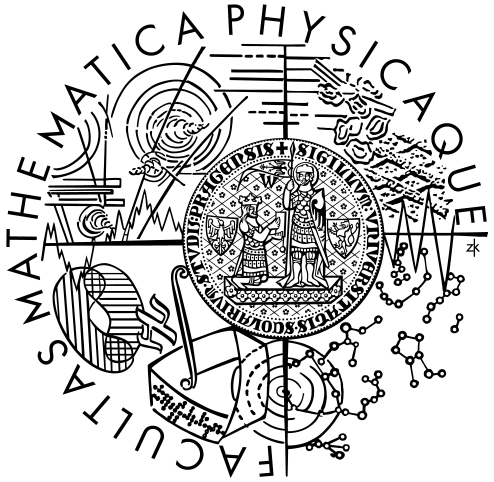
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

Diplomová práce



Matouš Voldřich

Vyhledávání tras v silniční síti

Katedra aplikované matematiky (KAM)
Vedoucí diplomové práce: Mgr. Martin Mareš
Studijní program: Informatika, Softwarové systémy

PODĚKOVÁNÍ

Největší díky jdou firmě PLANstudio a především Pavlu Žemlíkovi za poskytnutí navigačních a jiných dat pro potřeby diplomové práce. Vedoucímu diplomové práce Mgr. Martinu Marešovi děkuji za spoustu dobrých rad a velkou pomoc při psaní této práce.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 8. srpna 2008 Matouš Voldřich

Obsah

Seznam obrázků

Abstrakt

Název práce: *Vyhledávání tras v silniční síti*

Autor: *Matouš Voldřich*

Katedra (ústav): *Katedra aplikované matematiky (KAM)*

Vedoucí diplomové práce: *Mgr. Martin Mareš*

E-mail vedoucího: *mj@ucw.cz*

Abstrakt: Cílem práce je analýza problému hledání optimální silniční trasy v mapě, návrh efektivních vyhledávacích algoritmů pro tento problém a jejich implementace v prostředí síťového serveru. Algoritmy by přitom měly brát v úvahu i požadavky reálného života, jako například zákazy odbočení, jednosměrné ulice a obdobná omezení.

Klíčová slova: vyhledávání silničních a cyklistických tras, Dijkstrův algoritmus, A star, A*

Abstract

Title: *Routing in road networks*

Author: *Matouš Voldřich*

Department: *Departement of Applied Mathematics (KAM)*

Supervisor: *Mgr. Martin Mareš*

Supervisor's e-mail address: *mj@ucw.cz*

Abstract: The aim of this work is the analysis and implementation of internet / network server capable of very quick route planning in road networks. Planning algorithms should support various road situations like forbidden (or enjoined) turns, one way roads and so on.

Keywords: routing in road networks, Dijkstra, A star, A*

Úvod

Na internetu je v současné době dostupná celá řada mapových aplikací. Tyto aplikace se nejčastěji zaměřují na zobrazení map a různých doplňkových informací užitečných pro běžného uživatele internetu. Může se jednat o zobrazení bodových informací v mapě (např. hrady, památky, firmy a jejich pobočky nebo zastávky hromadné dopravy), nebo třeba vyznačené turistické trasy. Další častou a nedílnou součástí mapových aplikací je vyhledávání silniční trasy mezi dvěma (a více) body, výpis itineráře nalezené trasy a její znázornění v mapě.

Tématem této diplomové práce je právě problematika vyhledávání silničních tras v prostředí moderních mapových internetových aplikací. Již čtvrtým rokem pracuji pro firmu PLANstudio s.r.o. (http://www.planstudio.cz), která se touto problematikou intenzivně zabývá a spravuje a aktualizuje navigační data pro Českou republiku a okolí. Jakožto hlavní programátor navigačních řešení jsem pro tuto firmu implementoval navigační server OMS (Online Mapový Systém - dále jen navigační server), který je momentálně využíván mapovými portály mapy.cz, mapy.idnes.cz a jinými zákazníky. Ve spolupráci s kartografy jsem také navrhl strukturu navigačních dat, která je používána pro přenos dat mezi kartografickými aplikacemi a navigačním serverem. Tato data mi byla poskytnuta pro potřeby diplomové práce.

Firma PLANstudio dále poskytla záznamy logů z navigačního serveru mapy.idnes.cz za poslední půlrok. Tyto záznamy obsahují informace o vyhledaných trasách, míře využití serveru a jiné informace popisující chování uživatelů.

V první kapitole diplomové práce stručně popíši architekturu moderní on-line mapové aplikace a představím požadavky, které jsou kladeny na navigační server.

V další kapitole se zaměřím na analýzu logů serveru mapy.idnes.cz a na základě poznatků z nich zjištěných navrhnu vhodné optimalizace serveru.

Třetí kapitola se zaměří na výběr vhodného algoritmu plánování tras a jeho úpravy pro různá kritéria vyhledávání. Budou popsány unikátní možnosti navigačních dat firmy PLANstudio jako jsou zákazy odbočení, použití silničních obchvatů nebo vyhledávání po cyklotrasách, a způsob jejich zapracování do plánovacích algoritmů. V neposlední řadě zde bude popsán způsob prezentace a popisu vyhledané trasy uživateli.

Ve čtvrté kapitole bude stručně popsána implementace navigačního serveru.

Pro potřeby zátěžového testování serveru a porovnání implementovaných algoritmů bude navržena a implementována speciální testovací aplikace. Implementace této aplikace a výsledky

testů provedených na implementovaném navigačním serveru budou prezentovány v páté kapitole. Kapitola se zčásti zaměří na srovnání rychlosti a efektivity Dijkstrova algoritmu a algoritmu A star.

Navigační server firmy PLANstudio je v současné době nasazen v ostrém provozu na několika serverech a je využíván desítkami klientů a aplikací. Jejich popis lze nalézt v šesté kapitole. Zároveň zde budou představeny a popsány ukázkové aplikace, které umožňují si všechny možnosti navigačního serveru osahat v praxi.

Navigační server firmy PLANstudio

Server prošel dlouhou čtyř letou evolucí, na jejímž konci je server schopný obstát v konkurenci s jinými navigačními servery na českém a slovenském trhu. Server byl i nebyl součástí implementace softwarového projektu[1] jménem OMS (Online mapový systém), který byl obhájen v září roku 2007. Do zadání projektu byla zahrnuta mapová část serveru zajišťující vykreslování vyhledaných tras do mapy. Navigační část, která má na starosti samotné vyhledávání tras, do projektu zahrnuta nebyla. Komise pro schvalování projektů explicitně trvala na vyškrtnutí této části ze zadání z důvodů přílišné náročnosti implementace.

Dlouhá praxe v oboru mi poskytla dobrý vhled do problematiky vyhledávání tras v prostředí internetu. Zvolení tohoto tématu za diplomovou práci jsem pojal jako možnost pro celkovou analýzu problematiky a implementaci nové a optimalizované verze navigačního serveru odstraňující mnohé nedostatky a chyby předchozích verzí. V práci se zaměřím především na problémy spojené se samotným vyhledáváním tras v prostředí internetu. Související problémy jako prezentace vyhledané trasy zde ale též budou zmíněny.

[1] Povinný předmět na MFF UK ve kterém tým studentů implementuje větší softwarové dílo.

Kapitola 1

Úvod do problematiky

1.1 Moderní on-line mapové aplikace

Jak bylo řečeno v úvodu, navigační server je jednou ze součástí mapové aplikace. Mapové aplikace a jejich architektura kladou na navigační server několik požadavků, proto je nutné alespoň stručně popsat jak takovéto aplikace fungují a jakým způsobem s navigačním serverem pracují.

Průkopníkem a světovým vůdcem ve vývoji internetových mapových aplikací je firma Google. Ta uvedením svých internetových map světa a inovativním a intuitivním ovládáním vyvolala ve světě on-line map malou revoluci. Google svou aplikací Google Maps (http://maps.google.com) svým uživatelům nabídla nejen přehledné mapy s možností zobrazení satelitních snímků, ale hlavně komfortní, rychlé a propracované ovládání mapy myší. V porovnání s ní se mapové aplikace konkurence zdála těžkopádná a pomalá. Spousta firem od té doby řešení Google Maps do určité míry převzala a byl tak nastolen nový standard.

1.1.1 Rozvoj internetových prohlížečů

Internetové aplikace se vyvýjely ruku v ruce s vývojem a rostoucími schopnostmi internetových prohlížečů. První verze byly založeny hlavně na HTML a protokolu HTTP (Hypertext Transfer Protocol[1]). Při jakékoliv operaci s mapou se musela stránka s mapou obnovit (znovu načíst ze serveru), aby se změny požadované uživatelem projevily v mapě. Toto řešení bylo pomalé a neefektivní, protože i v případě, že došlo ke změně pouze malé části mapy, musel se vždy znovu načíst celý mapový výřez.

Vývoj internetových prohlížečů, rozšiřující se podpora skriptovacích jazyků[2] a zrychlování výpočetního výkonu klientských počítačů umožnila přesunout více prezentační a část aplikační logiky ze serveru na internetový prohlížeč. Zatímco v prvních verzích mapových aplikací se ma-

[1] http://www.w3.org/Protocols/rfc2616/rfc2616.html

[2] Například JavaScript (ECMAScript) - http://www.ecma-international.org/publications/standards/Ecma-262.htm

pový výřez vytvářel na serveru a klientovi se poslal až výsledný obrázek s mapou a zakreslenými informacemi, moderní řešení sestavují mapový výřez až v internetovém prohlížeči z menších částí získaných ze serveru. Jednou stažené menší části mapy se obvykle dají použít i po změně / posunu mapového výřezu a dotáhnou se jenom části, které v přechozím mapovém výřezu nebyly viditelné. Mapové aplikace tedy využívají vyrovnávací paměť prohlížeče k uskladnění jednou použitých částí mapy pro pozdější použití.

Větší a širší podpora kaskádových stylů[3] umožnila složení těchto částí do mapy viditelné uživatelem až na straně prohlížeče. Tyto části jsou obvykle tvořeny stejně velkými bitmapovými dlaždicemi (obvykle ve formátu PNG), které se s pomocí relativní pozice umístí na správné místo ve viditelném mapovém výřezu. Pro efektivnější využití vyrovnávací paměti se mapové dlaždice mohou skládat nad sebe v několika vrstvách.

Spodní vrstvu tvoří neprůhledné dlaždice s mapovými podklady. Tato vrstva je zobrazena vždy a zůstává neměnná, proto je u ní možné velmi efektivně využít vyrovnávací paměť internetového prohlížeče.

Nad mapovou vrstvu jsou pokládány vrstvy, které do mapy dokreslují dodatečné informace vyžádané uživatelem. Může se jednat o vyznačené cyklistické a turistické trasy, body zájmu znázorněné ikonkou, nebo právě trasy vyhledané navigačním serverem.

Navigační server musí být tedy schopen poskytnout dostatek informací pro to, aby bylo možné trasu dostatečně podrobně znázornit v mapě, nebo musí být schopen tuto vrstvu sám vytvořit.

1.2 Požadavky na funkce navigačního serveru

Hlavní funkcí navigačního serveru je vyhledávání silničních (a jiných) tras podle uživatelem zadaných kritérií. Vyhledanou trasu je pak nutné uživateli dostatečně popsat a znázornit zakreslením do mapy.

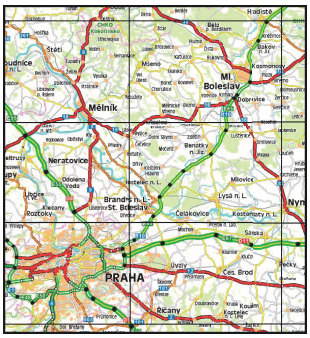
1.2.1 Zadání startu, cíle a průjezdního bodu

Prvním krokem při plánování trasy je volba, odkud a kam a případně přes co má trasa vést. Tyto informace lze označit jako průjezdní body trasy nebo stručněji *destinace*. Pokud je destinace první v trase, nazveme ji počáteční destinací, pokud je poslední, nazveme ji koncovou. Ostatní destinace označíme jako průjezdní.

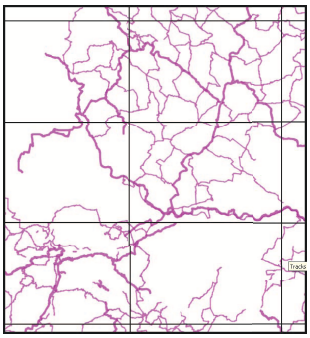
V mapových aplikacích lze destinace obvykle zadat čtyřmi způsoby:

1. vyhledáním v databázi místopisu,
2. vyhledáním v databázi bodů zájmů,
3. volbou pozice destinace v mapě,
4. zadáním GPS souřadnice.

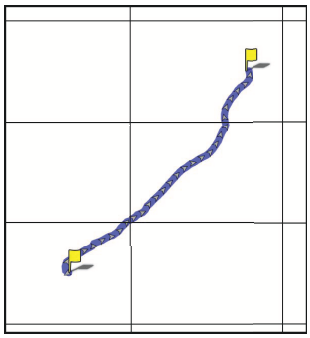
[3] CSS - Cascading Style Sheets - http://www.w3.org/Style/CSS/.



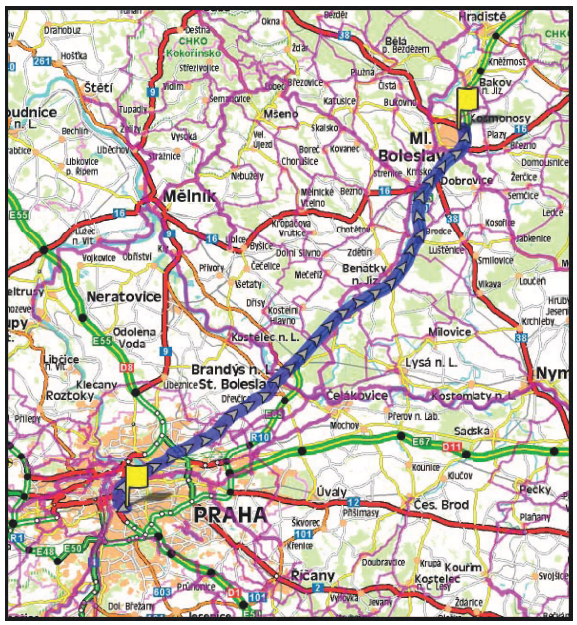
(a) Mapové dlaždice



(b) Zvýrazněné cyklotrasy



(c) Vyhledaná trasa



(d) Výsledný vzhled mapy po naskládání vrstev přes sebe.

Obrázek 1.1: Vrstvy mapové aplikace.

Zadání destinace vyhledáním v místopisu

Zahrnuje vyhledávání v názvech měst, místních názvů, ulic a čísel popisných. Vyhledávání probíhá obvykle ve dvou krocích za pomocí formuláře. V prvním kroku uživatel zadá vyhledávaný název do editačního políčka a odešle formulář. Server místopisu vyhledá zadaný název v databázi místopisu a vrátí možné výsledky. V případě, že bylo nalezeno více výsledků, je uživateli v druhém kroku umožněn výběr, obvykle volbou výsledku ze seznamu.

Zadání destinace vyhledáním v databázi bodů zájmů

On-line mapové aplikace často nad mapou zobrazují body zájmu. Tyto body jsou zakresleny do mapy ikonkou a informují uživatele o poloze obchodů, restaurací, benzínových pump, turistických zajímavostí atd. Uživatel pak obvykle kliknutím na ikonku je schopen přidat daný bod do destinací trasy.

Zadání souřadnicí

Většina uživatelů je zvyklá zadávat a číst polohu v mapě s pomocí zeměpisné šířky a délky. Tento systém souřadnic je znám pod označení WGS84[4] a je uznávaným celosvětovým standardem. Používají ho například v současné době velmi oblíbené GPS systémy. Server by tedy měl tento formát podporovat a umožnit přidání destinace na souřadnici zadanou uživatelem.

Zadání destinace volbou pozice v mapě

Posledním oblíbeným způsobem zadání destinace je volba bodu z viditelného výřezu mapy. Tento způsob je velmi intuitivní a uživatelsky přívětivý, protože umožňuje umístit destinaci kamkoliv v rozsahu aktuálně zobrazované mapy prostým kliknutím tlačítka myši.

1.2.2 Kritéria vyhledávání trasy

Vedle zadání destinací je nutné, aby uživatel zvolil kritérium, podle kterého chce trasu vyhledat. Volba kritéria zahrnuje typ dopravního prostředku, pro který je trasa vyhledávána (automobil, jízdní kolo), a způsob vyhledání trasy (nejkratší, nejrychlejší). U každého kritéria mohou být zadány dodatečné parametry, které pak kladou na vyhledanou trasu určité omezující podmínky.

Automobilové trasy

Většina mapových aplikací na internetu v současné době podporuje dvě základní kritéria umožňující vyhledání silniční nejkratší a nejrychlejší automobilové trasy.

Kritérium „nejkratší“ má za úkol vyhledat co do délky nejkratší trasu procházející všemi destinacemi. Tato trasa je však pro běžné řidiče automobilů často nepřijatelná, protože obvykle

[4] World Geodetic System 1984 - http://earth-info.nga.mil/GandG/publications/tr8350.2/tr8350_2.html

vede přes místní a nekvalitní komunikace. Vyhledaná trasa často také obsahuje zbytečně moc odbočení a je tedy navigačně složitá a nepřehledná.

„Nejrychlejší“ kritérium vyhledá takovou trasu, kterou vozidlo projede s minimálním časem (za dodržení rychlostních limitů). Toto kritérium tedy preferuje dálnice a kvalitnější komunikace, na kterých lze dosahovat vyšších rychlostí. Výsledná trasa je tedy přehlednější, protože se snaží držet kvalitnějších komunikací a místní použije, až když je skutečně potřeba. Na druhou stranu, ale nejrychlejší trasy jsou téměř vždy delší, než trasy nejkratší a jejich projetí může vyžadovat více pohonných hmot. Proto se ke dvojici těchto kritérií přidává ještě tzv. „ekonomické“ kritérium, které tvoří kompromis mezi nejkratším a nejrychlejším kritériem.

Důležitým parametrem při vyhledávání automobilové trasy je povolení nebo zakázání použití placených silničních úseků (dálnice a některé rychlostní komunikace). Tato funkce umožní plánovat trasy řidičům podle toho, zda mají nebo nemají zakoupenou dálniční známku.

Cyklistické trasy

Firma PLANstudio má kromě automobilové silniční sítě k dispozici i síť cyklostezek a místních komunikací. Jedním z požadavků je tedy i návrh a implementace plánování tras pro cyklisty. Cyklistická a automobilová síť se z větší části překrývá. Cyklisté však mají povolen vjezd na lesní cesty nebo cyklostezky, a naopak mají zakázaný vjezd na dálnice nebo speciální úseky cest, jako některé tunely.

Cyklisté se často chtějí držet značených cest a vyhnout se silnicím, proto kritéria pro vyhledávání cyklotras zahrnují různé stupně preference značených cest.

Silnice vyšších tříd jsou často plné automobilů, proto jedním z dodatečných parametrů cyklistické trasy je omezení silničních komunikací vyšších tříd.

1.2.3 Vyhledání trasy

Po definování destinací a volbě kritéria přichází na řadu samotné vyhledání trasy. Výpočet a plánování trasy musí být rychlé, aby celková odezva internetové mapové aplikace při vyhledání trasy byla dostatečně krátká. Uživatelé na internetu jsou zvyklí na „okamžitou“ reakci a dlouhé odezvy jsou považovány buď za otravný nedostatek nebo rovnou za chybu.

Plánování tras na internetu slouží hlavně pro orientační účely. Výsledky jsou určeny „laickým“ uživatelům a nemusí být tedy stoprocentně přesné. Rychlost odpovědi a vyhledání rozumné trasy vzhledem k zadaným kritériům je tedy přednější, než vyhledání optimální trasy.

1.2.4 Popis vyhledané trasy

Po vyhledání trasy je nutné trasu uživateli znázornit v mapě a vhodně popsat. Popis musí být snadno pochopitelný, stručný, ale zároveň musí obsahovat všechny informace potřebné k bezproblémovému absolvování trasy.

Itinerář trasy

Základním způsobem popisu trasy je zobrazení itineráře. Itinerář je detailní popis jednotlivých úseků trasy s podrobnými navigačními pokyny, které uživatele postupně provedou od startu k cíli. Obsahuje značení komunikací v trase, jejich třídu, na jakých křižovatkách a jakým směrem se má odbočit atd. Dále obsahuje informace o délce trasy a času, který je potřebný k jejímu projetí.

Itinerář musí být přehledný a přesný, ale zároveň nesmí obsahovat zbytečné informace. Pro řidiče je většinou zbytečné a matoucí, pokud jsou v itineráři uvedené všechny křižovatky v trase. Je vhodné uvádět pouze ty, které jsou pro navigaci důležité. Například ty, na nichž dochází k odbočení na jinou komunikaci, k prudké změně směru atd.

Zobrazení trasy v mapě

Navigační server musí být schopen poskytnout o trase informace, které umožní její vykreslení do mapy. Bude nutné navrhnout způsob práce s aplikaci, která toto vykreslení obstará. Ať už exportem vektoru bodů trasy nebo jinak.

1.2.5 Nasazení serveru, optimalizace na vysokou zátěž

Server musí být nasaditelný jak v prostředí operačního systému Windows, tak Unix / Linux.

Dnešní internetové servery často běží nad počítači s vícejádrovým procesorem. Navigační server by měl být schopen toto využít a podporovat vícevláknové zpracování požadavků.

1.2.6 Doplňkové služby

Do této kategorie požadavků spadají méně tradiční služby navigačního serveru.

Poskytování informací o komunikacích

Do mapy se pro nedostatek prostoru zakreslují pouze značení a mezinárodní značení silnic. Občas je ale nutné zjistit podrobnější informace o určité komunikaci, jako maximální rychlost, typ komunikace, nebo podrobnější informace o značení či nadmořskou výšku. Tyto informace by měly být dostupné intuitivně po kliknutí do mapy nad určitou komunikací.

Stejnout funkci lze využít i při tzv. lokalizaci vozidel. Úkolem je vyhledat informace dostatečně popisující polohu vozidel. Poloha vozidel je často brána přímo z GPS přístrojů a je tedy ve formátu WGS84. Což nás přivádí k dalšímu požadavku.

Převody souřadnic

Navigační server by měl být schopen pracovat s několika souřadnicovými systémy standardně používanými na území České republiky.

Neomezený počet destinací v trase

Většina mapových serverů umožňuje při plánování trasy zadat pouze dvě až tři destinace. Při plánování trasy volbou destinací kliknutím do mapy ale může být omezení maximálního počtu destinací v trase velmi omezující. Obzláště při detailním plánování cyklotras se často počet destinací může vyšplhat do řádů desítek. Pokud se počet destinací neomezí, bude si uživatel moci doslovně „vykolíkovat“ svou trasu v mapě a plánovat ji přesně podle svých představ.

Vyhledání tras z jedné destinace do všech ostatních

Zákazníci se občas ptají po funkci, která by vyhledala silniční vzdálenosti z jednoho bodu do množiny destinací. Vezměme si třeba příklad „betonárky“. Zákazník si chce koupit beton, ale neví do jaké betonárky to má nejblíže. Zadá tedy do mapy svoji polohu a zažádá o vyhledání nejbližších betonárek ve svém okolí. Vzdušná vzdálenost by v tomto případě byla matoucí, protože helikoptérou se pro beton zatím nelétá. Pro určení silniční vzdálenosti je tedy nutné vyhledat trasy z polohy zadané zakazníkem do všech „blízkých“ betonárek.

1.2.7 Logování

Činnosti serveru musí být monitorovány a zaznamenávány do logových souborů. Samotný proces monitorování a logování však nesmí příliš zpomalit samotné zpracovávání požadavků.

Logování bylo implementováno již v předchozí verzi navigačnho serveru, který byl nasazen a využívá mapovým serverem mapy.idnes.cz. Firma PLANstudio poskytla tyto záznamy pro potřeby diplomové práce. Ze záznamů lze vyčíst jaké byly vyhledávány trasy, jaké služby navigačního serveru byly nejvíce využívány, zda byl server využíván rovnoměrně či nárazově atd. Podrobnou analýzu těchto záznamů lze nalézt v kapitole 2.

1.3 Navigační data

Jak bylo řečeno v úvodu, jednou z činností firmy PLANstudio je vytváření a správa navigačních dat. Tato data byla plně poskytnuta pro potřeby diplomové práce a testování navigačního serveru. V této kapitole bude popsána struktura těchto dat a možnosti, které poskytují.

1.3.1 Souřadnice

Veškeré infomace o poloze jsou v datech uvedeny v souřadném systému UTM (Universal Transverse Mercator). UTM ([7]) je systém souřadnic, který zobrazuje polohu na Zemi do roviny reprezentované dvourozměnou mřížkou. Rozděluje Zemi do 60 poledníkových zón, kde každá zóna pokrývá 6° zeměpisné šířky. Středem zóny prochází tzv. středový poledník. Tímto poledníkem je proložen válec, který je poté pomocí mapového zobrazení „narovnán“ do roviny

(transverzní Mercatorovo zobrazení). Střed souřadnic je pro každou zónu jiný a je umístěn v průsečíku středového poledníku zóny a rovníku. Od tohoto středu jsou vzdálenosti měřeny v metrech. Po ose x rostou od středového poledníku směrem na východ (tzv. eastings) a po ose y rostou od rovníku směrem na server (tzv. northings).

Systém byl zvolen pro své výborné vlastnosti:

- má ortonormální bázi,
- jeho jednotkou je 1 metr.

Veškeré výpočty vzdáleností mezi souřadnicemi jsou tedy velmi jednoduché a vystačíme si s Pythagorovou větou. Nevýhodou tohoto systému je, že každá souřadnice je vázána na konkrétní UTM zónu. Navigační data Evropy pokrývají zóny 29 až 36, což znamená, že různé části navigačních dat by měly být správně uvedené v různých zónách. Pak by se ale velmi zkomplikoval výpočet vzdálenosti mezi souřadnicemi v různých zónách.

