GEOGRAFICKÝ ROUTING

Martin Marek
Fakulta jaderná a fyzikálně inženýrská, České vysoké učení technické v Praze
marekma3@fjfi.cvut.cz

ABSTRAKT:
Příspěvek podává základní přehled o různých přístupech k vyhledávání optimálních (nejrychlejší, nejkratší, nejlevnější atd.) tras v grafech, které reprezentují geografické údaje (routing). Jsou představeny základní deterministické algoritmy (Bellmanův-Fordův, Dijkstrův) a především jejich modifikace pro speciální případy. Druhý pohled na řešení problému pak přinášejí heuristické (A*, ALT, Arc Flags) a hierarchické algoritmy (HHR, HNR, TNR, vyhledávání pomocí zkratek). Zmíněno je také využití různých algoritmů v programátorské, zejména open-source, praxi. V příspěvku je navrženo možné začlenění statistických dat a jejich následné použití při hledání optimálních cest z hlediska využití v reálném prostředí.

ABSTRACT
The article gives a basic overview on different approaches used to search for optimal (fastest, shortest, cheapest etc.) paths in graphs that represent geographic data (routing). It presents basic deterministic algorithms (Bellman-Ford, Dijkstra) and especially their special cases modifications. Another look at the problem brings heuristic (A*, ALT, Arc Flags) and hierarchical algorithms (HHR, HNR, TNR, Contraction hierarchy). Mentioned is the use of different algorithms in, especially open-source, programming practices. This paper proposes a possible inclusion of statistical data and their subsequent use in the search for optimal paths in light of utilizations in the real environment.

KLÍČOVÁ SLOVA:
Routing, navigace, Dijkstrův algoritmus, A*, ALT, Arc Flags, hierarchické metody routingu, OpenStreetMap, online routing, offline routing, statistická data

1. ÚVOD

Geoinformačních systémy (GIS) jsou složené z dílčích částí, které dohromady tvoří komplexní celek. Jako takové jsou ideálním prostředím pro aplikaci různých programovacích technik a technologií. Ať už se jedná o data (uložení mapových podkladů), zobrazovací metody (vykreslování cest) nebo třeba vyhledávání cest.
Vyhledávání cest neboli *routing* je prvkem těchto systémů, který se dotýká většiny jeho součástí. Obvykle se při routingu hledá optimální cesta podle nějakého požadavku, např. nejrychlejší, nejkratší nebo třeba nejlevnější. Základ routingu tvoří vyhledávací algoritmus. Různé systémy používají různé přístupy, podle toho, na co jsou zaměřené. Jiné požadavky jsou kladeny na vyhledávání cest v reálném čase (GPS navigace) a jiné požadavky jsou určeny pro plánování cesty (offline routing). Routing (a to nejen geografický, ale například i síťový) je pravděpodobně nejviditelnější aplikací grafů a grafových algoritmů.
Tento příspěvek je založen především na pracích D.Schultese [1] a L. Fedorové [2].

2. DETERMINISTICKÉ ALGORITMY

Základními algoritmy, které se při vyhledávání cest používají, jsou Dijkstrův algoritmus a Bellmanův-Fordův algoritmus. První jmenovaný dosahuje sice výkonnostně lepších výsledků, ale narozdíl od druhého jmenovaného ho nelze použít na grafy se záporným ohodnocením hran. V geografických aplikacích se sice předpokládá kladné ohodnocení hran,

