Hledání nejlepší cesty v reálných silničních sítích

Jan Rygl

208072@muni.cz

28.10.2007

1 Úvod

1.1 Cíl práce

Hlavním cílem práce je popsat současné výsledky v oblasti algoritmů hledajících nejkratší cestu na reálných silničních sítích. Budou zde popsány výhody a nevýhody daných přístupů a jejich vhodnost na daných sítích.

1.2 Použité definice

Silniční síť – pro účely této práce nechť je síť reprezentována neorientovaným grafem $G(V, E)$, kde V je množina vrcholů a E množina hran. Navíc pro každou hranu $\varepsilon \in E$: $\omega(\varepsilon) > 0$, kde ω je funkce přiřazující každé hraně její ohodnocení.

Strom nejkratších cest (a shortest path tree) je strom, jehož kořenový uzel je startovní pozice a vzdálenost do všech ostatních uzlů je minimální (např. Dijkstrův algoritmus generuje strom nejkratších cest).

2 Problémy

2.1 Testování algoritmů

Přestože problematika algoritmů hledajících nejkratší cestu je zkoumána již delší dobu, většina výsledků se opírá o testováni na uměle vytvořených datech nebo o použití speciálních datových struktur jako je čtvercová síť.

To přináší řadu nevýhod, neboť:

1. Realistická síť má typicky velmi malé průměrné větvení (zdroje uvádí hodnotu 2.6), zatímco uměle vygenerovaná síť dosahuje průměrného větvení až 10.

2. Realistické sítě nejsou dostatečně jednolité. Přesněji, v síti lze nalézt několik diametrálně odlišných podoblastí. Málo početnou, avšak pro výpočty důležitou skupinou, jsou města a předměstí, pro něž je typická velmi hustá dopravní síť. Po celé síti jsou pak rozmístěny malé oblasti značící menší města a vesnice. Všechny oblasti jsou spojeny velmi řídkou silniční sítí.

3. Uměle generované sítě přidělují ohodnocení hran (např. vzdálenost měst) jednotným algoritmem, který zpravidla generuje náhodné číslo z daného intervalu. Ve skutečném světě se však setkáváme v některých oblastech s extrémními příklady, kdy mají všechny hrany nízké ohodnocení (krátké ulice ve městech), či naopak relativně vysoké (dálnice, silnice první třídy).

Z těchto důvodů bude nadále kladen důraz na algoritmy, které byly otestovány na skutečných mapách.

2.2 Znovupoužitelnost výsledků

Při použití vyhledávacích algoritmů v praxi může docházet k opakování dotazů. Může se tedy zdát, že je výhodné uchovávat si dotazy v paměti, aby se urychlilo vyhodnocování. K tomu účelu by dobře sloužila struktura strom nejkratších cest, neboť většina lidí cestuje především mezi významnými místy (např. velkými městy, hranicemi a podobně) a tudíž ve většině případů patří startovní pozice do omezené množiny míst.

Problém je však následující. Představme si, že 10 000 lidí bude chtít vyjet za dané dopravní situace z Prahy do Vídně. Jelikož však existují desítky městských čtvrtí a v každé z nich několik hlavních tahů, z kterých lze vyjet, pro každé místo bychom museli vytvořit nový strom nejkratších cest. Takových míst může být kolem 1000 a tudíž by daný strom využilo jen průměrně 10 lidí. Kdybychom chtěli tento problém obejít vytvořením stromu až od určité vzdálenosti (např. až po všech silničních přípojkách z Prahy na dálnici), nevyhnuli bychom se ruční optimalizaci pro každou oblast, nelze tedy mluvit o obecném algoritmu.

Použití stromů nejkratších cest může být nevýhodné ještě z těchto důvodů:

1. Dynamičnost prostředí – jelikož každý uživatel potřebuje vyjet v jiný čas, může se stát, že cesta propustná v čase t bude již v čase $t + h$ neprůjezdná. Tudíž by se pro každou změnu prostředí musel generovat nový strom.

2. Předpočítaná data omezují efektivitu použití různých algoritmů na určité síti. Nemá smysl hledat cestu rychlou heuristikou, když už máme spočítanou přesnou cestu, i když spotřebovala mnohem více času.

3. Silniční síť může být natolik rozsáhlá, že je neúnosné udržovat informace o stromech nejkratších cest v paměti.

3 Vybrané algoritmy

3.1 Kritéria výběru

Algoritmus shledaný jako optimální v jedné situaci se takto nemusí jevit za jiných podmínek. Proto je dobré vždy stanovit několik kritérií, podle kterých algoritmus hodnotíme:

1. Časová složitost.

