Žilinská univerzita v Žiline
Fakulta riadenia a informatiky

Tomáš Majer, Ing.

Autoreferát dizertačnej práce

Problémy trasovania v rozsiahlych dopravných sieťach

na získanie akademického titulu „philosophiae doctor“ (v skratke PhD.)
v študijnom programe doktorandského štúdia
aplikovaná informatika

v študijnom odbore
9.2.9 aplikovaná informatika

Žilina, apríl 2013

Dizertačná práca bola vypracovaná v externej forme doktorandského štúdia **na Katedre matematických metód, Fakulte riadenia a informatiky Žilinskej univerzity v Žiline**.

Predkladateľ: Ing. Tomáš Majer
Žilinská univerzita v Žiline
Fakulta riadenia a informatiky
Katedra matematických metód

Školiteľ: doc. RNDr. Stanislav Palúch, CSc.
Žilinská univerzita v Žiline
Fakulta riadenia a informatiky
Katedra matematických metód

Oponenti: prof. RNDr. Jan Černý, DrSc.
Vysoká škola ekonomická v Praze
Fakulta managementu J. Hradec

prof. Ing Karol Matiaško, PhD.
Žilinská univerzita v Žiline
Fakulta riadenia a informatiky

doc. Mgr. Juraj Pekár, PhD.
Ekonomická univerzita v Bratislave
Fakulta hospodárskej informatiky

Autoreferát bol rozoslaný dňa:

Obhajoba dizertačnej práce sa koná dňa o h. pred komisiou pre obhajobu dizertačnej práce schválenou odborovou komisiou v študijnom odbore **9.2.9 aplikovaná informatika**, v študijnom programe **aplikovaná informatika**, vymenovanou dekanom Fakulty riadenia a informatiky Žilinskej univerzity v Žiline dňa

prof. Ing. Martin Klimo, PhD.
predseda odborovej komisie
študijného programu **aplikovaná informatika**
v študijnom odbore **9.2.9 aplikovaná informatika**

Fakulta riadenia a informatiky
Žilinská univerzita v Žiline
Univerzitná 8215/1
010 26 Žilina

Úvod

Pri riadení dopravných systémov, ale aj v iných odvetviach inžinierskej praxe, je veľmi často potrebné vyriešiť úlohu nájdenia optimálnej trasy v dopravnej sieti. Táto úloha je často len čiastkovou úlohou rozsiahlejšieho problému a na vyriešenie celého problému je potrebné nájsť optimálnu trasu medzi dvomi rôznymi bodmi dopravnej siete aj niekoľkotisíckrát. Preto je veľmi dôležité poznať efektívny algoritmus na nájdenie najkratšej trasy v dopravnej sieti pri zohľadnení ďalších obmedzujúcich podmienok.

Vďaka prudkému nárastu výpočtovej sily a bežnej dostupnosti výpočtovej techniky je možné modelovať dopravnú sieť čoraz väčšiu a pritom aj čoraz presnejšie a podrobnejšie. S tým je ale spojený aj nárast veľkosti vstupných dát a aj požiadaviek na kvalitu týchto dát. Aj najlepší algoritmus bude dávať nesprávne výsledky, ak budú nepresné vstupné dáta.

Cieľom dizertačnej práce bolo na základe poznatkov získaných štúdiom známych rýchlych algoritmov na nájdenie najkratšej cesty v rozsiahlom orientovanom grafe navrhnúť a implementovať algoritmus na nájdenie najkratšieho prípustného sledu, ktorý bude rešpektovať zakázané manévre zadané ako vstupný údaj dopravnej siete. Algoritmus musí byť ľahko implementovateľný a pritom dostatočne rýchly aj na mobilných zariadeniach ako napríklad navigačné prístroje. V práci som sa nezaoberal komunikačným modulom pre takéto zariadenia, či už grafickým alebo hlasovým.

1 Súčasný stav problematiky

1.1 Základné pojmy

Na vytvorenie modelu dopravnej siete som využil prostriedky teórie grafov. Všeobecný úvod do tejto matematickej disciplíny je veľmi pekne spracovaný v učebnici [11]. Základné poznatky potrebné pre modelovanie dopravných procesov sú zhrnuté aj v knihách zameraných na riadenie dopravy [2] a [3].

Definícia 1 *Digrafom* nazveme usporiadanú dvojicu $G = (V, H)$, kde V je neprázdna konečná množina a H je množina usporiadaných dvojíc typu (u, v) takých, že $u \in V$, $v \in V$ a $u \neq v$, t. j.

$$H \subseteq \{(u, v) \mid u \neq v,\ u, v \in V\} \subset V \times V$$

Prvky množiny V nazývame vrcholmi a prvky množiny H orientovanými hranami digrafu G.

Definícia 2 Digraf $G = (V, H)$ nazveme hranovo ohodnoteným, ak každej orientovanej hrane $h \in H$ je priradené reálne číslo $c(h)$ nazývané cena hrany h alebo tiež ohodnotenie hrany h. Za hranovo ohodnotený digraf budeme teda pokladať usporiadanú trojicu $G = (V, H, c)$, kde V je množina vrcholov, H množina hrán a $c : H \to \mathbf{R}$ je reálna funkcia definovaná na množine H.

Definícia 3 Nech $G = (V, H)$ je digraf, $v \in V$, $h \in H$. Vrchol v je incidentný s hranou h, ak je v jedným z vrcholov hrany h. Hrany $h,\ k \in H$, $h \neq k$ sú priľahlé (susedné), ak majú spoločný jeden vrchol. Vrcholy u, v, sú priľahlé (susedné), ak $(u, v) \in H$ alebo $(v, u) \in H$.

Nech $G = (V, H)$ je digraf, $u \in V$, $v \in V$, $h \in H$. Hovoríme, že orientovaná hrana h vychádza z vrchola u, alebo že vrchol u je začiatočný vrchol orientovanej hrany h, ak $h = (u, x)$ pre niektoré $x \in V$. Hovoríme, že orientovaná hrana h vchádza do vrchola v, alebo že vrchol v je koncový vrchol orientovanej hrany h, ak $h = (y, v)$ pre niektoré $y \in V$. Orientovaná hrana h je incidentná s vrcholom v, ak hrana h vchádza do vrchola v alebo vychádza z vrchola v.

Symbolom $H^+(v)$ alebo H_v^+ budeme označovať množinu všetkých orientovaných hrán digrafu G vychádzajúcich z vrchola v, symbolom $H^-(v)$ alebo H_v^- budeme označovať množinu všetkých orientovaných hrán grafu G vchádzajúcich do vrchola v, $H(v) = H^+(v) \cup H^-(v)$ je množina všetkých incidentných hrán s vrcholom v.

Symbolom $V^+(v)$ alebo V_v^+ budeme označovať množinu koncových vrcholov všetkých hrán z $H^+(v)$, symbolom $V^-(v)$ alebo V_v^- množinu začiatočných vrcholov všetkých hrán z $H^-(v)$. Platí $V(v) = V^+(v) \cup V^-(v)$.

Definícia 4 Nech $G = (V, H)$ je digraf. *Orientovaný sled* v digrafe G je ľubovoľná alternujúca (striedavá) postupnosť vrcholov a hrán tvaru

$$(v_1, (v_1, v_2), v_2, (v_2, v_3), v_3, \ldots, (v_{k-1}, v_k), v_k).$$

Orientovaný ťah v digrafe G je taký orientovaný sled v digrafe G, v ktorom sa žiadna hrana neopakuje. *Orientovaná cesta* v digrafe G je taký orientovaný sled v digrafe G, v ktorom sa žiaden vrchol neopakuje.

Pri digrafoch budeme používať aj skrátené termíny *sled*, (*ťah*, *cesta*) namiesto *orientovaný sled*, (*orientovaný ťah*, *orientovaná cesta*).

Ak $v_1 = u$ a $v_k = v$, t. j. ak prvým vrcholom sledu je u a posledným v, budeme hovoriť, že ide o sled z u do v, alebo skrátene u–v sled. (Podobne u–v ťah, u–v cesta).

Definícia 5 Nech $m(u, v)$ je u–v sled v hranovo ohodnotenom digrafe $G = (V, H, c)$. *Dĺžkou sledu* $m(u, v)$ alebo tiež *cenou sledu* nazveme súčet ohodnotení jeho hrán, pričom ohodnotenie každej hrany započítavame toľkokrát, koľkokrát sa táto hrana v slede vyskytuje. Dĺžku sledu $m(u, v)$ budeme značiť $\mathrm{d}(m(u, v))$.