ale například v síťovém routingu tomu tak být nemusí. Kupříkladu Bellmanův-Fordův algoritmus je použit v RIP (routing information protocol) používanému pro komunikaci mezi routery v síti a umožňuje jim reagovat na změnu topologie sítě [3].
Podle [4] je možné **Dijkstrův algoritmus** [5] chápat jako prohledávání stromu do šířky, ovšem s určitou modifikací, takzvanou relaxací cesty. Mějme orientovaný graf (silniční síť obsahuje i jednosměrné silnice) a v něm výchozí uzel *s*, ze kterého se chceme dostat do cílového uzlu *t.* Na počátku uloží algoritmus všechny uzly do prioritní fronty a nastaví jim vzdálenost na nekonečno, vyjma uzlu s, kterému nastaví hodnotu 0. Algoritmus pak pracuje v následujícím cyklu: Pomocí hladového přístupu se z fronty vybere ten, ke kterému vede nejkratší cesta (tj. ten na začátku fronty), označme ho *u*. Pro sousední uzly *v* bodu *u* se určí vzdálenost pomocí $d(v) = min(d(v), d(u) + w(u,v))$, kde $w(u,v)$ je vzdálenost z *u* do *v*. Tomuto procesu se říká *relaxace* hran. Uzel *u* se pak nazývá uzavřený. Algoritmus končí, když nezůstaly žádné neprobádané uzly, nebo když mají všechny uzly ve frontě vzdálenost rovnou nekonečnu (tj. původní graf byl nesouvislý).
Bellmanův-Fordův algoritmus je velice podobný Dijkstrovu algoritmu. Také využívá relaxaci hran, ale na rozdíl od Dijkstrova algoritmu nerelaxuje pouze hrany vedoucí z uzavíraného uzlu *u*, ale všechny hrany všech uzlů. Bellmanův-Fordův algoritmus pracuje v čase $O(m.n)$ kde *m* je počet hran a *n* počet uzlů grafu.
Dijkstrův algoritmus pak běží v čase $O(n^2)$. Tato rychlost je vztažená k použití prioritní fronty pro ukládání dosud neuzavřených uzlů. Je tedy možné tento algoritmus optimalizovat ukládáním neprobádaných uzlů do datových struktur, v kterých jsou rychlejší operace vyhledávání a vyjmutí. Jako vhodné se jeví binární haldy se složitostí $O(m.ln(n))$ nebo Fibonacciho haldy s dokonce ještě lepší složitostí $O(m + n.ln(n))$. Jejich použití má ale význam pouze při použití na řídkých grafech (s počtem hran menším než $O(n^2)$), což obvykle pro geografická data platí.
Mírného neobecného (tj. neasymptotického) zlepšení se dá dosáhnout ukončením algoritmu po nalezení nejkratší cesty do cílového uzlu *t* (čili jeho uzavřením) a tím pádem zmenšením prohledávaného prostoru. Logicky další možností, jak urychlit nalezení nejkratší cesty, je použití **symetrického Dijkstrova algoritmu**. Ten funguje úplně jako klasický, ale vyhledávání probíhá z obou stran zároveň. Z počátečního bodu *s* se jde standardním způsobem, z cílového bodu *t* se jde proti orientaci hran. Prohledávaný prostor je v tomto případě obvykle ještě menší.

3. HEURISTICKÉ ALGORITMY

I s použitím symetrického Dijkstrova algoritmu se průměrná složitost nezlepší, dosáhne se pouze zlepšení v určitých případech. Problém tkví v příliš velkém prostoru, který musí algoritmus prohledat. Výše uvedené postupy prohledávají cesty i úplně nevhodným směrem. S tímto nedostatkem se pokouší vyrovnat algoritmus zvaný **A*** (anglicky **A star**) [6]. Je založen na stejném postupu jako Dijkstrův algoritmus, ale obsahuje navíc heuristiku pro výběr dalšího vhodného uzlu nejkratší cesty. Jestliže v původním algoritmu se minimalizovala funkce $f = d(u)$ (vzdálenost do uzlu *u*) pro výběr dalšího bodu nejkratší cesty, v A* se minimalizuje funkce $f = d(u) + h(u)$, kde právě $h(u)$ je heuristická funkce nějak ohodnocující správnost postupu. Funkce $h(u)$ musí být přípustnou heuristikou, musí vést k optimálnímu řešení a nesmí nadhodnocovat vzdálenost k cíli, tj. hodnota musí být maximálně rovna nejkratší reálné vzdálenosti z bodu *u* do cílového bodu *t*. Těmto podmínkám dobře vyhovuje dvourozměrná euklidovská vzdálenost