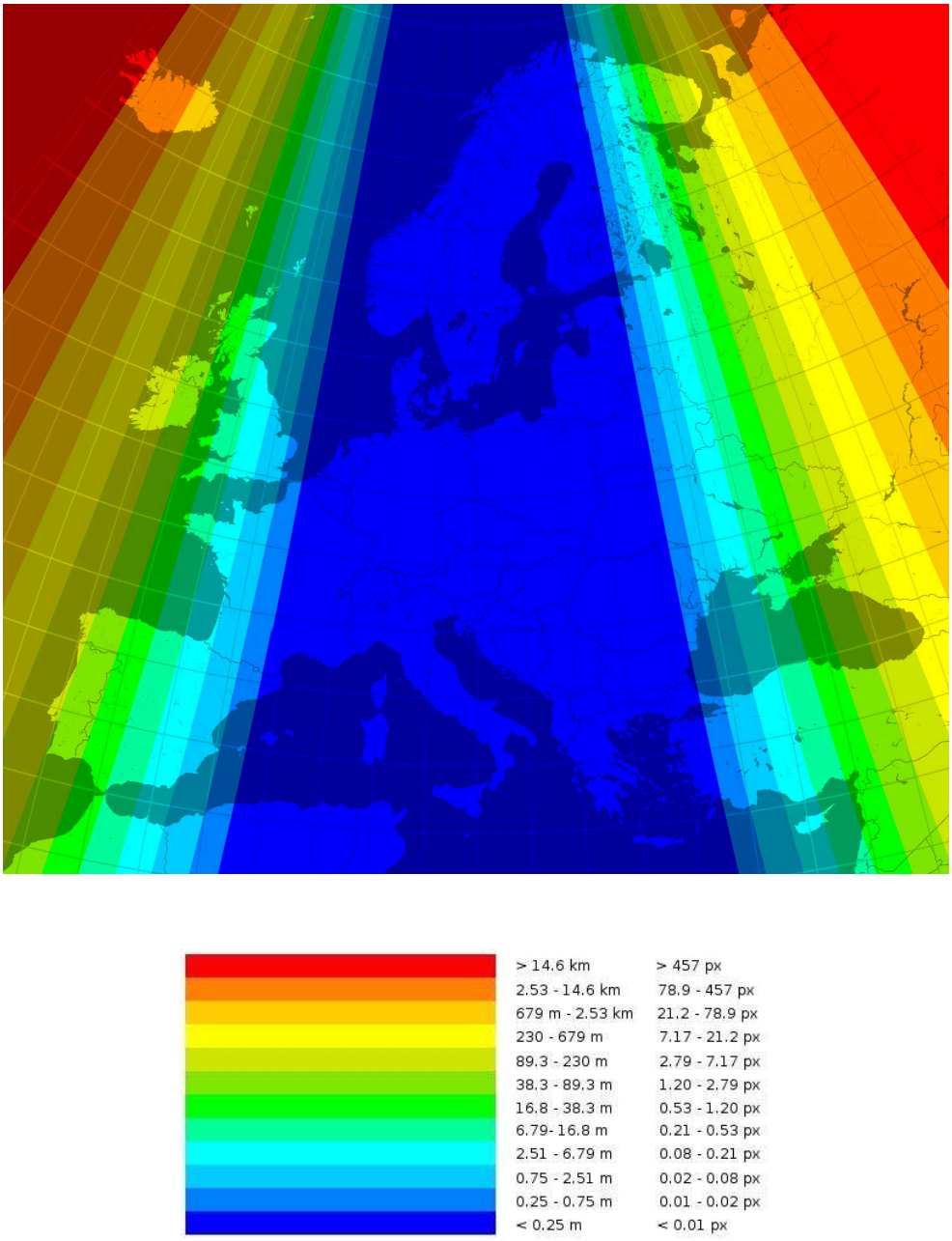
Řešením je „roztažení“ jedné UTM zóny (v tomto případě se jedná o zónu 33) do rozsahu celé Evropy. Tím získáme souřadnicový systém schopný obsáhnout celá navigační data, ale na úkor menší přesnosti mimo území zóny 33. Nepřesnosti se projeví hlavně při převodech z a do jiných souřadných systémů (např WGS84). Čím větší vzdálenost od zóny, tím větší nepřesnost.

Vývoj odchylek souřadného systému UTM v pásu 33 nad mapou Evropy zachycuje obrázek 1.2. Na krajích (červené oblasti) dochází k nepřesnostem, které jsou větší jak $14,$ *km*. V oblasti, kterou pokrývají navigační data, se však nepřesnosti pohybují v rozmezí $0{,}25$ *m* (celá oblast střední Evropy) do $89\,m$ (Portugalsko, Irsko). Zvolená reprezentace souřadnic je tedy na většině území dostatečně přesná.

1.3.2 Struktura navigačních dat

Navigační data jsou dodávána v několika textových souborech. Každý ze souborů obsahuje záznamy jednoho typu ve formátu CSV (jeden záznam na řádku, hodnoty oddělené speciálním znakem). Každý záznam je opatřen jednoznačným textovým nebo číselným identifikátorem. Identifikátory vycházejí z interních dat firmy PLANstudio a umožňují navázání záznamů navigačních dat na záznamy v kartografických databázích. Soubory navigačních dat jsou mezi sebou propojeny za pomocí těchto identifikátorů (vazby jsou znázorněny na obrázku 1.3 na stránce 16). Pro identifikaci měst a sídel je použito kromě interního identifikátoru také identifikátor UIR-ADR[5].

[5]Identifikátor města z Územně identifikačního registru adres - http://forms.mpsv.cz/uir/default2.jsp



Obrázek 1.2: Přesnost UTM souřadnic pásu 33 v mapě Evropy. Různě barevné zóny představují rozsahy chyb, které mohou vzniknout při práci se souřadnicemi UTM v pásu 33 v této oblasti. Schéma je dílem Ondřeje Procházky a je použito s jeho svolením.

Seznam souborů

Název souboru	Popis
location.txt	Seznam křižovatek
routes.txt	Seznam úseků komunikací
crossing_category.txt	Seznam kategorií křižovatek
route_category.txt	Seznam kategorií úseků komunikací
vectors.txt	Seznam vektorů úseků komunikací
cyclo.txt	Seznam značených cyklostezek
direction.txt	Seznam navigačních ukazatelů a návěští
states.txt	Seznam států
cities/*.txt	Seznamy sídel (měst, obcí a místních názvů)

Nejdůležitějšími typy záznamů jsou křižovatky a úseky komunikací.

Křižovatky

Křižovatka reprezentuje místo, kde začíná nebo končí jeden a více úseků komunikací. U každé křižovatky je evidován její název, název pro cyklisty, poloha a kategorie. Název obsahuje orientační informace pro automobily (např. název obce nebo číslo dálničního exitu), název pro cyklisty může obsahovat např. název cyklorozcestníku nalézajícího se na této křižovatce. Kompletní seznam parametrů a popis struktury záznamu úseku komunikace souboru locations.txt lze nalézt v tabulce 1.1.

Kategorie křižovatky obsahuje informace o typu a vzhledu křižovatky. Může jím být například kruhový objezd, světelná křižovatka, nájezd atd.

Parametr	Sloupec	Popis	Typ parametru
lineNum	1	číslo řádku	celé číslo
IDK	2	ID křižovatky	řetězec
Název	3	název křižovatky	řetězec
IDKC	4	kategorie křižovatky	celé číslo
Cyklo	5	název křižovatky pro cyklisty	řetězec
X	6	souřadnice X	desetinné č.
Y	7	souřadnice Y	desetinné č.
cityId	8	ID sídla ve kterém se křižovatka nalézá	řetězec

Tabulka 1.1: Popis sloupců záznamu křižovatky v souboru locations.txt.

Úseky komunikací

Úsek komunikace popisuje cestu mezi dvěma různými křižovatkami. K dispozici je celá řada informací od délky úseku (v kilometrech), kategorie silnice, přes různé typy značení, po navigační

parametry. Kompletní seznam parametrů a popis struktury záznamu úseku komunikace souboru routes.txt lze nalézt v tabulce 1.2. Jednotlivé parametry úseků budou rozebrány podrobně.

Parametr	Sloupec	Popis	Typ parametru
IDU	1	ID úseku	řetězec
IDK	2	ID výchozí křižovatky	řetězec
IDK	3	ID cílové křižovatky	řetězec
Číslo komunikace	4	číslo silnice	řetězec
Popis komunikace	5	dodatečný popis / název ulice	řetězec
Mezinárodní	6	mezinárodní značení	řetězec
IDUC	7	ID kategorie úseku	řetězec
Směr	8	viz. kapitola 1.3.2	řetězec
Příznaky	9	různé příznaky úseku cesty	řetězec
Délka	10	délka úseku v km	desetinné č.
Rychlost	11	skutečná předepsaná rychlost v km/h	celé číslo
Výška	12	výškové omezení v metrech	desetinné č.
Cyklotrasa	13	cyklistická značení oddělená mezerou	řetězec
Manévr K1	14	viz. kapitola 1.3.2	řetězec
Manévr K2	15	viz. kapitola 1.3.2	řetězec
Navigace K1	16	navigační povely výchozí křižovatky	řetězec
Navigace K2	17	navigační povely cílové křižovatky	řetězec
Uzavírka	18	časově omezená uzavírka úseku	řetězec
Město ve směru K1	19	viz. kapitola 1.3.2	řetězec
Město ve směru K2	20	viz. kapitola 1.3.2	řetězec
Turistika	21	turistická značení oddělená mezerou	řetězec
Obchvat	22	viz. kapitola 1.3.2	celé číslo
Mýto	23	příznak zda je placeno mýto	boolean
Stát	24	dvoupísmenná zkratka státu	řetězec
Obec	25	Název obce ve kterém se úsek nalézá	řetězec

Tabulka 1.2: Popis sloupců záznamu úseku komunikace v souboru routes.txt.

Obecné parametry úseků tvoří:

- textový identifikátor jednoznačný v celém modelu navigačních dat (nemění se ani při jejich výměně),
- počáteční a koncová křižovatka určující, odkud a kam úsek vede,
- délka úseku v kilometrech,
- identifikátor kategorie úseku,
- město a stát, ve kterém se úsek nachází.

Identifikátor kategorie úseků odkazuje do seznamu kategorií ze souboru route_category.txt. Pro každou kategorii úseků je definována maximální povolená rychlost automobilu v obci a mimo obec a doporučená rychlost pro cyklistu. Hodnota této implicitní maximální rychlosti automobilu je potlačena, pokud je definován parametr maximální povolené rychlosti (sloupec 11).

Pro každý úsek může být nastaveno několik typů **značení**:

- silniční,
- silniční mezinárodní,
- cyklistické,
- turistické.

Parametry cyklistického a turistického značení mohou obsahovat více hodnot oddělených mezerou nebo čárkou.

Navigační parametry

Navigační parametry omezují nebo upřesňují navigaci na úsecích nebo možnosti odbočení na křižovatkách úseku.

Směr úseku (sloupec 8) určuje zda je úsek průjezdný oběma směry (sloupec 8 se rovná „0“) nebo pouze ve směru od počáteční do koncové křižovatky (= „1“). Směr může být upresněn pro konkrétní dopravní prostředek. Například hodnota „1 C0“, říká, že úsek je jednosměrný pro automobily, ale obousměrný pro cyklisty (nejčastější případ).

Parametry manévrů (sloupce 14 a 15) omezují možnosti odbočení na jedné z křižovatek úseku komunikace. Lze tak definovat zákazy odbočení nebo naopak příkázaný směr jízdy. Zákaz odbočení je složen z písmena „Z“, které je následováno identifikátorem následujícího úseku, na který je zakázáno odbočit. Příkázaný směr jízdy se definuje stejně, pouze s písmenem „P“. V definici jednoho manévru může být buď jeden a více zákazů nebo právě jeden příkaz. Manévry jsou definovány pro obě křižovatky úseku zvlášť. Manévr K1 (sloupec 14) určuje manévry při příjezdu z úseku do výchozí křižovatky. Manévr K2 (sloupec 15) určuje manévry při příjezdu z úseku do křižovatky cílové.

Návěští (sloupce 19 a 20) poskytuje řidiči informace o větších městech ve směru pohybu vozidla. Návěští může obsahovat více měst najednou (například Brno, Vídeň). Návěští jsou podobně jako manévry definovány pro oba směry úseku zvlášť.

Obchvat města (sloupec 22) určuje, zda je úsek komunikace součástí obchvatu konkrétního sídla (obce či města). Sídlo je určeno identifikátorem UIRADR. Tato informace je užitečná při vyhledávání trasy „přes“ určité město.

Vektory úseků

Každý úsek komunikace má přiřazen vektor souřadnic ze souboru vectors.txt. Tento vektor popisuje jak ve skutečnosti úsek vypadá a umožňuje ho vykreslit do mapy. Souřadnice vektoru jsou uloženy vždy v pořadí od výchozí křižovatky do cílové, přičemž každý vektor má minimálně

dva body jejichž pozice je shodná s pozicemi počáteční a cílové křižovatky. Některé vektory mohou mít až několik desítek bodů. U každé souřadnice vektoru je uvedena i její nadmořská výška.

Vektory představují objemově majoritní část navigačních dat. Podílí se na celkovém objemu z 85,5% - 90%. V datech, která byla k dispozici při psaní diplomové práce zabírala 120 MB z celkem 137 MB.

Sídla

Seznam sídel obsahuje seznam měst, obcí a místních názvů. Pro každé sídlo je definován jeho identifikátor (IDC), typ, název, poloha a hodnota UIRADR. Typ sídla určuje zda se jedná o město (MA), část města (MC), obec (OA) nebo část obce (OC).

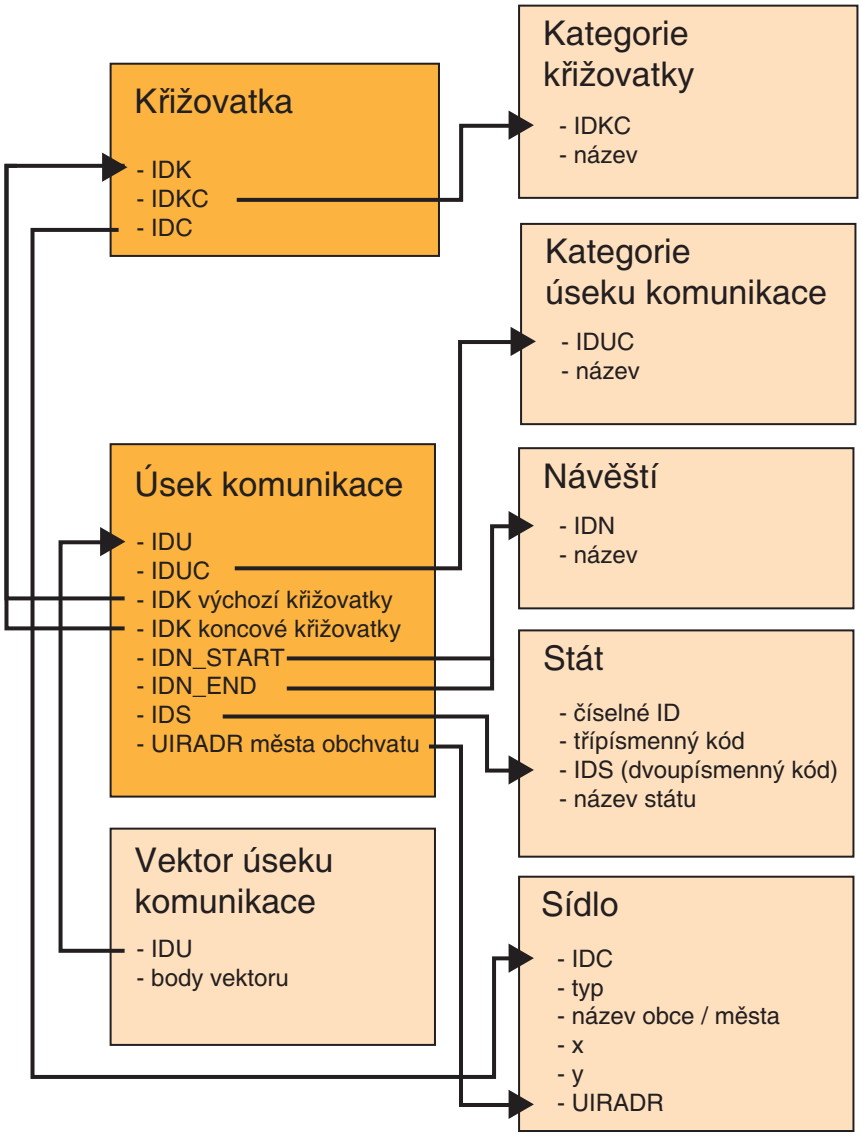
Sídlo je reprezentováno bodem v mapě. Tento bod ukazuje buď na důležité náměstí daného města, obecní úřad nebo jinou důležitou budovu.

1.3.3 Podrobnost modelu

Navigační model je v různých oblastech různě detailní. Na území České republiky je silniční model téměř kompletní (do úrovně místních komunikací a ulic), na území Slovenské republiky model obsahuje komunikace první a druhé třídy, místní komunikace a dálnice. V jiných zemích je však model mnohem méně podrobný a obsahuje pouze dálnice a silnice první, popřípadě druhé, třídy. Cyklistické komunikace jsou v současné době dostupné pouze na území České republiky.

Model dat a jeho podrobnost lze prohlédnout v ukázkové aplikaci (kapitola 6.3). Po kliknutí na odkaz „Využití komunikací ve vyhledaných trasách“ bude nad mapou zobrazena průhledná vrstva všech komunikací v modelu. Pomineme-li různou barevnost, tak vrstva poskytuje velmi dobrý přehled o podrobnosti modelu v různých oblastech.

V době psaní diplomové práce bylo v datech celkem 116687 křižovatek a 168148 úseků komunikací.



Obrázek 1.3: Vazby mezi soubory navigačních dat.

Kapitola 2

Analýza logových záznamů

On-line mapová aplikace mapy.idnes.cz využívá jakožto jeden z klientů navigační server OMS firmy PLANstudio. Veškerá činnost serveru a požadavky klientů jsou logovány a ukládány do textových souborů ve formátu CSV. V době zahájení jejich zpracování byly k dipozici logové záznamy za období 1.8.2007 - 31.3.2008[1].

V logových záznamech jsou uložené záznamy o aktivitách všech aplikací využívajících starý navigační server. V analýze se zaměřím pouze na klienty aplikace mapy.idnes.cz. Záznamy ostatních klientů budou zanedbány.

Zpracováním a analýzou těchto souborů lze získat přehled o oblíbených kritériích, četnosti opakování vyhledaných tras, průměrnou dobu práce s navigačním serverem atd. Nejprve bude stručně popsána struktura logů a logových záznamů.

Struktura logových záznamů

Logový soubor navigačního serveru je textový soubor. Pro každý den existuje právě jeden logový soubor. Jeho název je tvořen podle masky YYYYMMDD.log, kde YYYY je rok, MM měsíc a DD den. V souboru jsou jednotlivé záznamy oddělené znakem nové řádky. Záznam logu pak obsahuje hodnoty oddělené znakem „|“ (svislítko). První sloupec je vždy tvořen jedním písmenem, které určuje typ logového záznamu. Význam dalších sloupců se pak různí s ohledem na typ.

Existuje celkem 8 typů logových záznamu. Pro potřeby diplomové práce jsou zajímavé pouze záznamy typu R a O.

request R

Oznamuje vyřízení požadavku. (celkovou dobu vyřízení, typ atd.)

routing O

Oznamuje vyhledání trasy. Záznam obsahuje informace o kritériu vyhledávání, destinacích, délce trasy, době výpočtu trasy a jiné. Ze záznamu lze zrekonstruovat vyhledanou trasu.

[1] Celkem 242 souborů zabírajících 1 657 MB.

Struktura logových záznamů a podrobný popis jejich sloupců je popsána v dokumentaci navigačního serveru (příloha A).

Ukázka z logů

```
O|075728|23220|17|190|424,346|4|3|1|174|24377|218|460317;5547176;cz_0649;...
R|075728|23220|17|getroutingsimpleitinerary|81.0.225.136|9|1|216
O|080401|23571|15|5|13,227|2|3|1|16|43|9|748894;5490887;;747814;5496154;|0
R|080401|23571|15|getroutinglist|77.93.192.166|1|0|2
R|080401|23220|17|getimagenomap|81.0.225.136|4|0|11
R|080401|23220|17|getimagenomap|81.0.225.136|4|0|11
```

Z této ukázky lze vyčíst následující:

- Klient využívající aplikaci mapy.idnes.cz (id=17) požádal o vyhledání trasy přes 2 destinace. Její výpočet trval 190 ms a výsledná délka byla 424,3 km.
- Poté si vyžádal itinerář, který tuto trasu popisuje.
- Mezitím jiný klient vyžívající aplikaci travelguide.cz (id=15) požádal o vyhledání trasy a vyžádal si její itinerář.
- První klient požádal o vykreslení několika průhledných dlaždic znázorňující trasu v mapě.

2.1 Zpracování logů a výsledky

Pro účely zpracování logů bylo nutné navrhnout a implementovat aplikaci, která logové záznamy zpracuje a vytvoří statistiky, které bude možné snadno zpracovat v tabulkovém procesoru Microsoft Excel. V něm pak budou provedeny dodatečné výpočty a vytvořeny grafy.

Pro implementaci aplikace byla zvolena technologie .NET a programovací jazyk C#. Tato kombinace umožňuje rychlý vývoj aplikací v prostředí MS Windows. Aplikace byla pracovně pojmenována LogFilter a tento název ji již zůstal. Lze ji najít na přiloženém CD (příloha B).

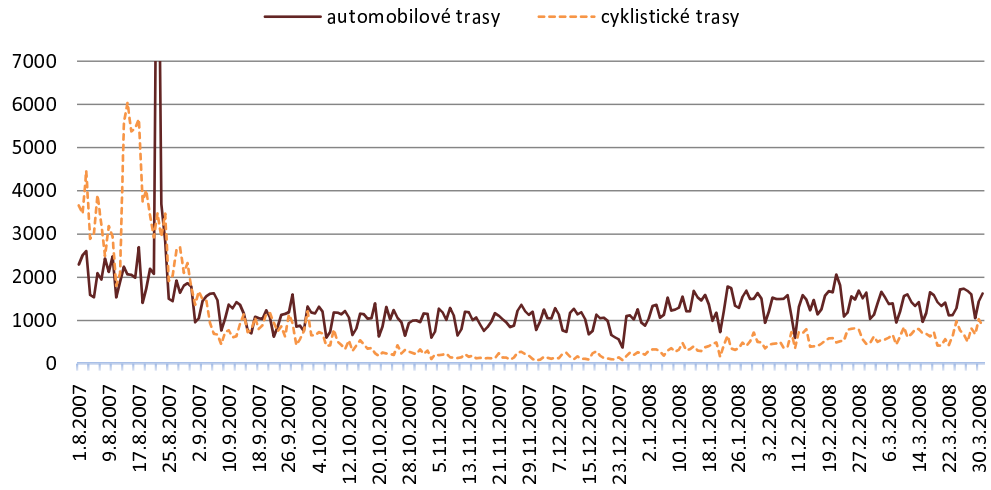
Při zpracování logů byl kladen důraz na zjištění informací o:

- rozložení zátěže serveru,
- využití různých kritérií vyhledávání,
- opakování vyhledání stejné trasy,
- využití úseků komunikací ve vyhledaných trasách,
- použitých destinací ve vyhledaných trasách.

2.1.1 Rozložení zátěže

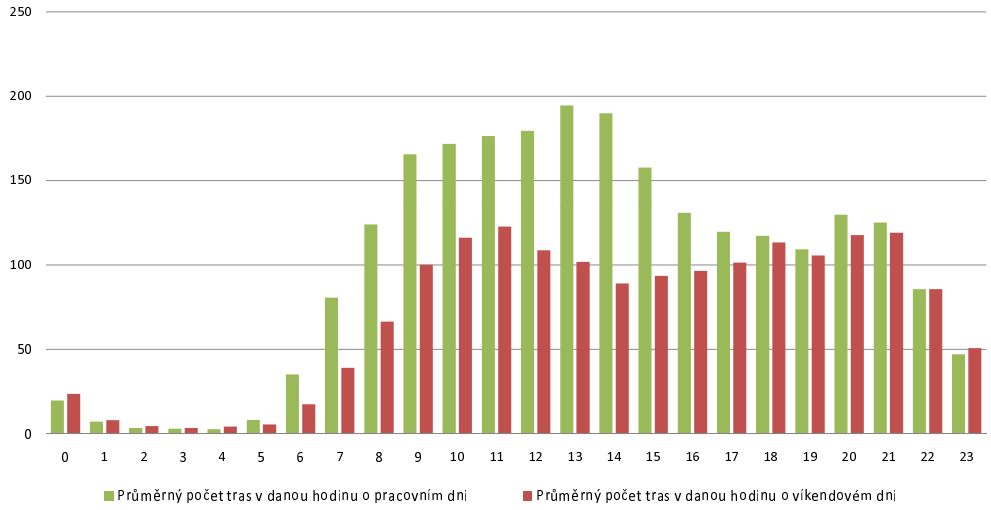
Počet vyhledaných tras v jednotlivých dnech ve zkoumaném období zachycuje graf 2.1. Server byl mnohem více využíván v letních měsících, v zimě došlo k útlumu a na jaře zátěž opět postupně narůstá. V letních obdobích převažovaly cyklistické trasy, jinak jasně dominují trasy automobilové.

Výrazné výkyvy v návštěvnosti jsou důsledkem reklamních kampaní internetového deníku www.idnes.cz.



Obrázek 2.1: Denní statistiky využití serveru aplikací mapy.idnes.cz.