Pre ťah a cestu, v ktorých sa podľa ich definície každá hrana môže vyskytovať len raz, môžeme zjednodušene definovať: Dĺžka $\mathrm{d}(m(u, v))$ ťahu alebo cesty) $m(u, v)$ je súčet ohodnotení ich hrán, t. j.

$$\mathrm{d}(m(u, v)) = \sum_{h \in m(u,v)} c(h).$$

Definícia 6 Nech μ je sled

$$\mu = (v_1, (v_1, v_2), v_2, (v_2, v_3), v_3, \ldots, v_{k-1}, (v_{k-1}, v_k), v_k).$$

Podsled sledu μ je ľubovoľná súvislá časť postupnosti μ, ktorá začína aj končí nejakým vrcholom, t.j. $(v_i, (v_i, v_{i+1}), v_{i+1}, \ldots, v_{j-1}, (v_{j-1}, v_j), v_j)$, kde $1 \leq i \leq j \leq k$. Rovnakým spôsobom by sme mohli definovať aj pojmy *podťah* a *podcesta*.

Definícia 7 *Zakázaný manéver* je sled deklarovaný ako zakázaný. Sled μ je *prípustný pri rešpektovaní zakázaných menévrov* (skrátene *prípustný*), ak μ neobsahuje ako podsled žiadny deklarovaný zakázený manéver.

Definícia 8 *Elementárny zakázaný manéver* je sled deklarovaný ako zakázaný, ktorý obsahuje len dve hrany, t.j. $\omega = (i, (i, j), j, (j, k), k)$.

1.2 Algoritmy na hľadanie najkratšej cesty

1.2.1 Dijkstrov algoritmus

Dijkstrov algoritmus je najznámejší algoritmus na nájdenie najkratšej u–v cesty v digrafe. Aj súčasné algoritmy väčšinou vznikli ako modifikácia tohto algoritmu. Popis algoritmu možno nájsť v [11].

1.2.2 Label set algoritmus

Label set algoritmus je vlastne len rýchla implementácia Dijkstrovho algoritmu, ktorá využívaj prioritný front na efektívne vyhľadávanie nespracovaného vrcholu s najmenšou dočasnou značkou.

Algoritmus 1 *Label set algoritmus* na nájdenie najkratšej u–v cesty v hranovo ohodnotenom digrafe $G = (V, H, c)$ s nezáporným ohodnotením hrán.

Krok 1. *Inicializácia.*
Polož $t(u) := 0$, $t(i) := \infty$ pre $i \in V,\ i \neq u$ a $x(i) := 0$ pre každé $i \in V$. Polož $\mathcal{E} := \{u\}$.

Krok 2. Vyber $r \in \mathcal{E}$ s najmenšou značkou $t()$ a polož $\mathcal{E} := \mathcal{E} - \{r\}$.
Ak $r = v$ STOP. $t(v)$ predstavuje dĺžku najkratšej u–v cesty.
Inak pre všetky hrany $h = (r, j) \in H^+(r)$ urob:
Ak $t(j) > t(r) + c(r, j)$, potom $t(j) := t(r) + c(r, j),\ x(j) := r,\ \mathcal{E} := \mathcal{E} \cup \{j\}$.

Krok 3. Ak $\mathcal{E} \neq \emptyset$, choď na Krok 2.
Ak $\mathcal{E} = \emptyset$, potom v je nedosiahnuteľný z u.

Nech je prioritný front $\mathcal{E}$ implementovaný ako Fibonacciho halda. Označme $n = |V|$, $m = |H|$ a $n_{\mathcal{E}} = \max|\mathcal{E}| \leq n$. Každý vrchol vyberiem z frontu $\mathcal{E}$ nanajvýš raz, t.j. prevedieme nanajvýš n operácii vybratia minimálneho prvku haldy so zložitosťou $O(\log(n_{\mathcal{E}}))$. Zlepšenie značky vrchola, ktorých môže byť nanajvýš m, vyžaduje vloženie prvku do $\mathcal{E}$ resp. zníženie priority prvku. Obe tieto operácie vo Fibonacciho halde majú konštantnú zložitosť $O(1)$. Výsledná zložitosť label set algoritmu je teda $O(n.log(n_{\mathcal{E}}) + m)$. Pre digrafy s počtom hrán rádovo $m \approx K.n$, ako napríklad model cestnej siete, bude zložitosť len $O(n.\log(n))$.

1.2.3 Obojsmerný Dijkstrov algoritmus

Obojsmerný algoritmus hľadá cestu v priamom smere z vrcholu u do vrcholu v a aj v opačnom smere z vrcholu v do vrcholu u naraz. Jeho asymptotická zložitosť je rovnaká ako pri jeho jednostrannej verzii. Z hľadiska priemerného počtu krokov by mal byť približne dvakrát rýchlejší a na nájdenie najkratšej cesty by mal prehľadať len polovičný počet hrán. Algoritmus je možné jednoducho paralelizovať do dvoch samostatných procesov.

1.2.4 A^* algoritmus

Tento algoritmus bol prezentovaný autormi Hart, Nilsson, Raphael v príspevku [9]. Je to asi prvý článok, ktorý sa zaoberal problémom hľadania najkratšej cesty aj z pohľadu hľadania heuristického a nielen exaktného algoritmu. Okrem toho na prehľadanie čo najmenšieho počtu vrcholov autori navrhli využiť aj niektoré informácie o modelovanej realite (napr. zemepisnej polohe uzlov dopravnej siete), ktoré sa do modelu (hranovo ohodnoteného digrafu) nemuseli preniesť. Algoritmus kladie dôraz na poradie prehľadávania vrcholov, pričom sa snaží v maximálne možnej miere nespracovávať vrcholy, ktoré nemôžu ležať na najkratšej u–v ceste.

Je to veľmi významná práca, na ktorú nadviazali vo svojich príspevkoch aj mnohí súčasní autori. Rozdiel oproti Label set algoritmu je v tom, že priorita vrcholu i v prioritnom fronte nie je hodnota značky $t(i)$ ale odhad dĺžky $u-v$ cesty, ktorá vedie cez vrchol i. Táto hodnota je najmenšia pre vrcholy digrafu, ktoré ležia na najkratšej $u-v$ ceste a preto budú algoritmom uprednostňované práve tieto vrcholy. Priorita vrchola i bude rovná $t(i)+\tilde{\mathrm{d}}(i,v)$, kde $\tilde{\mathrm{d}}(i,v)$ je dolný odhad vzdialenosti cieľového vrcholu v od aktuálneho vrcholu i. Tento odhad môžeme počítať zo známych zemepisných súradníc vrcholov.

Haversine metóda výpočtu vzdialenosti bodov na povrchu Zeme Táto metóda aproximuje zemský povrch povrchom gule. Na výpočet vzdialenosti využíva vzťah medzi vzdialenosťami troch bodov na povrchu gule a uhlom, ktoré tieto body zvierajú. Poloha bodov na povrchu gule je zadaná ich sférickými súradnicami a jedným z týchto bodov (ten, pri ktorom meriame uhol) je severný (alebo južný) pól.

Označme λ_1, λ_2 zemepisnú dĺžku a φ_1, φ_2 zemepisnú šírku bodov, ktorých vzdialenosť nás zaujíma. Potom pre vzdialenosť medzi nimi, ich vzdialenosť od pólu a uhol $\Delta\lambda =$

$|\lambda_1 - \lambda_2|$, ktorý zvierajú pri póle platí

$$\text{haversin}\left(\frac{d}{R}\right) = \text{haversin}(\varphi_1 - \varphi_2) + \cos(\varphi_1)\cos(\varphi_2)\text{haversin}(\Delta\lambda), \tag{1}$$

kde

$$\text{haversin}(x) = \sin^2\left(\frac{x}{2}\right) = \frac{1-\cos(x)}{2},$$

d je neznáma vzdialenosť medzi bodmi a R je polomer Zeme. Označme

$$h = \text{haversin}(\varphi_1 - \varphi_2) + \cos(\varphi_1)\cos(\varphi_2)\text{haversin}(\Delta\lambda), \tag{2}$$

potom

$$d = R \cdot \text{haversin}^{-1}(h) = 2R \cdot \arcsin(\sqrt{h}). \tag{3}$$

1.2.5 ALT algoritmus

ALT je skratka pre A^* — Landmark — Triangle inequality. Ako už napovedá názov, jadrom je A^* algoritmus. Príspevok [6], v ktorom bol tento algoritmus prvý krát prezentovaný, bol publikovaný v roku 2005. Niektoré ďalšie podrobnosti algoritmu a jeho implementácie boli prezentované v [7].