2. Paměťová složitost.

3. Optimálnost řešení.

4. Snadnost implementace.

3.2 Dijkstrův algoritmus

Dijkstrův algoritmus je algoritmus sloužící k nalezení nejkratší cesty v grafu. Je konečný (pro jakýkoliv konečný vstup algoritmus skončí), protože v každém průchodu cyklu se do množiny navštívených uzlů přidá právě jeden uzel, průchodů cyklem je tedy nejvýše tolik, kolik má graf vrcholů.

Popis algoritmu:

Mějme graf G, v němž hledáme nejkratší cestu. Řekněme, že V je množina všech vrcholů grafu G a množina E obsahuje všechny hrany grafu G. Algoritmus pracuje tak, že si pro každý vrchol v z V pamatuje délku nejkratší cesty, kterou se k němu dá dostat. Označme tuto hodnotu jako $d[v]$. Na začátku mají všechny vrcholy v hodnotu $d[v] = \infty$, kromě počátečního vrcholu s, který má $d[s] = 0$. Nekonečno symbolizuje, že neznáme cestu k vrcholu.

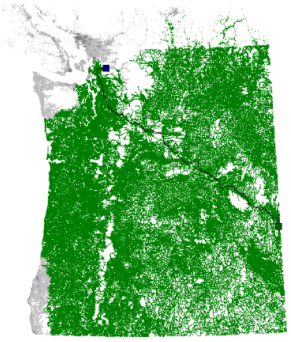
Dále si algoritmus udržuje množiny Z a N, kde Z obsahuje už navštívené vrcholy a N dosud nenavštívené. Algoritmus pracuje v cyklu tak dlouho, dokud N není prázdná. V každém průchodu cyklu se přidá jeden vrchol $vmin$ z N do Z, a to takový, který má nejmenší hodnotu $d[v]$ ze všech vrcholů v z N.

Varianty algoritmu v závislosti na použité datové struktuře (převzato z [1]), pro srozumitelnost ponechány v angličtině:

Název	Časová složitost *	Vznik
DIKQ Naive implementation	$O(n^2)$	Dijkstra (1959)
Using buckets structure		
DIKB Basic implementation	$O(m + nC)$	Dial (1969)
DIKBM With overflow bag	$O(m + n(C/a + a))$	Cherkassky et al. (1993)
DIKBA Approximate buckets	$O(mb + n(b + C/b))$	
DIKBD Double buckets	$O(m + n(b + C/b))$	
Using heap structure		
DIKF Fibonacci heap	$O(m + nlog(n))$	Fredman and Tarjan (1987)
DIKH k-array heap	$O(mlog(n))$	Corman et al. (1990)
DIKR R-heap	$O(m + nlog(C))$	Ahuja et al. (1990)

** n, počet síťových uzlů; m, počet hran; C, maximální váha hrany; a,b vstupní parametry.*

Hlavní nevýhoda algoritmu plyne z nutnosti prohledávat všemi směry, viz. obrázek níže.



(zdroj [4])

3.3 Obousměrný Dijkstrův algoritmus (Bidirectional Dijkstra's Algorithm)

Obousměrný dijkstrův algoritmus modifikovuje Dijkstrův algoritmus. Vlastní vyhledávání se skládá ze dvou fází. Během první fáze střídáme dvě jednosměrná prohledávání. První je vedeno ze startovní pozice s a druhé pak z cílové pozice d. Obě vytváří strom nejkratších cest z bodu s, respektive d, který nazveme strom S, respektive strom D. Algoritmus je ukončen ve chvíli, kdy je průnik stromů D a S neprázdný. Nalezená cesta je pak kombinace cest $(s - \{D \cap S\}) \cup (\{D \cap S\} - d)$.

3.4 Upravený A* algoritmus

Jedná se o klasický algoritmus A* s modifikovanou heuristickou funkcí:

V A* používáme přípustnou heuristickou funkci danou vztahem $(n) = g(n) + h(n)$, tedy ohodnocení uzlu n odpovídá vzdálenosti uzlu od startovní pozice $g(n)$ plus euklidovské vzdálenosti daného uzlu od cílové pozice $h(n)$. Tato heuristika je přípustná, neboť vrací vzdálenost vždy menší či rovnu skutečné vzdálenosti.

Avšak když zvolíme konstantu k a jí danou heuristickou funkci vynásobíme

$$(n) = k * [g(n) + h(n)],$$

můžeme sice ztratit optimální řešení, ale během měření na reálných silničních sítích se ukázalo, že vhodně zvolená konstanta dokáže algoritmus urychlit a ušetřit paměťové nároky s minimálním vlivem na efektivitu řešení.

