Pedagogická fakulta
Faculty of Education

Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

Jihočeská univerzita v Českých Budějovicích

Pedagogická fakulta

Katedra informatiky

Diplomová práce

MMR – optimalizace cesty

Vypracoval: Bc. Vilém Janda

Vedoucí práce: Ing. Václav Novák, CSc.

České Budějovice 2014

Abstrakt

Cílem práce je navrhnout a realizovat optimalizační software, který bude umožňovat:

- Nalezení optimální cesty pro odečítače a to v kombinaci jízdy autem a ostatní dopravy
- Trajektorie musí být dynamicky přepočítána v průběhu cesty.
- Naprogramování příslušné aplikace do ručních terminálů.

Součástí práce je nalezení a implementace vhodného optimalizačního algoritmu. Na závěr budou provedeny testy aplikace v terénu a její vyhodnocení

Abstract

The aim is to design and implement optimization software that will allow:

- Finding the optimal way for the reader in combination car and other transport
- The trajectory must be dynamically recalculated during your journey.
- Programming of the application to the mobile terminals.

Part of the work is to find and implement a suitable optimization algorithm.To the end the outdoor application tests will be performed and evaluated.

Klíčová slova

Visual studio 2008, C#, .Net Compact Framework 3.5, Programovací jazyk, Smart device application, GPS, optimalizace cesty, problém obchodního cestujícího

Keywords

Visual studio 2008, C#, .Net Compact Framework 3.5, Programming language, Smart device application, GPS, journey optimalization, travelling salesman problem

Zadání práce

Společnosti shromažďující data o spotřebě energii, a to jak o spotřebované elektřině, tak plynu či vody, se dostávají do problému při selhání odečtů pomocí dálkových systémů RMR - Remove meter reading a SMR Smard Meter Reading. Vznikne tak na daném sledovaném území nesourodá množina neodečtených míst, které je potřeba objet a obejít s ručním terminálem a provést příslušné odečty. Navíc je potřeba do tohoto systému zahrnout i mimořádné nebo kontrolní odečty.

Úkolem diplomanta je navrhnout a realizovat optimalizační software, který musí umožňovat:

- Nalezení optimální cesty pro odečítače a to v kombinaci jízdy autem a ostatní dopravy
- Trajektorie musí být dynamicky přepočítána v průběhu cesty.
- Naprogramování příslušné aplikace do ručních terminálů.
- Student provede rovněž testy v terénu a příslušně je vyhodnotí.

Součástí práce je též nalezení vhodného optimalizačního algoritmu pro svoji aplikaci.

Poděkování

Děkuji vedoucímu práce za rady a vedení při její tvorbě.

Prohlášení

Prohlašuji, že svoji diplomovou práci jsem vypracoval/a samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své diplomové práce, a to v nezkrácené podobě ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne 30. dubna 2014

Obsah

1. Úvod

Společnosti shromažďující data o spotřebě energii, a to jak o spotřebované elektřině, tak plynu či vody, se dostávají do problému při selhání odečtů pomocí dálkových systémů RMR - Remove meter reading a SMR Smard Meter Reading. Vznikne tak na daném sledovaném území nesourodá množina neodečtených míst, které je potřeba objet a obejít s ručním terminálem a provést příslušné odečty. Navíc je potřeba do tohoto systému zahrnout i mimořádné nebo kontrolní odečty.

Cílem práce je navrhnout a realizovat optimalizační software, který musí umožňovat nalezení optimální cesty pro odečítače. Požadavkem je, aby toto hledání zahrnovalo kombinaci jízdy autem a ostatní dopravy (chůze). Trajektorie bude dynamicky přepočítávána v průběhu cesty. Tento software bude implementován do ručních terminálů, otestován v terénu a vyhodnocen.

2. Cíle práce a metodika

Tato práce se zaměřuje na jednookruhový okružní dopravní problém (problém obchodního cestujícího) a jeho řešení. Bude uveden přehled některých metod, které se k řešení tohoto problému dají využít. Metody budou vycházet z literatury a dále budou navrženy jejich úpravy sloužící k optimalizaci řešení tohoto problému.

Jedná se o optimalizace na úrovni řešení algoritmu a optimalizace na technické úrovni při získávání vstupních dat pro výpočet řešení.

Cílem práce je nalezení vhodného optimalizačního postupu pro vyřešení jednookruhového okružního dopravního problému, a to tak, aby splňoval zadaná kritéria:

- Vstup pro více uzlů
- Dynamický přepočet
- Rychlost výpočtu, která umožní do 10 s zobrazit výsledek operace
- Kombinace jízdy autem a pěší dopravy

Dalším cílem je návrh a realizace aplikace, která tento algoritmus implementuje. Návrh bude vycházet z UML diagramů představujících procesy fungující v dané firmě, která realizuje vlastní odečty. Na jejich základě se realizují Use Case diagramy určující chování a funkcionalitu výsledné aplikace. Příkladem funkcionality je: „Nalezení optimální cesty pro odečítače, a to v kombinaci jízdy autem a ostatní dopravy“, „Trajektorie musí být dynamicky přepočítána v průběhu cesty“. Dále bude proveden návrh objektové struktury aplikace pomocí Class diagramů, následně pak realizace této aplikace.

Na závěr se provede vyhodnocení aplikace vůči zadaným kritériím, a to jednak vůči kritériím vlastního optimalizačního algoritmu, ale i oproti kritériím a požadované funkcionalitě vlastní aplikace.

K vyhodnocení efektivity optimalizačního algoritmu se provede měření doby potřebné pro nalezení optimální trasy pro zadaný počet uzlů určených k hledání.

3. Teorie grafů

3.1. Graf

Teorie grafů je jednou z nejvýznamnějších součástí současné diskrétní matematiky. Pod pojmem grafu se nachází struktura, která je složena z vrcholů (uzlů představující pevné body grafu) a hran. Hrana spojuje vždy dvojici vrcholů navzájem mezi sebou. [2]

Graf je definován jako dvojice 2 množin: množiny vrcholů a množiny hran. V některých případech se používá termín uspořádaná dvojice, tímto termínem je myšleno, že nejprve jsou uvedeny vrcholy a teprve pak hrany.

V případě, že dvojice vrcholů jsou spojeny více než jednou hranou, říkáme takovým grafům multigrafy. [11]

Dle definice *„graf je uspořádaná dvojice G = (V,E), kde V je množina vrcholů a E je množina hran – množina vybraných dvouprvkových podmnožin množiny vrcholů*“. [12]

3.2. Stupeň vrcholu

U grafu nás bude zajímat, kolik z kterého vrcholu vychází hran, neboli kolik má vrchol sousedů. Tuto informaci definuje pojem stupeň vrcholu.

„Stupněm vrcholu u v grafu G rozumíme počet hran vycházejících z u“. [2][12]

3.3. Podgraf, Isomorfismus

Podgraf představuje část grafu. *„Podgrafem grafu G rozumíme libovolný graf H na podmnožině vrcholů V(H) ⊆ V(G), který má za hrany libovolnou podmnožinu hran grafu G majících oba vrcholy ve V(H). Tedy H ⊆ G.*“ [12]

„Indukovaným podgrafem je podgraf takový, který obsahuje všechny hrany grafu F mezi dvojicemi vrcholů z V(H).“ [12]

„Izomorfismus určuje podobnost grafů. Jde o bijektivní (vzájemně jednoznačné) zobrazení f: V(G) -> V(H), pro které platí, že každá dvojice vrcholů u,v ∈ V(G) je spojená hranou v G právě tehdy, když je dvojice f(u), f(v) spojená hranou v H.“ [12]

„Grafy G a H jsou isomorfní, pokud mezi nimi existuje isomorfismus.“[12]

3.4. Orientované grafy

Využijeme je v případech, ve kterých potřebujeme pro každou hranu určit její směr. Orientovaný graf je uspořádaná dvojice vrcholů (D) = (V, E), kde E ⊆ V x V. Relace orientovaných grafů nemusí být symetrické. [12]

Orientovaná hrana se v obrázcích kreslí šipkou.

3.5. Další pojmy teorie grafů

„Podgraf H ⊆ G, který je isomorfní nějaké kružnici, říkáme kružnice v G“. [2]

„Podgrafu H ⊆ G, který je isomorfní nějaké cestě, říkáme cesta v G“. [2]

„Podgrafu H ⊆ G, který je isomorfní nějakému úplnému grafu říkáme klika v G“. [2]

„Podmnožině vrcholů X ⊆ V(G), mezi kterými nevedou v G žádné hrany, říkáme nezávislá množina X v G“. [2]

„Indukovanému podgrafu H ⊆ G, který je isomorfní nějaké kružnici, říkáme indukovaná kružnice v G“. [2]

3.6. Implementace grafů

Dalším důležitým bodem je způsob zápisu grafů pro počítačovou implementaci. (řeší bod 5.2). V základu máme dva způsoby.

Prvním způsobem je matice sousednosti neboli dvourozměrné pole g[][], kdy g[i][j]=1 představuje hranu mezi vrcholy *i* a *j*.

Druhým způsobem je výčet sousedů, kde se opět užije dvourozměrné pole g[][] a k němu pole h[], které představuje stupně vrcholů, kdy g[i][0],g[i][1],…g[i][h[i]-1] udává seznam sousedů vrcholu i. [12]

4. Problém obchodního cestujícího

Jde o známý kombinatorický problém. Historicky pochází z roku 1759, kdy matematik Leonhard Euler začal zkoumat a více se zabývat touto problematikou. Největší rozvoj a zájem o tento problém nastal s rozvojem výpočetní techniky a lineárního programování. Jde o klasický problém při optimalizaci ideálních tras v kombinaci s technologií GPS a navigací. Další praktické oblasti, kde má tento problém velké využití, je logistika, krystalografie, plánování atd. [1]

4.1. Definice Problému

„Problém obchodního cestujícího (Travelling salesman problem nebo také Travelling salesperson problem) je zadán takto:

Je dáno n měst a vzdálenosti mezi nimi. Úkolem je najít okružní cestu přes všechna města s minimální celkovou vzdáleností (tj. najít uzavřenou hamiltonovskou cestu).

Celková složitost problému je úměrná počtu všech permutací z n prvkové posloupnosti a je dána výrazem O(n!).“[2][12]

Dle definice L. Kučery: „*Problém obchodního cestujícího je formulován takto: Je dána množina M a pro každé dva její prvky x, y je dáno číslo d(x, y), které budeme nazývat vzdáleností x a y. Cílem je určit, v jakém pořadí má obchodní cestující projet prvky množiny M („města“) tak, aby prošel každým městem právě jednou a pak se vrátil do místa, kde cestu začal, a urazil při tom vzdálenost co možná nejmenší.*

Jinými slovy, hledáme uspořádání prvků množiny M do posloupnosti x1, …, xn, která obsahuje každý z prvků M právě jednou a takové, že součet

d(x1, x2)+ d(x2, x3)+…+ d(xn-1, xn)+ d(xn, x1) je nejmenší možný.“ [5][1]

Problém je možno definovat také pomocí grafů; základem je úplný graf s nejméně třemi body, kterými je nutno projít, pro méně bodů by nemělo smysl se problémem zabývat. Graf musí mít kladně váhově ohodnoceny hrany a cílem je získat hamiltonovský cyklus takový, aby součet ohodnocených hran byl co nejmenší. [3]

Dále je v tomto případě nutno zabývat se orientováním hran grafu (zda je ulice jednosměrná či nikoli), pokud jsou některé hrany grafu orientované a jiné ne, řešení úlohy je NP-těžké.

$$
\begin{aligned}
10! &= 3628800 \approx 0.36*10^{7} \\
20! &= 2432902008176640000 \approx 0.24*10^{9} \\
&\dots \\
100! &= 93326215443944152681699238856266700490715968264381621468 \\
&\quad 59296389521759999322991560894146397615651828625369792082 \\
&\quad 722375825118521091686400000000000000000000000 \approx 0.93*10^{158}
\end{aligned}
$$

Stirlingův vzorec $n! \approx \sqrt{2\pi n}\,.n^{n}.e^{-n} = \sqrt{2\pi n}.e^{n.(\ln n-1)}$

Obrázek 1: Stirlingův vzorec, převzato z [5][1]

Problém obchodního cestujícího lze dále dělit na jednookruhový nebo víceokruhový. Pro potřeby této práce se budeme zabývat pouze první variantou. Varianta víceokruhových úloh se v praxi vyskytuje při plánování trasy více vozidel z jednoho plánovacího místa, kterými je potřeba všechny plánované uzly pokrýt. [1]

4.2. Složitost algoritmu

Obecně jde o NP-úplný problém. [6]

Třída NP matematicky znamená úlohy, které lze v polynomiálním čase *„redukovat na úlohu určování splnitelnosti logických formulí“*[1]. NP „nedeterministicky polynomiální“ je třída problémů, jež lze řešit pomocí polynomiálního nedeterministického algoritmu. [9]

„Nedeterministický algoritmus se od deterministického liší tím, že v některých nebo ve všech krocích výpočtu je možné se zcela libovolně rozhodnout pro některé z několika možných pokračování výpočtu. Výpočet tedy není jednoznačně určen počáteční konfigurací a místo výpočet bychom měli spíše říkat systém možných výpočtů.“ [5][1]

„Nedeterministický algoritmus neurčuje pro daná data jedinou posloupnost operací, nýbrž celý systém přípustných výpočtů, které dávají různé výsledky.“ [5][1]

Jinými slovy není pouze jeden jednoznačný výsledek, ale množina těchto výsledků, ze kterých je nutné vybrat ten nejvýhodnější.

Pro určité úlohy existuje polynomiální neboli deterministický algoritmus. Tyto úlohy patří do třídy NP. [1]

Existuje seznam problémů, které je možné navzájem redukovat v polynomiálním čase převodem z jednoho na druhý, jinak řečeno všechny jsou stejně algoritmicky složité. Pro tyto úlohy se užívá název NP-úplné. Dnes jich v této třídě známe několik stovek. Ukázalo se, že všechny tytu úlohy třídy NP lze redukovat na třídu NP-úplné. Tedy NP-úplné úlohy jsou nejsložitější z třídy NP. [6][1]

Z definice s využitím teorie grafů *„může být požadavek projít všechny uzly grafu nahrazen požadavkem projít všechny hrany*“. Tento postup poprvé navrhl Kuan [7] a bývá

označován jako *„problém čínského listonoše“*.[1] Lze použít představu, že průchod po hranách odpovídá *„průjezdu ulicemi, které musí listonoš absolvovat“* [1] (obecně se takovýmto úlohám říká problém listonoše). Byl nalezen *„polynomiální algoritmus pro případy, kdy je graf neorientovaný“* (neexistují jednosměrné ulice) [8] [1], nebo kdy naopak jsou všechny hrany orientované, (tj. všechny ulice jsou jednosměrné). [1]

Pouze pro tyto dva případy okružních úloh lze efektivně nalézt přesné řešení. *„Jsou-li některé hrany neorientované a některé orientované (některé ulice jednosměrné a jiné nikoliv), zůstává úloha obecně NP- těžká.“* [1]

4.3. Řešení z teorie grafů

4.3.1. Hamiltonovská cesta

Hamiltonovská cesta představuje kostru grafu.