Zaujímavé je, že na odhad vzdielenosti medzi dvoma vrcholmi nie sú použité doplnkové informácie z modelovanej reality, ale len predvypočítané údaje z modelu (digrafu). Metóda využíva predvypočítané vzdialenosti všetkých vrcholov digrafu k tzv. landmarkom – rádovo len niekoľko špeciálne vybraných vrcholov grafu. Autori v príspevku [6] prezentujú aj spôsob, akým možno tieto landmarky vybrať. Tie sú volené rovnomerne zpomedzi vrcholov čo najďalej od centra grafu (napr. na okraji modelovanej dopravnej siete). Na základe týchto predvypočítaných vzdialeností je potom možné vypočítať veľmi dobrý dolný odhad vzdialenosti dvoch vrcholov.

Veta 1 Nech $\mathrm{d}(u, v)$ je dĺžka najkratšej $u - v$ cesty v hranovo ohdnotenom digrafe $G = (V, H, c)$. Potom pre všetky trojice vrcholov $u, v, l \in V$ platí *trojuholníková nerovnosť*

$$\mathrm{d}(u, v) + \mathrm{d}(v, l) \geq \mathrm{d}(u, l). \tag{4}$$

Jednoduchou úpravou (4) dostávame vzťah pre dolný odhad dĺžky $u - v$ cesty

$$\mathrm{d}(u, v) \geq \mathrm{d}(u, l) - \mathrm{d}(v, l). \tag{5}$$

Veta 2 Nech $L \subset V$ je množina landmarkov pre hranovo ohodnotený digraf $G = (V, H, c)$. Potom dolný odhad dĺžky najkratšej u–v cesty môžeme určiť ako

$$\lambda(u, v) = \max_{l \in L} \Big\{ \Big(\mathrm{d}(u, l) - \mathrm{d}(v, l) \Big) \Big\} \leq \mathrm{d}(u, v). \tag{6}$$

1.2.6 REACH algoritmus

REACH algoritmus bol publikovaný v príspevku [8] v roku 2004. Ide o modifikáciu Dijkstrovho algoritmu. Na základe funkcie „reach“ (dosiahnuteľnosť) $r : V \to \mathbb{R}$ predvypočítanej pre každý vrchol $i \in V$ hranovo ohodnoteného digrafu $G = (V, H, c)$ a dolného odhadu vzdialenosti vrcholu i od cieľového vrcholu $v \in V$ hľadanej u–v cesty definuje funkciu, ktorá dokáže identifikovať vrcholy, cez ktoré nemôže viesť najkratšia u–v cesta a teda nemusia byť spracované (označené dočasnou značkou resp. vložené do frontu).

Definícia 9 Nech $G = (V, H, c)$ je hranovo ohodnotený digraf s nezápornými ohodnoteniami hrán. Nech P je najkratšia u–v cesta z nejakého vrchola $u \in V$ do nejakého vrchola $v \in V$ taká, že prechádza vrcholom w.
Potom dosiahnuteľnosť vrchola w na ceste P je

$$r(w, P) = \min\{d(u, w), d(w, v)\}, \tag{7}$$

kde $d(i, j)$ predstavuje dĺžku najkratšej i–j cesty.
Dosiahnuteľnosť vrchola w v digrafe G je

$$r(w, G) = r(w) = \max_{Q}\{r(w, Q) | Q \text{ je najkratšia cesta taká, že } w \in Q\}. \tag{8}$$

Potom modifikovaný Dijkstrov algoritmus spracováva len tie vrcholy $w \in V$, pre ktoré

$$r(w) \geq t(u, w) \text{ alebo } r(w) \geq \hat{d}(w, v), \tag{9}$$

kde $t(u, w)$ je dĺžka najlepšej doteraz nájdenej u–w cesty (značka t pri vrchole w) a $\hat{d}(w, v)$ je dolný odhad najkratšej w–v, získanej napr. pomocou zemepisnej polohy vrcholov w a v.

Presné vyčíslenie funkcie dosiahnuteľnosti r vyžaduje uplnú maticu vzdialeností medzi ľubovoľnými vrcholmi digrafu G, čo je pri grafoch s rádovo niekoľkými desiatkami miliónov vrcholov takmer nemožné. Na potreby REACH algoritmu však stačí horný odhad hodnoty tejto funkcie.

1.2.7 Algoritmus na hľadanie K–najkratších ciest

V prípadoch, keď kritérium optimálnej cesty je ťažko formalizovateľné do presného tvaru, môže byť výhodné vyhľadať K–najkratších ciest a samotný výber optimálnej cesty z nájdených možností ponechať na človeka. Plesník v [13] formuloval algoritmus na nájdenie K–najkratších ciest ako opakované hľadanie najkratšej cesty v podgrafoch s redukovaným počtom hrán. Tento algoritmus je náročný na implementáciu a jeho výpočtová zložitosť je $O(Kn^3)$. Palúch v [12] navrhol iný algoritmus na nájdenie K–najkratších ciest, ktorý využíva K značiek pre každý vrchol.

1.2.8 Algoritmus na hľadanie najkratšieho prípustného ťahu pri rešpektovaním zakázaných odbočení

Tento algoritmus bol prezentovaný Peškom v monografii [10] a rieši úlohu nájdenia najkratšieho prípustného ťahu v hranovo ohodnotenom digrafe $G = (V, H, c)$ – modeli dopravnej siete, pričom sú rešpektované dopravné obmedzenia – zakázané odbočenia, ktoré pozostávajú len z dvoch hrán.

Nájsť najkratší prípustný u–v ťah v digrafe G so zakázanými odbočeniami $\mathcal{Z}$ znamená nájsť najkratšiu cestu v hranovom digrafe G_H skonštruovanom z grafu G. Úlohu hľadania najkratšieho prípustného ťahu v digrafe so zakázanými odbočeniami sa tak podarilo previesť na klasickú úlohu hľadania najkratšej cesty, na ktorú môžeme použiť príslušné známe algoritmy.

2 Výsledky práce

2.1 Porovnanie známych algoritmov

Implementácie vybraných algoritmov na hľadanie najkratšej cesty som testoval na bežnom počítači PC s 2x2 jadrovým procesorom Intel(R) i3-2120 @3.3GHz, 4GB operačnej pamäti a 64-bitovým operačným systémom Debian Linux 6.0 s verziou kernelu 2.6. Okrem tejto platformy som vytvorený program spustil aj na „doštičke“ Raspberry Pi s jednojadrovým procesorom ARMv6l BCM2708 spoločnosti Broadcom, 256 MB operačnej pamäte a operačným systémom Raspbian Linux pre MIPS architektúru s verziou kernelu 3.2. Namerané výsledky sú zapísané v niekoľkých nesledujúcich tabuľkách.

Namerané výsledky na PC s 2x2 jadrovým procerom Intel(R) i3

Algoritmus	Jednosmerný				Obojsmerný - sériový			
	n	l	t_1	t_2	n	l	t_1	t_2
Dijkstrov	109 840	194,881	**29,3**	29,3	85 570	194,881	**23,6**	23,6
A^*	48 810	194,881	**22,1**	22,1	32 895	*195,234*	**15,8**	15,8
Reach	11 222	194,881	**6,3**	6,3	9 840	194,881	**6,4**	6,4
Reach + A^*	6 314	194,881	**4.0**	4.0	4 853	*195,200*	**3.9**	3.9

Namerané výsledky na Raspberry Pi s procesorom Broadcom ARMv6

Algoritmus	Jednosmerný				Obojsmerný - sériový			
	n	l	t_1	t_2	n	l	t_1	t_2
Dijkstrov	109 840	194,881	**538,7**	537,6	85 570	194,881	**498.7**	497.7
A^*	48 810	194,881	**470,8**	469,9	32 895	*195,234*	**372.4**	371.6
Reach	11 222	194,881	**126,1**	125,9	9 840	194,881	**135.3**	135.0
Reach + A^*	6 314	194,881	**79,6**	79,4	4 853	*195,200*	**81.4**	81.2

n – počet spracovaných vrcholov, l – priemerná dĺžka najkratšej cesty [km],
t_1 – monolitický čas výpočtu [ms], t_2 – spotrebovaný čas CPU [ms]

Tabuľka 1: Rýchlosť jednosmerných a obojsmerných verzií vybraných algoritmov

Asi najdôležitejšie z tohto experimentu je zistenie, že obojsmerná verzia A^* a ani jeho

spojenie s Reach algoritmom nenašli vždy najkratšiu cestu. To znamená, že podmienka ukončenia obojsmernej verzie prevzatá z Dijkstrovho algoritmu nestačí na nájdenie najkratšej cesty obojsmerným algoritmom A^*.

Zaujímavé je, že obojsmerná verzia algoritmov nevyšla podľa očakávania dvakrát rýchlejšia ako jednosmerná, pre Reach algoritmus a jeho kombináciu s A^* algoritmom vyšla v niektorých prípadoch dokonca ešte pomalšia.