Další možnou úpravou je obousměrnost. Ve většině meření na reálných sítích se zjistilo, že obousměrné algoritmy dávají lepší výsledky (dosahuje se rychleji cíle, respektive prohledává se méně "zbytečných" uzlů).

4 Postupy pro zefektivnění hledání

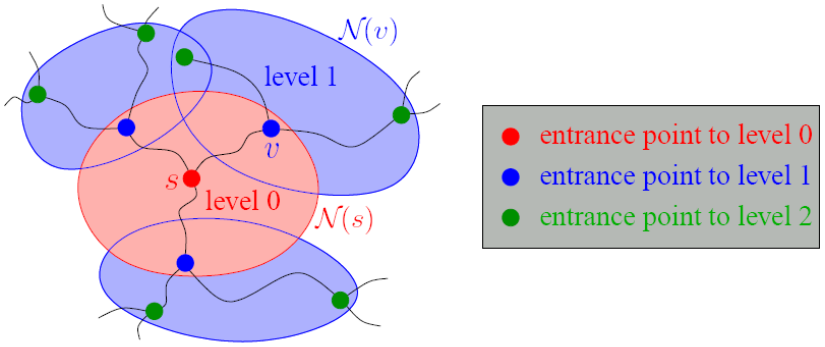
4.1 Silniční hierarchie (Highway hiearchy)

Základní myšlenkou algoritmu silniční hierarchie je, že mimo lokální oblasti budeme brát v potaz pouze nejvýznamnější hrany sítě. Lokálními oblastmi zde rozumíme množinu uzlů sítě v určitém okolí od startovního a cílového uzlu. Formálně pro každý uzel v grafu G definujeme množinu V, která obsahuje všechny uzly v jistém okolí v.

Pro větší efektivitu pak hiearchii dělíme na několik úrovní. Nejnižší hierarchií, úrovní G_0, myslíme původní graf G.

Přestup na vyšší hiearchii znamená, že zvolíme určité oblasti T_i takové, že $T_i \subseteq G_n, i \in \mathbf{N}$ a mezi každou oblastí T_i najdeme nejkratší cestu $t_{ij}, i, j \in \mathbf{N}$. Poté každou cestu k takovou, že $(k \notin T_i) \wedge (k \neq t_{ij})$ vypustíme a získáme graf $G_{n-1} \subseteq G_n$.

Implementaci zajístíme nejefektivněji použitím obousměrného Dijkstrova algoritmu. Algoritmus spustíme s nastavenou hiearchií G_0 pro oba směry. Ve chvíli, kdy zpracováváme vrchol, který neleží v sousedství startovní pozice, přesuneme se o jednu hiearchii výše ($G_n \Longrightarrow G_{n+1}$) pro daný směr. Pro usnadnění algoritmu tedy definujeme pro každou oblast množinu vrcholů, jenž znamenají přechod o úroveň výše (viz. obrázek níže, zdroj [3]).



Největší nevýhodou tohoto algoritmu jsou obrovské nároky na předpočítání nejkratších cest mezi danými hierarchiemi.

Druhým problémem pak může být fakt, že ve chvíli, kdy se algoritmus spuštěný ze startovní pozice setká s algoritmem z cílové pozice, nemůžeme algoritmus okamžitě ukončit, neboť není zajištěna efektivita řešení. Pokud bychom chtěli zaručit efektivitu řešení, byli bychom nuceni použít celou sadu výpočetně náročných kritérií na ukončení algoritmu a znehodnotili bychom rychlost hledání. Možným zefektivněním je zaručení, že oba směry Dijkstrova algoritmu vždy prohledávají v danou chvíli ve stejné hieararchii. Tj. nestane se nám, že když hledáme například cesty z Prahy do Veverské Bitýšky, směr Praha VB prohledává ve vysoké hieararchii (pro ilustraci na úrovni dálnic), zatímco směr VB Praha zatím prochází lokální silnice. Na obrázku níže by pak okamžité ukončení navrátilo cestu, kdy autem dojedeme po dálnici z Prahy až do centra Brna a z něho pak zpět do VB. Kdyby prohledávání v obou směrech mělo stejnou hieararchii, je větší pravděpodobnost, že se oba směry potkají na dálnici a není nutné se vracet.

4.2 ALT, použití orientačních bodů

Zkratka ALT vychází se z použití **A*** algoritmu, orientačních bodů (**L**andmarks) a trojúhelníkové nerovnosti (**T**riangle inequality). Použijeme tedy A* algoritmus s modifikovanou heuristickou funkcí.