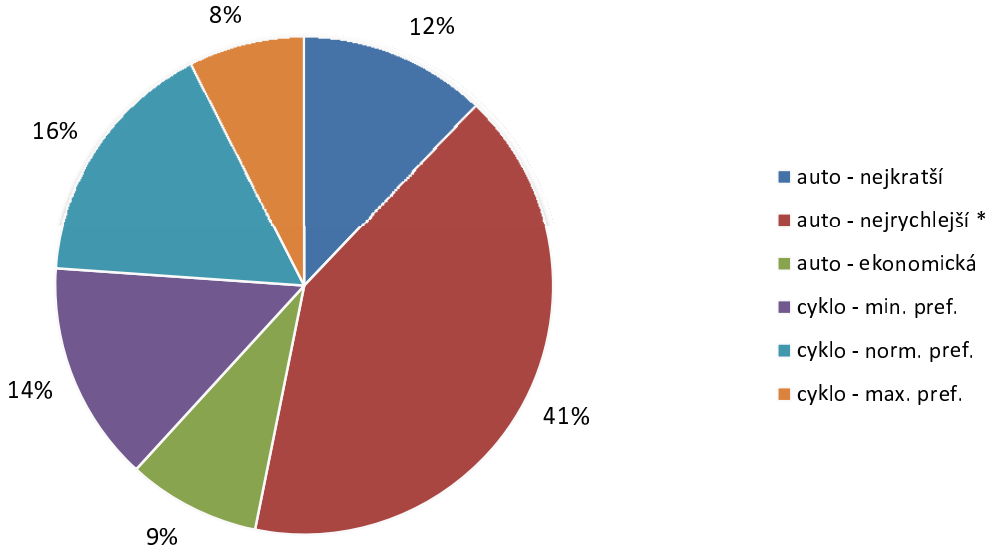
Graf 2.2 zachycuje hodinové rozložení vyhledávání tras. Obsahuje dva sloupce. Jeden pro všední dny, druhý pro dny víkendové. Server je více využíván ve všední dny ($\varnothing$ 2 384 vyhledaných tras/den, 99,34 tras/hodina) než o víkendu ($\varnothing$ 1 694 vyhledaných tras/den, 70,59 tras/hodina). Je zajímavé, že rozdíl mezi pracovními a víkendovými dny je patrný především mezi 6 a 16 hodinou, což vcelku přesně odpovídá pracovní době.



Obrázek 2.2: Hodinové statistiky využití serveru aplikací mapy.idnes.cz.

2.1.2 Využití různých kritérií vyhledávání

Součtem přes všechny záznamy „O“ podle 8 sloupce určujícího kritérium vyhledané trasy lze snadno zjistit oblíbená kritéria vyhledávání tras. Graf znázorňující poměr využití jednotlivých kritérií při vyhledávání lze vidět na obrázku 2.3. Drtivě nejpoužívanějším je vyhledávání nejrychlejší trasy (41% všech vyhledaných tras).



Obrázek 2.3: Využití kritérií ve vyhledaných trasách mapového serveru mapy.idnes.cz. Hvězdičkou jsou označena implicitní kritéria pro daný dopravní prostředek.

Srovnání mezi automobilovými trasami a cyklotrasami co do jejich průměrné délky trasy, počtu destinací a doby výpočtu popisuje tabulka 2.1.

	Auto	**Cyklo**
Průměrná délka trasy (km)	324,045	96,806
Průměrný počet destinací v trase	2,202	2,620
Průměrná doba výpočtu (ms)	224,307	82,641

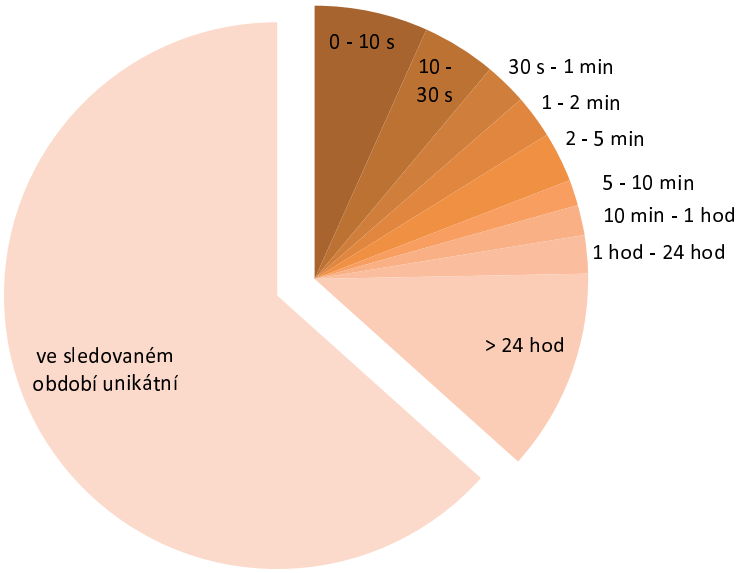
Tabulka 2.1: Srovnání základních statistik automobilových a cyklistických tras

Je zřejmé, že vyhledané cyklotrasy jsou mnohem kratší než trasy automobilové. Uživatelé je však plánují přes více destinací. I tak, ale málokdy uživatelé využili maximální počet destinací v trase povolených na serveru idnes.cz (sedm destinací). Uživatelské rozhraní na mapách serveru idnes je však v tomto směru značně nepřátelské a neumožňuje přidat destinaci do jednou vyhledané trasy. Znemožňuje tak její postupné plánování.

2.1.3 Opakování vyhledání stejné trasy

Z hlediska optimalizace je užitečné vědět, zda a jak často server vyhledává stejné trasy. Tato informace nám pomůže při rozhodování, zda má smysl implementovat vyrovnávací paměť vyhledaných tras. Místo opakovaného vyhledávání stejné trasy se trasa spočítá jednou a ostatní požadavky budou načteny z vyrovnávací paměti.

Při vytváření statistik byly počítány časové rozdíly mezi trasami se stejnou definicí (viz. kapitola 3.3.1) (neúplné logové záznamy typu „O“ bez definice destinací byly vynechány). Podle velikosti rozdílu pak byly trasy rozděleny do několika intervalů. Percentuální rozložení tras opakujících se v určitém časovém intervalu ve sledovaného období (1.8.2007 - 31.3.2008) za



Obrázek 2.4: Koláčový graf četnosti opakování stejné trasy v časových intervalech.

$19{,}13\%$ dotazů se opakuje během prvních 5 minut. Implementováním krátkodobé vyrovnávací paměti by tedy bylo možné celou $1/5$ všech vyhledaných tras načítat přímo z ní.

2.1.4 Využití komunikací

Poslední statistikou, která má význam pro pozdější analýzu, je využití navigačních dat ve vyhledaných trasách. Jaké silnice jsou nejčastěji v trasách využívány? Existují nějaké, které nejsou využity vůbec nebo velmi zřídka? Odpovědi na tyto otázky pomohou při rozhodování, zda má smysl celý model dat držet v paměti. A pokud nemá, tak jakým způsobem s ním pracovat, aby načítání dat z disku nebo databáze aplikaci příliš nezpomalovalo.

V logových záznamech tyto informace bohužel nejsou zaznamenány. Nebyl však problém tyto statistiky zpětně získat. Postup byl následující:

Interval	Počet tras	Procento z celku
0 - 10 s	16911	6,70%
10 - 30 s	11047	4,38%
30 s - 1 min	6343	2,51%
1 - 2 min	6455	2,56%
2 - 5 min	7508	2,98%
5 - 10 min	3837	1,52%
10 min - 1 hod	4496	1,78%
1 hod - 24 hod	5716	2,27%
více jak 24 hod	30145	11,95%
Celkem	**92458**	**36,64%**

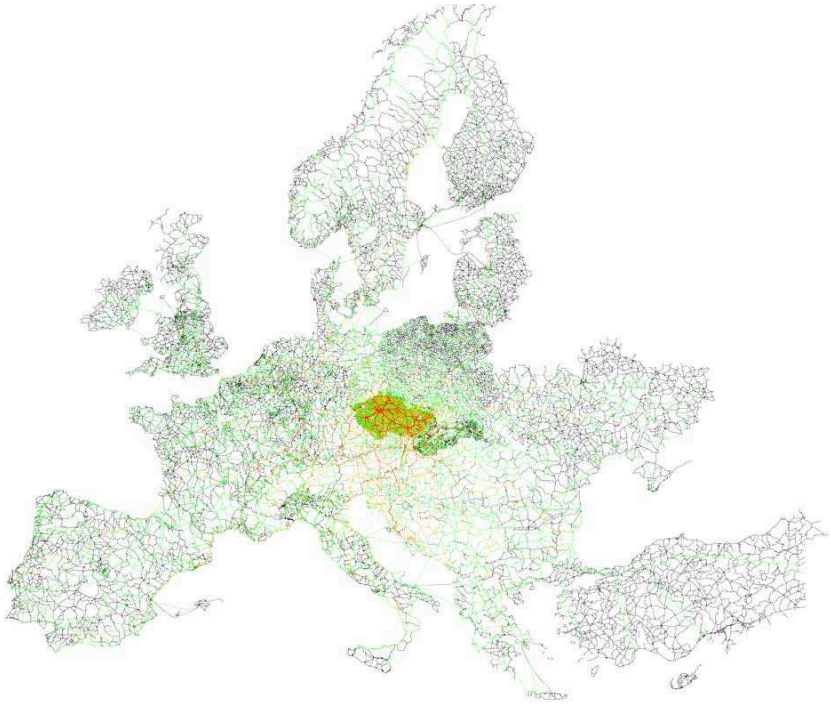
Tabulka 2.2: Četnosti opakování stejných tras v časových intervalech v období 1.8.2007 - 31.3.2008.

1. ze záznamu „O“ byla vytvořena úplná definice trasy,
2. na server byl poslán požadavek o vyhledání této trasy,
3. byl vyžádán její itinerář ve formátu XML,
4. analýzou tohoto XML byl pak získán seznam využitých komunikací v trase, které byly započteny do statistik.

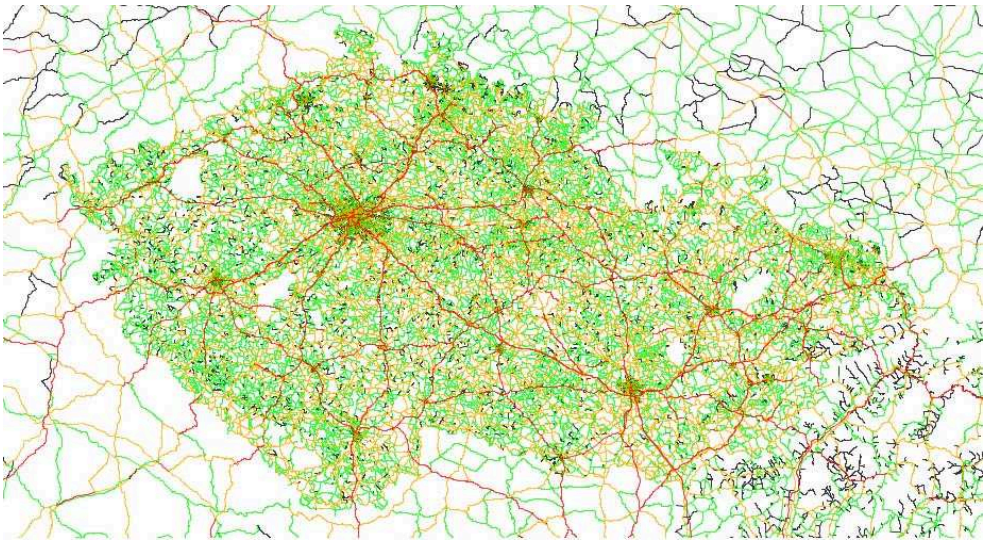
Výsledky byly velmi překvapivé. Z celkového počtu $168\,148$ úseků komunikací v navigačních datech bylo ve sledovaném období po vyhledání $250\,503$ tras využito pouze $101\,480$ úseků ($60{,}35\%$). Počet využitých úseků komunikací vyhledaných během jedné hodiny se pohybuje od desítek v nočních hodinách po maximálně $12\,199$ použitých různých úseků ($7{,}25\%$) naměřených během špičky. Průměrně je pak během jedné hodiny využito $3\,851$ úseků, což představuje pouze $2{,}29\%$ z celkového počtu úseků v navigačních datech.

Velká část datového modelu se tedy využije buď velmi zřídka nebo vůbec. Jak bylo řečeno v kapitole 1.3.2, největší část navigačních dat je tvořena vektory úseků cest. Pokud by tyto vektory byly načítány do paměti podle potřeby a v paměti bylo drženo pouze „rozumné“ množství těchto vektorů, mohli bychom ušetřit velké množství operační paměti. Na druhou stranu však navigační data nejsou v současné době nikterak velká a bez problémů se do paměti vejdou (cca 137 MB). V současné době tedy nebude nutné se tímto problémem zabývat, ale pokud do budoucna objem dat naroste, pak touto cestou bude možné udržet paměťové nároky navigačního serveru v rozumných mezích.

Grafické znázornění využití komunikací v mapě Evropy je k vidění na obrázku 2.5. Tyto informace lze též interaktivně zobrazit v ukázkové aplikaci popsané v kapitole 6.3.



(a) Evropa



(b) Česká republika

Obrázek 2.5: Využití komunikací ve vyhledaných trasách. Červená značí velmi často využívané komunikace, oranžová často využívané, zelená zřídka, černá nikdy nevyužité komunikace.

2.1.5 Oblíbené destinace

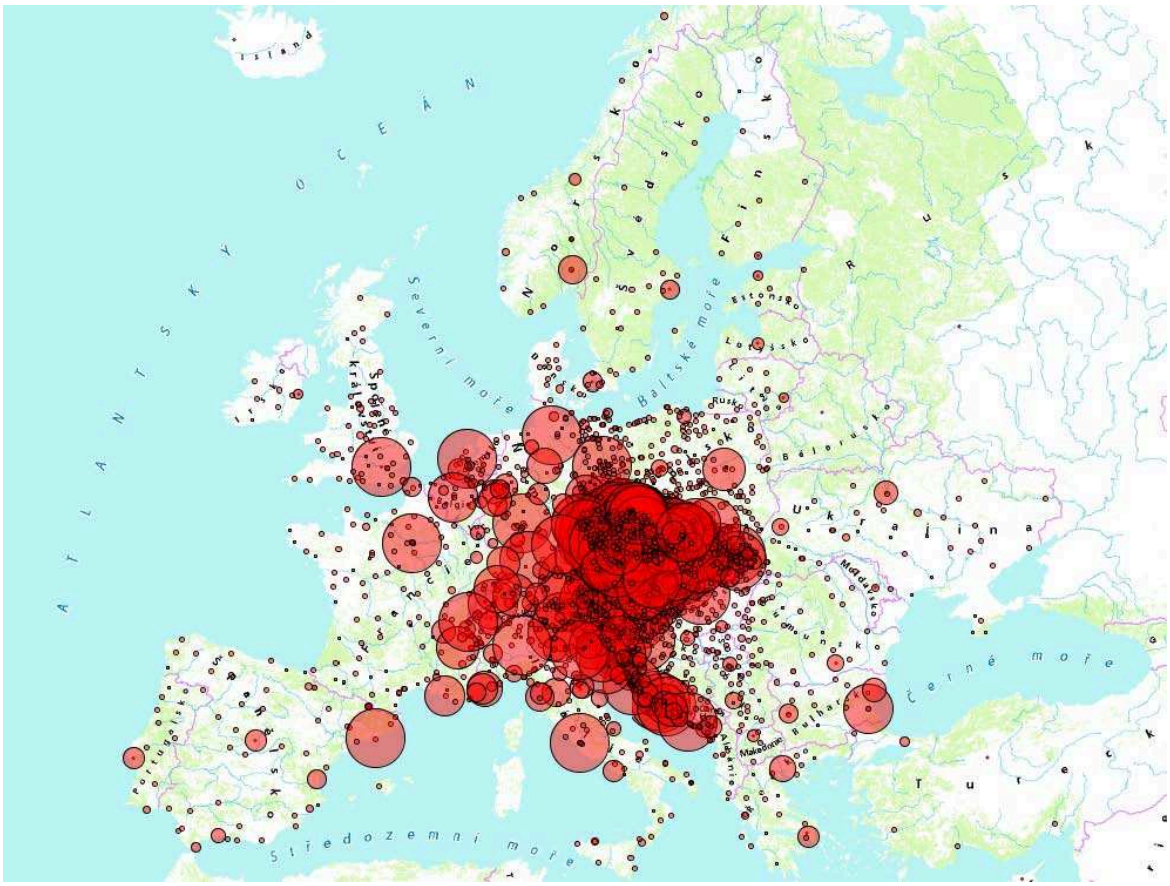
Jaké destinace uživatelé nejčastěji volí při vyhledávání tras? „O“ záznam o vyhledané trase obsahuje definice jejích destinací (viz. kapitola 3.3.1). Definice destinací obsahuje její souřadnici a případně identifikátor města. Pro každou destinaci s unikátní definicí byl spočítán její výskyt mezi všemi vyhledanými trasami.

Z těchto dat pak byla vytvořena průhledná mapová vrstva znázorňující destinace v mapě průhledným kolečkem. Čím větší poloměr kolečka, tím častěji byla destinace v trasách využita. Všechny destinace zobrazené nad mapou Evropy jsou k vidění na obrázku 2.6. Tyto informace lze též interaktivně zobrazit v ukázkové aplikaci popsané v kapitole 6.3. Kokrétkně jsou k dispozici průhledné vrstvy s destinací využitých ve všech automobilových nebo cyklistických trasách.

Obecně lze říci, že destinace se při plánování příliš neopakují. Pouze 856 destinací překonalo hranici 100-násobného opakování. Tabulka 2.3 obsahuje deset nejvyhledávanějších destinací. Pro každou souřadnici bylo určeno město, které je touto souřadnicí reprezentováno.

Souřadnice (UTM)	**Počet vyhledání**	**Město**
460317;5547176	21156	Praha
617187;5450367	11620	Brno
736603;5525630	5809	Ostrava
383019;5511610	5154	Plzeň
662686;5495891	4874	Olomouc
461504;5424844	4458	České Budějovice
559534;5562386	4307	Hradec Králové
555743;5543182	3692	Pardubice
503747;5623547	3435	Liberec
658351;5335475	3272	Bratislava

Tabulka 2.3: Nejoblíbenější destinace na serveru mapy.idnes.cz.



Obrázek 2.6: Oblíbené destinace zakreslené do mapy Evropy.

Kapitola 3

Analýza

V této kapitole budou zvoleny vhodné technologie a architektura serveru. Bude zvolen plánovací algoritmus a popsány optimalizace a úpravy tohoto algoritmu s ohledem na požadavky kladené na navigační server a navigační data, která jsou k dispozici. V poslední kapitole pak bude navržen způsob popisu vyhledané trasy s pomocí navigačních povelů.

3.1 Technologie

Pro implementaci navigačniho serveru byl zvolen programovací jazyk Java a technologie javovských servletů.

3.1.1 Java

Java byla zvolena pro svou multiplatformnost, programovací komfort a snadnou nasaditelnost. Díky multiplatformnosti je programátor odstíněn od individualit různých operačních systémů a architektur. Java poskytuje s pomocí virtuálního stroje stejné prostředí na všech systémech. Jazyk má velmi dobře vyřešeny problémy se znakovými sadami a kódováním češtiny.

Z hlediska programování jazyk Java nabízí přímo ve standardních knihovnách spoustu objektů pro běžnou i méně běžnou práci. Má velmi kvalitní knihovny pro práci s grafikou (AWT) s přímou podporou průhledných obrázků s alfa kanálem a pokročilými možnostmi vykreslování. Jazyk je navíc podporován širokou veřejností a je pro něj dostupná spousta knihoven třetích stran. Dalším velkým plusem je výborně vyřešena technologie vzdáleného ladění aplikace přímo na serveru. V kombinaci s NetBeans[1], nebo jiným kvalitním vývojovým prostředím jazyka Java, je možné velmi snadno ladit aplikaci přímo tam, kde je potřeba, s veškerým programátorským komfortem bodů přerušení, krokování atd.

[1] http://www.netbeans.org/

Technologie .NET byla zavržena kvůli nejisté funkčnosti pod systémem Unix / Linux[2]. PHP, Perl a jiné skriptovací za běhu interpretované jazyky byly zamítnuty pro přílišnou pomalost. S ohledem na rychlost běhu aplikace se uvažovalo o implementaci v jazyce C. Ten byl nakonec zamítnut z důvodu přílišné náročnosti na implementaci. Jazyk je příliš nízkoúrovňový a při programování se začíná prakticky od nuly. Nabízí se však možnost implementovat v jazyce C pouze určité „výpočetně náročné“ části, které by byly volány z Javy skrze interface JNI.[3]

3.1.2 Java servlety

Navigační server je určen pro práci v prostředí internetu. Jako komunikační protokole byl proto zvolen protokol HTTP (Hypertext Transfer Protocol[4]), který je v prostředí internetu standardem. Technologie javovských servletů (viz. [4]) umožňuje jeho implementaci skrze velmi jednoduché rozhraní. Při vyřizování HTTP požadavků (GET / POST) webový server volá servisní metody servletu a jejich výstup je pak odeslán zpět jako odpověď na daný požadavek.

Servlety mají jasně definovaný životní cyklus, jejichž podrobný popis naleznete v [4]. Tento cyklus je řízen kontejnerem, ve kterém jsou servlety nasazeny. V jeden okamžik je však inicializována maximálně jedna instance třídy servletu. Servisní metody této instance jsou pak volány vlákny vyřizující jednotlivé požadavky klientů. V jeden okamžik tedy s jednou instancí servletů může pracovat více vláken najednou.

Vzhledem k tomu, že navigačni server pracuje s velkým objemem dat (viz. kapitola 1.3.3), je naprosto vyloučené, aby se tato data načítala při každém požadavku znovu a znovu. Technologie servletů umožňuje jejich načtení a inicializaci při startu webového serveru a jejich sdílení během zpracování jednotlivých požadavků. Servlet může být držen v paměti po celou dobu běhu serveru. Tato efektivita je vykoupena nutností brát ohledy na problémy přístupu více vláken ke sdíleným prostředkům instance servletu.

3.1.3 Apache Tomcat

Servlety běží nad libovoným webový serverem, který tuto technologii podporuje. Server Apache Tomcat byl zvolen, protože je uznávaným standardem, je zdarma a snadno se instaluje a konfiguruje. K dispozici jsou instalační balíčky jak pro operační systém Windows, tak pro většinu distribucí systému Unix / Linux.

---

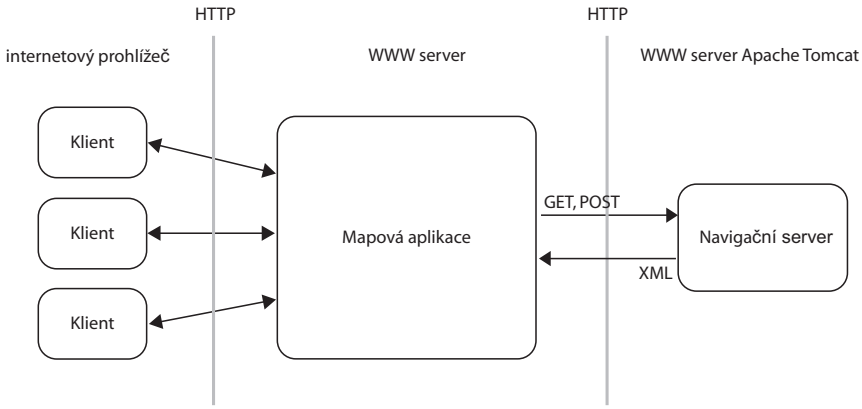
[2] V době kdy se o tomto rozhodovalo byl projekt Mono (http://www.mono-project.com/) umožňující provoz .NET aplikací na různých platformách v ranných stádiích vývoje

[3] Java Native Interface - http://java.sun.com/j2se/1.3/docs/guide/jni/

[4] http://www.w3.org/Protocols/rfc2616/rfc2616.html

3.1.4 Architektura

Jak bylo řečeno v úvodu, navigační server je jednou ze součástí on-line mapové aplikace. Architektura on-line mapové aplikace využívající navigační server je k vidění na obrázku 3.1. Ze schématu je patrné, že navigační server nekomunikuje s klienty (internetovými prohlížeči) přímo, ale pouze skrze mapovou aplikací. Tato architektura umožňuje umístit navigační server na jiný fyzický server nebo rozložit zátěž instalací navigačního serveru na několik strojů. Zároveň je však možné, aby více mapových aplikací přistupovalo k jednomu navigačnímu serveru. Architektura je tedy dobře škálovatelná.



Obrázek 3.1: Architektura mapové aplikace.

3.1.5 Rozhraní serveru

Vzhledem k velkému množství různých funkcí poskytovaných navigačním serverem jsou standardní možnosti HTTP rozhraní nedostačující. Rozhraní musí být dostatečně robustní a zároveň odolné vůči chybám na obou stranách. Server by měl být schopen spolupracovat s většinou současných programovacích jazyků a prostředí. Rozhraní by mělo být jednoduše použitelné.

V úmyslu přicházela implementace rozrhaní skrze vhodný protokol SOAP[5] (například jako webovou službu nebo RPC). Tyto protokoly jsou však příliš robustní a jejich použití na straně klientské aplikace často vyžaduje speciální knihovny (nebo spoustu práce) a zkušenosti programátorů s konkrétní technologií.

3.1.6 Příkazové rozhraní

Misto použití těchto standardů bylo nad protokolem HTTP navrženo jednoduché textové příkazové rozhraní. Navigační server zpracovává dva parametry HTTP požadavku. V paramteru *commands* uživatel definuje co chce, aby server vykonal. V parametru *return* určí co a v jakém formátu má server poslat na výstup.

[5]Simple Object Access Protocol - http://www.w3.org/TR/soap/

V obou parametrech je použit jednoduchý systém funkcí. Funkce navigačního serveru odpovídají funkcím nebo příkazům v klasickém programovacím procedurálním jazyku, tedy:

- každá funkce má svůj jedinečný název,
- každá funkce může mít libovolné množství parametrů,
- funkce může mít definován minimální počet parametrů,
- parametry mají jasně daný typ (celé číslo, desetinné číslo, řetězec, logická hodnota),
- parametry jsou od názvu funkce i od sebe navzájem oddělené středníkem,
- funkce vykoná přesně specifikovanou činnost s ohledem na své parametry.

Funkce, které lze používat v parametru *commands*, nazveme **příkazové funkce**. V parametru může být najednou uvedeno více funkcí oddělených znakem nové řádky. Funkce se pak vykonají v pořadí, ve kterém byly zadány. Funkce parametru *return* nazveme **návratové funkce**.