Namerané výsledky na PC s 2x2 jadrovým procerom Intel(R) i3

Algoritmus	Obojsmerný - sériový				Obojsmerný - paralelný			
	n	l	t_1	t_2	n	l	t_1	t_2
Dijkstrov	85 570	194,881	**23,6**	23,6	84 991	194,881	**17,3**	28,6
A^*	32 895	*195,234*	**15,8**	15,8	32 828	*195,220*	**12,9**	19,8
Reach	9 840	194,881	**6,4**	6,4	9 810	194,881	**7,4**	8,7
Reach + A^*	4 853	*195,200*	**3,9**	3,9	4 841	*195,195*	**5,2**	5,1

Namerané výsledky na Raspberry Pi s procesorom Broadcom ARMv6

Algoritmus	Obojsmerný - sériový				Obojsmerný - paralelný			
	n	l	t_1	t_2	n	l	t_1	t_2
Dijkstrov	85 570	194,881	**498,7**	497,7	84 819	194,881	**466,5**	464,1
A^*	32 895	*195,234*	**372,4**	371,6	32 791	*195,202*	**351,2**	350,4
Reach	9 840	194,881	**135,3**	135,0	9 806	194,881	**128,4**	128,0
Reach + A^*	4 853	*195,200*	**81,4**	81,2	4 861	*195,194*	**78,4**	78,1

n – počet spracovaných vrcholov, l – priemerná dĺžka najkratšej cesty [km],
t_1 – monolitický čas výpočtu [ms], t_2 – spotrebovaný čas CPU [ms]

Tabuľka 2: Rýchlosť sériových a paralelných obojsmerných verzií vybraných algoritmov

Počty spracovaných vrcholov v paralelných verziách sú rôzne od počtu spracovaných vrcholov v sériovej verzii ale aj medzi sebou v opakovaných spusteniach. Je to dané tým, že pomer množstva spracovaných vrcholov v priamom a v opačnom smere sa môže líšiť v závislosti od prideleného procesorového času pre jednotlivé vlákna a teda počet spracovaných vrcholov v paralelných verziách algoritmov je závislý od aktuálnej dispozície voľných jadier procesora v operačnom systéme.

Paralelizácia obojsmerných algoritmov nepriniesla očakávané zrýchlenie. Pre Reach algoritmus aj jeho kombináciu s A^* dokonca vychádza celkový čas výpočtu väčší ako pri sérovej verzii, takže prostriedky investované do potrebnej réžie na synchronizáciu vlákien sa nám pri týchto algoritmoch nevrátili. V prípade Dijkstrovho algoritmu je paralelná verzia približne o štvrtinu rýchlejšia, čo pri výpočte vo väčších digrafoch bez GPS súradníc jednotlivých vrcholov môže byť významným prínosom.

2.1.1 Max metóda na odhad vzdialenosti bodov na povrchu Zeme

Aj táto metóda rovnako ako Haversine metóda predpokladá, že povrch Zeme je povrchom gule. Metóda najskôr vypočíta dĺžku oblúkov, ako keby obidva body mali rovnakú zemepisnú šírku alebo rovnakú zemepisnú dĺžku. Výsledný dolný odhad vzdialenosti bude

maximum z dĺžok týchto dvoch oblúkov

$$d = R \cdot \max\left\{|\varphi_1 - \varphi_2|, |\lambda_1 - \lambda_2| * \cos(\max\{\varphi_1, \varphi_2\})\right\}. \tag{10}$$

Tento odhad bude menej tesný, ale výpočet bude rýchlejší. Ak si predvypočítame polomer jednotlivých rovnobežiek, vôbec nebudeme potrebovať výpočty v pohyblivej desatinnej čiarke. Celkový účinok na rýchlosť nájdenia najkratšej cesty teda môže byť lepší ako pri presnejšej ale pomalšej Haversine metóde. Implementácia funkcie na výpočet vzdialenosti pomocou max metódy je na obrázku Obr. 1.

```
int dist_max( int lat1, int long1, int lat2, int long2 )
{
    int dx = PARALLEL_RADIUS[ MAX( lat1, lat2 ) / 100000 ]
           * (long long) abs( long1 - long2 ) / 100000;
    int dy = PARALLEL_RADIUS[ 0 ]
           * (long long) abs( lat1 - lat2 ) / 100000;
    return MAX( dx, dy );
}
```

Obr. 1: Funkcia na dolný odhad vzdialenosti pomocou max metódy

Namerané výsledky na PC s 2x2 jadrovým procerom Intel(R) i3

Algoritmus	Haversine				Max			
	n	l	t_1	t_2	n	l	t_1	t_2
A^*	48 810	194,881	**22,1**	22,1	55 352	194,881	**18,0**	18,0
Reach	11 222	194,881	**6,3**	6,3	11 615	194,881	**4,3**	4,3
Reach + A^*	6 314	194,881	**4,0**	4,0	7 176	194,881	**3,1**	3,1

Namerané výsledky na Raspberry Pi s procesorom Broadcom ARMv6

Algoritmus	Haversine				Max			
	n	l	t_1	t_2	n	l	t_1	t_2
A^*	48 810	194,881	**470,8**	469,9	55 352	194,881	**437,1**	436,3
Reach	11 222	194,881	**126,1**	125,9	11 615	194,881	**102,2**	102,0
Reach + A^*	6 314	194,881	**79,6**	79,4	7 176	194,881	**72,4**	72,2

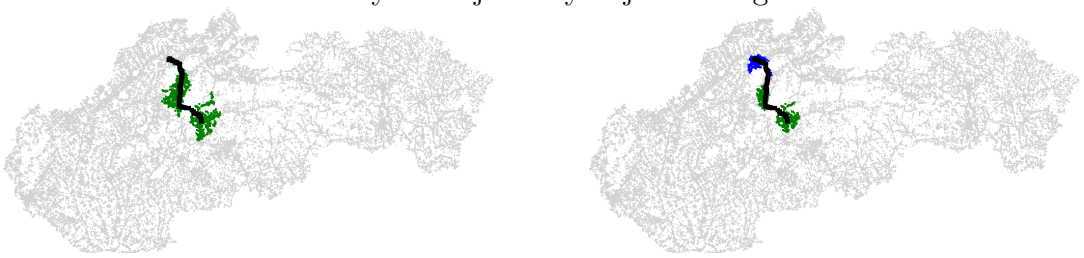
n – počet spracovaných vrcholov, l – priemerná dĺžka najkratšej cesty [km],

t_1 – monolitický čas výpočtu [ms], t_2 – spotrebovaný čas CPU [ms]

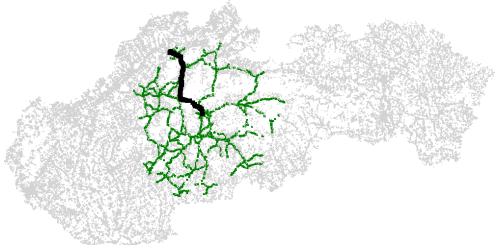
Tabuľka 3: Porovnanie Haversine a Max odhadu vzdialenosti v jednosmerných algoritmoch

Z výsledkov je zrejmé, že použitím Max metódy sa vo všetkých testovaných prípadoch spracuje viac vrcholov, ale vďaka rýchlosti výpočtu odhadu je absolútny čas potrebný na nájdenie najkratšej cesty vždy menší. Tento experiment teda jednoznačne ukázal, že metóda Max je na odhad vzdialeností vo všetkých verziách všetkých algoritmov výhodnejšia ako metóda Haversine.

Jednosmerný a obojsmerný Dijkstrov algoritmus



Jednosmerný a obojsmerný A^* algoritmus



REACH algoritmus

Obr. 2: Znázornenie spracovaných vrcholov vybranými algoritmami

2.2 Vplyv usporiadania vrcholov na rýchlosť výpočtu

Nech je digraf taký rozsiahly, že sa celý aj s medzivýsledkami potrebnými pre beh algoritmu nezmestí do operačnej pamäte. Vtedy je nevyhnutné zorganizovať údajovú štruktúru tak, aby bola do pamäte načítaná len časť digrafu – niekoľko dátových blokov. Vrcholy digrafu chceme usporiadať tak, aby v každom bloku bolo čo najviac vrcholov spracovávaných za sebou. Tak sa vyhneme potrebe načítať do pamäte iný blok a teda tým minimalizujeme aj počet operácií načítania bloku z disku.

