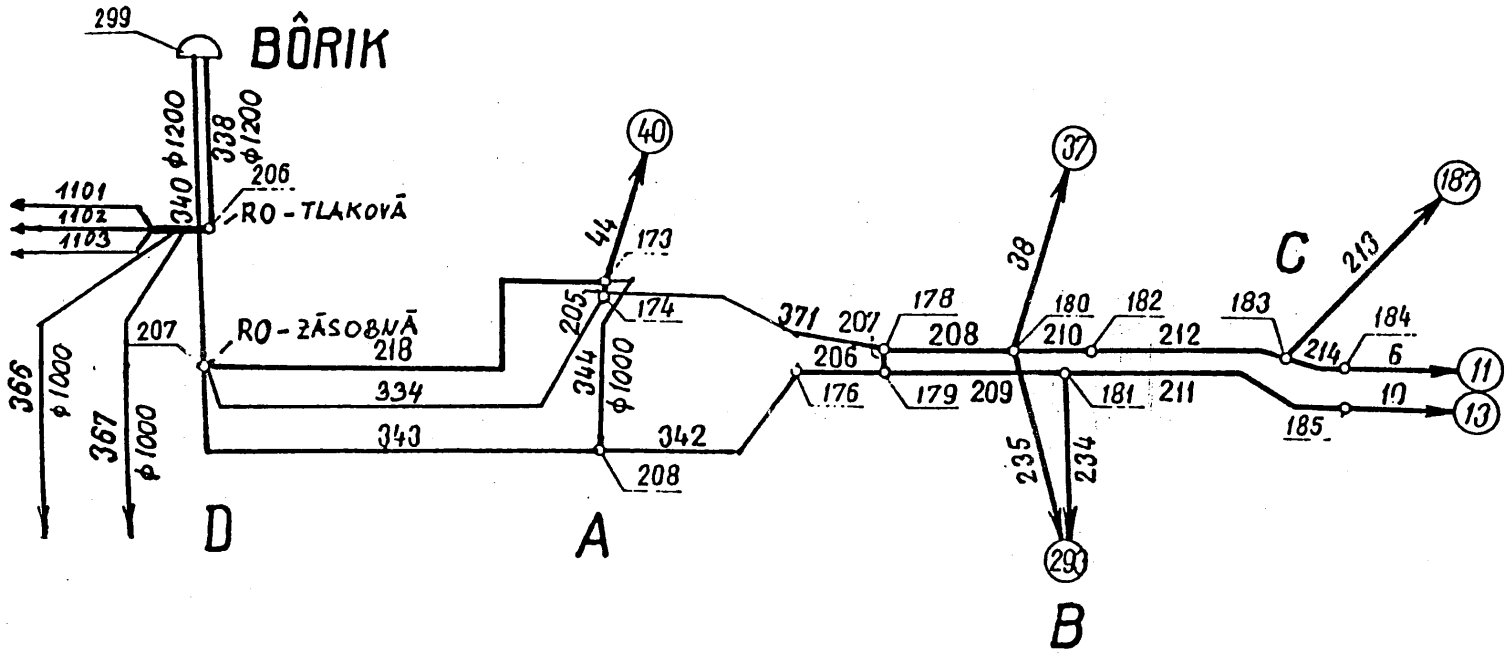

PETRŽALKA
C. S.


VYR. VODOJEM
KARLOVA VES
ČS
303
1
3
5101
1101
5101
5102
1102
5102
5103
1103
5103
RO BÔRIK


ČS. PODUNAJSKÉ BISKUPICE
ČS
306
5301
6
249
248


ČS KUTLÍKOVA - PETRŽALKA
ČS
333
5216
433
267
NOVÁ
5202
268
233
243
STARÁ


ČS. PEČNIANSKY LES
ČS
366
5401
266
244
369


299
BÔRIK
340 ϕ1200
338 ϕ1200
206
RO - TLAKOVÁ
1101
1102
1103
207
RO - ZÁSOBNÁ
218
366 ϕ1000
367 ϕ1000
334
343
344 ϕ1000
205
44
40
173
174
342
208
D
A
371
207
206
176
178
208
179
209
38
37
C
180
182
183
210
212
213
187
184
214
6
11
181
211
10
13
185
235
234
293
B