Odstraníme-li ze zadaného grafu hranu, která neleží na nejkratší hamiltonovské cestě, řešení úlohy, tedy nalezení nejlevnější hamiltonovské cesty, se nezmění. [2] [12]

4.3.2. Hledání nejkratší hamiltonovské cesty

1. $w_{min} := \infty$;
2. *Najdeme minimální kostru K daného grafu G.*

   (a) Tvoří-li tato kostra hamiltonovskou cestu (tj. všechny vrcholy v této kostře mají stupeň ≤ 2), pak tato kostra je nejkratší hamiltonovskou cestou (kdyby totiž existovala nějaká kratší hamiltonovská cesta, byla by zároveň kratší kostrou, což by byl spor).

   (b) K obsahuje vrchol x tak, že $d_K(x)=k \geq 3$, a tedy některá z hran z $H_K(x)$ neleží ve výsledné nejkratší hamiltonovské cestě. Sestrojíme zmenšené grafy $G_1, G_2, \ldots, G_k$ takové, že $H(G_i) = H(G) - \{h_i$ vycházející z vrcholu x$\}$, $i=1, \ldots,k$ a zkusíme vyhledat minimální kostry zmenšených grafů.

i. G_i vynětím hrany přestal být souvislý, tj. neobsahuje již žádnou kostru (není třeba dále zkoumat).

ii. Minimální kostra grafu G_i tvoří hamiltonovskou cestu, délku (cenu) této cesty srovnáme s délkou dosud nejkratší hamiltonovské cesty a minimum si zaznamenáme

if *$w(K(G_i)) < w_{min}$*

then *begin $w_{min} := w(K(G_i))$*

zaznamenání hamiltonovské cesty

end*;*

iii . Nejlevnější kostra graf u G_i netvoří hamiltonovskou cestu.

if *$w(K(G_i)) > w_{min}$*

then *podúlohu dále nezkoumáme, protože vytržením hrany z původního grafu bez respektování optimálního pořadí vypouštění hran nemůže délka minimální kostry klesnout (kostra zmenšeného grafu je i kostrou v původním grafu), řešení podúlohy G_i tedy určitě nebude lepší než w_{min} (bude-li vůbec existovat)*

else *rekurzivně vyvoláme krok (b) pro G_i, tj. vyřazujeme hrany z G_i, atd.[12]*

„Uvedený algoritmus je použitelný pouze pro úlohy „menšího“ rozsahu, protože problém nalezení hamiltonovské cesty patří do třídy NP-úplných úloh, jejichž složitost s rozsahem úlohy exponenciálně roste. Při „velkém“ rozsahu úlohy je nutné použít aproximativní nebo heuristické metody. Oblíbené jsou zejména stochastické heuristické metody (metaheuristiky), mezi něž patří genetické algoritmy (genetic algorithms, GA), simulované žíhání (simulated annealing, SA), zakázané prohledávání (tabu-search, TS) a horolezecký algoritmus (hill climbing, HC).“ [2]

4.3.1. Hladové a grafové algoritmy

K dispozici máme takzvané hladové algoritmy (greedy algorithms). Principy těchto algoritmů jsou jednoduché, umožňují nenáročnou a jasnou realizaci, ale v praxi jsou výpočetně poměrně hodně náročné. Jsou navrženy pro řešení problému hledání nejkratší cesty v grafu, nikoliv problému obchodního cestujícího. [1]

Grafové algoritmy (Dijkstrův, nebo Floyidův algoritmus) se požívají pro hledání nejkratší cesty v grafu.

4.3.2. Metoda nejbližšího souseda

Jde o nejjednodušší metodu pro hledání kostry grafu. Její princip spočívá v určení počátečního bodu, ze kterého se následně hledá nejkratší hrana, která z tohoto bodu vychází. Následně se pokračuje opět hledáním nejkratší hrany do dalšího bodu atd. Pro zpřesnění trasy se tato metoda může aplikovat na všechny obsažené uzly a porovnávat jejich výsledné hodnoty. [1]

4.4. Přehled dalších metod, které vytvářejí řešení

4.4.1. Metoda výhodnostních čísel

Metoda výhodnostních čísel je velice starý, ale stále používaný způsob pro řešení okružních úloh (problém obchodního cestujícího).

Pracuje tak, že nejprve označí počáteční uzel pomocí indexu 0. Dále se vypočítá číslo (nebo tzv. výhodnostní číslo) pro každou dvojici uzlů. Dle těchto čísel se trasy seřadí podle jejich výhodnosti a přidávají se do výsledného okruhu. Tímto postupem na závěr algoritmu nalezneme cestu, která obsahuje všechny hledané body. [1]

4.4.2. Vogelova metoda

Opět jde o metodu hledání kostry grafu. Vypočítají se diference pro řádek a sloupec. Z řádku, který má tuto diferenci největší, vybereme sloupec s minimální hodnotou. Trasu zapíšeme a odstraníme tento řádek i sloupec.

Takto odstraníme i trasy, které tvoří kruh, nebo které jsme již zařadili. Dále postupujeme obdobně, vypočítáme diference, vybereme řádek, který má největší diferenci a minimální hodnotu sloupce. Celý postup opakujeme, dokud nezbydou poslední 2 trasy. Ty se pak aplikují do okruhu. [1]

4.5. Metody zpřesňující řešení

Metody, které vytvářejí řešení, vytváří řešení od úplného počátku; oproti tomu metody, které zlepšují řešení, dostanou na počátku již nějaké vyhotovené řešení a to postupně upravují k lepšímu výsledku. [1]

Metody tohoto typu pracují na principu nahrazení několika hran okruhu pomocí stejného počtu jiných hran. A to tím způsobem, aby nakonec opět vznikl okruh, který ale bude mít nižší hodnotu účelové funkce. Iterací tohoto postupu se dosáhne optimálních výsledků. Iterace probíhá, dokud je metoda schopna výměny. [1]

4.6. Stochastické heuristické metody

„*Postup heuristické metody je formulován tak, aby účelová funkce dosahovala vysoké (u maximalizace, u minimalizace nízké) hodnoty. U většiny heuristických metod je odhad odchylky dosažené hodnoty účelové funkce od optimální hodnoty účelové funkce vysoký, nicméně praktické výsledky jsou uspokojivé.*

Heuristické metody na rozdíl od exaktních jsou rychlé, polynomiální a nevykazují obtíže i při řešení rozsáhlých úloh. Navíc, protože nejsou exaktně formulované, umožňují flexibilní úpravy pro různé varianty typových úloh, pro specifické podmínky. Spíše než metody bychom je měli nazývat výpočetní postupy, neboť jakákoliv odchylka v algoritmu je také heuristickou metodou“ [4][1]

4.6.1. Genetický algoritmus

Genetické algoritmy patří do třídy evolučních algoritmů, které mimo ně zahrnují také evoluční programování, evoluční strategii a genetické programování. Jsou to vyhledávací algoritmy založené na mechanismu přirozeného výběru a principech genetiky. [3]

P(0) := {P_1, P_2, … , P_{Npop}};
P_{min} := Permutace_minimalizující_délku z P(0);
t := 0 ;
while *t < počet iterací* ***do***
begin repeat *Binarní_turnajový_výběr(P(t), rodič1, rodič2);*

```
                potomek := Modifikované_2_bodové_křížení(rodič1, rodič2);
                potomek := Posuvná_mutace(potomek)
        until not (potomek in populace P(t));
        Pmax := Permutace_maximalizující_délku z P(t);
        P(t +1) :=P(t) - {Pmax} u {potomek};
        if  Délka(potomek) < Délka(Pmin)
                then Pmin := potomek;
        t := t +1
end;  { Pmin  je aproximací permutace měst s minimální délkou cesty }
```

nastavení parametrů:

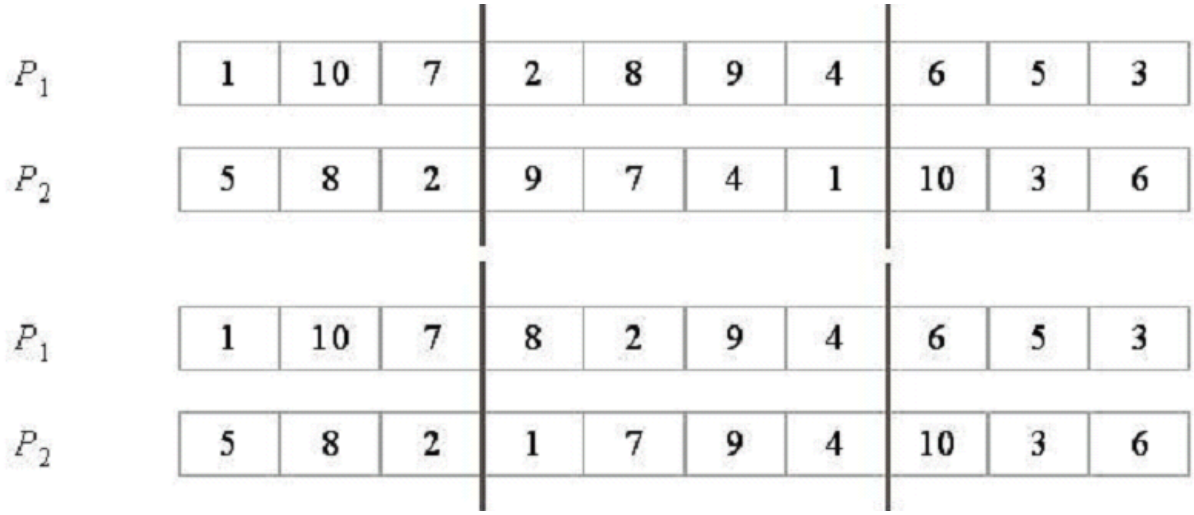
velikost populace $N_{pop} \in$ *<50, 200>*

počet iterací: 1000 - 50000

generování náhodné permutace

```
Randomize;
for i := 1 to n do
        perm[i] := i;
for i := n downto 2 do  { náhodné výměny dvojic prvků}
        begin j := 1+Trunc(i*Random)
                pom := perm[i];  perm[i] :=  perm[j];  perm[j] := pom
        end;
```

GA: modifikované 2-bodové křížení (modified two point crossover)



Obrázek 2: Princip modifikovaného 2- bodového křížení, převzato z [2]

GA: mutace(mutation)

- *vzájemná výměna dvou náhodně zvolených genů (exchange mutation)*
- *posuvná mutace (shift mutation)*

Obrázek 3: Posuvná mutace, převzato z [2]

Převzato z [2]

4.6.2. Další metody

Mezi další metody v této kategorii by patřila metoda Simulovaného žíhání, což je metoda, která prohledává stavový prostor a vychází z principu simulace žíhání oceli.

Dalšími metodami jsou Horolezecký algoritmus, Tabu-Search, evoluční strategie nebo různé další modifikace evolučních algoritmů. Bližší výzkum těchto metod přesahuje rozsah této práce. Dále se tedy budeme zabývat genetickým algoritmem popsaným výše. [2]

4.7. Problémy při řešení

Hlavní problém řešení nespočívá v tom, že by nebylo možné nalézt teoretický postup pro řešení daného problému, ale složitost problému je především ve výpočetní stránce.

„Existuje řada úloh z teorie grafů a operační analýzy, u nichž nejsme schopni zaručit nalezení optimálního řešení ani na nejvýkonnějších počítačích a většina z nich patří do třídy tzv. NP-úplných problémů. Někteří autoři dokonce o NP-úplných problémech mluví jako o úlohách z praktického hlediska neřešitelných.“ [5][1]

„Hranice mezi rychlým a pomalým algoritmem je tak neostrá, že proti každé striktní definici bude možné vznést řadu oprávněných námitek.

Čas, který potřebují ke zpracování vstupních dat velikosti n algoritmy považované za rychlé, bývá zpravidla shora omezen funkcemi typu n, logn, n^2, n^3 apod. Jako horní odhad lze vždy proto použít polynom.

Na druhé straně typický představitel pomalých algoritmů, metoda „hrubé síly“, která probírá všechny možnosti, vyžaduje alespoň 2^n kroků, probírají-li se všechny podmnožiny dané množiny o n prvcích, n! kroků, probíráme-li všechny permutace, a n^n kroků, probíráme-li všechna její zobrazení do sebe. Odhad počtu kroků je tedy alespoň exponenciální.

Budeme proto považovat algoritmus za rychlý, jestliže existuje polynom p tak, že počet kroků, které algoritmus provede při výpočtu, je omezen číslem p(n), kde n je velikost vstupních dat. Pokud takový odhad neexistuje, budeme algoritmus považovat za pomalý.

Abychom ilustrovali rozdíly mezi algoritmy pracujícími v polynomiálně omezeném čase a exponenciálními algoritmy, ukážeme v následující tabulce čas potřebný ke zpracování vstupních dat velikosti n, jestliže počet operací, které je nutné provést, je udán funkcí f(n) a provedení jedné operace trvá jednu mikrosekundu.“[5] [1]

Tabulka 1: Čas potřebný pro zpracování dat v závislosti na počtu nutných operací. (převzato z: [5] [1])

Operací	*N*							
	20	*40*	*60*	*80*	*100*	*200*	*500*	*1000*
N	*20 s*	*40 s*	*60 s*	*80 s*	*0,1 ms*	*0,2 ms*	*0,5 ms*	*1 ms*
Nlogn	*86 s*	*0,2 ms*	*0,35 ms*	*0,5 ms*	*0,7 ms*	*1,5 ms*	*4,5 ms*	*10 ms*
$_n2$	*0,4 ms*	*1,6 ms*	*3,6 ms*	*6,4 ms*	*10 ms*	*40 ms*	*0,25 s*	*1 s*
$_n3$	*8 ms*	*64 ms*	*0,22 s*	*0,5 s*	*1 s*	*8 s*	*125 s*	*17 min*
$_n4$	*0,16 s*	*2,56 s*	*13 s*	*41 s*	*100 s*	*27 min*	*17 h*	*11,6 dní*
$_2n$	*1 s*	*11,7 dní*	*36600 let*	*$3,6.10^9$ let*				
n!	*77000 let*							

„Další tabulka ukazuje rozdíl mezi polynomiálními a exponenciálními algoritmy. Ukazuje zvětšení rozsahu zpracovatelných úloh, které odpovídá zvětšení výpočetní rychlosti použitého počítače 10x, 100x, 1000x za předpokladu, že původně bylo možno v daném časovém limitu zpracovat vstupní data velikosti n = 100.“ [5] [1]

Tabulka 2: Zkrácení doby výpočtu při zvětšení výpočetní rychlosti (převzato z: [5] [1])

f(n)	*Zrychlení výpočtu*			
	1x	*10x*	*100x*	*1000x*
N	*100*	*1000*	*10000*	*100000*
Nlogn	*100*	*702*	*5362*	*43150*
n2	*100*	*316*	*1000*	*3162*
n3	*100*	*215*	*464*	*1000*
n4	*100*	*177*	*316*	*562*
2n	*100*	*103*	*106*	*109*
n!	*100*	*100*	*100*	*101*

„Z tabulek je vidět, že pro exponenciální algoritmy je typická existence mezní velikosti vstupních dat, nad níž je úloha neřešitelná i v případě, že by došlo ke zvýšení rychlosti počítače o několik řádů.