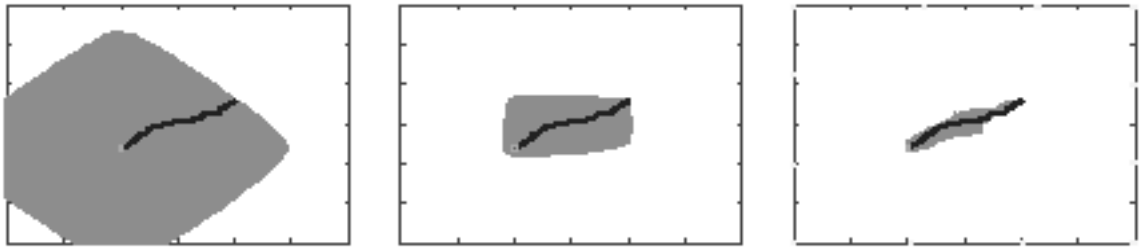
$$e(\vec{u}, \vec{t}) = \sqrt{(u_1 - t_1)^2 + (u_2 - t_2)^2}$$

která představuje vzdušnou vzdálenost mezi bodem *u* a cílem. Ačkoliv tento algoritmus neklade další nároky na paměť a vyhledává bez předchozí přípravy, je rychlejší než Dijkstrův.

Složitost samozřejmě vychází z použité heuristiky, ale v nejhorších případech nebude horší než obecné prohledávání do šířky. Nevýhodou je, že je vhodný pouze pro vyhledávání nejkratších cest. Pokud bychom potřebovali nejrychlejší cestu, museli bychom euklidovskou vzdálenost dělit nejvyšší dovolenou rychlostí všech hran (silnic) z celého grafu, což je velice skromný odhad. I zde je možné použít symetrickou verzi algoritmu, ale výkonnostní nárůst není signifikantní.

Vylepšením algoritmu A* je použití takzvaných orientačních bodů (*landmarks*) v algoritmu **ALT** (**A***, **L**andmark, **T**riangle inequality) [7] z dílny Microsoftu. Algoritmus využívá předpočítávání dat: Zvolí několik význačných bodů (označme je L) a vypočte pro ně vzdálenosti od všech možných bodů u oběma směry (od i k orientačním bodům) $d(u,L)$ a $d(L,u)$. Přitom platí trojúhelníkové nerovnosti $a = d(u,L) - d(t,L) < d(u,t)$ a $b = d(L,u) - d(L,t) < d(u,t)$. Pro vyhledávání se použije standardní algoritmus A*. Pro heu-ristiku se pak použije největší z těchto horních mezí, tj. $h(u) = max(a,b)$.

Podle [1] je pro výpočet nejkratších cest pro západní Evropu (při 16 orientačních bodech a náhodných bodech s a t) 27krát rychlejší než běžný A*. Velikost prohledávaného prostoru dobře ilustruje následující obrázek.

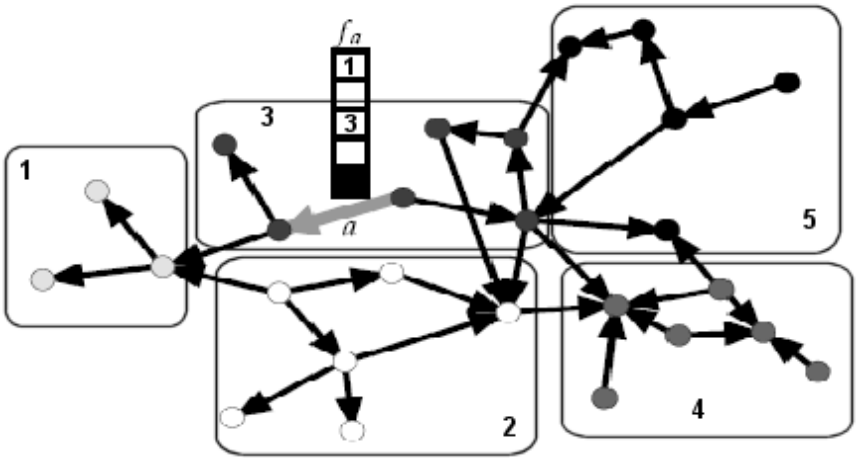


Obrázek 1: Šedivě jsou vyobrazeny uzly navštívené Dijkstrovým algoritmem (vlevo), algoritmem A (uprostřed) a algoritmem ALT (vpravo) nad stejnými vstupními daty. Černou barvou je vyznačena nejkratší cesta. Převzato z [5].*