Urobil som experiment, v ktorom som vrcholy digrafu usporiadal do blokov nasledovnými spôsobmi:

1. Pôvodné usporiadanie vrcholov digrafu bez zmeny tak, ako mi boli údaje poskytnuté.

2. Náhodné usporiadanie vrcholov.

3. Usporiadanie vrcholov tak, že sa digraf – model cestnej siete – rozdelil do dlaždíc podľa GPS súradníc. Veľkosť dlaždice bola zvolená $0,1° \times 0,1°$ a dlaždice sa zapisovali

do súboru postupne zdola nahor a zľava doprava (platí pre Európu so severovýchodnými GPS súradnicami).

4. Usporiadanie vrcholov v poradí, v akom sú spracovávané pomocou Dijkstrovho algoritmu pri hľadaní ciest z vrcholu u do všetkých ostatných vrcholov grafu, pričom vrchol u leží v strede digrafu (má čo možno najmenšiu excentricitu) – vrcholy sú usporiadané podľa vzdialenosti od tohoto bodu vzostupne.

5. Usporiadanie vrcholov v poradí, v akom sú spracovávané pomocou Dijkstrovho algoritmu pri hľadaní ciest z vrcholu u do všetkých ostatných vrcholov grafu, pričom vrchol u leží na okraji digrafu (má veľkú excentricitu).

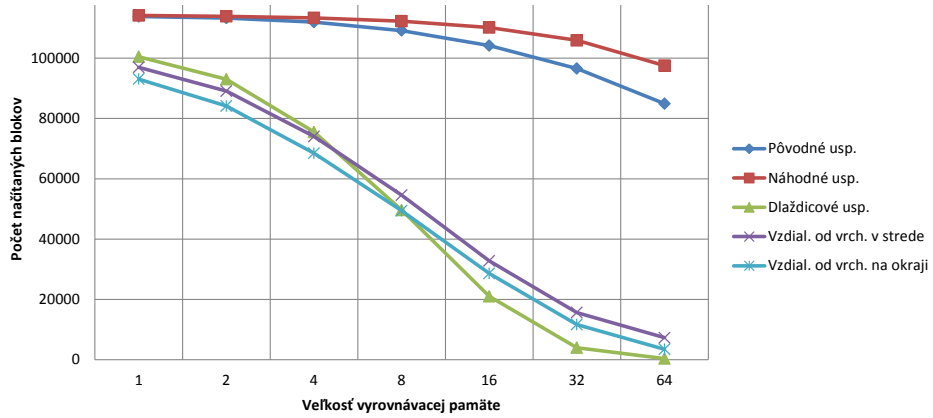
Experiment som urobil v digrafe slovenskej cestnej siete. Náhodne som vybral tisíc rôznych dvojíc vrcholov, medzi ktorými som potom hľadal najkratšiu cestu pomocou Dijkstrovho algoritmu pre všetkých päť variantov usporiadania vrcholov digrafu. Vrcholy som v každom variante vyhľadal cez ich GPS súradnice aby som zabezpečil, že v každom variante digrafu vyhľadávam najkratšie cesty medzi tými istými vrcholmi, aj keď v rôznych digrafoch majú iné poradové čísla. Vo všetkých variantoch digrafu boli dĺžky nájdených najkratších ciest naozaj zhodné.

Experiment som opakoval pre rôzne nastavenie veľkosti vyrovnávacej pamäte a sledoval som počet načítaných blokov z disku (spolu za všetky polia **S**, **V** a **C**) pre každý variant digrafu. Namerané údaje uvádzam v tabuľke Tabuľka 4 a ich grafické znázornenie na obrázku Obr. 3.

Usporiadanie vrcholov	Veľkosť vyrovnávacej pamäte [blok]						
	1	2	4	8	16	32	64
Pôvodné usp.	113 818	113 313	111 970	109 157	104 166	96 537	84 894
Náhodné usp.	114 157	113 894	113 356	112 282	110 142	105 925	97 539
Dlaždicové usp.	100 491	93 017	75 522	49 625	21 023	3 974	364
Vzdial. od vrch. v strede	96 958	89 080	74 029	54 560	32 784	15 639	7 311
Vzdial. od vrch. na okraji	93 078	84 139	68 481	49 410	28 616	11 656	3 462

Tabuľka 4: Počet načítaných blokov behom výpočtu v závislosti od veľkosti vyrovnávacej pamäte

Môžeme si všimnúť, že pre menšie veľkosti vyrovnávacej pamäte vychádza spomedzi týchto piatich variantov najlepšie ten, kde sú vrcholy usporiadané v poradí ich spracovania pomocou Dijkstrovho algoritmu. Postupným zväčšovaním vyrovnávacej pamäte sa ale do čela dostáva dlaždicové usporiadanie podľa GPS súradníc. To si vysvetľujem tým, že dlaždice boli v niektorých prípadoch príliš veľké a počet vrcholov v nich prevýšil počet vrcholov v dátovom bloku. Tento nedostatok sa zväčšovaním vyrovnávacej pamäte postupne stratil. Najhoršie dopadlo podľa očakávania náhodné usporiadanie vrcholov, ale usporiadanie v pôvodnom poradí na tom nebolo oveľa lepšie.



Obr. 3: Závislosť počtu načítaných dátových blokov od veľkosti vyrovnávacej pamäte

2.3 Výpočet dosiahnuteľnosti vrcholov digrafu

Gutmanov REACH algoritmus [8] potrebuje mať pre každý vrchol digrafu v vypočítanú (a zadanú ako vstupný údaj) dosiahnuteľnosť vrchola $R(v)$ definovanú vzťahom 8 na strane 8. Výpočet dosiahnuteľnosti je veľmi náročný na čas, lebo na jej určenie potrebujeme vypočítať a analyzovať všetky najkratšie cesty medzi ľubovoľnými vrcholmi v digrafe.

Algoritmus na výpočet dosiahnuteľnosti pre všetky vrcholy hranovo ohodnoteného digrafu $G = (V, H, c)$ by sa dal jednoducho formulovať takto:

Krok 1. *Inicializácia.*
$\forall i \in V$: $R(i) = 0$

Krok 2. Pre každú najkratšiu $u - v$ cestu $\mu(u, v)$ a každý vrchol $x \in \mu(u, v)$ na tejto ceste polož

$$R(x) := \max\{R(x), R(x, \mu)\} = \max\{R(x), \min\{\mathrm{d}(u, x), \mathrm{d}(x, v)\}\}.$$

Na výpočet dosiahnuteľnosti budeme potrebovať určiť všetky najkratšie cesty v digrafe. Urobíme to tak, že budeme postupne pre každý vrchol $u \in V$ opakovane spúšťať *point-to-all* Dijkstrov algoritmus a tak nájdeme najkratšie cesty $\mu(u, v)$ medzi každou dvojicou vrcholov $(u, v) \in V \times V$. Dosiahnuteľnosť $R(x, \mu(u, v))$ vrchola $x \in \mu(u, v)$ na ceste $\mu(u, v)$ vypočítame zo značiek $t(v)$ a $t(x)$

$$R(x, \mu) = \min\{\mathrm{d}(u, x), \mathrm{d}(x, v)\} = \min\{t(x), t(v) - t(x)\}. \tag{11}$$

Označme $R_u(x)$ dosiahnuteľnosť vrcholu x na cestách z vrcholu u, t.j.

$$R_u(x) = \max_{v \in V,\ x \in \mu(u,v)} R(x, \mu(u, v)), \text{ kde } \mu(u, v) \text{ je najkratšia } u - v \text{ cesta.} \tag{12}$$

Nech vrchol $x \in V$ leží na dvoch najkratších cestách $\nu(u, v)$ a $\xi(u, w)$ z vrcholu u. Potom prvá časť oboch ciest je rovnaká a jej dlžku môžme označiť $d = \mathrm{d}(u, x)$. Dosiahnuteľnosť vrcholu x na ceste ν je

$$R(x, \nu) = \min\{\mathrm{d}(u, x), \mathrm{d}(x, v)\} = \min\{d, \mathrm{d}(u, v) - d\} \tag{13}$$

a podobne

$$R(x, \xi) = \min\{\mathrm{d}(u, x), \mathrm{d}(x, w)\} = \min\{d, \mathrm{d}(u, w) - d\}. \tag{14}$$

Ak by platilo, že $\mathrm{d}(u, v) \geq \mathrm{d}(u, w)$, potom aj $R(x, \nu) \geq R(x, \xi)$ a $R_u(x) \geq R(x, \xi)$. To znamená, že ak budeme najkratšie cesty z vrcholu u do vrcholov $v_i \in V$ analyzovať v poradí od najdlhšej po najkratšiu (resp. vrcholov v_i v poradí od najvzdialenejšieho po najbližší k vrcholu u), značku $R_u(x)$ nastavíme pri prvom výskyte vrcholu x na nejakej $u - v_i$ ceste a už ju nebudeme meniť. Ak pri spracovaní nejakej cesty narazíme na vrchol x, ktorý už má nenulovú značku $R_u(x)$, značku už budú mať aj všetky vrcholy na $u - x$ ceste a teda analýzu cesty môžem ukončiť.