Je možné namítnout, že algoritmus, který pracuje v čase n^{100} nebo $2^{100}n$ je také nepoužitelný, i když časové omezení je polynomiální a ve druhém případě dokonce lineární. Ukazuje se ale, že pokud se pro nějakou úlohu „ze života“ podaří nalézt algoritmus

s polynomiálním časovým omezením, pak stupeň polynomu nebývá větší než 3 nebo 4 a i jeho koeficienty mívají „rozumnou" velikost."[5][1]

Při hledání rychlého a efektivního algoritmu pro určitou úlohu můžeme postupovat tak, že tento problém převedeme na jinou úlohu, pro kterou je již v požadované složitosti znám odpovídající algoritmus řešení. (viz. 3.2)

5. Analýza řešení

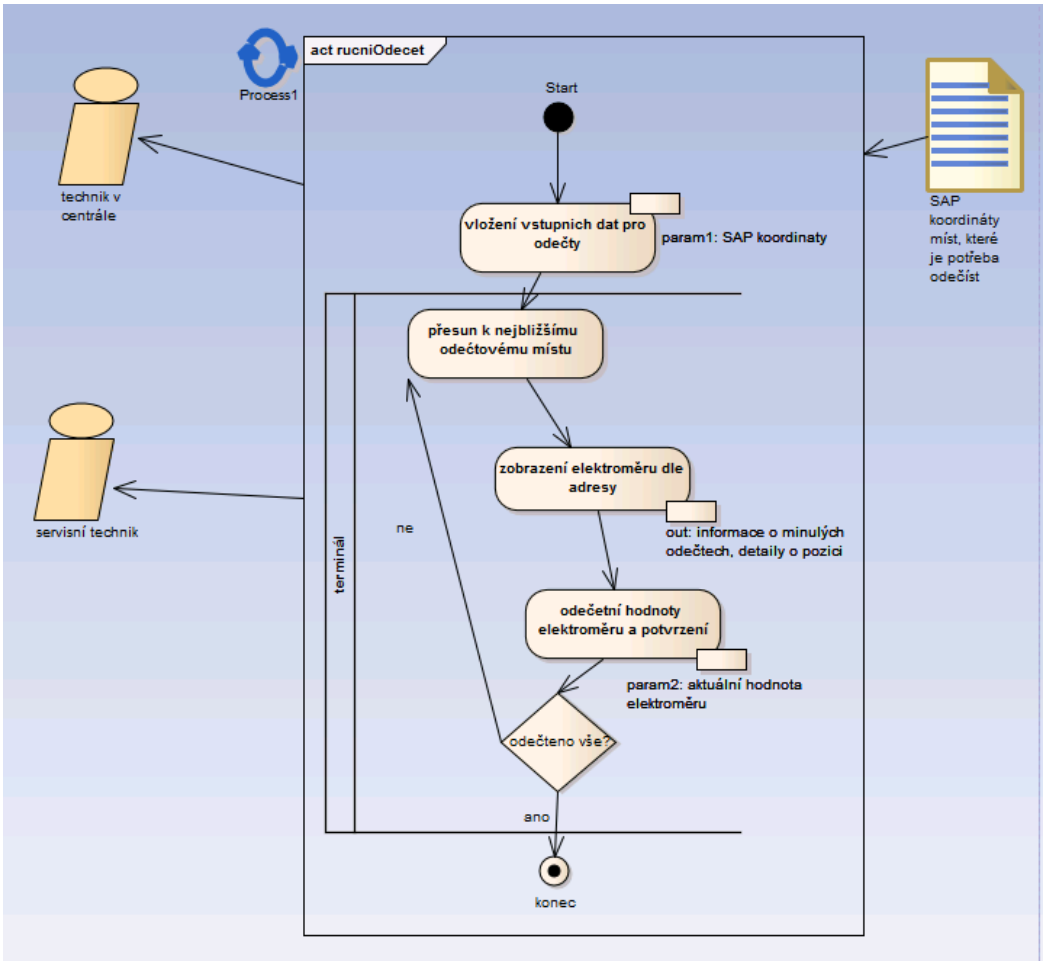
5.1. Procesní model

Slouží ke komplexnímu pohledu na podnikový informační systém. Ukazuje chování probíhající procesy s hlediska dynamiky procesních vláken. Není zahrnut do modelovacího jazyka UML.

Výhodou tohoto modelu je možnost celkového pohledu na důležité souvislosti v systému, možnost sdružování procesů nebo jejich paralelizace.

Dle definice elementární proces reprezentuje aktivitu jako reakci na firemní událost zpravidla v jednom místě a čase.

5.1.1. Zjednodušený konceptuální procesní model



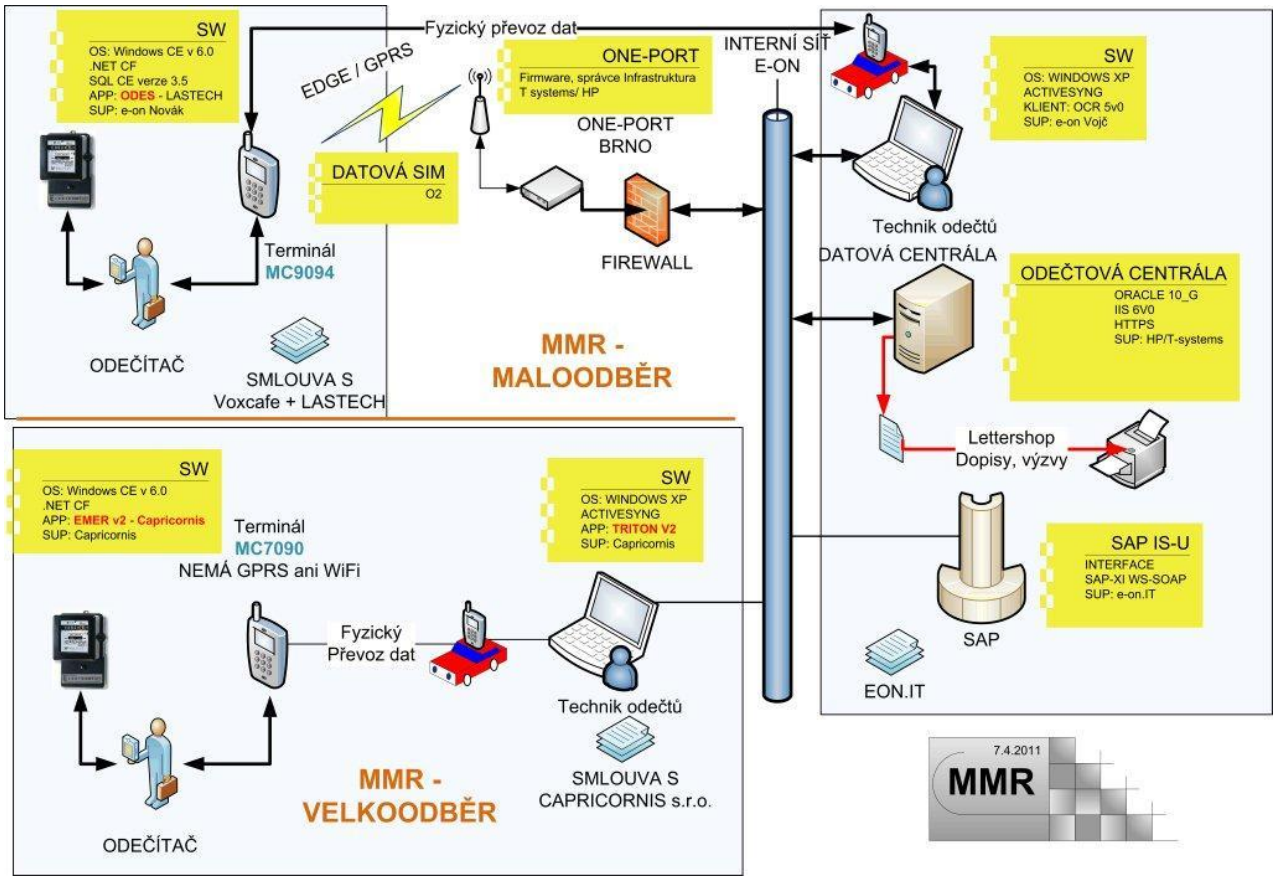
Obrázek 4: Konceptuální procesní model

V diagramu jsou patrné 2 role: technik v centrále, který připravuje a nahrává do terminálu data pro potřeby odečtu, a technik, který pomocí ručního terminálu provádí ruční odečet.

V první fázi dojde k vložení vstupních dat do terminálu. V této fázi je možné provést optimalizaci trasy a do terminálu nahrávat data již v pořadí, které odpovídá optimálnímu průchodu.

V další fázi technik pomocí terminálu obchází jednotlivá odečtová místa a provádí odečty. Aplikace by ho měla navigovat od aktuální pozice vždy k následujícímu odečtovému místu.

5.1.2. Reálný procesní model



Obrázek 5: Reálný procesní model, zdroj Eon-IS

Tento model popisuje reálné fungování dané firmy při provádění odečtů pomocí MMR (manual meter reading).

Liší se odečty prováděné v maloodběru a velkoodběru, kdy velkoodběr je řešen fyzickým převozem dat od odečítacích techniků a je prováděn výrazně pravidelněji než maloodběr.

Tato práce se zabývá technologií používanou pro maloodběr, kdy dochází k přenosu dat z a do terminálu buď fyzicky v centrále technikem odečtů nebo za pomocí GPRS spojení přes datovou síť.

5.2. Analýza aplikace

5.2.1. Základní východiska aplikace

Ze zadání víme, že cílem práce je nalezení vhodného optimalizačního algoritmu a návrh a vytvoření aplikace pro vyřešení jednookruhového okružního dopravního problému. Tak aby tato aplikace splňovala zadaná kritéria:

- Vstup pro více uzlů
- Dynamický přepočet (zredukován na dynamický přepočet trasy k příštímu bodu na trase)
- Rychlost výpočtu, která umožní do 10 s zobrazit výsledek operace
- Kombinace jízdy autem a pěší dopravy
- Aplikace musí být schopna nalézt cestu, i pokud je odečítač mimo silnici (kombinace jízdy autem a ostatní dopravy)
- Možnost trajektorii dynamicky přepočítat během cesty
- Nalezení bodu bez nutnosti znát cílovou GPS pozici
- Všechny výpočty musí probíhat offline, bez nutnosti připojení k internetu
- Pozice v obrazovém podkladu se musí zobrazit s přesností „na dům“
- Určit aktuální pozici z GPS modulu a pamatovat si poslední nalezenou pozici v případě ztráty signálu

Z procesního diagramu dále vyplynula možnost rozdělit aplikaci do dvou menších úkolů. Prvním úkolem je vypočítat vhodné pořadí bodů trasy během jejich nahrávání do terminálové aplikace. V druhé části, v terminálu, pak bude k dispozici seznam bodů a úkolem aplikace bude nalézt vždy jen trasu k následujícímu bodu a tuto zobrazit uživateli.

Tento krok umožní výrazné zrychlení odezvy aplikace v terminálu, protože již nemusí počítat celou trasu obchodního cestujícího, ale „pouze“ naviguje k nejbližšímu bodu.

Z tohoto důvodu byla aplikace rozdělena na dvě části, kdy první část bude umožňovat seřadit požadované body určené k odečtu dle optimální trasy. Rychlost této části bude vždy výrazně závislá na počtu hledaných bodů (jde o exponenciální závislost). Jde tedy o jedno z mála řešení v situaci, kdy požadujeme konstantní čas odezvy, ale zároveň i možnost trasu dynamicky plánovat pro stovky až tisíce bodů.

Druhá část bude instalovaná v terminálu a bude umožňovat navigaci na následující bod dané trasy.

V této kapitole budou popsány vybrané samostatné části řešeného problému, ze kterých bude později sestaven funkční celek jako výsledná aplikace. Jedná se o problémy týkající se realizace datových zdrojů a jejich umístění a zvolení metody výpočtu optimalizace trasy.

5.2.2. Vstupní data potřebná pro výpočet

Pro možnost realizovat tuto aplikaci je potřeba mít k dispozici základní data, která umožní provést optimalizační výpočet, a realizovat a prezentovat tyto informace uživateli.

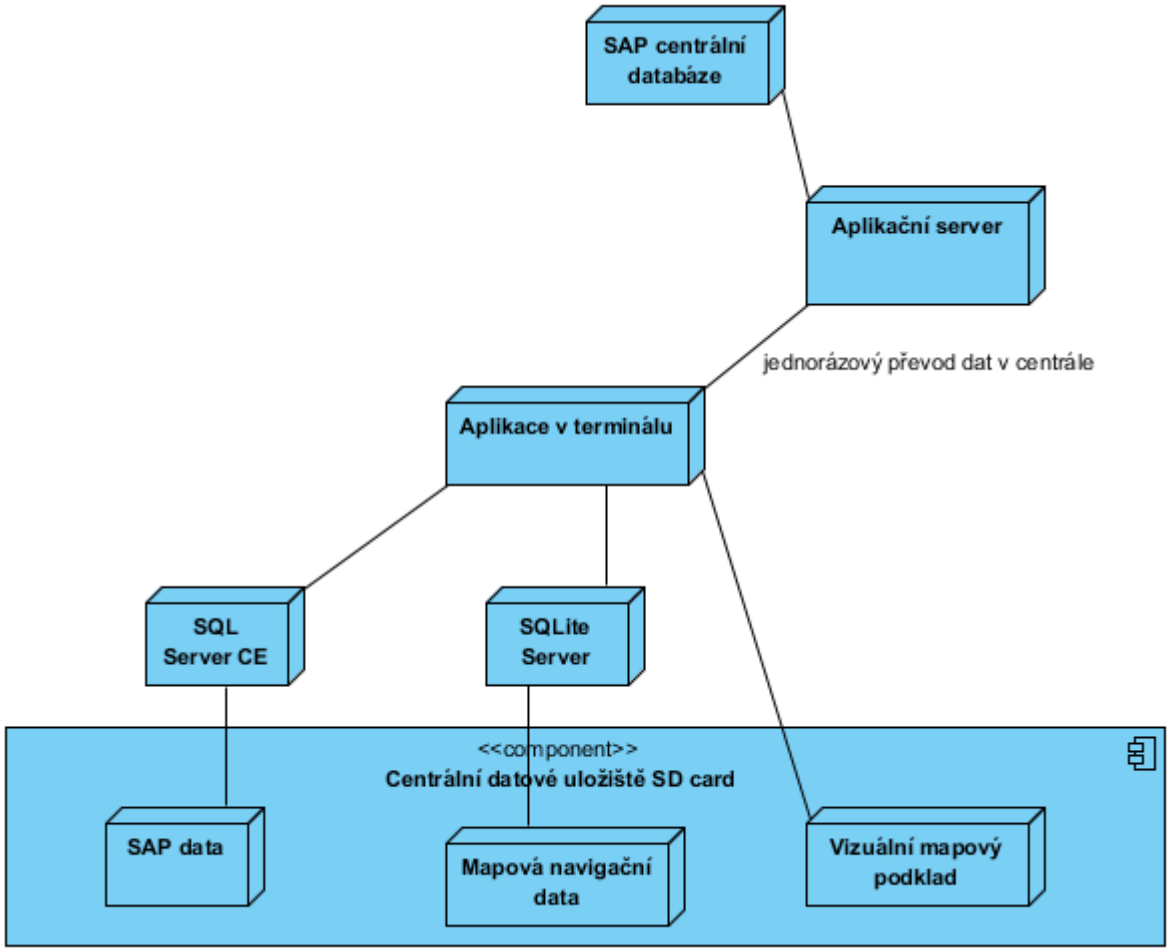
Detailněji popisuje diagram nasazení.