Zásadní nevýhodou celého přístupu ALT je ohromný nárůst dat. Pro každý uzel je potřeba držet v paměti $2|L|$ dalších dat. Ačkoliv předzpracování těchto dat je jednoduché a vyhledávací nárůst signifikantní, spotřebovává algoritmus při použití na velkých grafech příliš mnoho paměti.

Tento nedostatek se pokouší vyřešit algoritmus **PCD** (Precomputed Cluster Distances) [8]. Ten místo orientačních bodů používá shluky (*clusters*) bodů a předpočítává nejkratší cesty mezi nimi. Vyhledávání pak probíhá obdobně jako v ALT s tím rozdílem, že pro výpočet horní meze pro heuristiku se použijí právě tyto vzdálenosti. Spotřeba paměti se sníží při zachování zhruba obdobné rychlosti jako je u ALT.

Jiný přístup k heuristickému vyhledávání nabízí technika zvaná **Arc Flags** [9] někdy též **Edge Flags**. Při předzpracování dat jsou uzly grafu rozděleny do regionů. U každé hrany je pak vyznačeno, zda přes ni vede nějaká nejkratší cesta do bodu z některého regionu. Při vyhledávání se použije Dijkstrův algoritmus a relaxují se pouze ty hrany, přes které vede nejkratší cesta do cílového regionu. Je samozřejmě možné použít modifikace algoritmu zmíněné výše. Zejména symetrický postup je vhodný, neboť při dosažení cílového regionu (nikoliv však cílového uzlu) by algoritmus ztrácel na výkonu. Problém ovšem nastává při předzpracování dat, neboť použitím vyhledání příznaků pro všechny páry bodů ztrácí algoritmus na smyslu. Jako vhodnější se jeví výpočet pro uzly sousedící s nějakým regionem (a příslušná úprava algoritmu). Pokročilejší metody pak jdou ještě dále a používají pouze jedno vyhledávání pro každý region [10].



Obr. 2: Graf je rozdělen do pěti regionů. Na hraně a *je pak ilustrován příznak* f_a*, značící že se přes ní dá dostat do prvního a třetího regionu. Převzato z [9].*

4. HIERARCHICKÉ METODY

Ideově na předchozí dva algoritmy (PCD a Arc Flags) navazuje celá rodina přístupů, jejichž společným znakem je hierarchické řešení problému. Pro rozsáhlé grafy se totiž nabízí řada zjednodušení, které transformují dané grafy na jiné, obvykle zjednodušené. Jakmile je pak nalezena nějaká optimální cesta v modifikovaných grafech, buď se graf postupně upravuje zpátky a cesta se upřesňuje, nebo se vezme nalezená cesta a přímo se převede do původního grafu. Na tuto problematiku existují jak deterministické, heuristické, tak i kombinované algoritmy. Většina těchto postupů je založena na algoritmech vyvinutých pro planární grafy. Silniční sítě jsou při určitém zobecnění takovýmito rovinnými grafy. Problémy můžou tvořit pouze křížení bez možnosti změny směru (tunely, nadjezdy apod.). Řešením je pak nahrazení takovýchto míst jedním uzlem, který nahradí celou oblast, a povedou do něj a z něj stejné hrany jako do této oblasti. Vyhledávání uvnitř této (většinou ne příliš velké) oblasti pak proběhne některým obvyklým způsobem po nalezení nejkratší cesty přes ni.