S pomocí příkazových funkcí lze například definovat destinace, zvolit kritéria a vyhledat trasu. Návratovou funkcí a jejími parametry se pak určí, co a v jakém formátu má být posláno na výstup (např. typ itineráře trasy). Nejlépe se vše vysvětlí na příkladu:

```
?commands=routingAddDestinationCoord;740736;5523726
routingAddDestinationCoord;725906;5508784
routingAddDestinationCoordCityId;677759;5480643;0663
routingSearchRoute;3;1;tr1
&return=getRoutingSimpleItinerary;tr1;3
```

První tři příkazové funkce definují destinace trasy. Čtvrtá příkazová funkce vyhledá trasu vedoucí přes tyto destinace (nejrychlejší s použitím placených úseků). Návratová funkce *getRoutingSimpleItinerary* pak na výstup pošle itinerář trasy ve stručném formátu s numerickými hodnotami zaokrouhlenými na tři desetinná místa.

Výhodou tohoto rozhraní je možnost jeho použití jak přes požadavek typu POST, tak typu GET. Problémový znak nové řádky lze nahradit jeho ekvivalentem „%0A“. Pro vyhledání trasy a získání jejího itineráře si tedy lze vystačit s vhodně definovaným URL. V některých případech postačí zadat pouze návratovou funkci a vynechat příkazové. Například pro získání informací o nejbližší komunikaci k dané souřadnici stačí poslat požadavek:

```
http://tomcat/servlet/?return=getNearestPathInfo;740736;5523726
```

Výstup

Všechny výstupy jsou formátovány v XML [6]. Tento formát poskytuje dostatečné možnosti pro přenos strukturovaných informací a je podporován většinou současných programovacích jazyků. Je určen především pro výměnu dat mezi aplikacemi. Umožňuje popsat strukturu dokumentu z hlediska věcného obsahu jednotlivých částí, ale nezabývá se sám o sobě vzhledem dokumentu nebo jeho částí. S jeho pomocí tedy lze dobře oddělit informace a jejich strukturu od jejich prezentace.

[6] Extensible Markup Language - http://www.w3.org/XML/

Výjimky

Jakákoliv výjimka, která nastane při vyřizování požadavku je oznámena klientské aplikaci speciálním chybovým XML, které je zasláno místo požadovaného výstupu. Výjimky upozorňují na celou řadu nepovolených operací:

- neznámý název funkce,
- špatný formát parametru funkce,
- chyby při definování destinací nebo plánování trasy,
- kritická vyjímka při pádu aplikace,
- a další.

Výsledné rozhraní je tedy dostatečně jednoduché, aby šlo použít v klasickém HTTP GET požadavku bez nutnosti formátování složitých XML příkazů, ale zároveň dostatečně robustní, aby umožnilo zadávání sady složitějších příkazů. Zároveň poskytuje dostatek ladících informací o chybách, které nastaly v průběhu vykonávání požadavku.

3.2 Vyhledávání tras

Hlavní funkcí navigačního serveru je samozřejmě vyhledávání tras. Nejprve shrňme požadavky z kapitoly 1.2.3 týkající se této problematiky. Server musí:

- umožňovat zadání destinace trasy jako libovolné souřadnice v mapě, počet destinací v trase nesmí být omezen,
- podporovat různá kritéria vyhledávání automobilových a cyklistických tras,
- vyhledávat trasy dostatečně rychle (výpočet a plánování trasy by měl trvat desítky, maximálně stovky milisekund).

Nejprve je nutné vybrat vhodný plánovací algoritmus. Při popisu algoritmů bude použita terminologie diskrétní matematiky. Z navigačních dat sestavíme orientovaný graf $G = (V, H)$, kde V je množina vrcholů a H je množina hran grafu G. Každý úsek komunikace bude v grafu reprezentován hranou h, každá křižovatka vrcholem v.

Dále zavedeme cenovou funkci f, která každé hraně H přiřazuje její cenu. Funkci $f(h)$ lze například položit rovno délce úseku komunikace, která je hranou h reprezentována. Funkce f pak je zobrazení z množiny H do množiny R^+. [7]

3.2.1 Vyhledávací struktura

Při plánování trasy bude velmi často nutné získat seznam hran, které vedou z vrcholu $v \in V$ do nejbližších sousedů. Bude tedy výhodné tyto informace předpočítat a vytvořit vyhledávací strukturu, která k těmto informacím umožní rychlý přístup (tzv. seznamy následníků).

[7] Úseky cest s nulovou nebo zápornou délkou neexistují.

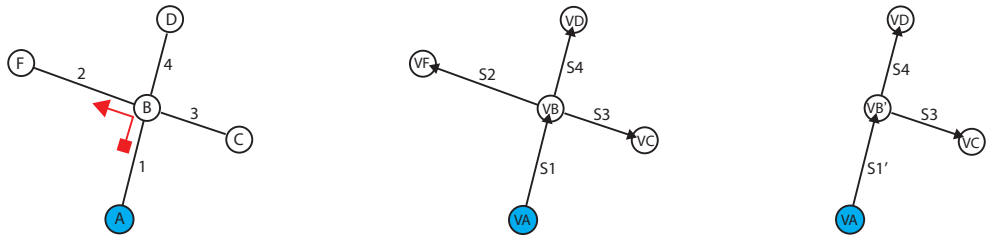
Pro každý vrchol v předpočítáme **množinu spojení** S_v **vrcholu** v, kde spojení $s \in S_v$ je dvojice (h, v_s), kde:

- h je hrana vedoucí z vrcholu v,
- v_s je vrchol, do kterého vede hrana h z vrcholu v.

Manévry

Při vyhledávání tras nám plánování komplikují tzv. manévry (1.3.2). Manévr je definován na dvojici (úsek, křižovatka) a omezuje možnosti odbočení na této křižovatce při příjezdu z daného úseku. Jedná se tedy o různé zákazy odbočení a přikázané směry jízdy.

Tato omezení výrazně komplikují plánování. Vezměme si situaci na obrázku 3.2 vlevo. Je zde znázorněna dopravní situace klasické křižovatky s křížením dvou komunikací ve vrcholu B. Z úseku 1 je však zakázáno odbočení na úsek 2. Při příjezdu z křižovatky A do B je tedy zakázáno odbočení na křižovatku F. Při příjezdu do B z křižovatek C a D však můžeme cestu do křižovatky F normálně použít. Aby nám tato omezení nekomplikovala a nezpomalovala plánovací algoritmus, je nutné upravit vyhledávací strukturu tak, aby odpovídala těmto omezením.



Obrázek 3.2: Schéma úpravy vyhledávací struktury podle manévru.

Pro každý vrchol v a každé jeho spojení $s \in S_v$, na kterém je definován manévr M, vytvoříme vrchol v_m. Tento vrchol bude obsahovat všechna spojení vrcholu v kromě těch, která jsou zakázána manévrem M. Spojení $s = (h, v_s)$ vrcholu v pak nahradíme spojením $s_m = (h, v_m)$.

Pro názornost je tato úprava znázorněna na obrázku 3.2. Nejprve je zduplikován vrchol VB do vrcholu VB'. Ze seznamu spojení vrcholu VB' pak jsou odstraněna všechna spojení, která jsou manévry zakázána při cestě z úseku 1 do křižovatky B. Ze seznamu spojení vrcholu VB' tedy vypadne spojení S2. Při příjezdu z křižovatky A do B tedy nebude odbočení na F vůbec k dispozici.

3.2.2 Plánovací algoritmus

Vyhledávání tras v silniční síti je speciálním případem vyhledávání cest v grafech a je řešena v rámci diskrétní matematiky a teorie grafů. Existuje celá řada algoritmů, mezi kterými bude nutné vybrat ten nejvhodnější. Při výběru plánovacího algoritmu je nutné brát ohledy na následující požadavky:

- rychlost algoritmu,
- podpora pro více kritérií a parametrů plánování,
- nároky na předzpracování dat.

Pro naše účely potřebujeme především algoritmus, který vyhledá optimální cestu z počátečního vrcholu do jednoho koncovéno vrcholu (tedy mezi dvěma vrcholy).

Dijkstrův algoritmus

První verze navigačního serveru implementovala Dijkstrův algoritmus. Hlavně pro jeho jednoduchost, univerzálnost a snadnou implementaci. Algroritmus pro daný počáteční vrchol vyhledá cesty s nejmenší cenou vedoucí z počátečního vrcholu do všech ostatních.

Pracuje se dvěma množinami vrcholů. První množina je tvořena **otevřenými vrcholy**, tedy vrcholy, pro které ještě nebyla vyhledána optimální cesta. Druhá množina obsahuje **uzavřené vrcholy**, pro které již optimální cesta byla vyhledána. U každého otevřeného vrcholu je evidována dočasná délka cesty z počátečního vrcholu do tohoto vrcholu. Při inicializaci je všem vrcholům nastavena dočasná délka na kladné nekonečno a počátečnímu vrcholu je přiřazena nulová délka. V každé iteraci algoritmu je vybrán a zpracován vrchol z množiny otevřených vrcholů s minimální dočasnou délkou cesty. Z tohoto vrcholu jsou pak prozkoumány hrany vedoucí do sousedních otevřených vrcholů. Pokud je dočasná délka sousedního vrcholu větší, než dočasná délka minimálního vrcholu plus cena zkoumané hrany, pak cesta do sousedního vrcholu vedoucí přes minimální vrchol má nižší cenu, než předchozí cesta do sousedního vrcholu. Minimální vrchol je poté označen za uzavřený a vyhozen z množiny nedefinitních vrcholů. Při každé iteraci algoritmu je tedy právě jeden otevřený vrchol prohlášen za uzavřený. Tento postup označme **zpracování vrcholu**.

Aby bylo možné po vyhledání optimální cesty „zrekonstruovat“ její průběh, je nutné u každého vrcholu evidovat z jakého vrcholu a za pomoci jaké hrany algoritmus do tohoto vrcholu dospěl. Zpětným průchodem z koncového vrcholu pak přes tato spojení dojdeme zpět do počátečního vrcholu a získáme seznam hran a vrcholů, které byly v cestě použity. Obrácením pořadí tohoto seznamu pak získáme posloupnost vrcholů a hran cesty z počátečního do koncového vrcholu.

Algoritmus pro daný počáteční vrchol vyhledá cesty s nejmenší cenou vedoucí z počátečního vrcholu do všech ostatních. Pro naše účely postačí vyhledání cesty do koncového vrcholu. Algoritmus tedy zastavíme v okamžiku, kdy je koncový vrchol prohlášen za uzavřený.

3.2.3 Haldy

Pro efektivní práci Dijkstrova algoritmu je velmi důležité vybrat vhodný způsob práce s otevřenými vrcholy. Existuje celá řada více či méně vhodných struktur, všechny spadají do katerorie tzv. hald. Haldy jsou datové struktury optimalizováné na získání minima z hodnot, které reprezentují.

Pro potřeby algoritmů potřebujeme, aby nad haldou byly efektivní následující operace:

- *insert* - přidání vrcholu do haldy,
- *decrease* - snížení hodnoty konkrétního vrcholu v haldě,
- *extractMin* - odstranění vrcholu s minimální hodnotou z haldy.

Algoritmus při každé iteraci provede jednou extractMin a podle počtu spojení, které z něj vedou, provede několikrát buď insert nebo decrease. V potaz byly brány tři typy hald: Fibonacciho, Binární a K-ární halda. Fibonacciho halda je popsána v publikaci [3].

Binární haldu si lze představit jako binární strom pro který platí, že je perfektně vyvážený, každý uzel může mít maximálně 2 potomky, jejichž hodnota je menší rovna hodnotě svého rodiče. Binární haldu je možné (a optimální) implementovat v poli. Rodič a potomek každého mohou být určeny s pomocí jednoduché aritmetiky s indexy pole. Pro každý prvek na indexu i se rodič nalézá na indexu $\lfloor (i-1)/2 \rfloor$, indexy potomků rodiče i lze najít na indexech $2i+1$ a $2i+2$.

K-ární halda je zobecnení binární haldy, ve které každý uzel může mít až k potomků. Rodiče pro prvek i lze nalézt na pozici $\lfloor (i-1)/k \rfloor$, potomky rodiče i pak na pozicích $ki+j$, kde $1 \leqq j \leqq k$.

Následující tabulka ukazuje asymptotické složitostí jednotlivých operací pro jednotlivé haldy (hvězdička označuje amortizovanou složitost, n je počet vrcholů v grafu, m počet hran):

Operace	**Binární**	**Fibonacci**	**K-ární**
deleteMin	$O(\log n)$	$O(\log n)^*$	$O(k \log_k n)$
insert	$O(\log n)$	$O(1)$	$O(\log_k n)$
decrease	$O(\log n)$	$O(1)^*$	$O(\log_k n)$
Dijkstrův alg.	$O(n \log n + m \log n)$	$O(n \log n + m)$	$O(kn \log_k n + m \log_k n)$

Z tabulky se zdá, že nejlepší volbou je Fibonacciho halda, jejíž asymptotické složitosti operací jsou nejnižší (ačkoliv některé amortizovaně). Průměrné hodnoty operací insert a decrease však u K-árních hald jsou však též O(1), protože při restrukturalizaci haldy často dojde pouze ke změnám na několika nejnižších úrovních a neprochází se celá výška stromu. Navíc díky možnosti reprezentace v poli ji lze implementovat efektivněji. Zátěžové testy ukázaly (viz. kapitola 5.2.3), že nejlepší volbou je K-ární halda s hodnotou $K = 3$ nebo $K = 5$.

Do haldy budeme vkládat pouze otevřené vrcholy, které mají dočasnou délku menší než nekonečno. Vrchol je tedy do haldy přidán až těsně před jeho prvním použitím a je z ní odstraněn poté, co je zpracován a prohlášen za uzavřený.

Nevýhodou Dijkstrova algoritmu je jeho přílišná obecnost. Algoritmus vyhodnocuje vrcholy striktně volbou a zpracováním aktuálního minima. Pokud bychom znázornili chování algoritmu při plánování nejkratší trasy z Českých Budějovic do Hradce Králove (obrázek 3.3), vidíme, že algoritmus prohledává prostor rovnoměrně ve všech směrech. Prozkoumaná oblast tedy tvoří nepravidelný kruh s postupně se zvětšujícím průměrem.

Je zřejmé, že algoritmus zkoumá zbytečně velký prostor. Nepočítá s informacemi, které jsou k dispozici při plánování tras v silniční síti a které by mu umožňovali lépe směrovat vyhledávání a zredukovat tak počet zkoumaných vrcholů. Takovýmto algoritmem je A star.

A star

A star (A^*) je svou stavbou téměř identický s Dijkstrovým algoritmem. Narozdíl od něj však při výběru vrcholu ke zpracování počítá kromě hodnoty cenové funkce f i s tzv. heuristickou funkcí, kterou budeme označovat písmenem h. Ke zpracování je pak vybírán vrchol s minimální hodnotou funkce g, kde

$$g(v) = f(v) + h(v).$$

Zatímco cenová funkce $f(v)$ vrací cenu cesty z počátečního vrcholu do vrcholu v, funkce heuristická $h(v)$ dává odhad ceny cesty z vrcholu v od cíle. Funkce g tedy dává odhad ceny cesty z počátečního vrcholu do cíle. Heuristická funkce, pokud je zvolena vhodně, umožňuje lépe směrovat vyhledávání a cesta je pak nalezena rychleji a za menšího počtu zpracovaných vrcholů.

Z článku [5] víme následující:

- A star vyhledá optimální trasu, pokud je heuristická funkce h optimální.
- Funkce h je optimální, pokud pro každý vrchol v nevrátí menší odhad ceny, než je reálná minimální cena cesty z vrcholu v do cíle.
- Pokud funkce h zobrazuje do otevřené množiny, pak musí být monotónní.

V případě vyhledávání silničních tras je ideální funkci h definovat jako eukleidovskou vzdálenost mezi křižovatkou reprezentovanou vrcholem v a vrcholu cílové destinace. Funkce h pak bude nabývat menších hodnot u vrcholů, které jsou blíže k cíli. Tyto vrcholy pak budou zpracovány dříve a algoritmus bude rychle postupovat „směrem“ k cílové destinaci. Takto definovaná heuristická funkce je jak optimální, tak monotónní a algoritmus pak dle [5] vyhledá optimální cestu.

Optimálnost heuristické funkce je vázána na cenovou funkci. Vzhledem k tomu, že různá kritéria plánování tras mají různé cenové funkce, je nutné navrhnout heuristickou funkci pro každou z nich zvlášť. Tímto problémem se bude zabývat kapitola 3.2.5.

Od Dijkstrova algoritmu se A star liší využitím heuristické funkce. Do haldy, která má hlavní slovo při volbě dalšího zpracovávaného vrcholu, vkládáme vrcholy ohodnocené funkcí $g = f + h$. U vrcholů však dále evidujeme pouze hodnotu $f(v)$, tedy cenu cesty do vrcholu v bez započtené heuristiky.

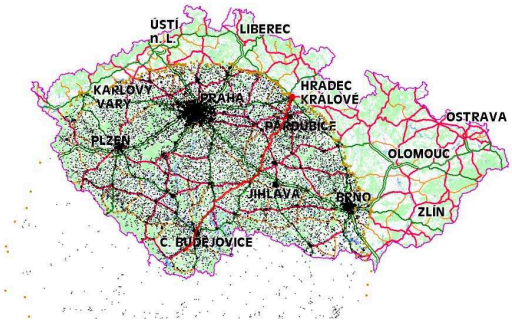
Narozdíl od Dijkstrova algoritmu nemáme zaručené, že do vrcholu v, který byl jednou prohlášen za uzavřený, neexistuje cesta s menší cenou. Při zpracování vrcholu je tedy nutné kontrolovat všechny cesty vedoucí z aktuálního vrcholu včetně těch, které vedou do již zpracovaných definitních vrcholů. Jeden vrchol tedy může být zpracován více jak jednou.

Algoritmus A star je kompromisem mezi „bezpečným“ vyhledáváním do šířky (cenová funkce f) a „rychlým“ vyhledáváním do hloubky (heuristická funkce h). Jak je ale patrné z obrázků 3.3, A star oproti Dijsktrovi zpracuje při vyhledávání trasy mnohem méně vrcholů. Důkladné porovnání obou algoritmů je součástí kapitoly zátěžové testování (kapitola 5). Algoritmus ve formě Java pseudokódu lze nalézt v algoritmu číslo 1 na stránce 35.

Algoritmus 1 A star

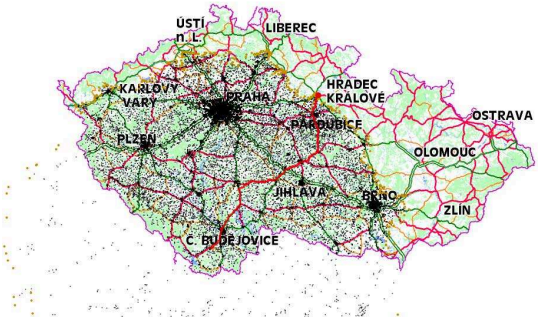
Vyhledávací kritérium je reprezentován třídou *Criterium*. Její metoda *f(L)* vrací hodnotu cenové funkce spojení *L*. Metoda *h(v, ENDV)* vrací hodnotu heuristické funkce, tedy odhad ceny cesty z vrcholu *v* do cílového vrcholu *ENDV*.
Spojení je zastoupeno třídou *Link*, která obsahuje jednak informace umožňující výpočet ceny tohoto spojení metody *f*, druhak pak obsahuje odkaz na vrchol, do kterého vede (člen *VK*).
Vrchol je reprezentován třídou *Vertex*. Jejími členskými proměnými jsou: *linkList* - seznam spojení vedoucí z daného vrcholu, *closed* - příznak určující zda je vrchol uzavřen, *prevV, prevL* - odkazy na předchozí vrchol a spojení v současné optimální cestě, *length* - cena současné cesty z počátečního do tohoto vrcholu, *heapValue* - odhad ceny optimální cesty z počátečního do cílového vrcholu procházející tímto vrcholem.
Halda je zastoupena třídou *Heap*, která má metody *insert*, *decrease*, *extractMin* popsané v kapitole 3.2.3. S vrcholy pracuje podle hodnot proměnné *heapValue*.

```
boolean findRouteAstar(Vertex STARTV, Vertex ENDV, Heap HEAP, Criterium CRIT)
{
  //inicializace
  for (V˜in VST) {
    V.heapValue = MAX_DOUBLE; V.length = MAX_DOUBLE;
    V.prevV = null; V.prevL = null; V.closed = false;
  }
  STARTV.length = 0; STARTV.heapValue = 0;
  HEAP.insert(STARTV);

  // hlavni vyhledávací cyklus
  while (!ENDV.closed) {
    if (HEAP.isEmpty())
      return false; // cesta nebyla nalezena

    Vertex VMIN = HEAP.extractMin();
    for (Link L in VMIN.linkList) {
      Vertex VK = L.VK;
      if (VMIN.length + CRIT.f(L) < VK.length) {
        VK.length = VMIN.length + CRIT.f(L);
        VK.heapValue = VMIN.length + CRIT.f(L) + CRIT.h(VK, ENDV);

        if (HEAP.contains(VK)) {
          HEAP.decrease(VK);
        } else {
          HEAP.insert(VK);
        }

        VK.prevV = VMIN;
        VK.prevL = L;
      }
    }

    VMIN.closed = true;
  }

  return true;
}
```

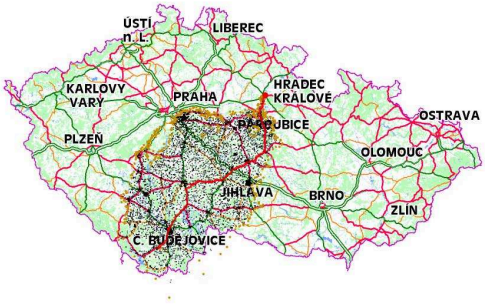


(a) Dijkstra - nejkratší trasa (43,3%)

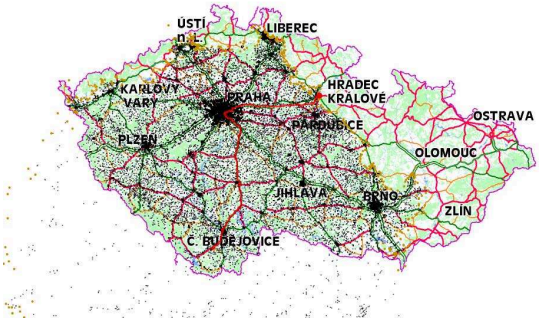
(b) A star - nejkratší trasa (9,1%)



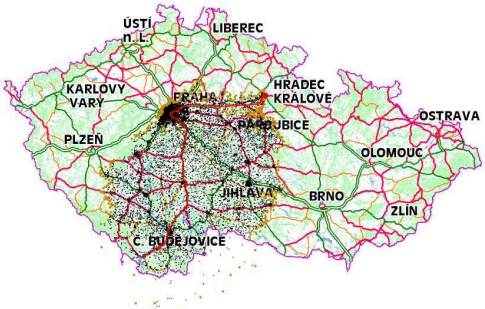
(c) Dijkstra - ekonomická trasa (44,2%)



(d) A star - ekonomická trasa (11,6%)



(e) Dijkstra - nejrychlejší trasa (49,1%)



(f) A star - nejrychlejší trasa (25,0%)

Obrázek 3.3: Vizualizace vrcholů zpracovaných vyhledávacími algoritmy v mapě. Uzavřené vrcholy jsou označené černou barvou, otevřené žlutou barvou a vyhledaná trasa červenou barvou. V závorce je vždy uvedeno procento zpracovaných vrcholů z celkového počtu vrcholů v grafu potřebných při vyhledání trasy.

Hlavní výhodou algoritmu A star oproti Dijkstrovi je lepší výběr zpracovávaných vrcholů. Při vyhledání stejné trasy tedy A star zpracuje mnohem menší počet vrcholů. Algoritmy popsané v publikacích [6] a [1] slibují další zefektivnění a zrychlení vyhledávání trasy, často dalším snížením počtu zpracovaných vrcholů.

SPAH

Algoritmus **SPAH** pracuje nad speciální hierarchickou strukturou pojmenovanou HiTi graf. Tento graf rozkládá původní graf navigačních dat do více menších podgrafů. Pro každý podgraf jsou vypočítány vzdálenosti mezi všemi jeho hraničními vrcholy (vrcholy, z nichž vede hrana do sousedního podgrafu). Následně je vytvořen graf, jehož vrcholy jsou tvořeny hraničními vrcholy podgrafů a hrany reprezentují optimální cesty mezi těmito vrcholy. Nový graf má tedy méně vrcholů, protože neobsahuje vnitřní vrcholy podgrafů. Tento postup lze opakovat i nad novým grafem a počet vrcholů dále redukovat. Získáme tím hierarchii grafů, díky které pak lze drasticky omezit množství zpracovaných vrcholů.

Algoritmus tedy vyžaduje časově i algoritmicky náročné předzpracování navigačních dat při kterém se vytvoří jednotlivé úrovně HiTi grafu. Tento graf je však vázán na jednu konkrétní cenovou funkci s jejíž pomocí byl vytvořen a lze ho pak tedy použít pouze pro jednu konfiguraci vyhledávacího kritéria a parametru. Vzhledem k tomu, že navigační server bude pravděpodobně pracovat s více jak 12 různými cenovými funkcemi (6 základních kritérií vyhledávání, ke každému je zadán dodatečný parametr), bylo by nutné předpřipravit 12 různých HiTi grafů. Na druhou stranu by bylo možné využít tento algoritmus prouze pro nejpoužívanější kritéria (vyhledání nejrychlejší trasy - 41% všech vyhledaných tras) a na zbytek použít klasický A star.

V publikaci [6] je důkladně popsáno nasazení algoritmu SPAH v serverovém prostředí (např. řeší optimalizace při výpočtu více tras v jeden okamžik - ISPAH) a je rozebrána možnost zapracování aktuálních dopravních omezení do vyhledávacích struktur bez nutnosti jejich úplného přepočítání. Dle měření autorů je SPAH oproti algoritmu A star mnohem efektivnější při rostoucí délce vyhledávané trasy. Při vyhledávání velmi dlouhých tras tak může být až 3x rychlejší rychlejší.)