5.2.2.1 Diagram nasazení

Jde o popis nasazení jednotlivých komponent a jejich vzájemné spolupráce. Diagram ukazuje rozmístění zdrojů a softwarové komponenty, které jsou jimi obsluhovány.

Jednotlivé části diagramu jsou rozebrány níže. Diagram také ukazuje potřebné technologie pro obsluhu datových zdrojů. V této části nejde tedy o podrobný rozbor datových zdrojů, ale spíše jejich souvislosti a umístění v cílové struktuře.

Všechna data umístěná v terminálu jsou uložena na SD kartě v předem definované struktuře. Výhodou je snadná manipulace a nezávislost na vnitřní paměti zařízení.



Obrázek 6: Diagram nasazení

5.2.2.1.a SAP data

Tato data představují interní databázi využívající výchozí technologii SQL Server compact edition. Jedná se o souhrn dat získaných z centrální SAP databáze při přípravě terminálu k použití.

Jsou zde obsaženy detailní informace o odečtových místech, včetně přesné adresy, popisu umístění odečtového místa, čísla elektroměru, posledních naměřených hodnot, informace o nepřítomnosti majitele během odečtu atp. Dále se zde nacházejí související navigační data importovaná z externího souboru.

Tato data obsahují i citlivé údaje, takže nejsou obsaženy v přílohách k této aplikaci. Náhradou těchto dat je soubor představující množinu GPS souřadnic určených k seřazení.

V ukázkové aplikaci přiložené k této práci se využívá pouze zjednodušený model těchto dat v podobě GPS souřadnic. Tyto souřadnice jsou ale ve stejném formátu jako v původním datovém modelu. Proto je možné úpravou importovacího mechanizmu napojit k aplikaci i originální data pro reálné využití.

V této modelové aplikaci se využívá soubor typu „inf“, který obsahuje pouze GPS souřadnice daných míst.

5.2.2.1.b Mapová navigační data

Jde o data, která představují model silniční sítě v cílovém místě. Na základě tohoto modelu se bude provádět vlastní navigace mezi jednotlivými body trasy.

Tato data využívají technologii SQLite kvůli obecnějšímu přístupu a kratším časům pro vyhledávání než jaké byly naměřeny s technologií SQL CE. Data jsou uložena v souborech, které obsahují informace odděleně pro jihočeský, resp. jihomoravský kraj.

5.2.2.1.c Vizuální mapový podklad

V tomto případě nejde o jeden centrálně umístěný soubor, ale o množinu obrázků představujících datový podklad pro sestavení požadované mapy.

Blíže je tato datová struktura popsána v dalších bodech. Další možností je využít jeden soubor – archiv, který by v sobě obsahoval komprimované obrázky pohromadě. Výhodou tohoto přístupu by byla rychlejší a snadnější manipulace s obrazovými soubory, nevýhodou by naopak byla nutnost při každém vykreslení mapy tento archiv rozbalit a prohledávat jeho obsah. Na základě testů se tato možnost ukázala jako řádově pomalejší, z těchto důvodů byla zvolena klasická adresářová struktura.

5.2.2.2 Hledané body

Jde o množinu bodů, které by měl odečítač za určený časový úsek obejít. Body jsou ve formátu obecných GPS souřadnic.

Data se získávají z databáze SAP umístěné centrálně a jsou ve formátu `SAP;4912.3038;01635.7469;`

Tento formát identifikuje původ dat, „SAP“ představuje data, která přišla přímo z centrální databáze. Data mohou mít též prefix identifikující odečítače a rok pořízení záznamu.

Další součástí bodu je zeměpisná pozice uvedená ve stupních „49“ a minutých „12,3038“ severní šířky.

Obdobně poslední část je „16“ stupňů a 35,7469 minut východní délky.

Hledaná souřadnice se dá tedy zapsat jako 49°12,3038’N a 16°35,7469’E.

5.2.2.3 GPS mapa

Další nezbytnou součástí je mapa, která obsahuje GPS souřadnice všech silnic a cest v celé hledané lokalitě. Tyto souřadnice musí být možné převést a vztáhnout vůči druhu silnice, ale také vůči jejímu tvaru, aby bylo možné pozdější vykreslení přesné trasy.

Dále je nutné, aby tato mapa byla v podobě vhodné pro realizaci grafových algoritmů. Jinými slovy aby tato GPS mapa hledané oblasti byla převoditelná na graf, který bude obsahovat uzly – křižovatky, kterými plánovaná trasa posléze povede, a hrany – cesty, které tyto křižovatky spojují.

Pak je také třeba mít další potřebné informace pro identifikaci cesty, například její typ, název, údaj, zda jde o jednosměrnou silnici…

Tato data jsou zásadní součástí systému a jejich umístění a způsob implementace bude mít velký vliv na celkovou rychlost výpočtu.

Volně využitelnou variantou, která by splňovala zadané parametry, je server OpenStreetMap.org. Jde o komunitou řízené a sbírané informace o silniční síti související data.

Tento server je financován neziskovou organizací OpenStreetMap Foundation, která byla založena v roce 2004 Steavem Coastem a primárně se zaměřovala na zmapování Velké Británie.

Na serveru OpenStreetMap.org je umístěno velké množství bodů a datových podkladů. K získání odpovídající datové vrstvy, tedy silniční sítě v cílovém regionu (jihočeský kraj), lze využít program Osmosis [16]. Tento program bude získávat data z obdélníka ohraničeného krajními GPS body kraje.

5.2.2.3.a Umístění dat v části seřazení bodů trasy

V první části aplikace, která provede vlastní seřazení dat, je k dispozici celý výpočetní výkon počítače, který bude hledané body seřazovat dle plánované trasy.

Důležitou podmínkou pro uložení dat v tomto případě je tedy jejich kompatibilita s algoritmem, který bude vyhledávat optimální trasy obchodního cestujícího, tedy řadit body dle jejich optimálního průchodu. Dále pak také jednoduchost přenosu dat mezi systémy a získání dat nových v případě jejich aktualizace.

Tato data je potřeba mít v co nejaktuálnější podobě s přípravou pro možnost napojit se přímo na zdrojové webové rozhraní. Proto se využívá i přes časovou neefektivnost zdroj v původním formátu osm.

5.2.2.3.b Umístění dat v terminálu

V terminálu nám jde primárně o velikost dat a rychlosti přístupu k nim. Z těchto dat bude vycházet i vykreslení optimální trasy na mapovém podkladu ve vlastní aplikaci. Dalším požadavkem na tato data je tedy možnost jednoduše na základě GPS souřadnice najít odpovídající uzel v grafu.

V těchto datech nepotřebujeme všechny detailní informace dostupné v původním souboru, jako je např. název uživatele, který data pořídil, atp.

Data je potřeba mít v přenositelné podobě, která bude snadno použitelná v terminálové aplikaci.

Na základě těchto požadavků byl zvolen databázový systém SQLite, který umožňuje data uložit do lokální databáze bez nutnosti mít nainstalovaný databázový server. Pro vlastní

převod dat do tohoto systému byl využitý systém OSMUpload [13], převádějící data z původní XML podoby do SQLite.

Po převodu těchto dat jsme získali strukturu, která v sobě již obsahuje předpřipravený graf pro vyhledávání na základě teorie grafů. Dále pak tento systém využívá technologii R-Strom (rTree). Jde o strukturu podobno B-stromům, která umožňuje rychlejší vyhledávání v indexovaných položkách, především v případě, kdy nehledáme pouze prvek s konkrétním indexem, ale hledáme množinu prvků, která patří do určitého intervalu. (Bližší popis zde: https://www.sqlite.org/rtree.html). Tato technologie bude dále využita při hledání GPS souřadnic, které jsou blízké aktuální pozici a náleží některé trase. Vyhledávací dotaz bude ještě optimalizován omezením počtu vstupních bodů na oblast, ve které má smysl reálně vyhledávat. (Nemá smysl například hledat bod trasy kilometry vzdálený od aktuální pozice).

5.2.2.4 Mapová data

Další nezbytnou součástí aplikace je způsob prezentace dat uživateli. Tato data se mohou prezentovat buď vektorovou dynamicky vykreslenou sítí silnic na základě dat z předchozího bodu, nebo pomocí bitmapových obrázků, které představují šachovnici na sebe navazujících obrázků, nad kterými se provádí další dodatečné vykreslení hledané trasy.

Z důvodu požadavku, který kombinuje hledanou trasu jízdy autem s ostatní dopravou, kdy hledaná místa k odečtení se často nacházejí mimo silnici popsanou v předchozích datech, byl zvolen druhý model.

Aplikace tedy obsahuje množinu na sebe navazujících bitmapových obrázků, které jsou uložené v adresářové struktuře popsané pomocí kartézských souřadnic pozic jednotlivých obrázků.

V odpovídajícím zvětšení vznikne pro cílové území (Jihočeský a Jihomoravský kraj) přibližně 14 000 souborů. Tato data je potřeba organizovat do podsložek, protože práce s takovým množstvím souborů v jediné složce je velmi pomalá. A dále bylo experimentálně

ověřeno, že SD karta, na kterou se data ukládají, formátovaná systémem FAT32, vykazuje problémy, pokud je v jedné složce více jak 5000 souborů.

Pro první řádek jsou data uložena v adresáři Map001_Y, který pak dále obsahuje množinu souborů jednotlivých obrázků v prvním X řádku. Soubory mají názvy odpovídající jejich pozici v Y sloupci např. Y001.

Každá složka dále obsahuje navigační soubor, který těmto obrázkům doplňuje odpovídající GPS souřadnice.

```
<?xml version="1.0" encoding="utf-8"?>
<maps>
  <mapNode map="Map0001_0001">
    <GpsXPomer value="0.00002800000000000" />
    <GpsYPomer value="-0.00002799999999999" />
    <GpsX value="13.50648017646371500" />
    <GpsY value="49.74759842367811300" />
  </mapNode>
```

Obrázek 7: Ukázka datové definice obrazových mapových dat

Soubor obsahuje složený název obrázku (na základě jeho pozice v souřadnicové síti) a souřadnici jeho levého horního rohu.

Dále pak obsahuje přepočtový koeficient, který říká, jakou hodnotu představuje v daném obrázku 1 pixel. Toto číslo je v rámci různých obrázků přibližně stejné, ale vzhledem k požadované velké přesnosti přepočtu obrazových dat na data navigační a naopak, je ještě dopočteno pro jednotlivé obrázky s přesností na 17 desetinných míst v datovém typu decimal.

Následuje ukázka přepočtu aktuální pozice ve stupních (pole *gpsPosXY*), souřadnice levého horního rohu cílového obrázku (pole *data.gpsPosImg1*) a převodního poměru *gpsYPomer*. Dále je proměnná *actualSize*, která přestavuje koeficient aktuálního zvětšení mapového podkladu.

Pro zrychlení výpočtu se používá metoda *calculatePosition(double[] gpsPosXY)*, která provede odhad čísla obrázku obsahující hledaný GPS bod.

```
int xCislo = Convert.ToInt32((gpsPosXY[0] - data.gpsPosImg1[0]) /
(gpsXPomer * actualSize.Width));
int yCislo = Convert.ToInt32((gpsPosXY[1] - data.gpsPosImg1[1]) /
(gpsYPomer * actualSize.Height));
```

5.2.2.5 Převodová data adresy na GPS souřadnice

Z důvodu požadavku „Nalezení bodu bez nutnosti znát cílovou GPS pozici“ je potřeba ještě jeden dodatečný datový podklad obsahující převod mezi názvem města a jeho GPS souřadnicemi.

```
<node id="25498965" name="OLOMOUC" lat="49.5954189" lon="17.2473483" uid="338451" version="13" xmlns="" />
<node id="26037142" name="TRENCIN" lat="48.8923583" lon="18.0393715" uid="29717" version="7" xmlns="" />
<node id="32665890" name="ROKYCANY" lat="49.7421482" lon="13.596083" uid="127170" version="5" xmlns="" />
<node id="86952050" name="HRADEK" lat="49.7114876" lon="13.6544120" uid="308" version="3" xmlns="" />
<node id="86976293" name="MIROSOV" lat="49.6881853" lon="13.6567822" uid="17615" version="4" xmlns="" />
<node id="86975143" name="SPALENE PORICI" lat="49.6122703" lon="13.6065309" uid="308" version="3" xmlns="" />
```

Obrázek 8: Ukázka definice dat pro překlad názvu na GPS

Tato data (viz ukázka výše) byla získána ze serveru OpenStreetMaps.org a převedena do xml podoby, která obsahuje pouze informace o upraveném názvu města a jeho pozici.

Všechny názvy měst byly převedeny do podoby psané velkými písmeny a bez diakritiky. Stalo se tak z důvodu minimalizace možnosti překlepů a neshod v názvech. V případě nalezení více výsledků systém umožní mezi těmito přepínat.

5.2.3. Výpočetní část

Když máme k dispozici všechna potřebná data, můžeme se zaměřit na analýzu provádění vlastního výpočtu a prezentace výsledků uživateli.

Výpočetní část lze rozdělit na dva podúkoly. (Blíže viz 5.2.1)

5.2.3.1 Optimalizace bodů trasy

Prvním úkolem je seřazení vstupních bodů popsaných v částí 5.2.2.2 tak, aby bylo možné na jejich základě v terminálu provádět navigační výpočet k nejbližšímu následnému bodu trasy.

Jedná se o řešení problému obchodního cestujícího. Je možné využít algoritmy uvedené v teoretickém úvodu.

Na základě hledaných parametrů byla zvolena metoda využití genetického algoritmu s využitím nástroje OsmSharp [17]. Tento nástroj implementuje základní i pokročilé algoritmy pro hledání optimální trasy, případně pro složitější navigační problémy jako je problém obchodního cestujícího.