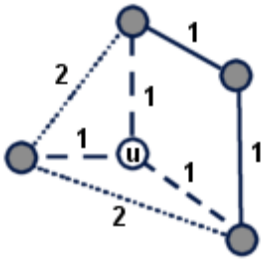
Základní algoritmus **Reach-based routing** (**RBR**) [11] vyhledává na základě dosahu, který je definován následovně $R(u) := max\{R_{st}(u) \mid s,t \in V\}$, kde $R_{st}(u) = min(d(s,u), d(u,t))$. Pomocí této funkce je vytvořena hierarchie uzlů. Pak se při průchodu některého z vyhledávacích algoritmů využívají ty cesty, které vedou přes uzly s vyšším dosahem.

Dalším základním hierarchickým algoritmem je **highway hieararchy routing** (**HHR**) [2][12]. Ten spoléhá na to, že na delší vzdálenost povede nejkratší cesta přes nějakou důležitou hranu (případně zkratku hran, viz níže). V reálném světě jsou takovými hranami například dálnice (odtud název). Algoritmus pracuje symetricky, v okolí startu a cíle se pohybuje po všech cestách, ve vzdálenějších částech (ve vyšších vrstvách hierarchie) pak upřednostňuje ty důležitější. Nevýhodou tohoto postupu ovšem je, že nemusí nalézt optimální cestu. Zároveň je potřeba nějak určit důležitost hran. Základní možností je určení podle reálných dat (silniční třídy), Schultes [1] ale navrhl řešení pro nalezení jak nejkratší cesty, tak pro automatizované ohodnocení priority hran.

Metoda **highway node routing** (**HNR**) [1][12] využívá stejné principy jako HHR, nevyznačuje však důležité hrany, ale důležité uzly (křižovatky), přes které povedou optimální trasy na delších vzdálenostech. Z nich se utvoří graf, který se pak používá při vyhledávání. Takovýchto grafů se může vyrobit více a vyhledávání probíhá na více úrovních.

Konečně **transit node routing** (**TNR**) [12][1] nejenže vytváří graf, ale zároveň přepočítává i všechna relevantní spojení mezi důležitými a ostatními uzly. Oproti HNR je pak vyhledávání rychlejší, ale za cenu mnohem vyšších paměťových nároků. Navíc i méně přizpůsobivý případným změnám v grafu.

Variantu s převzetím nalezené cesty v modifikovaném grafu využívá **vyhledávání pomocí zkratek** (**Contraction hierarchy, CH**) [12]. *Zkrácení* uzlu *u* je nahrazení posloupností všech nejkratších cest jdoucích přes *u* hranami s hodnotou rovnou součtu nahrazovaných hran. Situaci ilustruje obrázek 3 převzatý z [2]: Čárkované hrany zkracovaného uzlu *u* byly nahrazeny tečkovanými zkratkami. Při postupném zkracování je nutné ukládat přes které uzly původní cesty vedly. Základním faktorem, podle kterého se vybírá, jaké uzly se mají krátit, je rozdíl mezi počtem nových hran (zkratek) a počtem odstraněných hran (čím větší rozdíl, tím lépe). Zároveň je třeba dávat pozor na to, aby byl graf zkracován rovnoměrně. Nad takto připraveným grafem je možné použít symetrický Dijkstrův algoritmus. Heuristikou je priorita hran, daná třeba tím zda vedou do důležitých uzlů, tj. do uzlů na které jsou napojené důležité zkratky. Po nalezení cesty se graf jednoduše rozbalí a označí se správná cesta. Zajímavým důsledkem tohoto přístupu je, že zkrácený graf včetně informací o zkratkách je paměťově méně náročný než graf původní. Zároveň rychlost vyhledávání naroste oproti nejlepšímu předchozímu hierarchickému algoritmu (TNR) pětinásobně.



Obrázek 3: Zkracování uzlu u.