Vhodné usporiadanie vrcholov v_i sa dá získať počas behu Dijkstrovho algoritmu. Stačí si zapamätať poradie, v akom sa stávali riadiacimi vrcholmi a použiť opačné poradie - stačí ich vkladať do frontu typu LIFO.

Toto znamená, že po vyhľadaní všetkých najkratších ciest z vrcholu u pomocou Label Set implementácie so zložitosťou $O(n \cdot \log n)$, určenie značiek $R_u(x)$ pre všetky vrcholy už bude mať zložitosť len $O(n)$. Vdaka tomuto sa nám podarilo znížiť zložitosť celého výpočtu dosiahnuteľnosti pre všetky vrcholy digrafu na $O(n^2 \cdot \log n)$.

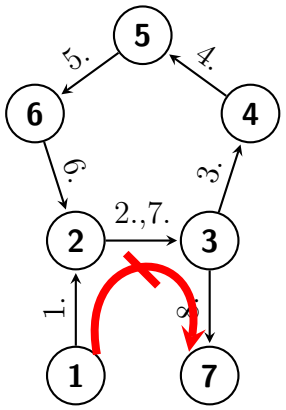
Algoritmus s vyššie uvedenými urýchleniami som opäť implementoval ako paralelný výpočet a otestoval najskôr na mojom PC s 2x2 jadrovým procesorom Intel(R) i3. Výpočet dosiahnuteľnosti pre vrcholy digrafu slovenskej cestnej siete trval približne 91 minút. Ten istý program som potom spustil na HPC Klastri Žilinskej univerzity, pričom výpočet prebiehal na 10–tich nodoch, t.j. v 120 paralelných procesoch. Výpočet trval presne podľa očakávaní 3 minúty a 20 sekúnd.

2.4 Hľadanie najkratšieho prípustného sledu

Diagram digrafu s definovaným zakázaným manévrom pozostávajúcim z troch hrán, kedy prípustný sled nie je ťahom, uvádzam na obrázku Obr. 4.

Keďže vo všeobecnom slede sa môže ktorýkoľvek vrchol alebo hrana opakovať ľubovoľný počet krát, pri reprezentácii sledu v pamäti počítača si už nevystačíme s jednoduchými značkami x ako to bolo pri ceste alebo ťahu. Riešením je pre každý vrchol mať viac značiek, rovnako ako v Palúchovom algoritme na hľadanie K–najkratších ciest [12].

Modifikácia tohto algoritmu tak, aby namiesto najkratšej cesty hľadal sled, je veľmi jednoduchá. Stačí vynechať podmienku, ktorá zakazuje duplicitné vloženie vrcholu do cesty. Všetky ostatné štruktúry algoritmu nie je potrebné meniť. A naopak, keďže hľadáme prípustný sled, ktorý má rešpektovať zakázané manévre, je potrebné zakázať vloženie vrcholu



$$
\begin{aligned}
G &= (V, H) \\
V &= \{1,2,3,4,5,6,7\} \\
H &= \{(1,2),(2,3),(3,4),(3,7),(4,5),(5,6),(6,2)\} \\
Z &= \{(1,(1,2),2,(2,3),3,(3,7),7)\} \\
\mu &= (1,(1,2),2,(2,3),3,(3,4),4,(4,5),5,(5,6),6, \\
&\quad (6,2),2,(2,3),3,(3,7),7)
\end{aligned}
$$

Obr. 4: Príklad najkratšieho prípustného sledu, ktorý nie je ťahom

do sledu, ak by sa tým v slede vytvoril podsled so zakázaným manévrom. Táto kontrola je pomerne jednoduchá. Pre každý vrchol si budeme udržiavať zoznam zakázaných manévrov, ktoré v tomto vrchole končia. Pri vkladaní vrcholu prejdeme všetky tieto zakázané manévre a vloženie vrcholu povolíme len vtedy, ak nenájdeme zhodu medzi nejakým zakázaným manévrom a koncom vytváraného sledu. Problém je, že dopredu nevieme určiť, koľko značiek pre každý vrchol budeme potrebovať. Ak by sme ich vytvárali príliš málo, nemuseli by sme prípustný sled nájsť. Ak by sme ich vytvárali veľa, algoritmus nám bude vytvárať krátke sledy v blízkosti počiatočného vrcholu u a nájdenie prípustného sledu do vrcholu v by trvalo príliš dlho a potrebovali by sme veľa pamäte na všetky tie značky.

Definícia 10 Nech $\mu = (v_1, (v_1, v_2), v_2, \ldots, (v_{k-1}, v_k), v_k)$ je sled v digrafe G. Potom všetky vrcholy v_i pre $2 <= i <= k-1$ nazveme *vnútorné vrcholy sledu* μ. Podobne všetky orientované hrany (v_i, v_{i+1}) pre $2 <= i < k-1$ nazveme *vnútorné hrany sledu* μ.

Poznámka Aj počiatočný vrchol sledu môže byť vnútorný, ak sa v slede opakuje. Napríklad v slede (1,(1,2),2,(2,3),3,(3,1),1,(1,4),4) by bola množina vnútorných vrcholov $\{1,2,3\}$.

Veta 3 Nech $G = (V, H, c)$ je hranovo ohodnotený digraf bez cyklov zápornej dĺžky. Nech $\mu = (u = x_1, (x_1, x_2), \ldots, (x_{k-1}, x_k), x_k = v)$ je prípustný sled z vrchola u do vrchola v s dĺžkou $\mathrm{d}(\mu)$.

Potom existuje taký prípustný $u-v$ sled ω s rovnakou alebo menšou dĺžkou $\mathrm{d}(\omega) \leq \mathrm{d}(\mu)$, v ktorom sa okrem vnútorných vrcholov zakázaných manévrov neopakuje žiadny iný vrchol $v \in V$.

Dôkaz. Ak sa v prípustnom slede μ z vrcholu u do vrcholu v neopakuje vrchol, ktorý nie je vnútorným vrcholom zakázaného manévru, sled μ je zároveň hľadaným prípustným sledom ω.

Nech sa v slede μ opakuje vrchol $x \in V$, ktorý nie je vnútorným vrcholom žiadneho zakázaného menévru. Potom nech x_i je prvý výskyt vrcholu x a x_j je posledný výskyt vrcholu x v slede μ.

$$\mu = (u = x_1, \ldots, (x_{i-1}, x_i), \underbrace{x_i, (x_i, x_{i+1}), \ldots, x_j}_{x}, (x_j, x_{j+1}), \ldots, x_k = v)$$

Pre sled $\mu_2 = (u = x_1, \ldots, (x_{i-1}, x_i), x, (x_j, x_{j+1}), \ldots, x_k = v)$, ktorý vznikne zo sledu μ vynechaním časti medzi vrcholmi x_i a x_j platí $\mathrm{d}(\mu_2) \leq \mathrm{d}(\mu)$, lebo digraf G neobsahuje cyklus zápornej dĺžky.

Keďže sled μ bol prípustný a vrchol x nie je vnútorný vrchol zakázaného menévru, potom aj sled μ_2 je prípustný, lebo všetky jeho netriviálne podsledy s aspoň dvomi hranami sú buď zároveň podsledmi prípustného sledu μ v tvare $(x_l, (x_l, x_{l+1}), \ldots, x_{m-1}, (x_{m-1}, x_m), x_m)$ pre $1 \leq l < m \leq i$ alebo $j \leq l < m \leq k$ alebo obsahujú vrchol x ako svoj vnútorný vrchol, ktorý podľa predpokladu nie je vnútorným vrcholom žiadneho zakázaného manévru.

Sled μ_2 je prípustný $u - v$ sled v digrafe G a pre jeho dĺžku platí $\mathrm{d}(\mu_2) \leq \mathrm{d}(\mu)$. Tento postup môžeme opakovať dovtedy, pokiaľ sa v slede μ_n bude opakovať nejaký vrchol, ktorý nie je vnútorným vrcholom zakázaného manévru. Takto získame prípustný sled ω, pre ktorý platí $\mathrm{d}(\omega) = \mathrm{d}(\mu_n) \leq \cdots \leq \mathrm{d}(\mu_2) \leq \mathrm{d}(\mu)$. □

Pri hľadaní prípustného sledu sa mi teda stačí obmedziť len na konštruovanie takých sledov, v ktorých sa opakujú len vnútorné vrcholy nejakého zakázaného manévru. Pre všetky ostatné vrcholy mi stačí počas výpočtu vytvoriť len jednu značku. Výsledný algoritmus na hľadanie prípustného sledu v digrafe potom bude vyzerať takto:

Algoritmus 2 *Multi label algoritmus* na nájdenie prípustného sledu v hranovo ohodnotenom digrafe $G = (V, H, c)$ s nezáporným ohodnotením hrán.