Ve výchozí podobě, z důvodu obecnosti, má pouze jednu formu výstupu a standartní vstup.

Pro využití v této aplikaci bude tedy nutné přepsat vstupní mechanizmus, aby uměl pracovat s daty popsané v bodě 5.2.2.2, a dále přepsat výstupní mechanizmus, který standartní navigační výstup převede do podoby, kdy vrátí zadané souřadnice dle 5.2.2.2 v již seřazené podobě odpovídající optimální trase.

5.2.3.1.a Popis výpočtu

```
// do the data processing.
MemoryRouterDataSource<PreProcessedEdge> osmData =
    new MemoryRouterDataSource<PreProcessedEdge>(tags_index);
PreProcessedDataGraphProcessingTarget targetData =
  new PreProcessedDataGraphProcessingTarget(osmData, interpreter, osmData.TagsIndex);
XmlDataProcessorSource data_processor_source = new XmlDataProcessorSource(data_stream);
DataProcessorFilterSort sorter = new DataProcessorFilterSort();
sorter.RegisterSource(data_processor_source);
targetData.RegisterSource(sorter);
```

Obrázek 9: Ukázka získání zdrojových dat a preprocesoru

K výpočtu pomocí nástroje OsmSharp je nejprve nutné napojit datové zdroje do výpočetní struktury. Je možné využít přímý vstup XML dat získaný jako *osm* soubor z portálu OpenStreetMaps.org. Dále je možné využít přímý vstup z tohoto webového systému.

Lze též využít již analyzovanou a upravenou strukturu z bodu 5.2.2.3.b.

V této ukázce využíváme nejjednodušší variantu s přímým vstupem *osm* dat, která je výhodná díky jednoduchosti změny datového zdroje. Pokud chceme nahradit zdrojová

mapová data, stačí tyto exportovat z OpenStreetMaps.org a napojit do systému. Nevýhodou tohoto postupu je nutnost předzpracování těchto dat před vlastním výpočtem. Zdrojové XML je hodně obsáhlé, takže aby byl výpočet vůbec možný, využívá se systém preprocesorů, které data nejprve předzpracují do operační paměti, kde vyrobí grafovou strukturu. Vlastní výpočet probíhá až nad těmito daty.

```
IRouter<RouterPoint> router = new Router<PreProcessedEdge>(osmData, interpreter,
    new DykstraRoutingPreProcessed(osmData.TagsIndex));
```

Obrázek 10: Ukázka vytvoření instance třídy Router

Dalším krokem je vytvoření instance třídy *Router*, která bude provádět vlastní propočet optimální trasy. Jako jeden z parametrů dostává způsob, kterým se propočítávají vzdálenosti mezi jednotlivými uzlovými body. Využívá se Dijkstrův algoritmus popsaný dříve. Tato část zabírá nejvíce času při řešení problému obchodního cestujícího, protože je nutné vypočítat vzájemnou vzdálenost mezi všemi hledanými body. Jde tedy o exponenciální časovou složitost.

Tento algoritmus se v upravené podobě využije i pro navigaci k jednotlivým bodům odečtu v terminálové aplikaci.

```
GeoCoordinate point = new GeoCoordinate(latitude, longitude);

RouterPoint resolved = router.Resolve(VehicleEnum.Car, point);
if (resolved != null && router.CheckConnectivity(VehicleEnum.Car, resolved, 100))
{
  points.Add(resolved);
}
```

Obrázek 11: Ukázka vložení hledaných GPS pozic

Nyní je nutné naplnit objekt vytvořený v předchozím bodě koordinačními daty, která budeme vyhledávat. Nejprve se provede parsování zdrojového souboru. Tato operace je závislá na přesném formátu vstupních dat, proto je napsaná konkrétně na vstupní data popsaná v bodě 5.2.2.1.a. Jde o rozebrání koordinátů z formátu SAP do podoby zeměpisné šířky a délky vyjádřené desetinným číslem ve stupních.

```
var tsp_solver = new RouterTSPAEXGenetic<RouterPoint>(router);
OsmSharpRoute tsp = tsp_solver.CalculateTSP(VehicleEnum.Car, points.ToArray(), true);
```

Obrázek 12: Ukázka spuštění výpočtu pro nalezení optimální trasy obchodního cestujícího

Dále vybereme typ routovacího algoritmu. Pro tento případ byl vybrán genetický algoritmus optimalizace trasy obchodního cestujícího. Princip genetických algoritmů je popsán v úvodu práce.

Vstupem pro tento algoritmus je objekt „router“, který v sobě již obsahuje informace o mapové síti v podobě navigačních dat a vybrané kombinace preprocesorů. Dále pak kolekce již předzpracovaných bodů, které se budou vyhledávat, a informace o způsobu vyhledávání trasy. V našem případě hledáme trasu pro automobilovou dopravu.

Výsledkem je pak množina bodů, které představují uzly výsledné trasy, která byla vyhodnocena jako optimální.

Výstupem je tedy XML soubor obsahující informace o trase. Tento soubor je ještě dodatečně zpracován v postprocesingu, kdy z informace o konkrétní nalezené trase získáme kolekci bodů, která odpovídá vstupní datům z hlediska formátu i datového obsahu. Tuto kolekci následně uložíme se zachováním vypočteného pořadí, kterým vede optimální trasa.

5.2.3.2 Navigace v terminálu

Terminálová aplikace dostane seřazenou sekvenci vstupních bodů, které je nutné postupně obejít. Aplikace tedy musí umět navigovat od aktuální pozice, na které se nachází odečítač k cílové pozici hledaného odečítaného místa.

Tato úloha je již řádové jednodušší než hledání celé trasy obchodního cestujícího.

K výpočtu lze použít standartní optimalizované způsoby hledání trasy mezi dvěma body. Nebo využít OsmSharp. Postup výpočtu pomocí nástroje OsmSharp je popsán v předchozím bodu, zde popíšeme variantu přímého výpočtu trasy mezi dvěma body na základě principů z dijkstrova algoritmu. Tato varianta využívá již optimalizovanou strukturu vstupních dat v databázi SQLite viz 5.2.2.3.b.

5.2.3.2.a Popis výpočtu

K provedení vlastního výpočtu je potřeba nejprve získat obě souřadnice, mezi kterými se má vypočítat optimální trasa. Počáteční souřadnicí je aktuální poloha odečítače, cílovou souřadnicí je pozice následujícího odečítaného místa. Pro oba tyto body je potřeba nalézt nejbližší odpovídající bod na silniční síti a nejbližší křižovatku tak, aby se mohl spustit algoritmus pro vyhledávání optimální trasy.

```
            var intersectionId =
                m_Database.LoadData(@"SELECT Intersection.Id
From Intersection
INNER JOIN Nodes on Nodes.Id = Intersection.NodeId
WHERE minX>= @param0 -" + LimiToSearchNode + @"
    AND maxX<= @param0 + " + LimiToSearchNode + @"
    AND minY>= @param1 -" + LimiToSearchNode + @"
    AND maxY<= @param1 + " + LimiToSearchNode + @"
ORDER BY ABS(@param0 - MinX),
    ABS(@param1 - MinY) " + (asc ? "ASC" : "DESC") + @"
LIMIT 1",
            coords[0],
            coords[1]);
```

Obrázek 13: Ukázka nalezení GPS na trase blízké zadané pozici

Tato ukázka využívá technologii RTree (viz 5.2.2.3.b). Na základě omezující podmínky v proměnné „LimitToSearchNode“, která definuje maximální vzdálenost, ke které se bude provádět vyhledávání, se omezí množina bodů určených k prohledávání jen na blízké okolí. Důvodem je časová optimalizace celé operace.

Dalším vstupem pro vyhledávání je pole GPS koordinátů s názvem „coords“, které obsahuje souřadnice hledaného místa. Díky tomuto omezení a faktu, že dotaz využívá strukturu RTree (což je struktura podobná binárním stromům), můžeme provést vlastní vyhledávání bodů, které splňují specifickou podmínku, jež stanoví, že body mají mít co nejmenší odchylku od hledaných koordinátů.

Velikost absolutní odchylky od hledaného bodu se určí pomocí seřazení bodů v části „ORDER BY“, kde jsou body řazené pomocí rozdílu absolutních hodnot x a y koordinátů.

Díky využití limitu počtu záznamů „LIMIT 1“ dojde k vrácení pouze jednoho záznamu, který nejlépe splňuje danou podmínku. Tedy jde o bod v grafu, který se nejvíce blíží hledanému místu.

```
var route = new Dijkstra(source, report, start);

NodeScore selScore;
var selNode = route.SelectNode(out selScore);
while (selNode != Int64.MaxValue)
{
    if (selNode == end)
    {
        return route.GetPath(end, false);
    }

    route.ProcessNode(selNode, selScore);
    selNode = route.SelectNode(out selScore);
}
```

Obrázek 14: Ukázka spuštění Dijkstrova algoritmu pro nalezení optimální trasy

Pokud jsme nalezli body z naší grafové struktury, které se blíží hledaným bodům, můžeme provést vlastní výpočet. Výpočet využívá Dijkstrův algoritmus s možností prohledávání ze dvou stran, které umožní paralelizovat výpočetní část na dva výpočty, jenž běží nezávisle a snaží se potkat ve společném bodě.

Výsledkem pak bude trasa, která vychází z prvního bodu a končí v bodě cílovém.

Dijkstrův algoritmus využívá principu prioritní fronty, v níž si uchovává všechny uzly řazené dle vzdálenosti od zdroje. Na začátku výpočtu má pouze zdroj vzdálenost 0, ostatní uzly mají vzdálenost nekonečno.

V každém dalším kroku algoritmus vybere uzel s největší prioritou, tedy nejkratší vzdáleností od již zpracované části a následně jej zařadí mezi zpracované uzly. (Blíže viz následující bod.) Následně algoritmus projde všechny ještě nezpracované potomky a přidá je do fronty, pokud tam ještě nejsou obsaženy, a ověří, zda nejsou blíže ke zdroji, než byly před zařazením právě zpracovávaného uzlu.

Pro všechny potomky ověřuje:

Vzdálenost (zpracovávaný) + délkaHrany(zpracovávaný, potomek) < vzdálenost(potomek)[14]

Jestliže nerovnost platí, danému potomkovi se nastaví nová vzdálenost a jeho předek se označí za zpracovávaný uzel, poté algoritmus vybere z fronty uzel s nejvyšší prioritou a celý krok opakuje až do okamžiku, kdy jsou zpracovány všechny uzly.

Nabízí se možné optimalizace v omezení počtu bodů určených ke zpracovávání, například omezením prohledávané oblasti jen na určitý okruh od hledaného místa. Tento okruh lze také omezit díky faktu, že odečítači se pohybují pravidelně po přibližně stejných trasách. Další možná optimalizace se týká rozhodovacího algoritmu, který ukončí výpočet v okamžiku, kdy už byla nalezena trasa, která splňuje zadané parametry, takže již není nutné prohledávat zbylou množinu uzlů. Těmito optimalizacemi se budeme blíže zabývat v závěru následující kapitoly.

5.2.3.2.b Výpočet vzdálenosti

K výpočtu vzdáleností mezi body a určení jejich výhodnosti z hlediska plánované trasy je potřeba mít možnost určit jejich vzdálenost. Tu lze buď vypočítat přesně na základě dat ze vstupního grafu nebo odhadnout pomocí trigonometrického výpočtu.

```
/// <summary>
/// Returns the distance in kilometers of any two latitude / longitude points.
/// </summary>
static public double Calc(double pos1X, double pos1Y, double pos2X, double pos2Y)
{
    const double R = 6371;
    var dLat = ToRadian(pos2Y - pos1Y);
    var dLon = ToRadian(pos2X - pos1X);
    var a = Math.Sin(dLat / 2) * Math.Sin(dLat / 2) +
            Math.Cos(ToRadian(pos1Y)) * Math.Cos(ToRadian(pos2Y)) *
            Math.Sin(dLon / 2) * Math.Sin(dLon / 2);
    var c = 2 * Math.Asin(Math.Min(1, Math.Sqrt(a)));
    var d = R * c;
    return d;
}
```

Obrázek 15: Ukázka výpočtu vzdálenosti mezi GPS body

Výše je popsán algoritmus trigonometrického výpočtu, který pomocí triangulace vypočítá vzdálenost v elipsoidu mezi dvěma body zadanými pomocí GPS souřadnic.

5.2.3.2.c Vykreslení nalezené trasy

S využitím dat z části 5.2.2.4 je nutné nalezenou optimální trasu vykreslit na připravený mapový podklad. Není možné kreslit jen body odpovídající jednotlivým uzlům, tedy v tomto pohledu křižovatkám, ale je potřeba vykreslit celou trasu od začátku až do konce včetně všech nepravidelností na silniční síti, jako jsou například zatáčky.

```
foreach (DataRow way in from
            DataRow way in wayId from
                DataRow way2 in wayId2
            where (long)way["WayId"] == (long)way2["WayId"]
          select way)
{
    finalWaId = (long)way["WayId"];
}

rows =
    m_Database.LoadData(@"
SELECT DISTINCT Nodes.MinX, Nodes.MinY, Nodes.Id as WayId,
    @param0 as IntersectionId
FROM WayNode
INNER JOIN Nodes on Nodes.Id = WayNode.NodeId
WHERE WayNode.WayId = @param0
", finalWaId).Rows;
                }
```

Obrázek 16: Ukázka získání bodů pro vykreslení nalezené trasy

Tato ukázka získá pro každou trasu, která je součástí výsledné plánované trasy, množinu GPS souřadnic, které tuto trasu určují a popisují. Jedná se o body pravidelně rozmístěné po celé silnici, které popisují reálný obraz dané trasy včetně nepravidelností.

Tyto body lze následně aproximovat křivkou, kterou lze již vykreslit na právě zobrazovaný mapový podklad.

Vykreslení bude probíhat dynamicky na základě právě zobrazovaného mapového podkladu. Z důvodu optimalizace by bylo značně neefektivní vykreslovat trasu jinde než na mapový podklad, který je právě zobrazován uživateli.

5.2.4. Integrace aplikace

Všechny výše uvedené body a postupy spolu musí komunikovat, aby vytvořily funkční celek. Detailně je návrh aplikace popsán v následující kapitole.

Aplikace, která bude součástí této práce, je modelová, neobsahuje z důvodů ochrany osobních údajů plné vstupní informace o odečtových místech a klientech. Z hlediska vnitřní komunikace ale obsahuje funkční celek.

V první fázi je aplikace pro načtení vstupních dat ve formátu SAP a jejich seřazení pomocí optimalizačního algoritmu do sekvence optimálního průchodu dané trasy.

Tyto seřazené body jsou vstupem do terminálové aplikace, která umožňuje navigovat vždy k následujícímu bodu. K tomuto využívá plná mapová a navigační data, takže může fungovat zcela bez závislosti na internetu.