5. KOMBINOVANÉ METODY

Za kombinovanou metodu se dá počítat výše zmíněný algoritmus **ALT**. Na něj navazuje kombinace s metodou Reach-based search, která se nazývá **REAL**. V ní se jako orientační body používají pouze body s dobrým dosahem. Metoda **SHARC** kombinuje přístup Arc Flags se zkratkami (*shortcuts*), ta je nasazována v případech, kde není možné použít symetrické vyhledávání. Metoda **CHASE** kombinuje Contraction Hierarchy, Arc FlagS a Highway nodE routing. Po zkonstruování hierarchie pomoci HNR a vytvoření podgrafu z uzlů nejvyšší úrovně je tento podgraf rozdělen na regiony pro které jsou vypočteny příznaky (*arc flags*). Vyhledávání pak probíhá symetricky dokud algoritmus nenarazí na uzel podgrafu, pak se spustí vyhledávání přes příznaky. [12] udává, že pro silniční síť západní Evropy jsou trasy nalezeny v časech nižších než 20 μs. Nejrychlejší současnou kombinovanou metodou je spojení **TNR a Edge Flags**, které dosahuje pro západní Evropu časů kolem 1.9 μs. Paměťové nároky jsou však obrovské.

Podrobný přehled a porovnání kombinovaných metod lze najít v [1] a [12].

6. VYHLEDÁVÁNÍ V PRAXI

Je důležité rozlišovat mezi hledáním trasy při plánování dopředu (obvykle online vyhledávání, tj. s možností připojení k externím datům) a při vyhledávání v reálném čase (offline vyhledávání,tj. bez možnosti dodatečného stahování dat), které probíhá třeba při GPS navigaci. Pro první možnost není obvykle problémem předpočítat velké množství dat, které se pak využijí později. V druhém případě je nutné brát ohled na to, že zařízení pracující v reálném čase (příruční navigace, telefony) obvykle nemívají dostatečnou paměťovou kapacitu. Oproti dřívějšku už ale dosahují slušných výpočetních výkonů (nejvýkonnější v současné době prodávaná zařízení mají 1GHz procesor a kolem 500 MB RAM).

6.1 Online implementace

Hojně využívaným online open-source GIS projektem je **OpenStreetMap** (**OSM**) [13]. OSM je použit jako mapový základ pro projekt nazvaný **OpenRouteService** (**ORS**) [14]. ORS ovšem nenabízí zdrojové kódy ke svému vyhledávání a uživatel má k dispozici pouze API.

Autoři na svých webových stránkách tvrdí, že používají algoritmus A*. Systém má však problémy s GUI a vyhledáváním delších tras, které nejsou optimální [2].
Dalším projektem využívajícím OSM je **YOURS** [15] využívající routovacího jádra *Gosmore*, které je hojně rozšířené u mobilních telefonů s operačním systémem Windows Mobile.
Na vyhledávacím stroji *pyroute* (viz 6.2) napsaném v pythonu je založeno několik projektů. Z online implementací vyhledávání je to **OpenStreetRouting** (**OSR**) [16], ten využívá knihovnu *pyroutelib2* [17], která implementuje A*. Ve fázi vývoje je pak projekt **PHProute** [18], který používá modifikovaný algoritmus A* z pyroute.
Velký výběr v použití vyhledávacího algoritmu projekt **pgRouting** [19] postavený na PostgreSQL, který umožňuje vyhledávání pomocí Dijkstrova algoritmu, A* a zde nezmíněného heuristického algoritmu *Shooting Star*.
Zajímavým projektem je **Routino** [20], které opět využívá OSM, ale liší se v použitém algoritmu. Nejkratší cestu totiž vyhledává pomocí principu HNR.
Odlišný přístup používá closed-source projekt **CloudMade** [21], pro vyhledávání cest používá open-source implementaci vyhledávání pomocí zkratek.
Existuje samozřejmě řada dalších online vyhledávačů tras, které jsou mnohem sofistikovanější, než zde uvedené. Za všechny například **Google Maps**, **ViaMichelin**, **Bing Maps**, nebo české **Mapy.cz** a **Amapy.cz**. Pro účely tohoto příspěvku však tvoří okrajovou záležitost, neboť se jedná o komerční software, pro který se neuvolňují informace o vnitřním fungování.