Pre každý vrchol $i \in V$ budeme evidovať jednu alebo viac definitívnych značiek $\mathcal{L}_i$. Každá značka bude usporiadaná štvorica (k, t, x, x_k), kde

k	je poradové číslo u–i sledu
t	je dĺžka tejto u–i sledu
x	je predposledný vrchol na tomto u–i slede
x_k	je poradové číslo sledu do predposledného vrcholu x

Do prioritného frontu $\mathcal{E}$ budeme vkladať usporiadané štvorice (w, t, x, x_k), kde

w	je kandidát na riadiaci vrchol
t	je dĺžka u–w sledu, zároveň je to aj priorita prvku frontu
x	je predposledný vrchol na tomto u–w slede
x_k	je poradové číslo sledu do predposledného vrcholu x

Krok 1. *Inicializácia.*

Polož $\mathcal{L}_u = \{(1,0,0,0)\}$ a $\mathcal{L}_i = \emptyset$ pre $\forall i \in V, i \neq u$.
Polož $\mathcal{E} := \emptyset$.
Pre všetky vrcholy $i \in V^+(u)$ polož $\mathcal{E} := \mathcal{E} \cup \{(i, c(u,i), u, 1)\}$.

Krok 2. Z frontu $\mathcal{E}$ vyber štvoricu $(w_{\min}, t_{\min}, x_{\min}, x_{\mathrm{k}\min})$ s minimálnou značkou $t_{\min}$.
Ak je množina značiek $\mathcal{L}_{w_{\min}}$ prázdna alebo ak je vrchol $w_{\min}$ vnútorným vrcholom nejakého zakázaného menévru, potom:

- $k = |\mathcal{L}_{w_{\min}}| + 1$
- $\mathcal{L}_{w_{\min}} := \mathcal{L}_{w_{\min}} \cup \{(k, t_{\min}, x_{\min}, x_{\mathrm{k}\min})\}$
- Pre $\forall i \in V^+(w_{\min})$ skontroluj, či vo vrchole i končí nejaký zakázaný manéver. Ak áno tak skontroluj, či nejaký zakázaný manéver nebude podsledom nájdeného sledu do vrcholu i. Ak vo vrchole i nekončí žiadny zakázaný manéver alebo žiadny takýto zakázaný manéver nebude podsledom nájdeného sledu do vrcholu i, potom $\mathcal{E} := \mathcal{E} \cup \{(i, t_{\min} + c(w_{\min}, i), w_{\min}, k)\}$.

Krok 3. Ak $\mathcal{E} \neq \emptyset$ a $\mathcal{L}_v = \{\}$, GOTO Krok 2.
Inak STOP. Ak $\mathcal{L}_v = \{\}$, potom prípustný $u - v$ sled neexistuje. Inak pomocou značky v $\mathcal{L}_v$ môžeme skonštruovať najkratší prípustný u–v sled $\mu(u,v)$:

$$\mu(u,v) = (u \ = w_1, (w_1, w_2), w_2, \ldots, w_{s-1}, (w_{s-1}, w_s), w_s = v).$$

Označme $\mathcal{L}_i[j]$ značku pri vrchole i, ktorej prvá zložka $k = j$. Teda ak $(k, t, x, x_{\mathrm{k}}) \in \mathcal{L}_i$, potom $\mathcal{L}_i[k] = (k, t, x, x_{\mathrm{k}})$. Zaveďme tiež označenie $\mathcal{L}_i[j]^{()}$ pre jednotlivé zložky značky, teda $\mathcal{L}_i[k]^{(t)} = t$, $\mathcal{L}_i[k]^{(x)} = x$ a $\mathcal{L}_i[k]^{(x_{\mathrm{k}})} = x_{\mathrm{k}}$.

Potom

$$\begin{aligned} w_s &= v, & k_s &= 1 \\ w_{s-1} &= \mathcal{L}_{w_s}[k_s]^{(x)}, & k_{s-1} &= \mathcal{L}_{w_s}[k_s]^{(x_{\mathrm{k}})} \\ w_{s-2} &= \mathcal{L}_{w_{s-1}}[k_{s-1}]^{(x)}, & k_{s-2} &= \mathcal{L}_{w_{s-1}}[k_{s-1}]^{(x_{\mathrm{k}})} \\ &\vdots & &\vdots \\ w_1 &= \mathcal{L}_{w_2}[k_2]^{(x)} = u \end{aligned}$$

Algoritmus som implementoval v programovacom jazyku `C#` a testoval na digrafe slovenskej cestnej siete. Trafiť vrcholy u a v tak, aby sa pri hľadaní najkratšieho $u-v$ prípustného sledu prejavil nejaký zakázaný manéver bolo veľmi náročné. Preto som algoritmus otestoval tak, že som ho nechal vypočítať a zobraziť najkratší prípustný sled, potom som časť tohto sledu zadefinoval ako zakázaný manéver a výpočet opakovane spustil.

3 Záver

V dizertačnej práci sa mi podarilo prehľadným spôsobom popísať známe publikované algoritmy na hľadanie optimálnej cesty v orientovanom grafe. Väčšinu týchto algoritmov som implementoval v programovacom jazyku `C#` a následne aj v jazyku `C`. Implementáciu algoritmov som experimentálne overil na digrafe slovenskej cestnej siete, pričom som porovnal a vyhodnotil rýchlosť jednosmernej, obojsmernej sériovej a obojsmernej paralelnej implementácie vybraných algoritmov. Výsledkom tejto časti mojej práce je, že najrýchlejším algoritmom na nájdenie najkratšej cesty v orientovanom hranovo ohodnotenom grafe je z pomedzi mnou implementovaných algoritmov jednosmerná kombinácia algoritmov Reach a A^*. Pomerne prekvapivo bola jedmosmerná sériová implementácia rýchlejšia ako paralelný výpočet toho istého obojsmerného algoritmu vo viacerých vláknach.

Tiež som navrhol vlastnú metódu na výpočtet dolného odhadu vzdialenosti dvoch bodov na povrchu Zeme z ich GPS súradníc. Použitím tejto metódy sa mi podarilo zrýchliť niektoré známe algoritmy a dá sa predpokladať, že v mobilných zariadeniach, ktoré nemajú hardvérovú podporu výpočtov v pohyblivej desatinnej čiarke by takýto spôsob odhadu vzdialenosti bol v porovnaní s klasickým výpočtom Haversine metódou ešte efektívnejší.

Navrhol som aj niekoľko spôsobov, ako usporiadať vrcholy digrafu vo vstupných údajoch pred samotným výpočtom tak, aby som znížil počet operácii čítania blokov z externej pamäte na zariadeniach s malou kapacitou operačnej pamäte. Ukázal som, že usporiadaním vrcholov v digrafe – modeli cestnej siete – podľa známych GPS súradníc je objem dát čítaných z externej pamäte podstatne nižší ako keby sa na začiatku výpočtu načítal do pamäte celý digraf. Ak by GPS súradnice o vrcholoch neboli k dispozícii, ako vhodná alternatíva sa ukázalo usporiadanie vrcholov podľa ich vzdielenosti (dĺžky najkratšej cesty) od pevne zvoleného vrcholu s veľkou excentricitou.

Úspešne som sa zaoberal aj návrhom algoritmu na výpočet dosiahnuteľnosti vrcholov pre potreby REACH algoritmu. Objavil a dokázal som niekoľko vlastností funkcie dosiahnuteľnosti, vďaka ktorým sa mi podarilo rádovo urýchliť výpočet dosiahnuteľnosti pre všetky vrcholy digrafu. Moja implementácia tohto algoritmu paralelizovaná pomocou `MPI` knižnice dokázala na HPC klusteri v 120 paralelných procesoch vypočítať dosiahnuteľnosť vrcholov pre slovenskú cestnú sieť s vyše 200 000 vrcholmi a 500 000 hranami za 3 minúty a 20 sekúnd a existuje reálny predpoklad, že pre cestnú sieť Nemecka s vyše 4 000 000 vrcholov by výpočet trval len o niečo dlhšie ako jeden deň.

Za svoj najväčší prínos považujem formuláciu exaktného a pritom rýchleho algoritmu na nájdenie najkratšieho prípustného sledu v dopravnej sieti s rešpektovaním zakázaných manévrov. Algoritmus som navrhol ako vlastnú modifikáciu Palúchovho algoritmu na nájdenie K-najkratších ciest. Pri návrhu algoritmu som použil mnou formulovanú a dokázanú vetu, pomocou ktorej sa mi podarilo podstatným spôsobom obmedziť počet vytvorených značiek vrcholov. Samozrejme som aj tento algoritmus implementoval a jeho funkčnosť experimentálne overil na slovenskej cestnej sieti.