V reálné aplikaci je tato část součástí větší terminálové aplikace, která mechanizmus importu dat spojuje do většího celku. Z tohoto důvodu se způsob spouštění a importu dat do reálné aplikace od modelu liší. Vlastní zobrazovací a optimalizační nástroje ale zůstávají stejné.

6. Návrh aplikace

6.1. Úvod

Tato kapitola popisuje vlastní zpracování, návrh a realizaci softwaru určeného pro optimalizaci a prezentaci navigačních dat.

Vychází z informací a postupů analyzovaných v předchozích kapitolách, kdy využívá již detailně rozpracované datové zdroje stejně tak, jako jednotlivě již hotové způsoby výpočtu uvedené v části 5.2.3. Následně z nich vytváří programově funkční celek, který má splnit zadaná kritéria.

6.2. Postup vývoje

Jako první po provedení obecné analýzy požadavků a technologické analýzy podkladů aplikace budou zpracovány UML diagramy, které popisují aplikaci v již co nejpodrobnější rovině.

Na základě těchto analýz budou zvoleny technologie a postupy vhodné pro vývoj. Dále je zaveden multiverzovací systém SVN pro správu a verzování zdrojových kódů.

Na závěr bude vytvořen instalační script pro snadnou instalaci aplikace a provede se testování.

6.3. UML diagramy

K návrhu aplikace jsou využity UML diagramy.

Jde o diagramy struktury: diagram tříd (class diagram), diagram komponent (component diagram), diagram nasazení (deployment diagram) uvedený již v předchozí kapitole.

A dále se jedná o diagramy modelující chování a interakce: diagram způsobů užití (use case diagram), sekvenční diagram (sequence diagram), diagram aktivit (acitivity diagram), diagram komunikace, stavový diagram.

6.4. Use Case diagram

V této kapitole budou jednotlivě specifikovány případy použití (use case) a jejich scénáře na základě požadavků a cílů aplikace.

Use Case slouží pro modelování typické interakce uživatelů se systémy, aby bylo možné lépe pochopit jejich konkrétní požadavky.

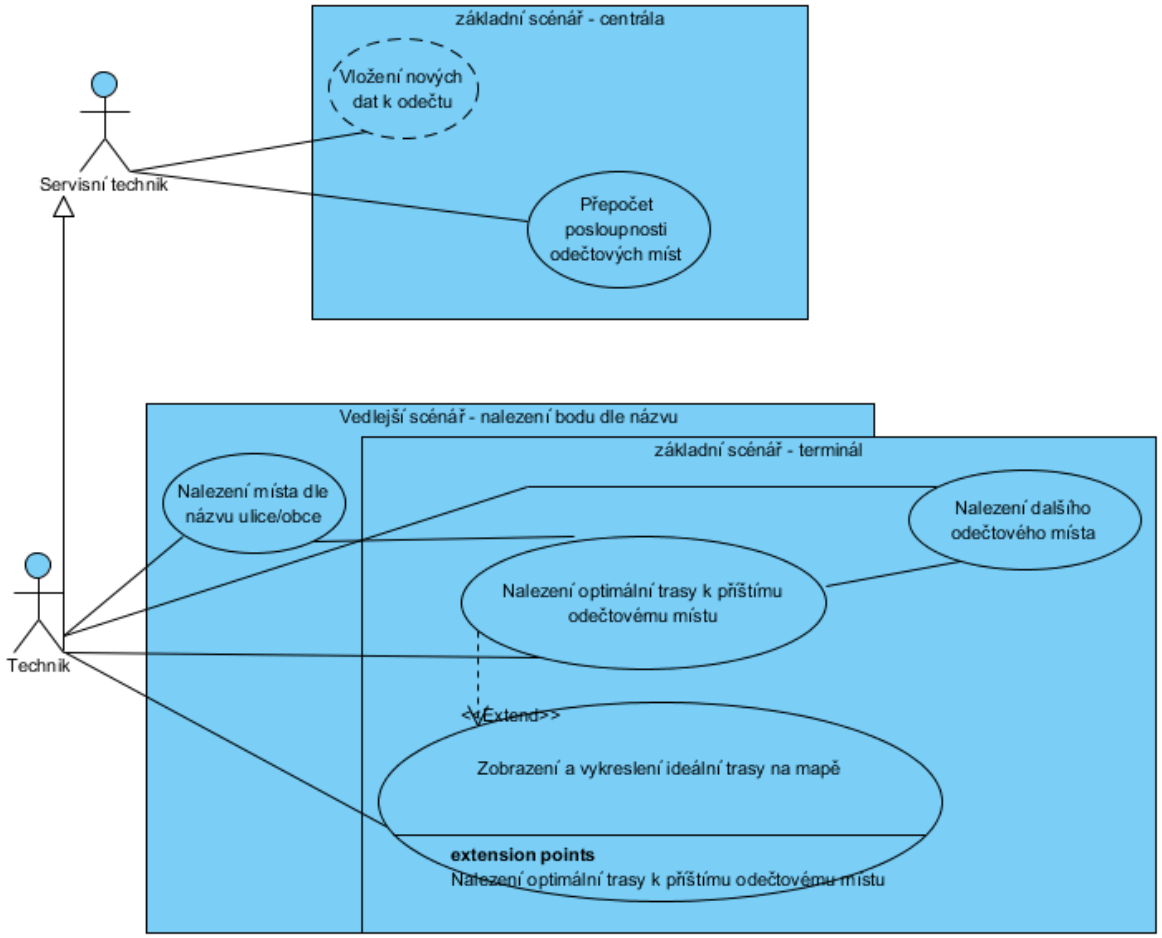
6.4.1. Use Case základního scénáře

Zde je popsán základní scénář způsobu užití aplikace. Scénář vychází s procesního diagramu, který popisuje způsob, jakým se provádí odečty.

Vstupují do něj dva druhy aktérů.

Prvním je Technik, který představuje „odečítače“ pracovníka firmy provádějící odečty a který se fyzicky pohybuje v terénu s mobilním terminálem s nainstalovanou mobilní aplikací.

Druhým je Servisní technik, zaměstnanec v centrále nebo servisním středisku dané firmy, který má přístup do společné databáze dat a provádí přípravu terminálu a nahrání dat na určité období odečtu.



Obrázek 17: Use case diagram

6.4.1.1 UC1: Vložení nových dat k odečtu

6.4.1.1.a Popis Use-Case

Popisuje způsob, jakým se získají data o odečtových místech, která nebyla odečtena automatickými technologiemi. Dále popisuje způsob vložení dat do terminálu.

6.4.1.1.b Uživatelé

Role	Popis role/uživatele
Servisní technik	Uživatel v servisním středisku s oprávněními ke vkládání nových dat

6.4.1.1.c Předpoklady

Je k dispozici soubor obsahující SAP souřadnice pro vložení do systému

6.4.1.1.d Hlavní tok

Krok	Popis kroku	Alternativní tok
1	Uživatel se připojí k terminálu z PC	
2	Načte soubor do systému	
3	Data se převedou do inf souboru pro vložení do terminálu	A-01
4	Program provede kontrolu dat	
5	Vypíše logovací informace a ukončí se	

6.4.1.1.e Alternativní tok A – plánování trasy

Krok	Popis kroku	Alternativní tok
1	Kontrola, zda data obsahují GPS souřadnice odečtových míst	
2	Vyvolání UC – přepočet posloupnosti odečtových míst	
3	Uložení posloupnosti pro potřeby navigace	
4	Zobrazí se informace o úspěšném přepočtu a uložení dat	

6.4.1.2 UC2: Přepočet posloupnosti odečtových míst

6.4.1.2.a Popis Use-Case

Po načtení dat z hlavního systému dojde k přepočtu optimální trasy. Optimální trasa je spočítána vlastním offline algoritmem, který bude možné v případě potřeby přenést i do mobilního zařízení.

Vygenerovaná posloupnost odečtových míst bude sloužit jako zdroj navigačních dat pro trasu, po které bude aplikace navigovat Technika.

6.4.1.2.b Uživatelé

Role	Popis role/uživatele
Servisní technik	Uživatel v servisním středisku s oprávněními ke vkládání nových dat

6.4.1.2.c Předpoklady

Do systému byla vložena data o odečtových místech
Data odečtových míst mají GPS souřadnice

6.4.1.2.d Hlavní tok A

Krok	Popis kroku	Alternativní tok
1	Uživatel se připojí k pre datovému souboru z PC	
2	Načte soubor do systému	
3	Data se převedou do inf souboru pro export do terminálu	
4	Program provede kontrolu správnosti dat a získá GPS informace o místech	B1
5	Dojde k propočtu optimální trasy	
6	Vypíší se informace o nalezené trase	
7	Trasa se zobrazí ke kontrole	
8	Program trasu uloží a umožní ji nahrát do terminálu	
9	Vypíše se log o úspěšnosti operace	

6.4.1.2.e Alternativní tok B – chybí GPS body

Krok	Popis kroku	Alternativní tok
1	Naleznou se odečtová místa bez GPS souřadnic	
2	Tato místa se vyřadí z optimalizačního algoritmu	
3	Vypíše se informace o místech bez GPS	
4	Pokračuje krokem A5	

6.4.1.2.f Datové položky

Tabulka 3: Datová položka typu plánované body trasy

Položka	Typ	Povinné	Význam	Příklad
Název	**string**	X	Název bodu trasy z interní databáze	T14
Lat	**decimal**	X	Zeměpisná šířka ve formátu stupněMinuty	4911.3245 (= 49° 11.3245’)
Lon	**decimal**	X	Zeměpisná délka ve formátu stupněMinuty	1424.123 (= 14° 24.123’)
Order	**int**		Pořadí v trase	(možno vyjádřit pořadím dat ve zdrojovém souboru)

6.4.1.3 UC3: Nalezení místa dle názvu obce

6.4.1.3.a Popis Use-Case

Aplikace v terminálu umožní Technikovi nalézt cílové místo na základě přímého zadání názvu obce. Tato technologie se použije i při lokalizaci odečtových míst, které nemají GPS pozici.

Systém bude obsahovat databázi měst a jejich GPS pozic.

6.4.1.3.b Uživatelé

Role	Popis role/uživatele
Technik	Uživatel, který obsluhuje terminál a provádí vlastní odečty

6.4.1.3.c Předpoklady

Systém obsahuje navigační data pro překlad názvu místa do GPS souřadnic

6.4.1.3.d Hlavní tok A

Krok	Popis kroku	Alternativní tok
1	Technik se připojí k terminálu	
2	Načte aktuální odečtové místo dle jeho adresy	B1
3	Místo se zobrazí na mapě	
4	Technik zobrazí formulář pro přímé hledání obce	
5	Dojde k zadání textového výrazu názvu obce	
6	Pro tento výraz se nalezneme GPS pozici	A9
7	Mapa se vycentruje na hledané místo	
8	Systém umožní přepínat v případě, že bylo nalezeno více cílových míst	
9	Pokud již nebylo nalezeno další místo, aplikace vypíše chybovou hlášku	
10	Mapa se vycentruje na aktuální pozici	

6.4.1.3.e Alternativní tok B – přímý vstup do mapové aplikace

Krok	Popis kroku	Alternativní tok
1	Technik otevře aplikaci pro zobrazování dat bez vybraného odečtového místa	
2	Mapa se vycentruje na aktuální pozici	
3	Pokračuje se přímým hledáním	A4

6.4.1.4 UC4: Nalezení dalšího odečtového místa

6.4.1.4.a Popis Use-Case

Aplikace v terminálu umožní Technikovi nalézt cílové místo na základě jeho GPS souřadnic a pořadí v tabulce trasy. Bude možné se přepínat mezi jednotlivými body trasy.

6.4.1.4.b Uživatelé

Role	Popis role/uživatele
Technik	Uživatel, který obsluhuje terminál a provádí vlastní odečty

6.4.1.4.c Předpoklady

Systém obsahuje naplánovaná data trasy odečtu dle definice v tabulce Trasy.

6.4.1.4.d Hlavní tok A

Krok	Popis kroku	Alternativní tok
1	Technik se připojí k terminálu	
2	Zobrazí se možná cílová místa určená k hledání dle pořadí	
3	Vybere se hledané místo dle pořadí s možností jeho změny	
3	Najde se vybrané odečtové místo	
4	Místo se zobrazí	
5	Dynamicky se přepočítá trasa od aktuální pozice k odečtovému místu	UC5
6	Trasa se zobrazí	UC6
7	Technik dorazí k cílovému místu	
8	Technik mapovou aplikace ukončí, aby bylo možné zadat odečet	

6.4.1.5 UC5: Nalezení optimální trasy k odečtovému místu

6.4.1.5.a Popis Use-Case

Aplikace v terminálu umožní Technikovi nalézt cílové místo, k němuž bude možné na základě aktuální pozice vypočítat nejkratší trasu. Tato trasa se bude dynamicky na vyžádání přepočítávat.

6.4.1.5.b Uživatelé

Role	Popis role/uživatele
Technik	Uživatel, který obsluhuje terminál a provádí vlastní odečty

6.4.1.5.c Předpoklady

Jsou k dispozici navigační data pro hledané místo
Systém obsahuje navigační data pro silniční síť v dané oblasti

6.4.1.5.d Hlavní tok A

Krok	Popis kroku	Alternativní tok
1	Technik se připojí k terminálu	
2	Nalezne aktuální odečtové místo	B1, UC4
3	Místo se zobrazí na mapě	
4	Získá se aktuální pozice, na které se terminál nachází	
5	Mapa se vycentruje na aktuální pozici	

6	Dojde k výpočtu optimální trasy k hledanému místu dle parametrů	
7	Trasa se zobrazí	UC6
8	Mapa zůstane vycentrována na aktuální pozici	
9	Při změně aktuální pozice bude umožněn provést přepočet	A4

6.4.1.5.e Alternativní tok B – přímý vstup do mapové aplikace

Krok	Popis kroku	Alternativní tok
1	Technik zadá nové cílové místo	
2	Mapa se vycentruje na aktuální pozici	
3	Pokračuje přímým hledáním	A3

6.4.1.6 UC6: Zobrazení ideální trasy na mapě

6.4.1.6.a Popis Use-Case

Aplikace bude obsahovat mapové podklady, na kterých se zobrazí navigační data. Stejným způsobem se bude zobrazovat i nalezená trasa.

Mapové podklady a všechny další potřebné informace musí být k dispozici offline.