6.2 Offline implementace

Některé offline vyhledávače používají stejné stroje jako online vyhledávače. Například knihovnu *pyroutelib2* používá **Pyroute** [17][22][23] nebo jeho následovník **Rana**. Oba tedy vyhledávají pomocí A*.
Zajímavým projektem je **TravelingSalesman** [24], který je plně modulární a každá jeho část se dá nahradit. Ve verzi od vývojářů je pro routing použit jejich plugin *OSMNavigation* využívající OSM. Pro vyhledávání optimálních cest nabízí Dijkstrův algoritmus, A* a několik jejich modifikací, včetně vyhledávání mezi více body najednou (a to jak startovními, tak cílovými).
Poměrně rozsáhlým projektem je **Navit** [25] s vyhledávání pomocí Dijkstrova algoritmu s Fibonacciho haldami.
Mezi významnější projekty, které používají A* jsou **GpsMid** [26], **True maps** [27] pro iPhone, **VGPS** [28] a **We-Travel** [29]. Pro J2ME platformu existuje routovací engine **J2MERouting** [30], který taktéž využívá A*.
Výrobci komerčního software obvykle opět přesně neuvádí, jaké routovací systémy využívají. Existují výjimky jako třeba [31], kde se uvádí, že pro navigační systém pro palubní počítač Toyoty Soarer z roku 1991 byl použit algoritmus TNR.

7. ZAKOMPONOVÁNÍ STATISTICKÝCH DAT

Všechny výše zmíněné algoritmy pracují nad daty, která se dají označit za statická. Ne vždy však údaje na mapě musí nutně odpovídat reálnému stavu. Silnice může být označená jako první třídy s vysokou maximální rychlostí, nicméně kvůli tomu, že se hodně klikatí, nebo protože je ve špatném technickém stavu, dosahují vozidla pouze zlomek této dané rychlosti. K vyřešení tohoto problému mohou mimo jiné přispět získaná data od samotných uživatelů navigačního software. Řada firem používá sledovací systém pro svá vozidla. Ve městech například taxislužby a kurýrní služby, mezi městy pak dopravci (autobusy, nákladní doprava). Z těchto údajů se pak dá vytvořit reálný model ohodnocení hran grafu. Je to možné několika způsoby. Buď přímo vyrobit graf z takovýchto údajů, pak by ovšem pro málo navštěvovaná místa chyběly údaje. Další možností je zkombinování těchto údajů s obvyklým grafem. Nevý-

hodou může být nutná častá aktualizace a tudíž časté přepočítávaní kompletních podkladů. Zároveň je třeba brát zřetel na to, že dynamická data se mění cyklicky (den x noc) a sezónně (námraza x sucho), musely by tudíž existovat podklady pro všechny tyto možnosti. Stacionární data by pak byla ukládána s každou verzí redundantně. Třetí možnost nabízí ukládat stacionární údaje a několik dynamických zvlášť a při vyhledávání brát v potaz jejich kombinaci. Vyhledávání by pak probíhalo ve více grafech zároveň a výsledná cesta by mezi nimi prolínala podle určitých parametrů. Například tam, kde by se vybíralo mezi dvěma možnými hranami, ale přitom rozdíly by nebyly oproti dynamickým datům nijak markantní, by se dala přednost statickým údajům. Obecně by pak platilo, že čím větší by měla daná dynamická hrana váhu (např. to, že byla mnohokrát změřena), tím spíše by se vzala při výpočtu v úvahu.

8. ZÁVĚR

Jak bylo ukázáno, existuje celá řada přístupů, pomocí kterých se dá řešit vyhledávání optimálních cest v grafech. Zatímco se dá předpokládat, že v uzavřených komerčních projektech jsou používány lepší vyhledávací algoritmy z důvodu konkurenčního prostředí, u open-source projektů běžná programátorská praxe za výzkumem zaostává. Většina software používá buď Dijsktrův algoritmus nebo A*, a to ač je možné používat předpočítaná data a tím výpočty významně urychlit, zejména v online aplikacích.
Navíc, jak bylo ukázáno v kapitole 7, je dalším nedostatkem, že se vývojáři málo zaměřují na opravdu praktické vyhledávání, když používají příliš hrubý model reality. Jeho zjemněním lze dosáhnout lepších řešení a tím pádem upřesnit výsledek pro praktické použití.