REAL

Publikace [1] popisuje celou řadu optimalizací, které pak zkombinuje v algoritmu **REAL**. Algoritmus opět vyžaduje předpočítání celé řady údajů. Nejzajímavější optimalizační technika využívá tzv. „dosah“ vrcholu (reach). Mějme optimální cestu P, která vede z počátečního vrcholu s do koncového vrcholu t přes vrchol v. Dosah vrcholu v vzhledem k cestě P, $r_P(v)$, je roven minimu z ceny cesty z $s \rightarrow v$ (prefix) a ceny cesty $v \rightarrow t$ (postfix). Dosah vrcholu v, $r(v)$ spočítáme jako maximum z hodnot $r_P(v)$ všech optimálních cest P vedoucích přes vrchol v.

Dosah vrcholů pak lze velmi efektivně využít k „ořezávání“ vrcholů, které nemá vůbec smysl zkoumat. Platí: pokud $\bar{r}(v) < \underline{dist}(s,v)$ a $\bar{r}(v) < \underline{dist}(v,t)$, pak vrchol v nemůže být na optimální

cestě z vrcholu s do vrcholu t a plánovací algoritmus tedy tento vrchol nemusi zpracovávat ($\bar{r}(v)$ je horní odhad dosahu vrcholu v, $\underline{dist}(a,b)$ je dolní odhad ceny optimální cesty $a \to b$).

Použití dosahů vrcholů pak lze velmi jednoduše zapracovat do algoritmu A star. Při zpracovávání vrcholu v máme k dispozici jak dolní odhad ceny cesty z $s \to v$ (evidujeme ji u každého vrcholu), tak dolní odhad ceny cesty $v \to t$ (hodnota heuristické funkce dává vždy dolní odhad ceny z vrcholu do cíle). Pokud jsou obě hodnoty menší než $\bar{r}(v)$, pak vrchol nebudeme zpracovávat. V zásadě tedy přibudne pouze jedna podmínka navíc.

Výpočet dosahů pro všechny vrcholy v grafu je výpočetně velmi náročný. Nejjednodušší způsob je vyhledaní optimálních cest mezi všemi vrcholy v grafu a určení dosahu aplikování definice. Takovýto výpočet by však trval na velkých grafech neúnosně dlouho. V publikaci [1] proto navrhují efektivní (a složitý) algoritmus, který běží v rozumném čase.

SPAH i REAL jsou vcelku složité algoritmy a jejich implementace spolu se zapracování některých úprav (obchvaty, dopravní omezení) by byla časově náročná. Pro svou flexibilitu, jednoduchost a dostatečnou rychlost byl tedy nakonec zvolen **A star**. Do budoucna by však v úvahu přicházela implementace některé z optimalizací navržených v [1].

3.2.4 Destinace

V kapitole 1.2.1 byly popsány různé možnosti zadávání destinací v mapové aplikaci. Třetí bod pojednává o možnosti přidávání destinace v libovolném bodě zobrazené mapy. Destinace zadány dalšími dvěmi způsoby (sídlo / adresa, bod zájmu) jsou obvykle v konečném důsledku reprezentovány souřadnicí. Pro oblasti je zvolen vhodný reprezentativní bod. V případě měst se často jedná o střed náměstí, v případě obcí polohu obecního úřadu nebo kostela. Bude tedy stačit vyřešit pouze tento nejobecnější způsob a předpokládat, že destinace může být umístěna na libovolné souřadnici. Z toho plyne několik problémů:

- Uživatel očekává, že trasa bude začínat co nejblíže k místu, které vybral v mapě. Není tedy možné, aby trasa začínala a končila na nejbližší křižovatce, která může být stovky metrů až kilometry daleko. Trasa musí tedy začínat v nejbližším bodě ležícím na **vhodné blízké komunikaci**. Vhodná blízká komunikace nemusí být nutně nejbližší. Její výběr je omezen dopravními omezeními dopravního prostředku, pro který se trasa vyhledává. Vhodný blízký úsek komunikace k dané destinaci nazveme **navigační úsek destinace**. Nejbližší bod k destinaci ležící na vektoru tohoto úseku pak označíme pojmem **navigační bod destinace**. Postup, kterým tuto komunikaci vyhledáme nazveme **vyhledání navigačního úseku**.
- Možností, kde může být destinace umístěna, je „nekonečně“ mnoho. Navigační úseky a body destinací tedy není možné předpočítat.
- Oba plánovací algoritmy vyhledávají cesty z vrcholu do vrcholu. Bude tedy nutné upravit algoritmy nebo vyhledávací strukturu tak, aby bylo možné začít vyhledávání v nejbližším bodě na navigačním úseku.

- K zadané souřadnici nemusí být vhodná blízká komunikace vůbec k dispozici. Vzhledem k různé hustotě navigačních dat se může stát, že nejbližší vhodná komunikace je desítky až stovky kilometrů daleko. V takovémto případě nemá smysl trasu vyhledávat a je nutné upozornit uživatele hozením vyjímky.

Vyhledání navigačního úseku a bodu destinace

Jak bylo řečeno v kapitole 1.3.2, každý úsek komunikace má přiřazen podrobný vektor. Ačkoliv byly tyto vektory původně určeny pro vykreslení úseků do mapy, lze je využít i pro vyhledání vhodné blízké komunikace k souřadnici destinace. Na vektoru je pak možné vyhledat navigační bod destinace, tedy nejbližší bod vzhledem k souřadnici destinace. Tento bod se pak využije při vyhledávání trasy jako její počátek nebo konec na místo původní souřadnice destinace.

Vzhledem k tomu, že destinace může mít libovolnou souřadnici, nelze navigační body jakkoliv předpočítat. Problémem tedy je: Jak velmi rychle vyhledat pro každou destinaci ve trase její navigační bod mezi 168 148 vektory úseků komunikací, kde každý vektor může obsahovat až stovky bodů? Je naprosto vyloučené, abychom procházeli každý vektor bod po bodu. Na místo toho využijeme vhodných prostředků, abychom nejprve odfiltrovali velkou část vektorů. Zůstane nám pouze malá množina kandidátů, kterou pak podrobíme důkladnějšímu zkoumání a vybereme z nich ten nejvhodnější.

K rychlému odfiltrování vhodných kandidátů bude použit R strom. R strom je stromová datová struktura podobná B stromům upravena pro efektivní uložení a vyhledávání prostorových informací. Jejich podrobný popis lze nalézt v [2]. V R stromu je každý prvek reprezentován tzv. minimální obálkou, což je nejmenší obdélník, který obsahuje všechny body reprezentovaného prvku. Tyto obálky jsou pak shlukovány podle určitého pravidla do skupin, která je pak reprezentována minimální obálkou vzhledem k obálkám prvků ve skupině. Skupiny se mohou dále seskupovat do skupin skupin, čímž vznikne hierarchická stromová struktura. V R stromu pak lze vyhledávat zadáním obdélníkové oblasti, přičemž vráceny jsou všechny prvky, jejichž minimální obálka má s vyhledávanou oblastí neprázdný průnik.

Pro každý vektor spočítáme nejmenší obdélník, který obsahuje všechny body vektoru (**minimální obálka vektoru**). Z obálek vektorů poté vystavíme R strom[8].

Pro vyhledání malé množiny kandidátských vektorů pak:

1. Vezmeme souřadnicový obdélník s vhodným poloměrem a se středem v bodu destinace.
2. S pomocí R stromu vyhledáme vektory, jejichž obálky mají s tímto obdélníkem neprázdný průnik.
3. Pokud nebyly nalezeny žádné vektory, postupně zvětšujeme poloměr obdélníku a vyhledávání opakujeme.

[8] Tato operace je časově náročná, proto je dobré strom předpočítat v rámci předzpracování navigačních dat.

Pokud se poloměr vyhledávání zvětší nad maximální hodnotu, vyhledávání zastavíme a hodíme vyjímku oznamující, že vhodný navigační úsek nebyl nalezen[9].

Vzhledem k tomu, že začínáme s obdélníkem o malém obsahu a zvětšujeme ho až v případě neúspěchu, nebude množina kandidátských vektorů nikdy příliš velká. V této množině pak musíme vyhledat nejbližší vektor a nejbližší bod ležící na tomto vektoru k bodu destinace (označme ho b_d).

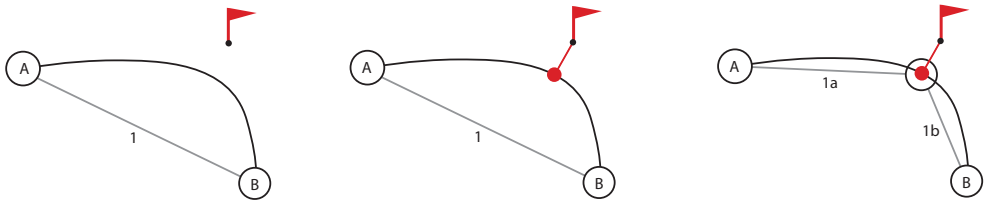
Pro každé po sobě jdoucí body vektoru v_i a $v_i + 1$ spočítáme vzdálenost bodů b_p a b_d, kde b_p je průsečíku úsečky určné body v_i a $v_i + 1$ a kolmice k této usečce, která prochází bodem b_d. Pokud průsečík existuje, pak vzdálenost bodu b_d od úsečky je rovna eukleidovské vzdálenosti průsečíku b_p a bodu destinace b_d. Pokud takto projdeme všechny úsečky sestavené z po sobě jdoucích bodů vektoru, získáme minimální vzdálenost bodu destinace od tohoto vektoru. Vyhledáním minima vzdáleností přes všechny vektory v kandidátské množině získáme hledaný navigační úsek a bod destinace.

Funkci vyhledávání vhodné blízké komunikace bude moci využít pro implementaci požadavku na poskytování informací o blízké komunikaci k souřadnici zadané uživatelem.

Plánování cesty s použitím navigačních bodů destinací

Algoritmy Dijkstra a A star umí vyhledat cesty, které začínají a končí ve vrcholech grafu. Jak ale bylo řečeno v předchozí kapitole, trasa musí začínat v co nejbližším bodě vzhledem k zadané destinaci. Aby toto bylo možné, bude nutné upravit ještě před spuštěním plánovacího algoritmu vyhledávací strukturu tak, aby navigační bod destinace byl reprezentován vrcholem v grafu G.

Postup úpravy navigačních dat a vyhledávací struktury je zachycen na schématu 3.4. K destinaci označené červenou vlaječkou je nejprve vyhledán nejbližší bod na vektoru navigačního úseku destinace. V tomto bodě je pak vytvořena křižovatka a úsek destinace je podle této křižovatky rozdělen na dvě části. Ve vyhledávací struktuře pak je pak hrana 1 mezi vrcholy A a B odstraněna a nahrazena hranami 1a a 1b a novým vrcholem reprezentující dočasnou křižovatku. Tento vrchol pak bude použit jako počáteční / koncový vrchol při plánování cesty z / do této destinace.



Obrázek 3.4: Rozdělení navigačního úseku podle navigačního bodu destinace.

[9] Maximální hodnota by se měla pohybovat v řádech kilometrů až desítek kilometrů. Vzdáleňejší komunikace už pak obvykle nemá smysl vyhledávat.

Je důležité si uvědomit, že v jeden okamžik může server paralelně vyhledávat více tras najednou. Úpravy však není nutné ve struktuře provádět. Na místo toho inicializujeme před zahájením vyhledávání vrcholy A i B. Vrchol A na hodnotou 1a, vrchol B hodnotou 1b.

Plánování cesty přes více destinací

Jedním z požadavků kladených na navigační server je plánování cest přes více destinací. Tato funkce umožní uživateli například detailní zadání průjezdních bodů trasy při plánování cyklistického výletu. Implementace této funkce není nikterak složitá. Plánování cesty vedoucí přes všechny destinace se rozdělí na plánování **dílčích cest** vedoucích vždy z přechozí destinace do následující. Při plánování cesty o N destinacích bude tedy nutné vyhledat celkem $N-1$ dílčích cest ($1 \rightarrow 2, 2 \rightarrow 3, \ldots, N-1 \rightarrow N$). Spojením dílčích výsledků pak získáme hledanou cestu, která postupně prochází všemi destinacemi.

Obchvaty

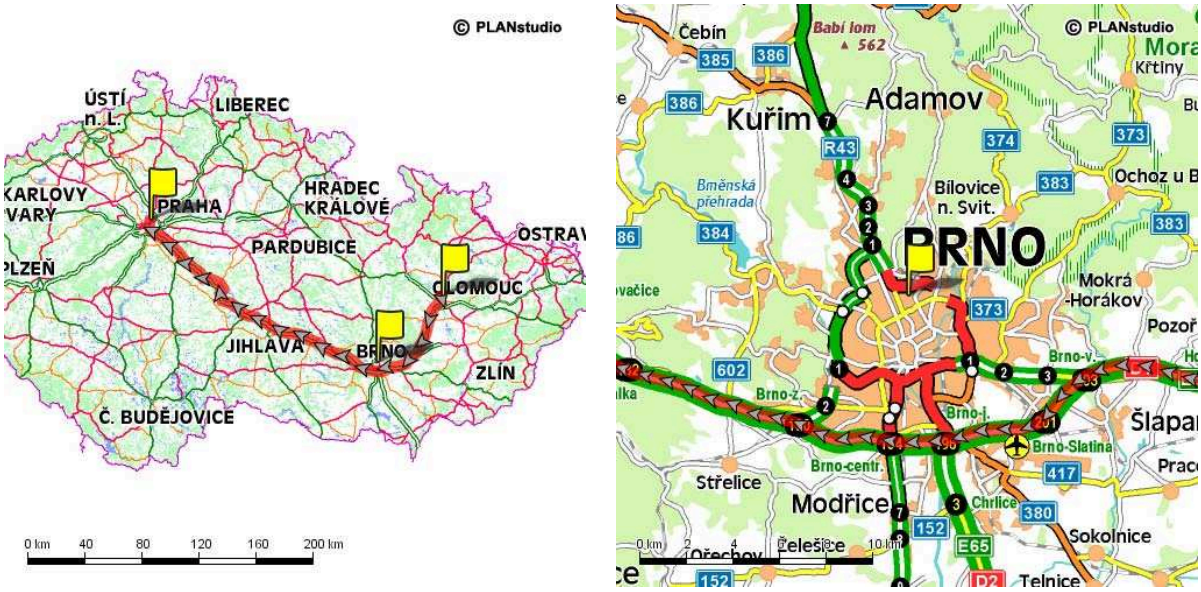
Ve kapitole 1.2.1 byl představen problém reprezentace celého města či obce jednou souřadnicí. Například město Praha má svůj reprezentativní bod umístěn na Vinohradské ulici[10]. Tento jednoduchý způsob je vhodný pro většinu případů užití, ale jak už to tak bývá, existuje situace, pro kterou tato reprezentace není vhodná. Vezměme v potaz například trasu z Českých Budějovic do Liberce přes Prahu. Plánovací algoritmus nejprve vyhledá cestu z Budějovic do Prahy a pak z Prahy do Liberce. Problém je, že Praha je reprezentovaná souřadnicí v centru a výsledná trasa tedy povede do bodu na Vinohradské ulici. Původní záměr uživatele zcela jistě nebyl zajíždět do centra aby uvízl v zácpách, ale pouze jet „přes“ respektive „okolo“ Prahy (jinak by zadal přesnější adresu).

Z tohoto důvodu bylo představeno vyhledávání cest s použitím **obchvatů** měst. U každého úseku komunikace v navigačních datech je definováno, zda náleží k obchvatu města (kapitola 1.3.2). Tyto úseky obvykle dohromady tvoří okruh okolo celého nebo části města. Při plánování cesty do tohoto města cesta velmi pravděpodobně povede přes křižovatku jednoho z těchto úseků.

Při plánování cesty přes destinaci d_c reprezentující dané město c pak stačí kontrolovat, zda vrchol, který se chystáme prohlásit za definitní, náleží k obchvatu města c. Pokud ano, pak vyhledávání dílčí cesty do destinace d_c ukončíme a z tohoto vrcholu pak začneme vyhledávat navazující dílčí cestu do následující destinace.

Pro názornost uvádím příklad vyhledání trasy z Olomouce do Prahy přes Brno. V Brně je použit obchvat, přičemž obchvat Brna je identifikován na základě UIRADR města Brna (582786). Výsledek je k vidění na obrázku 3.5. Z detailu vpravo je patrné, že algoritmus ukončil vyhledávání cesty z Prahy do Brna na úseku dálnice D1, který je definován jako součást obchvatu města Brna. Z této křižovatky na dálnici pak pokračoval v plánování cesty do města Olomouc.

[10] WGS84 souřadnice: 50˚4’32”N 14˚26’43”E



Obrázek 3.5: Vyhledaná trasa s obchvatem města Brna. Vlevo celá, vpravo detail obchvatu okolo Brna.

3.2.5 Kritéria plánování tras

Navigační server musí podporovat několik způsobů (kritérií) vyhledávání trasy (viz. požadavky v kapitole 1.2.2). Podle použitého dopravního prostředku lze kritéria rozdělit do dvou skupin: na automobilové a cyklistické. Každý dopravní prostředek má určitá omezení vycházející z reálných omezení tohoto dopravního prostředku na silničních komunikací:

- Automobily nemohou využívat lesní pěšiny a silnice vyhrazené cyklistům.
- Cyklisté nemohou využívat dálnice a rychlostní silnice.

Tato omezení ovlivňují nejen plánování tras, ale také vyhledávání vhodné blízké komunikace k destinacím trasy (viz. kapitola 3.2.4). Trasa automobilu nesmí začít na lesní pěšině a cyklista zase nemůže začít na dálnici.

Ke každému kritériu vyhledávání trasy lze podle požadavků definovat jeden parametr, který nějakým způsobem omezuje nebo penalizuje určité úseky komunikací. U automobilových kritérií se jedná o omezení placených úseků silnic. U cyklistických o omezení silnic první třídy.

Pro každou kombinaci kritéria a parametru musí být zvolena vhodná cenová funkce f, která pak bude použita plánovacím algoritmem při vyhledání trasy. Ke každé cenové funkci je potřeba určit i heuristickou funkci, která je vzhledem k cenové funkci optimální.

Nejkratší automobilová trasa

Toto kritérium má za úkol vyhledat nejkratší cestu ze startu do cíle. Budeme se tedy snažit minimalizovat celkovou délku vyhledané trasy. Zvolíme tedy cenovou funkci f takto:

$$f(s) = \text{délka úseku s v km}$$

Heuristickou funkci pak postačí definovat jako vzdušnou vzdálenost vrcholu do cíle. Tato vzdálenost nikdy nemůže být menší, než silniční vzdálenost do cíle (ledaže se jedná o chybu v datech).

$$h(v) = DIST(V, ENDV).$$

Nejrychlejší automobilová trasa

Toto kritérium má za úkol vyhledat cestu ze startu do cíle, kterou automobil projede co nejrychleji. Budeme se tedy snažit minimalizovat celkový čas potřebný k absolvování vyhledané trasy.

$$f(s) = \frac{\text{délka úseku s v km}}{\text{maximální rychlost v km/h automobilu na úseku S}}$$

Zjištění maximální povolené rychlosti probíhá následovně. Pokud je rychlost explicitně definována v záznamu úseku, pak je použita. V opačném případě se použije implicitní maximální rychlost definovaná v kategorii komunikace tohoto úseku. U každé kategorie komunikací jsou definovány dvě maximální rychlosti pro komunikace ve městě a mimo město. S ohledem na to, zda se konkrétní úsek nalézá či nenalézá ve městě je pak zvolena správná hodnota maximální rychlosti.

Abychom zajistili správnost a optimálnost heuristické funkce, musíme ji definovat následovně:

$$h(v) = \frac{DIST(V, ENDV)}{\text{maximální povolená rychlost na všech komunikacích}}.$$

Maximální povolená rychlost evidovaná v datech firmy PLANstudio je 130 km/h. S takto definovanou heuristickou funkcí máme jistotu, že žádný její odhad nebude nadhodnocen (menší než je skutečná cena cesty do koncového vrcholu) a algoritmus tím pádem vyhledá optimální (nejrychlejší) cestu.

Ekonomická automobilová trasa

Toto kritérium je kompromisem mezi nejkratší a nejrychlejší automobilovou trasou. Chceme vyhledat takové trasy, které sice budou preferovat rychlejší komunikace, ale nikoliv za cenu velkých zajížděk. Abychom tohoto docílili, bude stačit vhodně shora omezit maximální rychlost vozidla. Hlavním kritériem bude opět doba potřebná k absolvování vyhledané trasy, ale při omezení maximální rychlosti nebudou dálnice a rychlostní komunikace už tak výhodné.

Praxe a experimenty ukázaly, že optimální omezení maximální rychlosti je dobré stanovit na 80 nebo 90 km/h. Při použití 80 jsou všechny silnice postaveny na stejnou úroveň s kvalitními silnicemi druhé třídy. Při maximální rychlost 90 km/h pak na úroveň silnic prvních tříd.

Heuristická funkce zůstane stejná, jako v případě vyhledávání nejrychlejší cesty. Maximální rychlost, kterou dělíme vzdálenost od cíle, můžeme zredukovat na hodnotu zvolenou v předchozím odstavci.

Trasu z Českých Budějovic do Hradce Králové vyhledanou podle všech tří automobilových kritérií znázorněné v mapě ČR lze nalézt na obrázku 3.6.

Obrázek 3.6: Porovnání tras vyhledaných podle různých automobilových kritérií. Červená trasa - nejkratší, zelená - nejrychlejší, modrá - ekonomická.

Cyklistická trasa

Při vyhledávání cyklistické trasy je hlavním kritériem její délka. Nejkratší cesta však velmi často vede po silnici a do kritéria je tedy nutné započítat určitou preferenci značených cyklotras. Cenovou funkci definujeme takto:

Pokud spojení s reprezentuje značenou cyklostezkou, pak

$$f(s) = \text{délka úseku s v km}$$

jinak

$$f(s) = \text{délka úseku s v km} * \mathit{penalizace}.$$

Neznačené úseky se tedy budou tvářit delší, než ve skutečnosti jsou a algoritmus bude preferovat úseky značených cyklostezek. Čím větší penalizace, tím více budou cyklostezky preferovány. Heuristická funkce bude stejná jako při vyhledávání nejkratší trasy.

3.2.6 Postup při vyhledání trasy

V předchozích kapitolách byly popsány různé úpravy plánovacího algoritmu. Pro přehlednost bude dobré dát vše vzájemně do kontextu a sepsat algoritmus, který pro dané kritérium a seznam destinací vrátí vyhledanou trasu (algoritmus 2).

Algoritmus nejprve inicializuje destinace a vyhledá k nim jejich navigační úseky a body. Poté získá / vytvoří vyhledávací strukturu, kterou upraví podle potřeb destinací a jejích navigačních bodů. Následně postupně vyhledá dílčí trasy mezi po sobě jdoucími destinacemi a dílčí výsledky spojí do celkového výsledku.

Algoritmus 2 Postup při vyhledání trasy

```
Vysledek vyhledej_trasu(kriterium, seznam_destinaci)
{
  inicializuj_destinace(seznam_destinaci);

  Vysledek vysledek = vytvor_prazdny_vysledek();
  Heap struktura = null;
  try
  {
    struktura = ziskej_a_priprav_vyhledavaci_strukturu(seznam_destinaci);

    for (int i=0; i~< seznam_destinaci.length - 1; i++)
    {
      Destinace dest_start = seznam_destinaci[i];
      Destinace dest_end = seznam_destinaci[i+1];

      Vysledek dilci_vysledek = vyhledej_cestu_astar(
        struktura, kriterium,
        dest_start, dest_cil
        );
      vysledek.pripoj_dilci_vysledek(dilci_vysledek);
    }

    return vysledek;
  } finally {
    if (struktura != null) vrat_vyhledavaci_strukturu(struct);
  }
}
```

3.2.7 Vyhledávání tras z jedné destinace do všech ostatních

Jedním z požadavků byla možnost vyhledat trasy z jedné destinace do všech ostatních. Máme-li zadaných N destinací, pak našim úkolem je vyhledat trasy $1 \rightarrow 2, 1 \rightarrow 3, ..., 1 \rightarrow N-1, 1 \rightarrow N$. Tento problém by šel řešit postupným vyhledáním všech těchto trasy jednu po jedné. Pokud si ale uvědomíme, že Dijkstrův algoritmus vyhledá optimální trasy z jednoho vrcholu do všech ostatních, můžeme této vlastnosti efektivně využít a vyhledat všechny trasy vedoucí z této destinace najednou. Dijkstrův algoritmus tedy spustíme z první destinace a vyhledávání ukončíme, pokud

vrcholy odpovídající destinacím 2 až N byly prohlášeny za definitní. Zpětným sledováním iterací algoritmu z jednotlivých vrcholů destinací pak postupně získáme všechny hledané trasy.

3.3 Vyrovnávací paměť vyhledaných tras

Při zpracování logových záznamů z ostrého provozu předchozí verze serveru (kapitola 2.1.3) bylo zjištěno, že velké množství vyhledávaných tras se ve krátkém časovém intervalu opakuje. Využití vyrovnávací paměti tedy může výrazně ulevit serveru od zbytečného znovuvyhledávání té samé trasy.

Úkolem vyrovnávací paměti je po určitou dobu uchovávat výsledky do ní vložené a v případě požadavku musí co nejrychleji odpovídající výsledek vyhledat a vrátit. Je tedy nutné, aby informace, které do ní chceme vkládat, byly jednoznačně a snadno identifikovatelné.

3.3.1 Definice destinací a trasy

Trasa je jednoznačně určena kritériem, podle kterého byla vyhledána, a seznamem destinací, přes které prochází.

Z kapitoly 3.2.4 víme, že k jednoznačnému určení destinace postačí její souřadnice a identifikátor města, které reprezentuje. Definicí destinace budeme od této chvíle nazývat řetězec

x;y;cityId,

kde x,y jsou souřadnice destinace ve systému UTM a cityId je identifikátor města, které je touto destinací reprezentováno. Id města může být prázdné, pokud souřadnice žádné město nereprezentuje.

Každému kritériu může být přiřazen jednoznačný číselný identifikátor. Parametr kritéria může být reprezentován logickou hodnotou. Každá trasa může obsahovat dvě a více destinací. S ohledem na tyto možnosti lze tedy trasu jednoznačně definovat řetězcem:

criterium;param;destdef1;destdef2[;destdef3;...],

kde destdef1, destdef2, atd. jsou definice destinací trasy. Např. nejrychlejší automobilovou trasu využívající placené komunikace z Prahy do Olomouce přes Brno pak lze jednoznačně zapsat řetězcem:

```
3;1;460317;5547176;CZ_0649;617187;5450367;CZ_0674;662686;5495891;CZ_0673
```

3.3.2 Využití vyrovnávací paměti

Algoritmus 3 je upraveným algoritmem vyhledání trasy (algoritmus 2), do kterého byla zapracována vyrovnávací paměť.