6.4.1.6.b Uživatelé

Role	Popis role/uživatele
Technik	Uživatel, který obsluhuje terminál a provádí vlastní odečty

6.4.1.6.c Předpoklady

Jsou k dispozici navigační data pro hledané místo
Systém nalezl optimální trasu k cílové oblasti
Systém obsahuje mapové podklady pro cílovou oblast

6.4.1.6.d Hlavní tok A

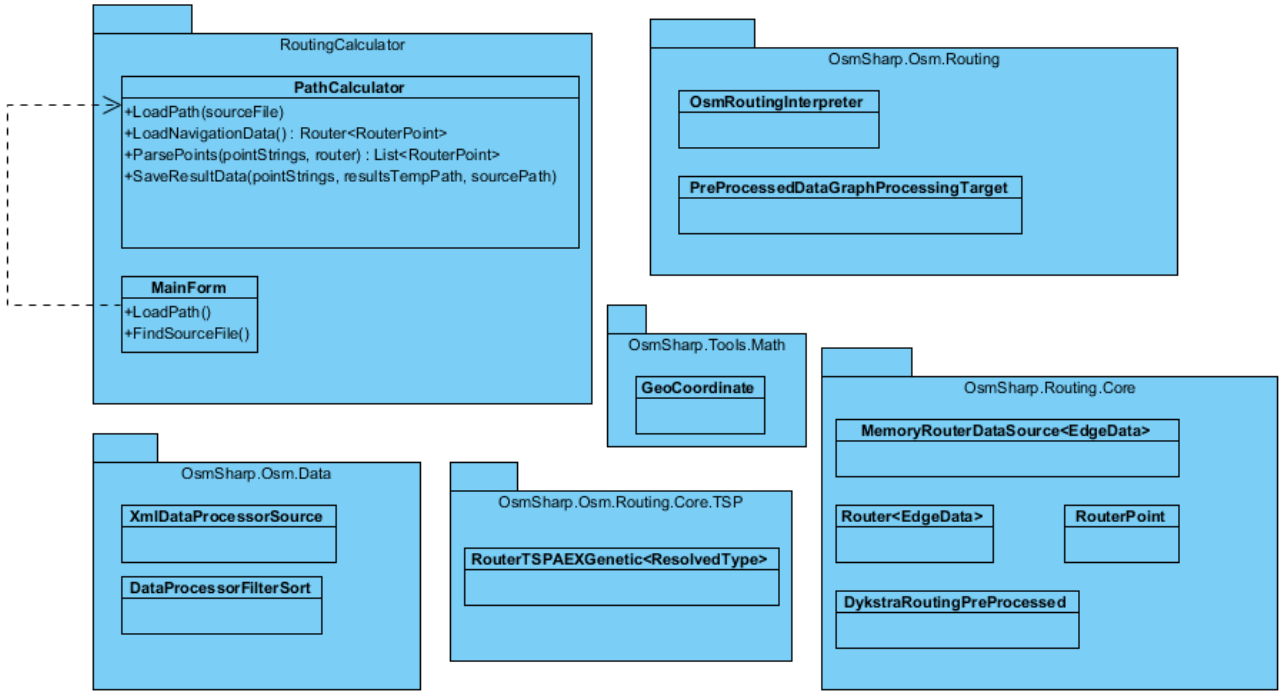
Krok	Popis kroku	Alternativní tok
1	Technik se připojí k terminálu	

2	Nalezne aktuální odečtové místo	UC4
3	Místo se zobrazí na mapovém podkladu	
4	Získá se aktuální pozice, na které se terminál nachází	
5	Mapový podklad se vycentruje na aktuální pozici	
6	Dojde k výpočtu optimální trasy k hledanému místu	UC5
7	Trasa se vykreslí a barevně označí do mapového podkladu	
8	Při pohybu v mapovém podkladu zůstane nalezená trasa zobrazená	
9	Mapa zůstane vycentrována na aktuální pozici	

6.5. Class diagramy

6.5.1. Seřazení bodů trasy

Jde o aplikační část spouštěnou v první fázi při převodu navigačních dat do terminálové podoby. Obsahuje základní algoritmus pro seřazení bodů trasy dle optimální nalezené trasy. Jedná se tedy o vlastní řešení problému obchodního cestujícího.



Obrázek 18: Class diagram - RoutingCalculator

6.5.1.1 Popis OsmSharp

Aplikace je složena z několika menších projektů ve formě dll knihoven. Z projektu OsmSharp, uvedeného dříve, jsou to části:

- OsmSharp.Osm.Routing – knihovna pro obsluhu základních routovacích algoritmů a tříd potřebných k jejich provozu. Využívá se třída *OsmRoutingInterpreter*, která slouží k vytvoření grafu silniční trasy na základě zadaných podkladů, dále pak třída *PreProcessedDataGraphProcessingTarget,* která obsluhuje techniku preprocesingu pro předzpracování těchto dat. Spolupracuje s následující knihovnou.

- OsmSharp.Routing.Core – kde jsou obsaženy základní třídy pro uchování těchto dat například *MemoryRouterDataSource*, která obsahuje aktuálně v operační paměti předzpracované informace pro rychlejší přístup k navigačním datům. Třídy *Router, IRouter, RouterPoint,* které reprezentují vlastní obecný nástroj pro provedení výpočtu, respektive jednotlivý hledaný bod pro výpočet. *DykstraRoutingPreProcessed* slouží k pomocnému zpracování fat za užití dijkstrova algoritmu.

- OsmSharp.Osm.Routing.TSP – obsahuje algoritmy přímo určené k řešení TSP (traveling salesman problém), tedy problému obchodního cestujícího. My využíváme především třídu *RouterTSPAEXGenetic*, která implementuje základní genetický algoritmus určený ke zpracování problému, jenž byl popsaný již dříve v předchozích kapitolách.

- OsmSharp.Osm.Data a OsmSharp.Tools.Math – obsahují další pomocné třídy pro uchování a obecné zadání koordinátů v systému GPS a předzpracování dat pro tvorbu grafu.

6.5.1.2 Popis RoutingCalculator

V hlavním projektu s názvem RoutingCalculator je zpracované jednoduché uživatelské rozhraní umožňující modelové načtení a zpracování dat. Dále pak třída *PathCalculator*, která

slouží k obsluze výpočetních funkcí a implementuje algoritmy pro dopracování vstupních dat, jejich optimalizaci a převod výstupních dat do podoby kompatibilní s terminálovou aplikací.

Ve třídě *PathCalculator* jsou metody:

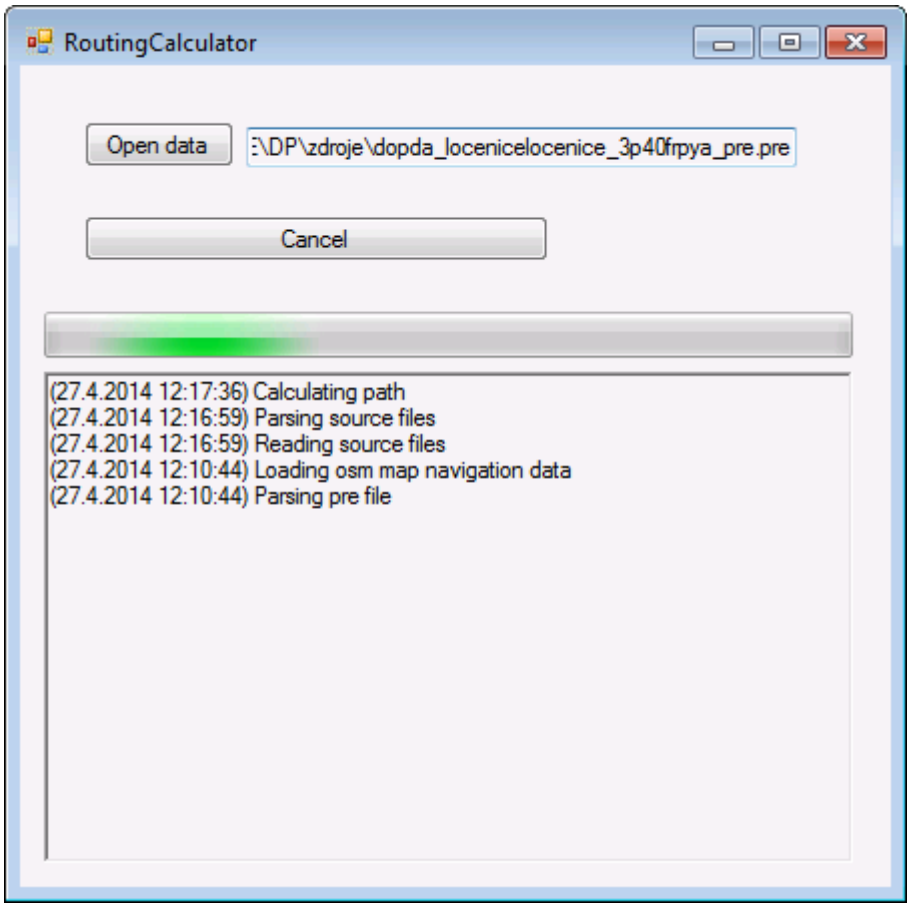
- *LoadPath(sourcePath)* – hlavní veřejná statická metoda, která jako parametr získává cestu ke zdrojovému souboru obsahující navigační data pro body, které je potřeba zoptimalizovat a nalézt optimální trasu. Tato metoda využívá ostatní dále uvedené metody ke své funkci.

- *LoadNavigationData()* – načítá zdrojová navigační data, která popisují silniční síť, respektive graf na základě této sítě vytvořený s využitím výše zmíněných technik preprocessingu. Tato metoda vrací instanci třídy pod interface *IRouter*, tedy již připravený objekt reprezentující graf a obecný nástroj pro práci s tímto grafem.

- *ParsePoints(string[] pointStrings, IRouter<RouterPoint> router)* – tato metoda zpracuje vstupní soubor, respektive pole řetězců, které představují hledané body trasy, jež je potřeba obejít. Dále pak instanci třídy typu *IRouter*, která umožňuje vytvářet a zadávat nové body trasy převedené do podoby *RouterPoint* tak, aby bylo v obecné podobě možné s těmito body již standartně v rámci systému pracovat. Metoda vrací kolekci těchto právě vložených bodů.

- *SaveResultData(string[] pointStrings, string resultsTempPath, string sourcePath)* – je metoda, která se volá po ukončení výpočtu a slouží k převodu výsledných dat do podoby využívané terminálovou aplikací. Jako parametr získá kolekci vstupních bodů, cestu k dočasnému souboru s vypočítanou trasou a cestu k uložení nového výsledného souboru. Na základě těchto informací upraví zdrojový soubor tak, že změní pořadí bodů trasy dle výsledků uvedených na cestě *resultTempPath*. Tento upravený soubor pak uloží na umístění *sourcePath* se sufixem „_Sorted“ .

Třída *MainForm* představuje jednoduché grafické uživatelské rozhraní, ve kterém je možné načíst odpovídající zdrojový soubor a spustit výpočet.



Obrázek 19: Grafické uživatelské rozhraní - RoutingCalculator

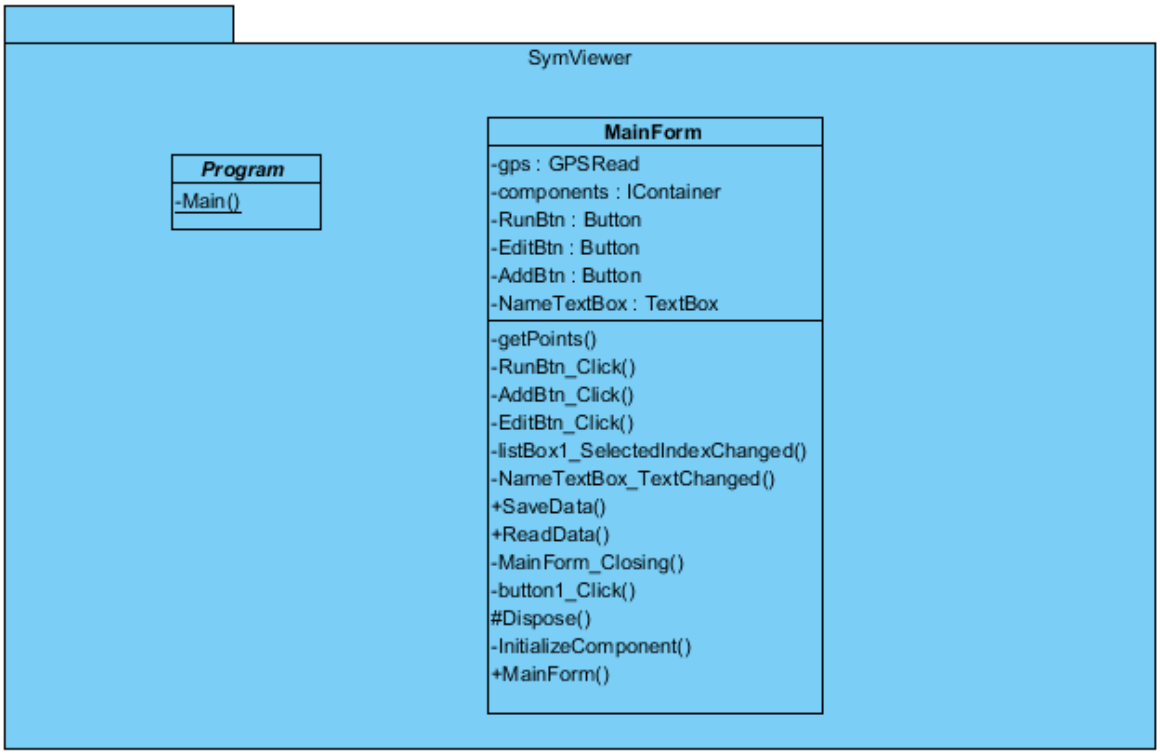
6.5.2. Terminálová aplikace

Jedná se o vlastní aplikaci v přenosném terminálu, která slouží k výpočtu optimální trasy k následujícímu odečtovému bodu a jejímu zobrazení. Skládá se ze tří větších komponent.

6.5.2.1 SymViewer

Jde o první část modelu terminálové aplikace. Slouží k volbě plánované trasy. Vstupem je seznam navigačních bodů získaných z předchozí aplikace. Dále umožňuje editaci, uložení a přidání bodů do této trasy.

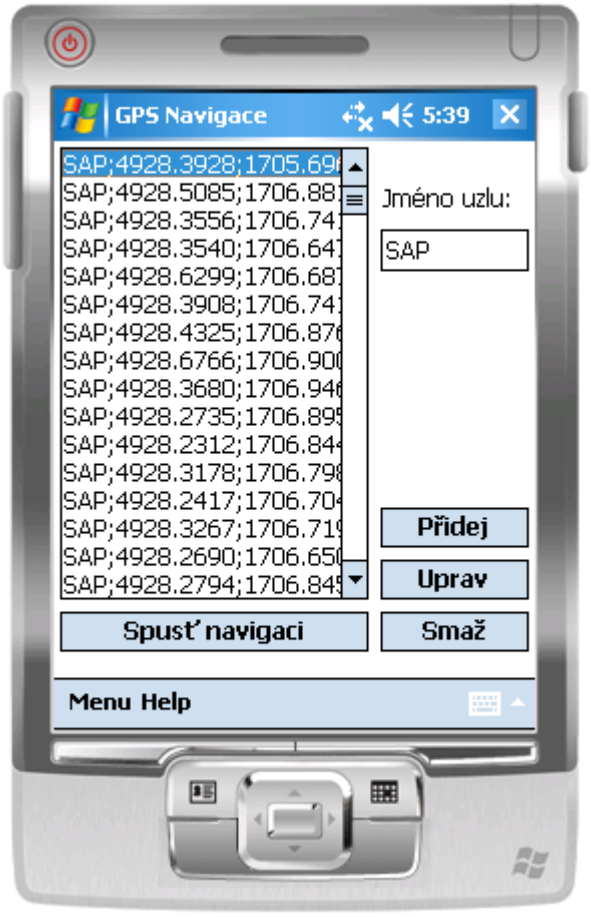
Hlavní funkcí je pak výběr a zobrazení bodu, ke kterému se bude počítat trasa pro navigaci.