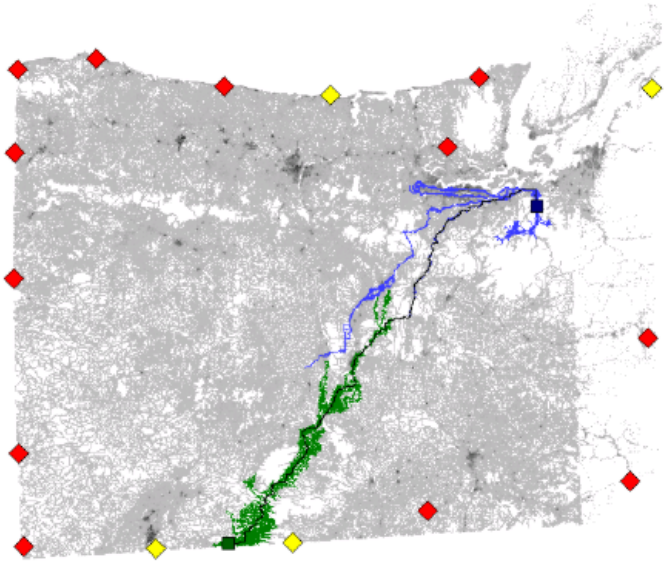
ALT používá malé množství uzlů l (orientačních bodů) takových, že $l \in L, L \subset V$, kde $G = (V, E)$. Pro každý vrchol $v \in V$ si pamatujeme každou vzdálenost $d(v, l), l \in L$, tj. známe vzdálenost z každého vrcholu do každého orientačního bodu. Trojuhelníková nerovnost je nutná pro zajištění korektní heuristické funkce, musí tedy platit, že:

$$Pro\ \forall u, v \in V, l \in L : (d(u, l) \leq d(u, v) + d(v, l)) \wedge (d(l, v) \leq d(u, l) + d(u, v))$$

Z toho plyne, že vzdálenost $d(u, v), u, v \in V$ je větší než $|d(v, l) - d(u, l)|$, a proto můžeme za

heuristické ohodnocení považovat funkci $h(s,t) = max_{l \in L}(|d(s,l) - d(t,l)|)$, kde s je prohledávaný uzel a t je cílová pozice.

Kvalita A* s takto definovou heuristikou velmi závisí na dobrém umístění orientačních značek. V optimálním případě může algoritmus fungovat jako na obrázku níže (zdroj [4]):



Zajímavostí je, že na určitých datech algoritmus dosahuje až 30-násobného urychlení oproti Dijkstrově algoritmu.

5 Závěr

K dispozici je několik nezávislých měření efektivity jednotlivých algoritmů. Jelikož každé měření probíhalo na jiných mapách a jiných měřítkách, jejich výsledky se nepatrně liší. Přesto lze dospět k několika doporučením.

Pokud chceme docílit vždy nejlepší cesty (tj. minimálního ohodnocení), měli bychom použít Dijkstrův algoritmus. Na malých mapách (regionální silniční sítě) nejlépe dopadla implementace používající datovou strukturu "bucket" (Dijkstra's Approximate Buckets – DIKBA), na větších mapách (definovány tak, že se délky cest od sebe liší více než 1500 násobně) lze použít i implementaci v angličtině pojmenovanou Dijkstra's Double Buckets (DIKBD).

Za situace, kdy chceme získat řešení rychle, lze použít různé verze algoritmu A* (obousměrný, ALT, násobení heuristické funkce konstantou). Tyto algoritmy mají nevýhodu v tom, že mohou vyžadovat předpočítávání, nebo mohou v malém procentě případů vracet chybné výsledky.

Silniční hierarchie má dobré výsledky jen v kombinaci s dalšími vylepšujícími algoritmy.

Dle testů bychom neměli používat Bellman–Ford–Moorův a Dijkstrův algoritmus s naivní implementací z důvodu jejich neefektivity na reálných silničních sítích.

Reference

[1] Shortest Path Algorithms: An Evaluation using Real Road Networks
F. Benjamin Zhan, Charles E. Noon

[2] A Computational Study of Routing Algorithms for Realistic Transportation Networks
Riko Jacob, Madhav Marathe and Kai Nagel

[3] Highway Hierarchies Star*
Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner

[4] Reach for A*: an Efficient Point-to-Point Shortest Path Algorithm
Andrew V. Goldberg

[5] http://www.cs.wikipedia.org

[6] http://www.en.wikipedia.org