V algoritmu se navíc vyskytují následující funkce:

- je_trasa_ve_vyrovnavaci_pameti - vrátí true, pokud je trasa s danou definicí ve vyrovnávací paměti,

Algoritmus 3 Postup při vyhledání trasy s využitím vyrovnávací paměti

```
Vysledek vyhledej_trasu_2(kriterium, seznam_destinaci)
{
  String TRASA_DEF = vytvor_definici_trasy(kriterium, seznam_destinaci);

  if (je_trasa_ve_vyrovnavaci_pameti(TRASA_DEF)) {
    return ziskej_trasu_z_vyrovnavaci_pameti(TRASA_DEF);
  }

  inicializuj_destinace(seznam_destinaci);

  Vysledek vysledek = vytvor_prazdny_vysledek();
  Heap struktura = null;
  try
  {
    struktura = ziskej_a_priprav_vyhledavaci_strukturu();

    for (int i=0; i~< seznam_destinaci.length - 1; i++)
    {
      Destinace dest_start = seznam_destinaci[i];
      Destinace dest_end = seznam_destinaci[i+1];
      String DILCI_TRASA_DEF = vytvor_definici_trasy(kriterium, dest_start, dest_cil);

      Vysledek dilci_vysledek = null;
      if (je_trasa_ve_vyrovnavaci_pameti(DILCI_TRASA_DEF))
      {
        dilci_vysledek = ziskej_trasu_z_vyrovnavaci_pameti(DILCI_TRASA_DEF);
      }
      else
      {
        dilci_vysledek = vyhledej_cestu_astar(struktura, kriterium, dest_start, dest_cil);
        vloz_trasu_do_vyrovnavaci_pameti(DILCI_TRASA_DEF, dilci_vysledek);
      }
      vysledek.pripoj_dilci_vysledek(dilci_vysledek);
    }

    vloz_trasu_do_vyrovnavaci_pameti(TRASA_DEF, vysledek);
    return vysledek;
  } finally {
    if (struktura != null) vrat_vyhledavaci_strukturu(struct);
  }
}
```

- ziskej_trasu_z_vyrovnavaci_pameti - načte trasu z vyrovnávací paměti a vrátí výsledek,
- vloz_trasu_do_vyrovnavaci_pameti - vloží vyhledanou trasu do vyrovnávací paměti pod její definici.

Jak je z algoritmu patrné, vyrovnávací paměť je dotazována na celou trasu, tak pro všechny její dílčí trasy zvlášť. První případ je důležitý pro situace, kdy uživatel vyhledává jednu trasu pořád dokola (například porovnává trasu vyhledanou různými kritérii a opakovaně přepíná z jednoho na druhé). Druhý případ je užitečný v případě, že uživatel trasu postupně mění. Například nejprve vyhledá trasu Praha → Brno, pak Praha → Brno → Zlín, pak Praha → Brno → Zlín → Olomouc. V takovémto případě pak bude nutné vyhledávat vždy pouze poslední dílčí cestu do nově přidané destinace. Ostatní budou načteny z vyrovnávací paměti, kam byly uloženy při předchozím vyhledávání.

Veškeré operace musí být rychlé, aby v konečném důsledku vyhledávání trasy spíše nezpomalily. Vzhledem k dostatku operační paměti na serveru bude vhodné držet vyhledané trasy v paměti. Rychlý přístup k nim pak lze implementovat například za použitím asociativního pole (hash map), ve kterém přiřadíme definici trasy k jejímu výsledku.

Aby se operační paměť časem nezahltila, bude nutné vyrovnávací paměť jednou za čas projít a odstranit staré záznamy. Čistící operaci bude nutné vykonávat pravidelně a tak, aby nezpomalovala vyřizování požadavků. Dle výsledků zpracování statistik (kapitola 2.1.3) bude postačovat, když vyrovnávací paměť bude evidovat pouze trasy, které byly naposledy vyhledány před méně jak pěti minutami. Není ale jisté, zda by se při velké zátěži paměť nezahltila. Idálním řešením proto bude držet trasy v paměti pouze kratší dobu a po jejím uplynutí trasy serializovat na pevný disk a z paměti odstranit. Na disku pak mohou být k dispozici mnohem delší dobu, aniž by hrozilo zahlcení paměti.

3.4 Výpis a formátování itineráře trasy

Jak bylo řečeno v seznamu požadavků (kapitola 1.2.4), trasu je po jejím vyhledání nutné uživateli vhodně popsat. Popis musí být snadno pochopitelný, co nejvíce stručný, ale zároveň musí obsahovat všechny informace potřebné k bezproblémovému absolvování trasy.

3.4.1 Výsledek vyhledání trasy

Výstupem plánovacího algoritmu je posloupnost křižovatek a úseků cest v pořadí od počáteční do koncové destinace. Křižovatku, ve které trasa začíná, nazveme **počáteční křižovatka** (K_0). Ke každému úseku pak přiřadíme křižovatku, do které tento úsek vede a vytvoříme tak dvojice **elementárních úseků trasy**:

$$(S_1 \rightarrow K_1), (S_2 \rightarrow K_2), ..., (S_n \rightarrow K_n),$$

kde S_i je úsek vedoucí z křižovatky K_{i-1} do K_i. Vzhledem k tomu, že trasa vždy začíná a končí ve křižovatce, obsahuje trasa jednu počáteční křižovatku K_0 a n elementárních úseků trasy.

3.4.2 Navigační úseky

Statistiky ukazují, že průměrná trasa vyhledaná na mapovém serveru mapy.idnes.cz má zhruba 102 úseků. V tabulce 3.1 lze nalézt přehled o průměrných a maximálních počtech úseků ve vyhledaných trasách s ohledem kritérum, podle kterého byla trasa vyhledána. Z tabulky lze vyčíst, že trasa může být tvořena až 818 úseky. Pokud bychom vytvářeli navigační povel z každého elementárního navigačního úseku v trase, byl by výsledný itinerář velmi dlouhý a tím pádem nepřehledný. Navíc, vezměme si trasu z Prahy do Brna. Většina trasy vede po dálnici D1 a pokud bychom uváděli každý elementární úsek, mohl by itinerář této části trasy vypadat následovně:

- *Pokračujte 8 km po D1 do křižovatky dálniční exit 15, pokračujte rovně.*
- *Pokračujte 24 km po D1 do křižovatky dálniční exit 16, pokračujte rovně.*
- *Pokračujte 15 km po D1 do křižovatky dálniční exit 17, pokračujte rovně.*
- ...
- *Pokračujte 800 m po D1 do křižovatky dálniční exit 31, odbočte doprava.*

Kritérium	**∅ počet úseků**	**∅ počet úseků**
Auto - nejkratší	142,03	801
Auto - nejrychlejší	119,03	541
Auto - ekonomická	131,00	541
Cyklo - minimální pref. cyklotras	75,80	780
Cyklo - průměrná pref. cyklotras	49,16	722
Cyklo - maximální pref. cyklotras	84,26	818

Tabulka 3.1: Průměrné a maximální počty úseků ve vyhledaných trasách podle jednotlivých kritérií.

Takovéto povely jsou z pohledu řidiče zbytečné a matoucí, protože popisují úseky trasy, na kterých řidič jede po stejné komunikaci a nijak nemění směr jízdy. Místo tohoto výčtu dálničních exitů by bylo vhodnější uvést pouze ten, na kterém vozidlo dálnici opouští a je tedy pro navigaci důležitý. Například: „*Pokračujte 84 km po D1 do křižovatky dálniční exit 31, odbočte doprava*“. Nedůležité úseky popisující cestu po dálnici D1 tedy sloučíme do jednoho celku, tento celek nazveme **navigačně důležitým úsekem trasy**.

Navigačně důležitý úsek trasy je posloupnost po sobě jdoucích dvojic elementárních úseků trasy, na kterých:

- nedochází ke změně značení komunikace,

- nedochází k prudké změně směru vozidla / odbočení.

Druhá podmínka je důležitá pro případy, kdy vozidlo po příjezdu na křižovatku sice pokračuje po komunikaci se stejným značením, ale mění směr jízdy. Na změnu je tedy nutné upozornit, aby uživatel nepokračoval rovně.

3.4.3 Navigační povely

Vyhledanou trasu lze uživateli velmi intuitivně popsat s pomocí tzv. **navigačních povelů**. Navigační povel imituje hlášení navigátora, který řidičovi průběžně radí, kam má jet, kdy má odbočit atd. Povely by měly řidiče informovat:

- po jaké komunikaci má jet (silniční a mezinárodní značení, kategorie, maximální rychlost),
- do jaké křižovatky vozidlo dojede po projetí úseku,
- navigační pokyny na této křižovatce (zda a kam odbočit),
- ujetou vzdálenost a čas trasy do této křižovatky.

Pro lepší srozumitelnost bude vhodné povel reprezentovat ve formě věty či souvětí. Sada povelů vytvořená z elementárních částí trasy by mohla vypadat následovně:

- *Začínáte v křižovatce* K_0.
- *Pokračujte 800 m po* S_1 *do křižovatky* K_1. *Odbočte doleva.*
- *Pokračujte 2 km po* S_2 *do křižovatky* K_2. *Odbočte doprava.*
- ...
- *Pokračujte 200 m po* S_n *do cílové křižovatky* K_n.

Pokud by elementární úseky S_1 až S_n náležely do stejného navigačně důležitého úseku, pak by povely vypadaly následovně:

- *Začínáte v křižovatce* K_0.
- *Pokračujte 800 m po* $S_{1\to n}$ *do cílové křižovatky* K_n. *Odbočte doleva.*

Úsek $S_{1\to n}$ vzikne sloučením úseků S_1 až S_n.

Každý navigační úsek (ať už elementární nebo navigačně důležitý) musí být řidiči dostatečně popsán. Musí vědět jak „dlouho“ a po jaké komunikaci má jet a do jaké křižovatky dojede. Na křižovatce pak musí být popsaná změna směru. Popis cesty po úseku do následájící křižovatky může obsahovat následující informace:

- délku úseku,
- čas potřebný k projetí úseku,
- překonané převýšení,
- automobilové, cyklistické a turistické značení,
- kategorii / typ komunikace,
- maximální povolenou rychlost automobilu,
- zda je úsek zpoplatněný či nikoliv,

- vektor souřadnic popisující navigační úsek bod po bodu.

Ne všechny tyto informace jsou užitečné pro všechny typy dopravních prostředků. Například pro automobil je zbytečné uvádět cyklistické a turistické značení, pro cyklistu zase, zda je úsek zpoplatněný nebo maximální povolená rychlost automobilu. Itinerář tedy musí být sestaven s ohledem na kritérium podle kterého byla trasa vyhledána a dopravní prostředek, kterému je itinerář určen.

3.4.4 Navigace na křižovatkách

Navigační úsek trasy končí na křižovatce, na které musí dopravní prostředek změnit směr či odbočit. Pro každou křižovatku lze z navigačních dat zjistit následující informace užitečné pro itinerář:

- popis křižovatky pro automobil, popis pro cyklistu,
- poloha křižovatky,
- typ křižovatky.

Navigační data rozlišují několik základních typů dopravních křižovatek: klasická křižovatka, kruhový objezd, dálniční nájezdy a výjezdy. Pro každou jsou vhodné jiné navigační povely.

Klasická křižovatka

Nejrozšířeňejším typem křižovatky je úrovňové křížení dvou a více komunikací (obrázek 3.7 vlevo). Křižovatka může být opatřena světelnou signalizací. Na tomto typu křižovatky je nutné hlásit, jakým směrem má řidič pokračovat dále, tedy zda a kam má odbočit. Příkaz by tedy mohl znít: „*Na křižovatce odbočte doprava*“ nebo „*Pokračujte přímo*“.

Směr odbočení lze určit ze změny úhlu pohybu vozidla na této křižovatce. Změnu úhlu spočítáme jako rozdíl mezi úhlem, pod kterým vozidlo do křižovatky vstoupí, a úhlem, pod kterým vozidlo křižovatku opustí. Tento úhel tedy může nabývat hodnot mezi -180° a $+180^\circ$. Pro různé intervaly úhlů pak zvolíme vhodné hlášení informující o změně směru:

- 0° – 15° - jeďte přímo,
- 15° – 45° - odbočte mírně doleva / doprava,
- 45° – 100° - odbočte doleva / doprava,
- 100° – 170° - odbočte ostře doleva / doprava,
- 170° – 180° - otočte vozidlo.

Pro výpočet úhlu odbočení na křižovatce K_i jsou potřeba vektory úseků S_i a S_{i+1}. Tyto vektory V_i a V_{i+1} mají společný bod ležící v místě křižovatky K_i. Ke spočítání úhlu pak stačí z obou vektorů vybrat body, které následují bezprostředně po sdíleném bodu a spočítat úhel, který tato trojice bodů ($V_i[1]$, Ki, $V_{i+1}[1]$) svírá.

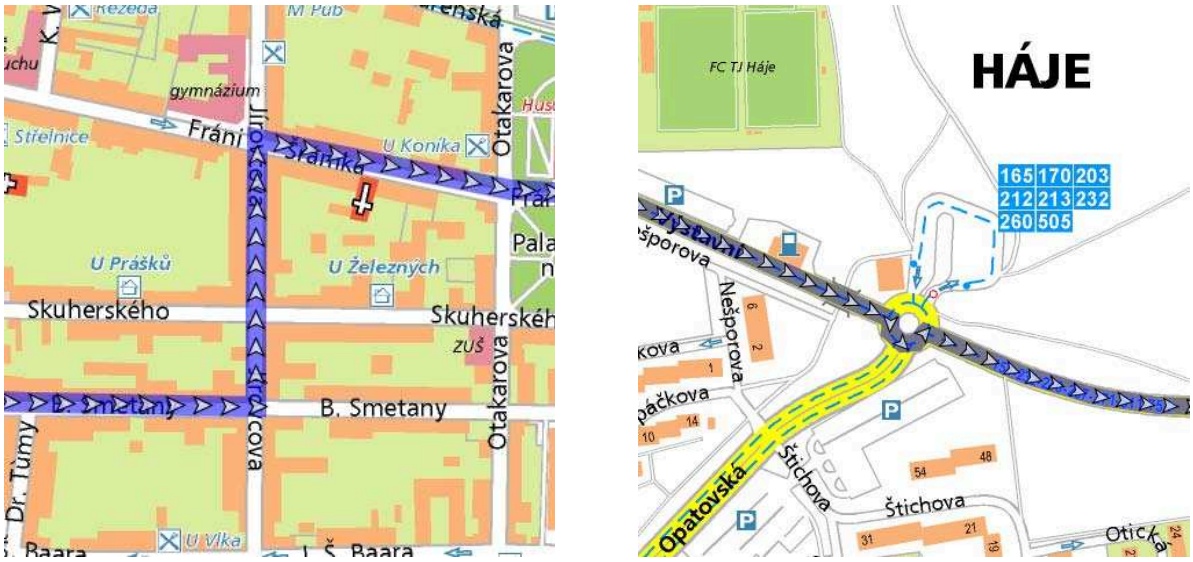
U některých křižovatek je definován stručný popis. Může se jednat o název části obce či města, nebo názvy ulic, které se na ní kříží. Tento popis může pomoci řidiči v orientaci a je tedy

dobré jej v povelu uvést alespoň jako dodatečnou informaci v závorce. Např: „Na křižovatce (Korunní/Blanická) odbočte doprava“.

Kruhový objezd

Kruhový objezd není nutné představovat. Co je zajímavé, je jeho reprezentace v navigačních datech (obrázek 3.7 vpravo). Kruhový objezd je tvořen tolika křižovatkami, kolik má objezd nájezdů/výjezdů. Tyto křižovatky jsou propojeny jednosměrnými úseky orientovanými proti směru hodinových ručiček. Pokud bychom použili pro popis těchto křižovatek standardní povely, obdrželi bychom toto: „*Odbočte doprava, pokračujte 20 metrů, pokračujte přímo, pokračujte 20 metrů, odbočte doprava*“. Zápis je dlouhý a zbytečně matoucí, protože rozkládá navigaci na kruhovém objezdu do několika povelů.

V tomto případě postačí řidiči poskytnout informace o tom, že najíždí na kruhový objezd a říci, který výjezd má zvolit. Vhodnější navigační povel popisující tuto dopravní situaci tedy je: „*Kruhový objezd opusťte na druhém výjezdu*“.



(a) Klasická křižovatka

(b) Kruhový objezd

Obrázek 3.7: Ukázky typů křižovatek v mapě.

Dálniční nájezd, exit

Dálniční nájezdy a exity slouží k nájezdu či výjezdu vozidel na a z rychlostních komunikací. Obvykle jsou realizovány vyhrazenými odbočovacími pruhy, které vozidlu umožní plynulý nájezd či výjezd. Nejedná se tedy o křižovatku v pravém slova smyslu, ale spíše o zařazení vozidla z odbočovacího pruhu do normálního a naopak. Ve většině případů je tedy zbytečné hlásit směr odbočení.

Hlášení směru však řidiči neuškodí a maximálně ho utvrdí v tom, že jede správně. Dálniční nájezdy a exity tedy lze hlásit stejným způsobem jako klasické křižovatky. Je ale dobré uvést popis křížovatky, který v tomto případě obsahuje číslo exitu / nájezdu a informace o tom, na kterou silnici se najíždí. Zde je důležité, aby směr odbočení byl popsán „odstupňovaně“. Hlášení „odbočte mírně doprava“ odpovídá situaci často mnohem lépe, než příkaz bez přídavného jména.

Křižovatky destinací

Jak bylo řečeno v kapitole 3.2.4, každá destinace má přiřazenou svoji křižovatku, ze které je zahájeno vyhledávání trasy, nebo ve které je vyhledávání ukončeno. V itineráři musí být tato křižovatka náležitě zvýrazněna a řidič by měl být upozorněn na to, že dorazil nebo projíždí okolo destinace trasy.

Hraniční přechody

Přejezd do jiného státu může být spojen s komplikacemi od čekání na hraničních přechodech, po nutnost koupě dálniční známky platné v dané zemi. Přechody hranic je tedy nutné zvýraznit tak, aby byly v itineráři jasně viditelné. Řidič jistě uvítá informaci, do kterého státu přechodem vstupuje.

3.4.5 Formát itineráře

Itinerář vyhledané trasy by měl být poskytnut v takovém formátu, aby z něj bylo možné snadno vytvořit v aplikacích zákazníka popis trasy odpovídající představám zákazníka. Není tedy možné, aby výstupem byl již naformátovaný popis trasy ve větách (třeba ve formátu HTML či PDF). Proto byl pro tyto účely zvolen formát XML (Extensible Markup Language).

Výhoda tohoto řešení je zároveň jeho nevýhodou. Vyžaduje implementaci na straně klientské aplikace (stažení XML, příprava formátování itineráře), což může být pro některé firmy odrazující. Součástí projektu by měla být knihovna, která implementuje zaslání požadavku na vyhledání trasy na servlety a zpracuje vrácený XML itinerář. Poslouží tedy jako základ či inspirace pro programátory klientských aplikací.

Kapitola 4

Implementace

V této kapitole bude popsána implementace navigačního serveru, od použitých technologií, knihoven, architektury serveru, až po popis důležitých tříd a postupů.

4.1 Použité technologie

Servlety jsou naprogramovány v jazyce Java v prostředí Java 1.5.x. Jako vývojové prostředí byla použita aplikace Netbeans ve verzi 6.1. Pro nasazení a testování servletů pak server Apache Tomcat 6.x.

4.1.1 Použité knihovny

Veškeré zdrojové zdrojové kódy jsou mým dílem s následujícími výjimkami:

PROJ.4

Knihovna implementující velmi přesné a rychlé převody mezi různými souřadnicovými systémy[1]. Autorem knihovny je Frank Warmerdam a knihovna je distribuována pod MIT License. Knihovna je implementována v jazyce C a je distribuována v podobě zdrojových kódů. Je nutné ji zkompilovat na stroji, pro který je knihovna určena. Servlety využívají knihovnu přes rozhraní JNI.

xmlenc

Light-weight XML output library for Java [2]. Knihovna pro rychlé a paměťově nenáročné vytvuření XML dokumentů.

[1] http://www.remotesensing.org/proj/
[2] http://xmlenc.sourceforge.net/

Spatial Index Library

Knihovna implementujíci R stromy a jiné prostorové vyhledávací struktury. Autorem knihovny je firma Navel Ltd. a knihovna je distribuována pod GNU Lesser General Public License [3]. Zdrojové kódy lze nalézt v balíku *spatialindex*. Z knihovny je využívána implementace R stromů.

Fibonacciho haldy

Třída implementujici Fibonacciho haldu. Je součástí Open Source knihovny JGraphT[4], která je distribuována pod GNU Lesser General Public License. Zdrojové kódy lze nalézt v souboru FibonacciHeap.java v balíku *planstudio.mapobjects.util*. Fibobacciho halda byla zvažována pro použití v plánovacích algoritmech.

4.2 Architektura a moduly serveru

4.2.1 Servlety

Pro implementaci HTTP rozhraní byla zvolena technologie java servletů. HTTP rozhraní je implementováno třídou *ServletRouting* (potomek třídy *HttpServlet*). Tato třída pak vyřizuje zpracování všech požadavků na navigační server. Zároveň se stará o načtení a inicializaci navigačnich dat a start modulů, které jsou poté serverem využívány.

Život servletů

Servlety se instalují na server jako každá jiná webová aplikace určená pro Apache Tomcat. Každá aplikace má svůj webový deskriptor (definovaný souborem web.xml), který určuje:

- URL aplikace,
- propojení URL a třídy servletu, který vyřizuje požadavků na toto URL,
- příznak, zda se servlety mají inicializovat při startu serveru.

Initializace

Servlet je při startu serveru Apache Tomcat načten a inicializován (metoda *init*). V této metodě jsou postupně:

- načtena základní nastavení serveru,
- načtena navigační data,
- inicializováno logování,
- inicializována vyrovnávací paměť,

[3] http://research.att.com/~marioh/spatialindex/index.html
[4] http://jgrapht.sourceforge.net/

- vytvořeno a spuštěno údržbářské vlákno.

Pokud během nějaké operace dojde k fatální chybě, hozená výjimka se propaguje až mimo metodu *init*. Servlet pak nebudou prohlášeny za práceschopné. Při příchodu požadavku na servlet, který nebyl úspěšně inicializován, se server bude pokoušet inicializovat servlet znovu. Tyto neúspěšné opakované inicializace mohou velmi zatěžovat server. Je proto důležité ověřit úspěch inicializace serveru zasláním pokusného požadavku.

Zpracování požadavku, kontext

Server při zpracování HTTP požadavků volá metody *doGet* a *doPost* instance třídy *ServletRouting*. Při zpracování požadavku:

- Je vytvořen kontext (třída *ContextRouting*), nad kterým se bude požadavek zpracovávat.
- Nad tímto kontextem jsou zpracovány příkazové funkce (definované v parametru „*commands*“). Veškeré mezivýsledky se ukládají do instance kontextu.
- Nad kontextem je spuštěna návratová funkce (definovaná v parametru „*return*“). Její výstup je poslán jako odpověď na požadavek.

Pro každý požadavek je tedy vytvořena jedna instance kontextu. Pokud v průběhu zpracování požadavku dojde k chybě, je na místo požadovaného výstupu vráceno XML popisující chybový stav (třída *ContextException*).

Třída *ContextRouting* se stará o zpracování všech příkazových i návratových funkcí. Implementuje tedy rozhraní popsané v kapitole 3.1.6. Zpracování funkce a jejích parametrů zajišťuje třída *ContextCommandRouting*. Ta provádí kontroly, zda má funkce definováno dostatek parametrů, zda jsou parametry ve správném formátu atd.

Při vykonání příkazové nebo návratové funkce je název funkce převeden na malá pismena. Metodou *getMethod* nad třídou aktuálního kontextu je získán odkaz na odpovídající metodu třídy *ContextRouting*. Tato metoda je pak vyvolána s parametrem instance *ContextCommandRouting* (algoritmus 4).

Algoritmus 4 Zpracování příkazové funkce

```
ContextCommandRouting cmd = new ContextCommandRouting(commandStringDef);

Method method = this.getClass().getMethod(cmd.commandName,
    new Class[] { ContextCommandRouting.class });
Object[] args = { cmd };
method.invoke(this, args);
```

Metody návratových funkcí mají navíc parametr výstupního proudu, do kterého zapisují svůj výstup (algoritmus 5).

Při zpracování funkcí kontext využívá sdílených dat a prostředků servletu. Odkazy na tato data a prostředky jsou kontextu dodány při jeho inicializaci.

Algoritmus 5 Zpracování návratové funkce

```
ContextCommandRouting outCmd = new ContextCommandRouting(outputCommandStringDef);

Method method = this.getClass().getMethod(outCmd.commandName,
    new Class[] { ContextCommandRouting.class, OutputStream.class  });
Object[] args = { outCmd, outStream };
method.invoke(this, args);
```

Údržbářské vlákno

Údržbářské vlákno (třída *ContextWorkerThread*) je vlákno, které se průběžně stará o:

- zápis logových záznamů z paměti na disk,
- serializaci tras ve vyrovnácací z paměti na disk,
- odstraňování nepoužívaných tras z vyrovnávací paměti,
- monitorování paměti využívané servlety a zápisu průběžných statistik.

Vlákno je spuštěno při inicializaci servletu a je ukončeno při jeho destrukci. Běží v nekonečné smyčce, ve které nejprve vykoná své činnosti a pak se uspí. Délka spánku při jedné iteraci je nastavitelná v základních nastaveních serveru.

4.2.2 Nastavení serveru

Nastavení serveru jsou uložena ve formátu standardního souboru vlastností. Jedná se o textový soubor s jedním parametrem na řádek. Parametr je řetězec ve formátu $parametr = hodnota$. Načtení a správu parametrů servletu má na starosti třída *AppRoutingSettings*. Při inicializaci servetu je vytvořena jedna instance této třídy, která je pak sdílena všemi kontexty.