Summary

This thesis deals with the shortest path problems in extensive transport network. One of the objectives of this work is to implement, compare and summarize well known shortest path algorithms and their modifications. The main focus is to design a fast algorithm for finding the feasible shortest trail in a directed graph. Feasible trail is such a route in transport network which respects the additional terms in the modeled reality – prohibited maneuvers (e.g. left-turn, U-turn) in the transport network.

A subset of known shortest path algorithms were implemented in the programming language `C#` and consequently the programming language `C`. These implementations have been verified on the digraph of the Slovak road network. The speed of unidirectional, bidirectional serial and bidirectional parallel implementation of selected algorithms have been compared and evaluated. The fastest algorithm was the unidirectional combination of algorithms Reach and A^*. Surprisingly the unidirectional serial implementation of Reach algoritm was faster than the multiple threads parallel implementation of the same bidirectional algorithm. I designed the new method of calculating lower bound of distance of two points on the earth's surface from their GPS coordinates. By using this method some known algorithms became faster. I suppose this method of distance estimating in the mobile devices without hardware support of floating point calculations is more efficient than the traditional Haversine method.

There are designed several ways how to organize digraph vertices in the input data before the calculation process so that the number of blocks read operation needed on devices with small capacity of main memory were minimized. I have shown that the arrangement of vertices in the digraph of the model of the road network by using of the known GPS coordinates can significantly lower the number of disk I/O operations. If the GPS coordinates of the vertices are not available the vertices arrangement in the order of the distance (shortest path length) from fixed vertex is a suitable alternative.

I have successfully dealt with the algorithm to calculate the reachness of vertices in directed graph needed for REACH algorithm. I have discovered and proved several properties of the reachness function, thanks to them I have accelerated the calculation of reachness for all vertices digraph. The new designed implementation of this algorithm, parallelized by using `MPI` library and processed on HPC cluster in 120 CPUs, calculates the reachness of over 200 000 vertices of the Slovak road network in 3 minutes and 20 seconds. There is assumption that calculating of reachnes in the German transport network with over 4 000 000 vertices would take little more than one day.

The greatest contribution of the thesis is the formulation of the exact and even fast algorithm for finding the feasible shortest trail in a transport network with the respect of prohibited maneuvers. The new algorithm is a modification of the existing Paluch's multilabel algorithm for finding K-shortest paths. In the process of designing the algorithm has been used me itself formulated and proven theorem which substantially reduces the number of created labels of vertices.

Literatúra

[1] Bellman,R.: On a routing problem, Quarterly of Applied Mathematics, 16, 1958, pp. 88-90

[2] Cenek,P. – Klima,V. – Janáček,J.: Optimalizace dopravních a spojových procesu, Edičné stredisko VŠDS, 1994, ISBN 80-7100-197-X

[3] Černý,J. – Kluvánek,P.: Základy matematickej teórie dopravy, VEDA, vydavateľstvo SAV, 1991, ISBN 80-224-0099-8

[4] Dijkstra,E.W.: A note on two problems in connexion with graphs. Numerische Mathematik, 1959

[5] Goldberg,A.V.: Reach for A^*: an Efficient Point-to-Point Shortest Path Algorithm, prezentácia, dostupná na
`http://www.cs.princeton.edu/courses/archive/spr09/cos423/Lectures/reach-mit.pdf`

[6] Goldberg,A.V. – Harrelson,Ch.: Computing the Shortes Path: A^* Search Meets Graph Theory, 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '05), Vancouver, Canada, 2005

[7] Goldberg,A.V. – Werneck,R.: Computing Point-to-Point Shortest Paths from External Memory, SIAM Workshop on Algorithms Engineering and Experimentation (ALENEX '05), Vancouver, Canada, 2005

[8] Gutman,R.: Reach–based routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks. In Proc. 6th International Workshop on Algorithm Engineering and Experiments, pp. 100–111, 2004

[9] Hart,P.E – Nilsson,N.J. – Raphael,B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths, IEEE Transactions of Systems Science and Cybernetics, Vol. SSC-4, No. 2, 1968

[10] Palúch,S. – Peško,Š.: Kvantitatívne metódy v logistike, EDIS vydavateľstvo ŽU, 2006, ISBN 80-8070-636-0

[11] Palúch,S.: Algoritmická teória grafov, EDIS vydavateľstvo ŽU, v tlači

[12] Palúch,S.: A Multi Label Algorithm for k Shortest Paths Problem, príspevok zaslaný do Communications 2/2009

[13] Plesník,J.: Grafové algoritmy, VEDA, vydavateľstvo SAV, 1983

Vlastné publikácie

[14] Majer, T.: A fast algorithm for the problem of shortest trail with prohibited maneuvers, Mathematical methods in economics 2011 : proceedings of the 29th international conference : book of abstracts. - Prague : Professional Publishing, 2011. - ISBN 978-80-7431-060-7. - S. 54.

[15] Majer, T.: An algorithm for the problem of shortest trail with prohibited maneuvers, In: Journal of Information, Control and Management Systems. - ISSN 1336-1716. - Vol. 9, No. 2 (2011), s. 101-108

[16] Majer, T.: Shortest trail problem with respect to prohibited maneuvers, In: Mathematical methods in economics 2010 : proceedings of the 28th international conference : September 8-10.2010. part II. - České Budějovice : University of South Bohemia, 2010. - ISBN 978-80-7394-218-2. - S. 418-422

[17] Majer, T.: Route assignment based on k-shortests paths problem, In: Quantitative methods in economics : multiple criteria decision making XV : proceedings of the international sientific conference : 6th-8th October 2010, Smolenice, Slovakia. - Bratislava : Iura edition, 2010. - ISBN 978-80-8078-364-8. - S. 117-126

[18] Majer, T.: Pažravá metóda na návrh liniek MHD, Dopravná infraštruktúra v mestách, 7. medzinárodná konferencia, Žilina 2010, ISBN 978-80-554-0260-4

[19] Majer, T.: A multi labelalgorithm for shortest trail problem, In: Diskretnaja matematika, algebra i ich priloženija : meždunarodnaja naučnaja konferencija : 19-22 oktjabria 2009 g., Minsk. - Minsk : Institut matematiki NAN Belarusi, 2009. - ISBN 985-6499-61-5. - S. 151-153

[20] Majer, T.: Simulačný model zaťaženia spojov MHD, Dopravná infraštruktúra v mestách, 6. medzinárodná konferencia, Žilina 2008, ISBN 978-80-8070-913-6

[21] Majer, T.: Kurz kryptografie s použitím otvoreného softvéru, APLIMAT 2008: Part IV, Open Source Software in Research and Education, ISBN 978-80-89313-04-4.

[22] Majer, T., Palúch, S.: Odhad prvkov OD matice na základe údajov z elektronického tarifného systému, Horizonty dopravy, ročník XV, číslo č, 2007, pp. 21-23, ISSN 1210-0978.

[23] Majer, T., Palúch, S.: K optimalizácii metskej a prímestskej pravidelnej osobnej dopravy, Vysoká škola jako facilitátor rozvoje společnosti a regionu, III. medzínárodní konference, Kunovice 2007, ISBN 80-7314-107-8.

[24] Majer, T.: Nonlinearity of S-BOXes of Multiplicative Inverse of Elements in Galois Field, Journal of Information, Control and Management Systems, Volume 5, No. 1, 2007, pp. 61-66, ISSN 1336-1716.

[25] Majer, T.: Optimálna regulácia uzavretého regulačného obvodu, Hydraulika a pneumatika, ročník VIII, číslo 4, 2006, pp. 20-21, ISSN 1335-5171

[26] Majer, T.: Passenger – Trip Assignment Model, Journal of Information, Control and Management Systems, Volume 4, No. 1, 2006, pp. 19-22, ISSN 1336-1716.

[27] Majer, T.: Cryptographic Properties of Generated S-Boxes, Journal of Information, Control and Management Systems, Volume 3, No. 2, 2005, pp. 139-144, ISSN 1336-1716.

[28] Majer, T.: Another Modification of Block Cipher IDEA, Journal of Electrical Engineering Vol. 54 No. 12/s, FEI STU Bratislava 2003, pp. 59-60, ISSN 1335-3632

[29] Majer, T.: Modification of Block Cipher IDEA, Journal of Electrical Engineering Vol. 53 No. 12/s, FEI STU Bratislava 2002, ISSN 1335-3632