PODĚKOVÁNÍ

Práce na tomto příspěvku byla podporována z grantů MŠMT LA08015 a SGS 10/094.

LITERATURA

[1] D. Schultes: *Route Planning in Road Networks*, Universität Fridericianazu Karlsruhe (TH), Karlsruhe, 2008

[2] L. Fedorová: *Vyhľadávanie najkratších ciest nad dátami z OpenStreetMap*, České vysoké učení technické v Praze, Praha, 2009

[3] C. Hendrik: RFC 1058, *Routing Information Protocol*, The Internet Society, červen 1988

[4] J. Kolář: *Teoretická informatika*, ČIS, Praha, 2004

[5] E. W. Dijkstra: *A note on two problems in connexion with graphs*, Numerische Mathematik 1 (s. 269–271), 1959

[6] P. E. Hart, N. J.Nilsson, B. Raphael: *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*, IEEE Transactions on Systems Science and Cybernetics SSC4 (s. 100-107), 1968

[7] A. V. Goldberg, C. Harrelson.: *Computing the shortest path: A* search meets graph theory*, Technical Report MSR-TR-2004-24, Microsoft Research, 2004

[8] J. Maue, P. Sanders, D. Matijevic: *Goal Directed Shortest Path Queries UsingPrecomputed Cluster Distance*, 5th Workshop on Experimental Algorithms (WEA), number 4007 in LNCS (s. 316-328), 2006

[9] E. Köhler, R. H. Möhring, H. Schilling: *Fast point-to-point shortest path computations with arc-flags*, 9th DIMACS Implementation Challenge, 2006

[10] M. Hilger: *Accelerating point-to-point shortest path computations inlarge scale networks*, Technische Universität Berlin, Berlín, 2007

[11] R. Gutman: *Reach-based routing: A new approach to shortest path algorithms optimizedfor road networks*, Workshop on Algorithm Engineering and Experiments (ALENEX) (s. 100-111), 2004

[12] R. Geisberger, P. Sanders, D. Schultes: *Contraction hierarchies: Faster and simpler hierarchical routing in road networks*, 7th Workshop on Experimental Algorithms, 2008

[13] *OpenStreetMap*, http://www.openstreetmap.org/

[14] *OpenRouteService*, http://wiki.openstreetmap.org/wiki/OpenRouteService

[15] *YOURS*, http://www.yournavigation.org/

[16] *OpenStreetRouting*, http://openstreetrouting.appspot.com/

[17] *pyroutelib2 - verze pyroutelib pro příkazovou řádku*, http://wiki.openstreetmap.org/wiki/Pyroutelib2

[18] *PHProute*, https://launchpad.net/phproute

[19] *pgRouting*, http://pgrouting.postlbs.org/

[20] *Routino*, http://www.gedanken.org.uk/software/routino/

[21] *CloudMade*, http://cloudmade.com/

[22] *pyroute*, http://wiki.openstreetmap.org/wiki/Pyroute

[23] *pyroutelib*, http://almien.co.uk/OSM/Routing/

[24] *Traveling Salesman*, http://sourceforge.net/apps/mediawiki/travelingsales/

[25] *Navit*, http://www.navit-project.org/

[26] *GpsMid*, http://gpsmid.sourceforge.net/

[27] *True Maps*, http://sites.google.com/site/truemaps/

[28] *VGPS*, http://www.digitalmobilemap.com/

[29] *We-Travel*, http://we-travel.co.cc/

[30] *J2MERouting*, http://www.j2megps.com/index.php?slug=j2merouting

[31] K. Ishikawa, M. Ogawa, S. Azume, T. Ito: *Map Navigation Softwareof the Electro Multivision of the '91 Toyota Soarer*, IEEE Int.Conf. Vehicle Navig. Inform. Syst (s. 463–473), 1991