4.2.3 Navigační data

Při inicializaci servletů jsou veškerá navigační data načtena do paměti. Aby tato operace zabrala co nejméně času, jsou data načítána z předpřipravených binárních souborů. Jejich příprava je popsána v kapitole 4.4.

Načtení a správu dat zajišťuje třída *MAPRouting*. Ta při načítání:

1. Zkontroluje, zda načítaná data odpovídají verzi navigace a jsou kompatibilní.
2. Postupně načte části navigačních dat:
   - křižovatky (třídy *MAPLocationList*, *MAPLocation*),
   - úseky cest (třídy *MAPRouteList*, *MAPRoute*),
   - vektory úseků cest sloužené bodů reprezentovaných třídou *MAPPoint*,
   - typy křižovatek a úseků,
   - vyhledávací strukturu,
   - R stromy úseků cest a měst,
   - a další.

Navigační data jsou pak sdílena mezi všemi kontexty.

4.2.4 Logování

Logování slouží k pořizování záznamů o:

- vykonaných přikazových funkcích,
- vykonaných návratových funkcích,
- vyhledaných trasách, jejich definici, statistikác plánovacího algoritmu,
- využití operační paměti, vyrovnávací paměti atd.
- dalších činnostech servletu a systému.

Logování je implementováno třídou *ContextLogger*. Od této třídy je vytvořena opět pouze jedna instance, která je sdílená mezi všemi kontexty. Při vkládání záznamů kontexty je tedy nutné dbát na vzájemné vyloučení přístupu. Logové záznamy jsou dočasně skladovány v paměti, protože přístup na disk při každém přidání zápisu by příliš zpomaloval. Na disk jsou zapsány až údržbářským vláknem.

Popis a struktura logových záznamů je detailně popsán v dokumentaci navigačních servletů (příloha A).

4.2.5 Převody souřadnic

Někteří klienti vyžadují při komunikaci se servlety použití jiného souřadnicového systému, než je ten, ve kterém jsou uložena navigační data. Je tedy nutné imlementovat převody mezi jednotlivými souřadnicovými systémy. K tomuto účelu byla zvolena knihovna PROJ.4.

Knihovna je implementována v jazyce C a je distribuována v podobě zdrojových kódů. Je tedy potřeba ji zkompilovat na stroji, pro který je určena. Postup při kompilaci je detailně popsán v návodech dodávaných spolu s knihovnou.

Servlety přistupují ke knihovně prostřednictvím rozhraní JNI. Knihovna PROJ.4 ve své distribuci sice obsahuje třídy, které toto rozhraní implementují, ale jejich použití způsobuje úniky paměti. Při častém volání převodních funkcí (v řádech 10 000) pak docházelo k vytuhávání servletů a následně k pádu celého serveru (s vyjímkou informující o nedostatku paměti). Rozhraní JNI bylo tedy přepsáno a úniky chyb opraveny.

Kontext využívá k převodům souřadnic metody třídy *ContextManagerProj*. Ta využívá k převodům rozhraní JNI implementované v třídě *Projections*, která je součástí balíku *org.proj* a musí být umístěna v jiném jaru, než zdrojové kódy servletu. Jar s PROJ knihovnou pak musí být umístěn v adresáři sdílených knihoven serveru Tomcat. Pokud by knihovna byla umístěna v aplikačním adresáři servletu, byl by k jejich restartu nutný restart celého serveru.

Třída *ContextManagerProj* dále zajišťuje správné formátování souřadnic v různých systémech a jejich tisk na výstup (WGS84 souřadnice musí být zaokrouhleny na 7 destinných míst, ostatní na dvě desetinná místa).

Definice projekcí

Každý typ souřadnic musí být knihovně PROJ definován za pomocí formátovaného řetězce. Definice pro jednotlivé typy souřadnic lze nalézt na internetu. Servlety v současné době používají následující 4 typy projekcí:

- **WGS84** - Zeměpisná šířka a délka v desetinném formátu.
- **S42** - S-1942, Gauss-Krügerovo zobrazení ve 3. pásu na Krasovského elipsoidu. Zobrazení velmi podobné UTM. Ve firmě PLANstudio bylo dříve používáno pro uložení mapových podkladů.
- **UTM** - Universal Transverse Mercator v pásu 33.
- **JTSK** - Souřadnicový systém S-JSTK (Křovákovo zobrazení na Besselově elipsoidu). Používá se hlavně ve starších vojenských mapách nebo v datech státní správy.

4.3 Vyhledání trasy

Postup při vyhledání trasy byl popsán v kapitole 3.2.6 a je implementován metodou *findRouteDestinations* třídy *MAPRouting*. Metoda nejprve inicializuje destinace, připraví záznamovou část vyhledávací struktury, vyhledá dílčí trasy a sloučí je do celkového výsledku, ze kterého je poté připraven itinerář trasy. Při vyhledávání jsou využívána předem načtená sdílená navigační data.

4.3.1 Inicializace destinací

Destinace je reprezentována třídou *MAPRoutingDestination*. Při inicializaci je postupováno dle kapitoly 3.2.4. Ke každé destinaci je vyhledán metodou *getNearestRoute* třídy *MAPRouteList* vhodný navigační úsek (nejbližší úsek komunikace) a nejbližší bod na vektoru tohoto úseku. V místě bodu je pak vytvořena křižovatka a navigační úsek je v bodu rozdělen na dvě části (metoda *setNearestRoute* třídy *MAPRoutingDestination*). Nová křižovatka spolu s oběma úseky pak pro potřeby plánování této trasy nahradí původní navigační úsek.

4.3.2 Vyhledávací struktura a plánování trasy

Vyhledávací struktura (třída *MAPSearchStruct*) je vytvořena na základě poznatků z kapitoly 3.2.1. Vrchol je reprezentován třídou *MAPVertex*, spojení pak třídou *MAPLink*. Každé spojení obsahuje:

- odkaz na úsek komunikace (*MAPRoute*), který je spojení reprezentuje,
- odkaz na vrchol, do kterého spojení vede,
- příznaky, zda je spojení použitelné automobilem a cyklistou,

- předpočítané hodnoty cenových funkcí pro ekonomické a nejrychejší kritérium[5].

Vyhledávací struktura patří mezi data sdílená mezi všemi kontexty. Nelze tedy do ní zapisovat a jakkoliv ji měnit. Pro účely uchovávání informací lokálních pro konkrétní plánování trasy musí být před spuštěním plánování připravena záznamová část vyhledávací struktury (třída *MAPHeapSearchStruct*). Tato třída pak pro každý vrchol ve vyhledávací struktuře eviduje jeho dočasné plánovací informace (třída *MAPHeapVertex*). K těmto patří:

- odkaz na plánovací informace předchozího vrcholu (a použité spojení) v aktuální optimální cestě vyhledané do tohoto vrcholu,
- dočasnou cenu této cesty,
- příznak zda je vrchol uzavřený.

Jelikož plánování probíhá nad záznamovou strukturou, jsou ve třídě *MAPHeapSearchStruct* implementovány všechny plánovací algoritmy. Jedná se o metody, které vyhledají cestu mezi dvěma destinacemi:

- *SearchRoute* - používá Dijkstrův algoritmus,
- *SearchRouteAstar* - používá A star (implementace se drží pseudokódu popsaného v algoritmu 1 na stránce 35) ,

a metody, které vyhledají cesty z destinace do jedné a více jiných destinací.

- *SearchRouteFromDestinationToAllOthers* - používá Dijkstrův algoritmus.

V algoritmech je využívána K-ární halda implementovaná třídou *DnaryHeap*.

Kritéria

Jedním z parametrů metody *findRouteDestinations* je instance třídy reprezentující vyhledávací kritérium. Třída implementující kritérium musí být potomkem abstraktní třídy *MAPRoutingCriterium* a musí přetížit všechny její metody. Mezi ty nejdůležitější patří:

- *boolean isUsable(MAPLink l)* - vrací true, pokud je spojení *l* použitelné tímto kritériem,
- *double getLength(MAPLink l, MAPVertex v)* - vrací hodnotu cenové funkce pro spojení *l* vedoucí do vrcholu *v*,
- *double getHeuristic(MAPLocation loc)* - vrací hodnotu heuristické funkce pro křižovatku *loc*.

Instance tříd kritérií jsou vytvářeny v metodě *getCriterium* třídy *MAPRouting* na základě číselného identifikátoru kritéria a parametru. Implementované třídy všech kritérií lze nalézt v balíku *planstudio.mapobjects.routingcriterium*.

[5]Tyto hodnoty se vyplatí předpočítat, protože dle statistik jsou tato kritéria využívána nejvíce.

4.3.3 Reprezentace výsledku

Po vyhledání trasy mezi dvěma destinacemi je její výsledek uložen do instance třídy *MAPSearchPartialResult*. Tato třída eviduje vyhledanou trasu v podobě uspořádaného seznamu křižovatek (*MAPLocation*) a úseků cest (*MAPRoute*) v pořadí start → cíl.

Trasa může vést přes neomezený počet destinací a může být složena z jedné a více jednoduchých (dílčích) tras. Celkový výsledek je pak reprezentován třídou *MAPSearchResult*, která obsahuje:

- seznam destinací, přes které byla trasa vyhledána,
- kritérium a parametr vyhledávání,
- seznam instancí třídy *MAPSearchPartialResult* s dílčími vyhledanými trasami,
- seznam všech úseků a křižovatek v celé trase (seznam vytvořený sloučením seznamů dílčích výsledků),
- statistické informace o trase (celkovou délku, čas, klesání, stoupání),
- minimální souřadnicový obdélník trasy.

4.3.4 Itinerář

Třída *MAPSearchResult* se stará o formátování a tisk itineráře trasy na výstup. Za výstupní formát bylo zvoleno XML (kapitola 3.4.5). Ačkoliv má jazyk Java zabudované knihovny pro práci s XML daty (rozhraní DOM a SAX), byla nakonec pro vytváření XML zvolena knihovna xmlenc. DOM (Document Object Model) drží celé XML v paměti, což může u velkých itinerářů přestavovat desítky až stovky kB. Knihovna xmlenc drží v paměti jenom malou část dokumentu a je tedy mnohem šetrnější. Na druhou stranu ale nenabízí takový komfort při stavbě dokumentů, jako DOM. Pro naše účely však postačuje.

Pro seskupování elementárních úseků trasy do navigačně důležitých úseků (kapitola 3.4.2) slouží třída *MAPSearchResultIterator*. Třída určuje, které skupiny po sobě jdoucích elementárních úseků budou tvořit navigačně důležitý úsek.

Třída *MAPSearchResult* implementuje několik typů itinerářů. Některé však existují pouze z důvodu zpětné kompatibility s předchozími aplikacemi firmy PLANstudio. Typy itinerářů a jejich formátování a struktura je detailně popsána v dokumentaci navigačních servletů (příloha A).

4.4 Zpracování navigačních dat

Navigační data jsou dodávána v relativně surových textových souborech (popis v kapitole 1.3). Před jejich použitím je tedy nutné je vhodně předzpracovat. Dobré je, že soubory dat zabírají jen 137 MB a bez problémů se vejdou do operační paměti.

Zpracování dat je implementováno metodou *serializeAll* třídy *MAPRoutingDataPreparator*. Metoda:

- zpracuje textové soubory a načte navigační data do objektů,
- vytvoří vazby mezi navigačními objekty,
- vytvoří vyhledávací strukturu,
- vytvoří R strom úseků cest,
- uloží zpracovaná data do binárních souborů.

Předzpracování dat a je časově velmi náročná operace. Pokud bychom servlety na webovém serveru inicializovali přímo ze zdrojových textových dat, mohla by tato operace trvat velmi dlouhou dobu v řádu jednotek až desítek minut. V případě výpadku serveru nebo nutnosti jeho restartu během dne by pak servlety nebyly po dlouhou dobu schopny vyřizovat požadavky. Předzpracování je tedy nutné provést ještě před jejich použitím v servletech. Pro tyto účely byla implementována konzolová aplikace (třída *Main*), která umožňuje spuštění této operace z příkazové řádky. Na serveru se tedy nepoužijí textová data, ale až soubory vzniklé jejich zpracováním. Inicializace navigačních servletů z předzpracovaných dat zabere 4 s. Příprava těchto dat trvá na stejném počítači 118 s.

Kontrola chyb v datech

Navigační data jsou vytvářena lidmi a ačkoliv prošla důkladnou kontrolou, obsahují různé chyby. Při zpracování souborů je tedy nutné dávat pozor na jejich konzistenci a správnost. Konkrétně je třeba dávat pozor na špatné provázání souborů mezi sebou, např. odkazy na neexistující identifikátory nebo překlepy a špatné hodnoty sloupců. Seznam chyb je po zpracování dat uložen do souboru, který může posloužit pracovníkům firmy při jejich opravě.

Optimalizace uložení dat

Souřadnice v navigačních datech jsou uloženy s přesností na metr a metr je zároveň základní jednotkou souřadného systému UTM. Pro uložení bodů souřadnic bude tedy postačovat celočíselný datový typ (int).

Hodnoty některých datových sloupců záznamů úseků cest se velmi často opakují. Například silniční a cyklistická značení nebo názvy měst, ve kterém se úsek nalézá. Tyto hodnoty tedy bude výhodné vyhledat a nahradit odkazem na hodnotu sdílenou všemi instancemi objektů úseků cest na místo původní hodnoty.

Kapitola 5

Zátěžové testování a porovnání plánovacích algoritmů

Hlavním cílem zátěžového testování implementovaného navigačního serveru je zjistit rychlost vyřizování požadavků na vyhledání tras. Bude zde ukázáno, jak se server chová při vyledávání různě dlouhých tras podle různých kritérií. Budou porovnány plánovací algoritmy Dijkstra a A star a rychlosti výpočtů při použití Fibonacciho a K-árních hald.

5.1 Metodika testování

Jak volit trasy, které budou použity při zátěžovém testování serveru? Vyhledávané trasy by měly být dostatečně náhodné, ale zároveň by měly odpovídat trasám, které jsou na serveru nejčastěji vyhledávány. V kapitole analýza logových záznamů (2) bylo řečeno, že logové záznamy obsahují záznamy tras vyhledaných v období 1.8.2007 - 31.3.2008. Z těchto záznamů lze o každé vyhledané trase vyčíst následující informace:

- kritérium a parametr trasy,
- destinace trasy,
- dobu trvání výpočtu v milisekundách,
- počet zpracovaných vrcholů při výpočtu,
- délku vyhledané trasy v km,
- počet úseků cest ve vyhledané trase.

Záznam o vyhledání trasy tedy obsahuje úplnou definici trasy (první a druhý bod) a na jejich základě tedy je možné zcela zrekonstruovat požadavky, kterými byly tyto trasy vyhledány, a použít je při zátěžovém testování nového serveru. Tyto trasy poskytují reálný a dostatečně náhodný vzorek dat a jsou tedy pro testování ideální.

5.1.1 Normalizace dat

V období zachycených záznamy logů bylo vyhledáno 528 662 tras. Z toho 326 682 automobilových a 201 980 cyklistických. Pouze z 252 342 záznamů lze vyčíst úplnou definici vyhledané trasy a použít ji pro její znovuvyhledání (logování destinací bylo přidáno na přelomu roku 2007 / 2008). Do testů byly pro zjednodušení zahrnuty pouze trasy se dvěma destinacemi (celkem 217 515 tras). Při vyřizování jednoho požadavku na vyhledávání trasy byl tedy plánovací algoritmus spuštěn vždy právě jednou.

Během období zachycených v záznamech došlo několikrát k aktualizaci (rozšíření) navigačních dat a počty úseků a křižovatek se skokově zvýšily o 5 až 10%. Části záznamů vypovídajících o efektivitě algoritmů (počet zpracovaných vrcholů atd) tedy nejsou vypovídající, protože trasy nebyly vyhledány nad stejnými daty. Aby bylo možné věrohodně srovnat oba algoritmy, bude nutné všechny trasy znovu vyhledat nad stejnými navigačními daty. Z původních záznamů se tedy použijí pouze definice tras.

5.1.2 Porovnání algoritmů

Vyhledání stejné trasy pomocí různých plánovacích algoritmů nám dává možnost porovnat chování a efektivitu obou algoritmů. V záznamu logů o vyhledané trase jsou uvedeny všechny podstatné informace o práci algoritmu. Porovnáním záznamů vytvořenými oběma algoritmy pak zjistíme:

- který algoritmus trasu vypočetl rychleji,
- který algoritmus trasu vypočetl efektivněji (zpracoval méně vrcholů),
- zda se vyhledané trasy liší (mají různou délku, nebo jiný počet úseků).

Pro srovnání efektivity Dijkstrova a A star algoritmu bude nejlepší sledovat vztah mezi počtem zpracovaných vrcholů při vyhledání trasy a počtem úseků cest ve výsledné trase. Pro porovnání jejich rychlosti pak vztah mezi dobou výpočtu a počtem úseků.

Porovnáním délek tras je možné ověřit, zda oba algoritmy vyhledaly stejné nebo podobné trasy. Pokud by se od sebe příliš lišily, signalizovalo by to potenciální problém v jednom z algoritmů.

5.1.3 Testovací aplikace

Pro zpracování statistik byla implementována aplikace v .NETu jménem LogFilter (viz. kapitola 2). Funkce pro zátěžové testování byly pro pohodlnost implementovány v téže aplikaci. Po spuštění testů aplikace:

- načte definice vyhledaných tras z logových záznamů,
- vytvoří 1 a více pracovních vláken, které budou zasílat a zpracovávat požadavky na server,
- každé z vláken pak postupně žádá o nezpracovanou definici trasy, vytvoří a pošle požadavek na její vyhledání a vyžádá si itinerář,

- je zaznamenán čas vyřízení požadavku a výsledná trasa a informace z jejího itineráře jsou započítány do statistiky.

Požadavky posílá testovací aplikace na server paralelně s pomocí 1 a více pracovních vláken. Je tedy možné otestovat chování aplikace na serveru, který má k dipozici více procesorů, nebo jeden vícejádrový procesor. Testovací aplikace komunikuje se serverem s pomocí HTTP protokolu. Je tedy možné a vhodné spustit navigační server na jednom stroji, testovací aplikaci na jiném a oba stroje propojit lokální sítí. Odpadnou tak chyby v měření, kdy testovací aplikace bere výkon a prostředky navigačnímu serveru.

Testovací aplikace pak statistiky ukládá do několika souborů (uvedeny jsou pouze ty důležité):

- ok.txt - seznam tras, jejichž délka byla při vyhledání oběma algoritmy stejná,
- diff.txt - seznam tras, jejichž délka byla při vyhledání oběma algoritmy různá, ale menší než určitá hodnota (v procentech),
- diffAboveTr.txt - seznam tras, jejichž délka byla při vyhledání oběma algoritmy různá, a větší než určitá hranice (v procentech),
- error.txt - seznam tras, při jejichž zpracování došlo k chybě,
- optimalization.txt - srovnání obou algoritmů (rozdíly mezi počty zpracovaných vrcholů),
- usedRoutePaths.txt - seznam úseků cest využitých ve vyhledaných trasách a četnost jejich užití.

Ze souborů diffAboveTr.txt a error.txt lze pohodlně vyčíst, které trasy si zaslouží podrobnější pohled a přezkoumání. Soubor usedRoutePaths.txt je využíván pro zjištění míry užití úseků cest v trasách vyhledaných na serveru (podrobně popsáno v kapitole 2.1.4).

5.1.4 Konfigurace serveru

Při testování navigačního serveru bylo použito PC s následující konfigurací:

- Velikost operační paměti: 2 GB (DDR2).
- Maximální velikost operační paměti dostupné pro Apache Tomcat: 1 GB.
- Procesor: Intel Core2 Duo E6750 na 2,66 GHz.
- Operační systém: MS Windows XP.
- Verze serveru Apache Tomcat: 6.0.16.
- Verze Java SDK: 1.6.0_05-b13.

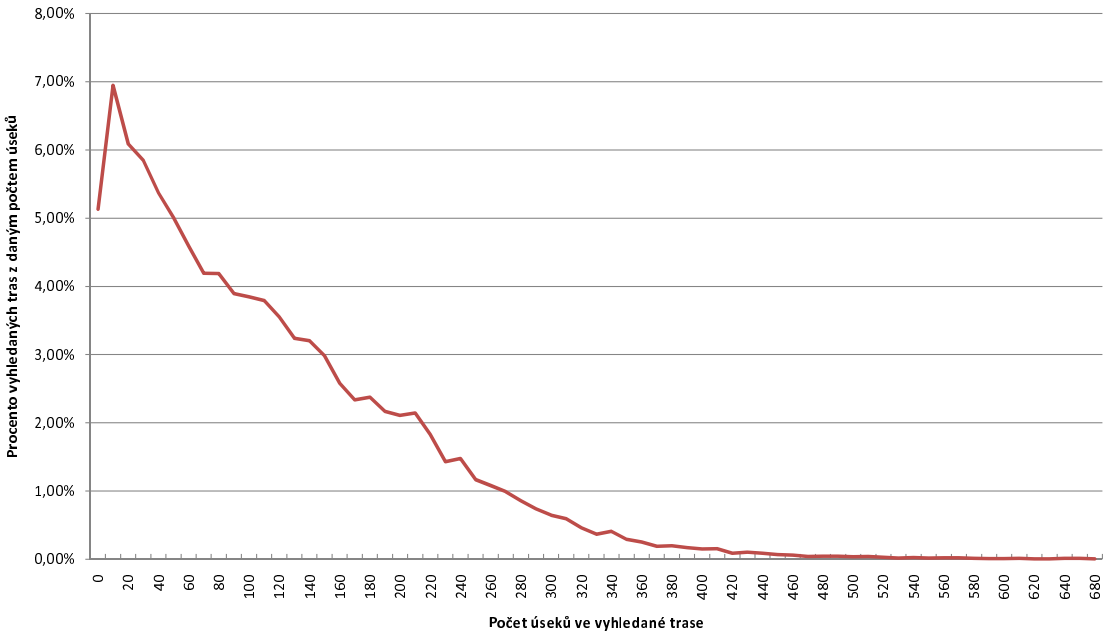
5.2 Výsledky

Do statistik bylo zahrnuto celkem 217515 tras. Informace o vyhledaných trasách, jejích kritériích, délkách a počtech úseků lze nalézt v tabulce 5.1.

Kritérium	Počet tras	∅ počet úseků	∅ délka (km)
auto - nejkratší	33234	149,15	272,27
auto - nejrychlejší	118402	121,49	296,83
auto - ekonomická	23342	137,80	332,38
cyklo - min. pref.	14939	79,23	74,65
cyklo - norm. pref.	18928	56,57	66,02
cyklo - max. pref.	8670	90,35	88,35
∅	**217515**	**117,67**	**253,24**

Tabulka 5.1: Statistiky tras vyhledaných při zátěžovém testování.

Vyhledané trasy měly různou délku a byly složeny z různého počtu úseků. Jejich rozložení je k vidění v grafu 5.1. Z grafu a tabulky 5.1 je patrné, že trasy s vysokým počtem úseků nejsou vyhledávány příliš často.



Obrázek 5.1: Počet vyhledaných tras s daným počtem úseků.

5.2.1 Srovnání počtů zpracovaných vrcholů

Ve srovnání s Dijkstrovým algoritmem potřebuje A star pro vyhledání trasy průměrně zpracovat o **50,86%** vrcholů méně. Průměrné počty zpracovaných vrcholů se liší s ohledem na kritérium, kterým byla trasa vyhledána. Tabulky 5.2 a 5.3 obsahují podrobné výsledky měření pro oba algo-

ritmy a jednotlivá kritéria. Zároveň obsahují průměrné doby vyhledání tras za použití Fibonacciho a K-ární haldy.

Z tabulek 5.1 a 5.3 je patrné, že ačkoliv nejkratší trasy obsahují více úseků než trasy nejrychlejší, jejich výpočet algoritmem A star trvá kratší dobu. Tento rozdíl je způsoben méně přesnou heuristikou funkcí použitou při vyhledávání nejrychlejších tras. Heuristika je postavena „opatrně“ s ohledem na omezení popsaná v kapitole 3.2.3. Pokud by byla zvolena „přesnější“ heuristika, mohl by Astar vyhledat nejrychlejší a ekonomické trasy efektivněji a rychleji.

	∅ **doba výpočtu (ms)**		
Kritérium	Fibonacciho	K-ární	∅ **počet zprac. vrcholů**
auto - nejkratší	99,39	29,69	39240,47
auto - nejrychlejší	152,67	32,15	40634,97
auto - ekonomická	150,38	35,69	44742,65
cyklo - min. pref.	41,28	15,89	13982,84
cyklo - norm. pref.	29,23	10,97	8596,53
cyklo - max. pref.	64,08	16,47	13450,63
∅	**122,36**	**30,61**	**35160,72**

Tabulka 5.2: Výsledky měření při vyhledání tras Dijkstrovým algoritmem.

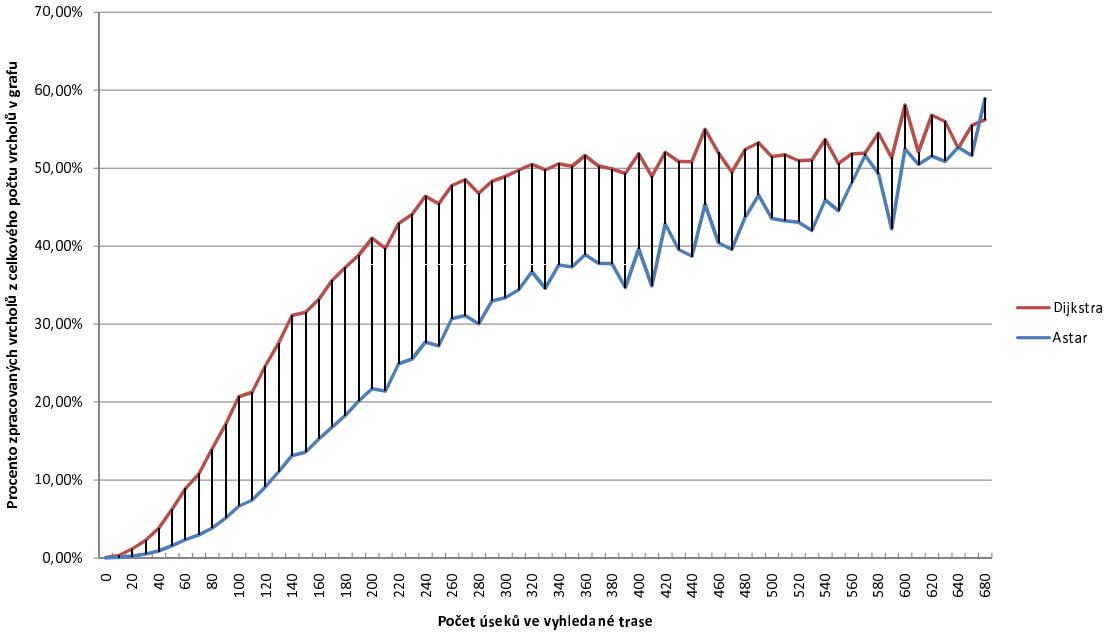
	∅ **doba výpočtu (ms)**		
Kritérium	Fibonacciho	K-ární	∅ **počet zprac. vrcholů**
auto - nejkratší	58,12	13,82	17501,37
auto - nejrychlejší	97,35	17,68	20907,09
auto - ekonomická	86,65	16,78	19488,13
cyklo - min. pref.	26,81	8,33	6865,59
cyklo - norm. pref.	17,39	6,17	3983,79
cyklo - max. pref.	48,39	9,74	7855,26
∅	**76,45**	**15,96**	**17277,20**

Tabulka 5.3: Výsledky měření při vyhledání tras algoritmem A star.

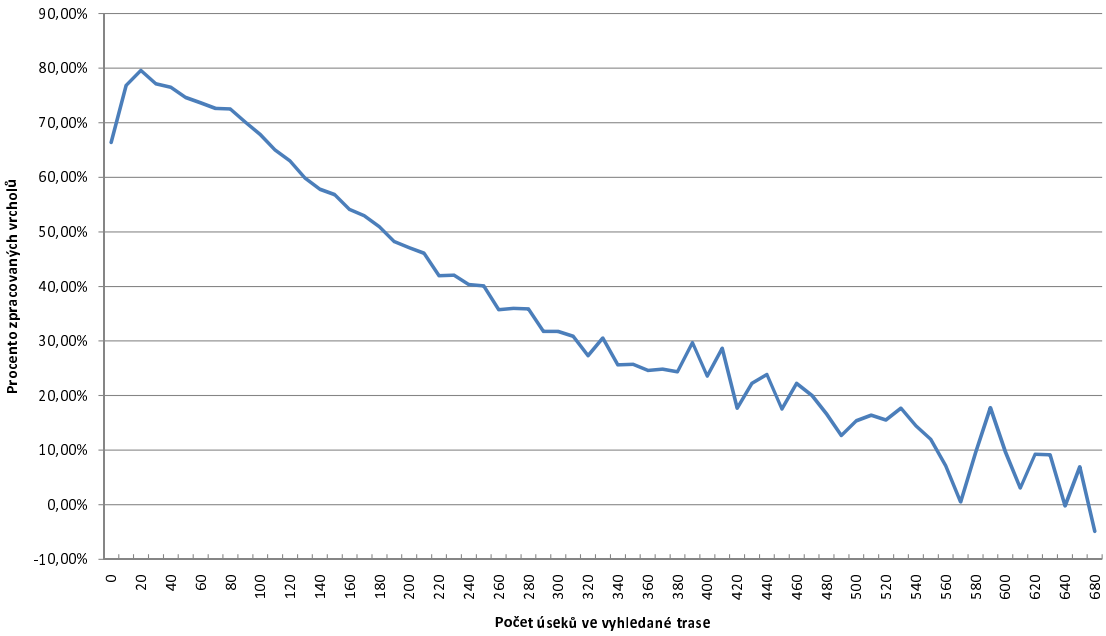
Počty zpracovaných vrcholů rostou v závislosti na počtu úseků ve vyhledané trase. Jak je vidět na grafu 5.2, A star je mnohem efektivnější na kratších trasách, ale na delších se efektivitou blíží k Dijkstrovi (viz percentuální porovnání počtů zpracovaných vrcholů v grafu 5.3).

5.2.2 Doba výpočtu

Z tabulek 5.1 a 5.3 lze vidět průměrné doby vyhledání trasy pro oba algoritmy a rozdíly mezi dobama při použití Fibonacciho a K-ární haldy. Nejrychlejší je pak varianta algoritmu A star



Obrázek 5.2: Porovnání počtu zpracovaných vrcholů v závislosti na počtu úseků v trase.

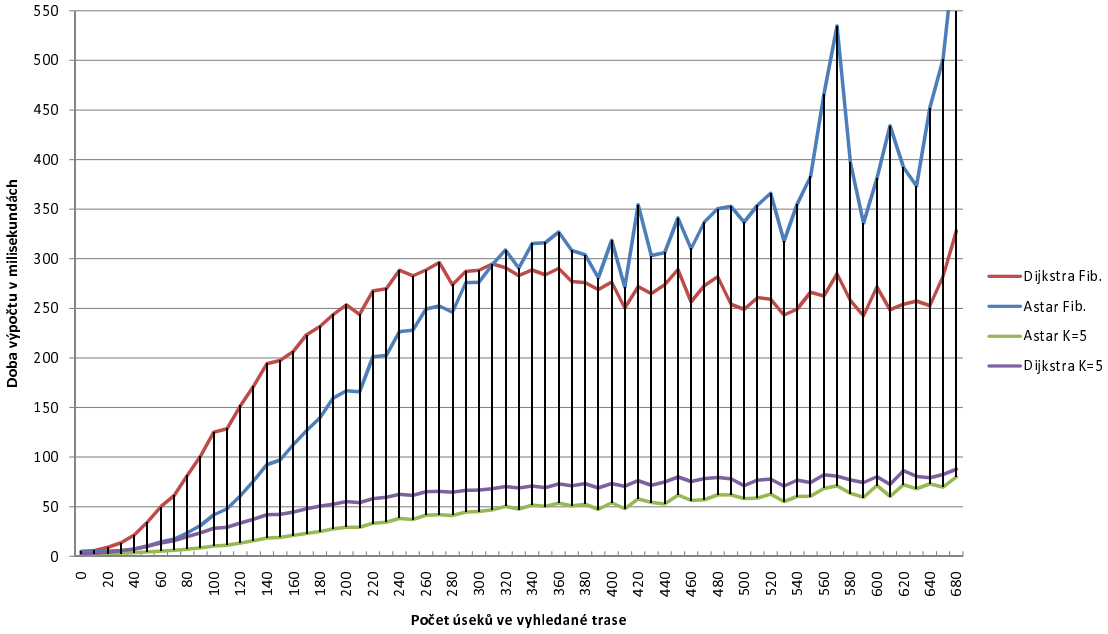


Obrázek 5.3: Percentuální rozdíl počtů zpracovaných vrcholů algoritmem A star oproti Dijkstrovo algoritmu v závislosti na počtu úseků ve vyhledané trase. Algoritmus A star zpracuje při výpočtu kratších tras o 70 - 80% vrcholů méně, než Dijkstra.

s využitím K-nární haldy (s $K = 5$), která všech $217\,515$ tras vyhledala v průměrném čase $15{,}96$ ms. Srovnání průměrných dob výpočtů jednotlivých variant je k vidění na obrázku 5.4.

Při použití K-ární haldy a Dijkstrova algoritmu byla trasa vypočítána v průměrném čase $30{,}61$ ms. A star byl v průměru o $47{,}86\%$ rychlejší s $15{,}96$ ms. Při použití Fibonacciho haldy byl A star oproti Dijkstrovi pouze o $37{,}52\%$ rychlejší (Dijstra: $122{,}36$ ms, A star: $76{,}45$ ms).

Průměrná doba zpracování jednoho vrcholu je u Dijkstrova algoritmu $0{,}8706$ μs u algoritmu A star je trochu vyšší s $0{,}9238$ μs. Tento drobný rozdíl je pravděpodobně způsoben jednak tím, že A star musí v každé iteraci zkoumat i uzavřené vrcholy (narozdíl do Dijkstry), ale také výpočtem heuristické funkce. Tato funkce obsahuje odmocninu (ve výpočtu eukleidovské vzdálenosti), která je pak v rámci této funkce při zpracování jednoho vrcholu počítána několikrát. Pro potřeby heuristické funkce ale zjištění přesné odmocniny není potřeba. Pokud bychom nahradili volání standardní funkce *Math.sqrt* rychlejší funkcí, výkon algoritmu A star by mohl vzrůst stejnou měrou. V současné době se při výpočtu trasy na uchování dočasné délky trasy používá typ double. Tak velká přesnost ale není při plánování trasy vůbec potřeba a bohatě by postačil i celočíselný typ int. Pro celá čísla pak existuje celá řada rychlých aproximací druhé odmocniny.



Obrázek 5.4: Doba trvání vyhledání trasy Dijkstrova a A star algoritmu v závislosti na počtu úseků v trase.

5.2.3 Srovnání hald

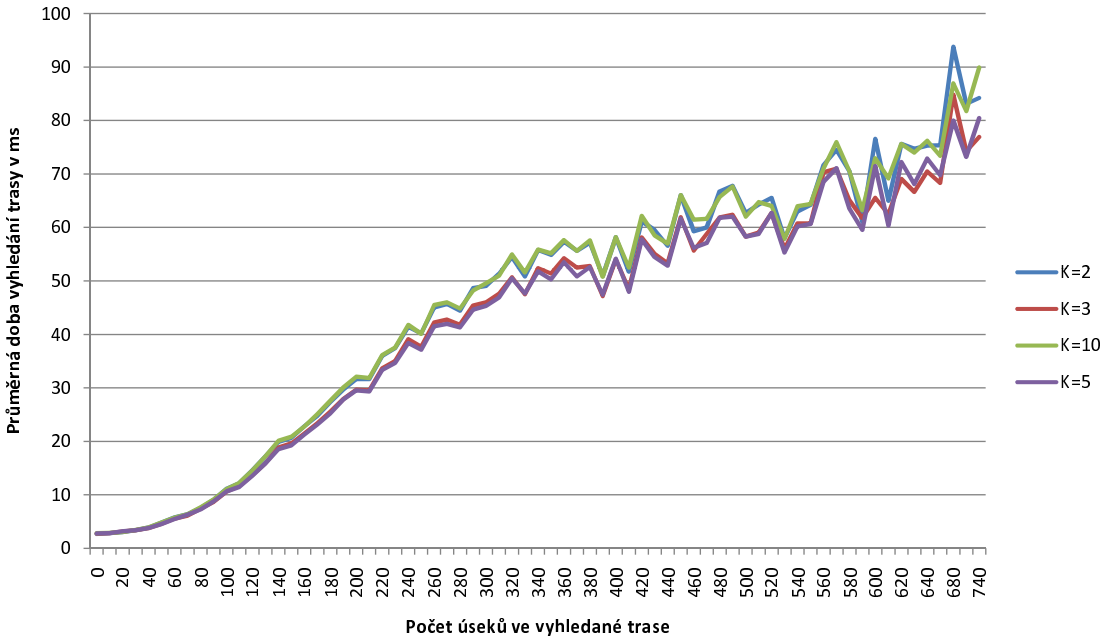
Testy jednoznačně ukázaly (viz. graf 5.4), že implementovaná K-ární halda je mnohonásobně rychlejší, než halda Fibonacciho. V tabulce 5.4 jsou uvedeny průměrné doby vyhledání tras při použití různých hald. V případě K-árních hald byly testy provedeny pro několik hodnot parametru

K. Nejrychleji byly trasy vyhledány pro $K = 5$. Pro $K = 3$ a $K = 4$ se však průměrné hodnoty příliš neliší.

Graf srovnávající chování hald pro trasy s různými počty úseků lze nalézt na obrázku 5.5.

Typ haldy	**∅ doba vyhledání trasy v ms**	**Rozdíl oproti nejlepšímu**
Fibonacci	77,164	483,39%
K-ární ($K = 2$)	16,974	106,33%
K-ární ($K = 3$)	16,091	100,80%
K-ární ($K = 4$)	16,183	101,38%
K-ární ($K = 5$)	15,963	100,00%
K-ární ($K = 6$)	16,419	102,86%
K-ární ($K = 10$)	17,026	106,66%

Tabulka 5.4: Průměrné doby vyhledání tras při použití různých hald.



Obrázek 5.5: Doba trvání vyhledání trasy algoritmem A star s K-ární haldou s různými hodnotami parametru K v závislosti na počtu úseků v trase.

Kapitola 6

Zhodnocení, nasazení a ukázkové aplikace

6.1 Zhodnocení implementovaného serveru

Výsledky zátěžových testů ukázaly, že server je schopen vyhledat trasu v průměrném čase $15{,}96$ ms. Doba vyhledání tras sice může přesáhnout (u tras s více jak 600 úseky) 70 ms, ale takovéto trasy jsou vyhledávány velmi zřídka (viz graf 5.1). Server je stabilní a je schopen zvládat velký počet dotazů.

Požadavek na server, ve kterém je trasa vyhledána a je vytvořen a vrácen její itinerář ve formátu XML, je vyřízen v průměrném čase $27{,}60$. Vzhledem k tomu, že server efektivně využívá více procesorů, je za jednu vteřinu teoreticky schopen vyřídit $36{,}23 \cdot N$ požadavků, kde N je počet procesorů.

Stará verze serveru trasu vyhledala v průměrném čase $144{,}47$ ms a požadavek vyřídila průměrně v $173{,}76$ ms. Nový navigační server tedy trasu naplánuje $9\times$ rychleji a požadavek vyřídí zhruba $6{,}3\times$ rychleji. Nejdelší doba odpovědi na požadavek klesla z $2\,672$ ms na 407 ms, tedy $6{,}6\times$. Díky implementované vyrovnávací paměti (jejíž využití bylo v průběhu zátěžových testů zakázáno) bude průměrná doba vyřízení požadavku ještě menší. Vzhledem k tomu, že do vyrovnávací paměti jsou ukládany i části vyhledaných tras, umí server efektivně vyhledávat i trasy, jejichž destinace jsou zadávany postupně (a trasa pokaždé znovu vyhledána), nebo trasy, ve kterých došlo ke změně polohy její destinace.

Byl představen a implementován popis vyhledané trasy s pomocí navigačních povelů. Oproti staré metodě je přehlednější a lépe popisuje všechny důležité úseky trasy.

Navigační server byl nasazen na ostrý server firmy PLANstudio, běží bez restartu několik měsíců a vyřizuje několik desítek tisíc požadavků denně.

Server je v současné době využíván desítkami klientů, mezi nejznámější z nich patří mapový portál www.mapy.cz, mapy.idnes.cz nebo mapy.pravda.sk. Podrobný seznam naleznete v kapitole 6.4.

6.2 Dokumentace, instalace

K serveru byla sepsána podrobná uživatelská a referenční dokumentace (příloha A). V této dokumentaci lze nalézt popis instalace navigačního serveru a jeho konfigurace. Dále obsahuje příklady a ukázky práce se serverem a popis všech příkazových a návratových funkcí. V neposlední řadě obsahuje též popis a strukturu výstupních XML.

Na přiloženém CD (příloha B) je k dipozici instalační balíček servletů navigačního serveru (war soubor). Navigační data této instalace jsou však omezena na Znojemský kraj.

6.2.1 Přímý přístup k navigačním servletům

Pro potřeby obhajby diplomové práce byla se svolením firmy PLANstudio zprovozněna aplikace umožňující testování příkazového rozhraní servletů na adrese.

http://netmap.planstudio.cz/servlet/.

Oponent a členové komise mohou využívat tuto aplikaci k testování rozhraní servletů. V případě potřeby je možné zřídit i přímý přístup k navigačním servletům, ale v takovémto případě je nutné zaslat a povolit IP adresu, ze které se bude k servletům přistupovat. Servlety jsou dostupné na URL:

http://tomcat.planstudio.cz:8080/omsmap/servlets/map.

Možností navigačního serveru lze však mnohem lépe odzkoušet prostřednictvým ukázkové aplikace (6.3), nebo v jiných aplikacích (6.4), které služeb navigačního serveru využívají.

6.3 Ukázková aplikace

Pro potřeby diplomové práce a předvedení možností navigačního serveru byla na adrese:

http://netmap.planstudio.cz/ceskoapi/

zprovozněna interaktivní internetová mapová aplikace umožňující snadné odzkoušení většiny možností implementovaného serveru v praxi (obrazovka ukázkové aplikace je zachycena na obrázku 6.1). Aplikace vyžaduje pro svůj běh internetový prohlížeč Firefox 2.0 a vyšší nebo Internet Explorer 6.0 a vyšší. V jiných prohlížečích aplikace může fungovat, ale nebyla v nich testována. Aplikace umožňuje:

- moderní zobrazení a ovládání map v grafickém webovém prohlížeči,
- zobrazení kontextového menu kliknutím pravého tlačítka nad mapou (zřejmě nefunguje na systémech Linux / Unix),
- přidání průjezdních bodů trasy přes kontextové menu (po přidání více jak dvou průjezdních bodů je trasa automaticky vypočtena / přepočtena),
- změnu kritéria a parametrů vyhledané trasy a její okamžité přepočítání,
- zobrazení vyhledané trasy v mapě,

- výpis popisu trasy ve formě navigačních povelů,
- zobrazení informací o nejbližší komunikaci,
- zobrazení značených cyklotras v podobě průhledné vrstvy kladené nad mapu,
- zobrazení statistických informací v podobě průhledných vrstvy kladených nad mapu (viz kapitoly 2.1.4 a 2.1.5),

Ovládání aplikace je podobné ovládání ostatních mapových aplikací na internetu, proto bude popsáno velmi stručně:

Ovládání mapy

Mapu lze posunout tažením myši v mapě (levým tlačítkem). Levým dvojklikem do mapy je mapa přiblížena (přepnuto na detailnější vrstvu). Pravým dvojklikem oddálena. Přiblížit a oddálit mapu lze též provést rolováním kolečka. Levým kliknutím je do mapy přidáno ukazovátko (zvýrazněno ikonkou zaměřovače). Pravým tlačítkem je zobrazeno kontextové menu umožňující operace nad zvoleným místem v mapě.

Přidání destinace

Pravým kliknutím do mapy vyvolejte kontextové menu. Zvolením položky „*Přidat destinaci*“ bude destinace přidána do trasy a poloha destinace bude zvýrazněna vlaječkou. Pokud trasa obsahuje více jak jednu destinaci, bude automaticky přepočítána.

Pokud se kontextové menu po kliknutí pravého tlačítka nezobrazí, pak lze postupovat následovně. Levým kliknutím do mapy je na tuto pozici vloženo ukazovátko. V pravé části pak kliknutím na odkaz „*Přidej jako destinaci*“ je na pozici ukazovátka vložena destinace.

Vyhledání trasy

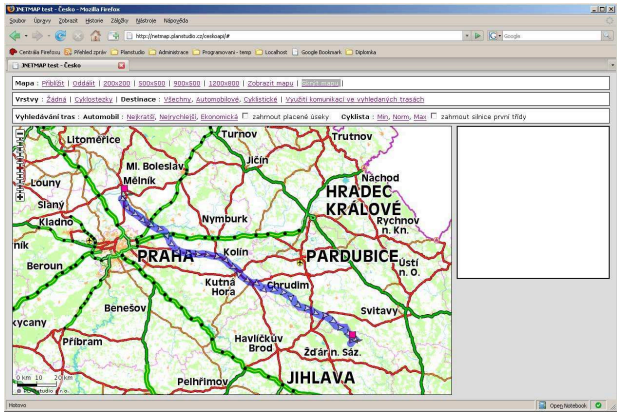
Po přidání dvou a více destinací je trasa automaticky vyhledána, znázorněna v mapě a do pravého pole vložen její itinerář. Po kliknutí na různá kritéria v poli „*vyhledávání tras*“, je trasa znovu vyhledána podle zvoleného kritéria či parametru.

Zobrazení průhledných vrstev nad mapou

Nad mapou lze zobrazit několik průhledných vrstev:

- **Cyklostezky** - značené cyklostezky na území ČR.
- **Destinace** - destinace, které byly použity v trasách v období 1.8.2007 - 31.3.2008 (všechny trasy, automobilové, cyklistické). Poloha destinace je do mapy zakreslena průhledným kolečkem. Čím větší poloměr, tím častěji se v trasách vyskytovala.
- **Využití komunikací ve vyhledaných trasách** - Červená značí velmi často využívané komunikace, oranžová často využívané, zelená zřídka, černá nikdy nevyužité komunikace.

V případě, že mapové podklady jsou příliš výrazné a splývají se zobrazenou vrstvou, pak je možné mapu schovat tlačítkem ”*Skrýt mapu*”.



Obrázek 6.1: Ukázková aplikace.

6.4 Klienti využívající navigační servlety

Navigační server byl již nasazen na ostrý server firmy PLANstudio a je využíván následujícími mapovými aplikacemi:

- *http://mapy.idnes.cz* - plánování silničních a cyklistických tras na území ČR, zobrazení výškového profilu vyhledané trasy,
- *http://www.czechtourism.com* - plánování silničních tras na území ČR,
- *http://mapy.pravda.sk*, - plánování silničních tras na území Slovenska,
- *http://www.nakole.cz* - plánování po cyklotrasách na území ČR,
- *http://www.betonserver.cz/betonarky* - vyhledání silničních vzdáleností z místa zadaného uživatelem k nejbližším betonárkám. Pro zobrazení mapy je využíváno mapové API od firmy Atlas.

Kromě toho jsou servlety součástí uzavřených mapových a informačních systémů:

- SV agency - lokalizace vozidel, vyhledávání tras.
- NAM - lokalizace vozidel.

6.4.1 www.mapy.cz

Největší firmou využívající navigační servlety je firma Seznam a.s., která servlety zakomponovala do vlastní mapové aplikace dostupné na adrese *http://www.mapy.cz*. V rámci této aplikace jsou servlety využívány pro vyhledávání silničních automobilových tras. Vzhledem k velké zátěži má firma Seznam servlety nasazené na několika vlastních serverech. Nasazení nové verze

navigačního serveru je plánováno na podzim 2008. V rámci aktualizace bude pravděpodobně přidáno plánování po cyklotrasách.

Firma vyjednává spolupráci s nově budovaným celorepublikovým systémem dopravních informací (*http://jsdi.dopravniinfo.cz*) a uvítala by možnost plánovat trasy s ohledem na aktuální informace o dopravních omezeních a uzavírkách. Původně toto byl jeden z požadavků na nový navigační server, ale firma Seznam nedodala včas testovací data ani jejich popis, a proto byl tento požavek z diplomové práce vyřazen.

Přílohy

A Dokumentace navigačních servletů

V dokumentaci navigačních servletů lze nalézt:

- popis instalace serletů,
- konfiguraci a nastavení servletů,
- popis logů a logových záznamů,
- praktické příklady a ukázky práce se servlety,
- popis všech příkazových a návratových funkcí servletu,
- popis XML vracených návratovými funkcemi servletu.

Dokumentaci lze nalézt na CD (příloha B) v adresáři **dokumentace**, nebo na internetové adrese:

http://data.voldrich.net/diplomka/omsroutingdoc.zip.

Součástí dokumentace jsou také XML soubory s ukázkami výstupů servletu. Na cd je naleznete v podadresáři

/dokumentace/ukazky XML/.

B Datové CD

K diplomové práci je přiloženo datové CD. Toto CD obsahuje:

/diplomova-prace.pdf
: Tento dokument ve formátu PDF.

/dokumentace/
: Dokumentace navigačních servletů (viz. příloha A).

/install/omsrouting.war
: Instalační balíček navigačních servletů.

/omsrouting/src/
: Zdrojové kódy navigačních servletů.

/omsrouting/libs/
: Knihovny potřebné pro zkompilování zdrojových kódů.

/algorithm-iterations/
: Videa znázorňující iterace Dijkstrova algoritmu a algoritmu A star při vyhledání silničních tras. Videa jsou ve formátu AVI a komprimované kodekem XVID. V podadresářích jsou také obrázky jednotlivých iterací algoritmů ve formátu PNG.

/logs/logfilter/
: Aplikace použitá při zpracování logových záznamů a zátěžového testování (*LogFilter*).

/logs/examples/
: Ukázky logových záznamů.

/logs/idnes.log
: Logové záznamy o trasách vyhledaných na serveru idnes.cz za období 1.8.2007 - 31.3.2008.

Obsah datového CD lze také nalézt na internetu na adrese:

http://data.voldrich.net/diplomka/omsroutingcd.zip.

Instalační balíček

Balíček omsrouting.war obsahuje navigační data v oblasti Znojemského kraje, tedy pouze obdélník s hraničními souřadnicemi 535697.00, 5375237.50 a 618172.00, 5451437.50 (UTM 33). Firma PLANstudio s.r.o. povolila použití navigačních dat v této oblasti pro distribuci spolu s diplomovou prací. Firma však nepovolila distribuci navigačních dat v původním textovém formátu, proto jsou k dispozici pouze data v již předzpracované podobě, které jsou využívány při inicializaci serveru (a jsou součástí instalačního balíčku). Servlety pracující nad úplnými navigačními daty jsou dostupné on-line (viz. poslední kapitola).

Literatura

[1] ANDREW V. GOLDBERG, HAIM KAPLAN, R. F. W. Efficient point-to-point shortest path algorithms. Tech. rep., Microsoft Research, October 2005.

[2] GUTTMAN, A. R-trees: a dynamic index structure for spatial searching. *Readings in database systems* (1988), 599–609.

[3] MICHAEL L. FREDMAN, R. E. T. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM 34*, 3 (July 1987), 596–615.

[4] SUN MICROSYSTEMS. *Java servlety*. http://java.sun.com/j2ee/, červen 2008.

[5] SUNGWON JUNG, S. P. Generalized best-first search strategies and the optimality of a*. *Journal of the ACM 32*, 3 (July 1985), 505–536.

[6] SUNGWON JUNG, S. P. Hiti graph model of topographical road maps in navigation systems. *IEEE Transactions on Knowledge and Data Engineering 14*, 5 (september/october 2002), 1029–1046.

[7] WIKIPEDIA. *Universal Transverse Mercator*. http://cs.wikipedia.org/wiki/UTM, červenec 2008.