Obrázek 20: Class diagram - SymViewer

6.5.2.1.a Popis

Hlavní částí této aplikace je třída *MainForm*, která slouží ke zobrazení uživatelského grafického rozhraní, které umožňuje: načíst aktuální GPS pozici, načíst plánovanou trasu, editovat plánovanou trasu, uložit plánovanou trasu, zobrazit navigaci k odpovídajícímu bodu trasy.



Obrázek 21: Grafické uživatelské rozhraní - SymViewer

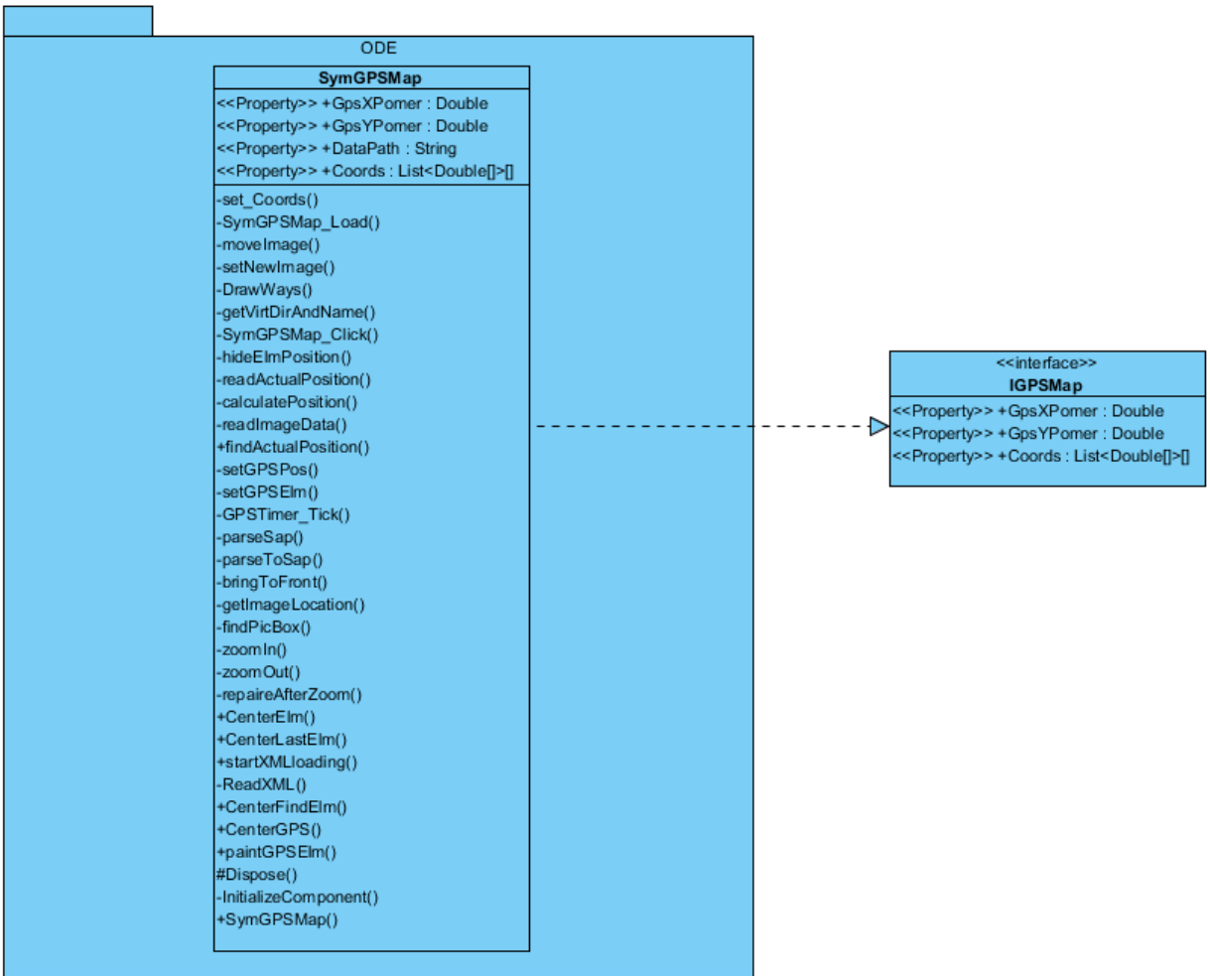
6.5.2.2 SymGpsMap

Další část této aplikace, která se stará o komplexní zobrazování map a mapového podkladu. Tato část se skládá ze dvou hlavních podčástí.

Obrázek 22: Grafické uživatelské rozhraní - SymGpsMap

6.5.2.2.a SymGpsMap

Hlavní část zobrazovací aplikace. Jde o třídu SymGpsMap, jež obsahuje množství metod, které slouží pro obsluhu mapové aplikace. Spolupracuje dále s ostatními moduly pro načítání a správu dat, načítání a synchronizace GPS pozice, přepočet plánované trasy…



Obrázek 23: Class diagram - SymGpsMap

Třída si uchovává informace o cestě ke zdrojovým datovým souborům, informaci o hledaných koordinátech a informaci o aktuálním korekčním zobrazovacím poměru pro X a Y rozměr. Tyto informace se získávají na základě souborů se zdrojovými daty pro mapové podklady (viz 5.2.2.4)

O zobrazování mapových podkladů se starají metody *setNewImage(), moveImage(), zoomIn(), zoomOut(), repaireAfterZoom(), paintGPSElm().*

Pro pravidelnou synchronizaci, obnovu a načítání aktuální GPS pozice se využívá timer, který vyvolává metodu *SymGPSMap_Click(), readActualPosition().*

Pro nalezení a centrování zobrazované mapy na odpovídající pozici elektroměru nebo na aktuální pozici dle GPS polohy se využívají metody *CenterFindElm(), CenterLastElm(), CenterGPS().*

K vyvolání a aktualizace přepočtu optimální trasy k hledanému místu se využívá metoda *DrawWays().*

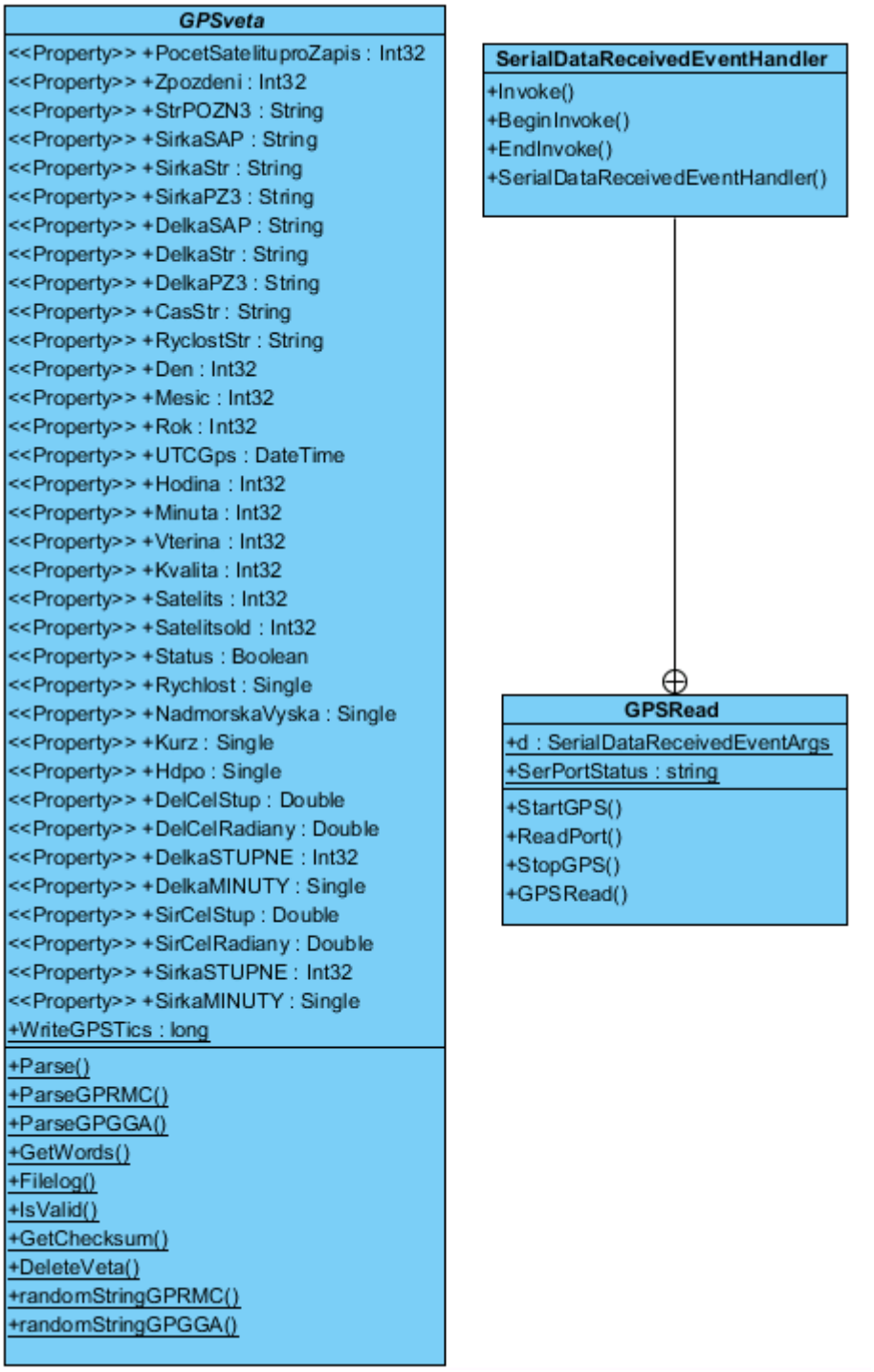
Tato třída je implementuje interface *IGPSMap* pro možnost jejího obecného využití ve vázaných projektech.

6.5.2.2.b GPSveta

Jedná se o komunikační můstek aplikace, která komunikuje se sériovým portem zařízení a získává z něj informace o aktuální GPS pozici. Tyto informace poskytuje připojený GPS modul.

Vlastní načtená a zpracovaná data jsou z GPS věty zobrazena v odpovídajících vlastnostech třídy *GPSveta*.

Periodické načítání dat a komunikaci se sériovým rozhraním zajišťuje třída *GPSRead.*



Obrázek 24: Class diagram - SymGpsMap (načtení aktuáůní GPS pozice)

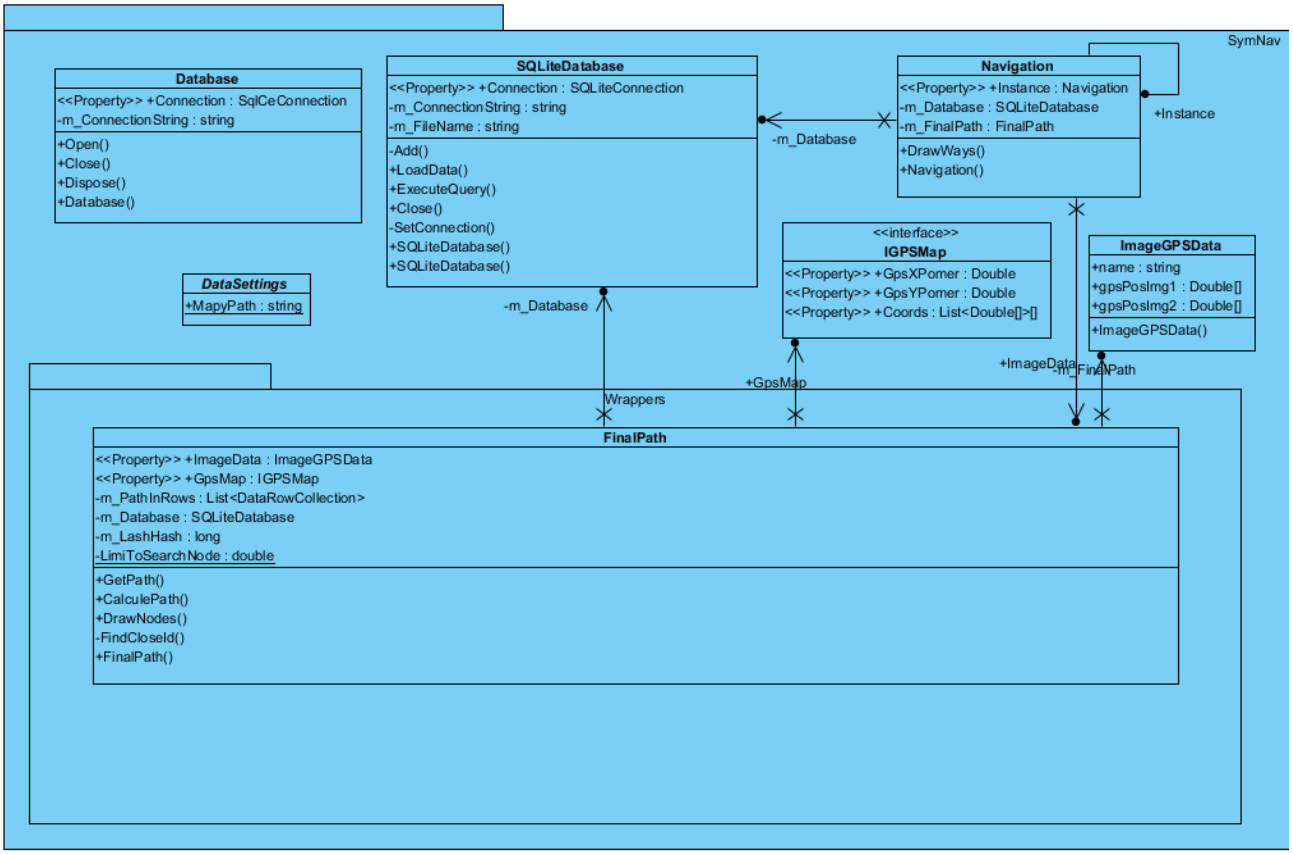
6.5.2.3 SymNav

Jedná se o poslední část aplikace, která zajišťuje vlastní výpočet optimální trasy k hledanému cílovému bodu. Poskytuje seznam bodů určených k zobrazení a upravuje

vizuální podklad daného zobrazovaného obrázku tak, že tyto body aproximuje křivkou, kterou vykreslí.

Tuto aplikaci opět pro přehlednost rozdělíme do dvou částí.

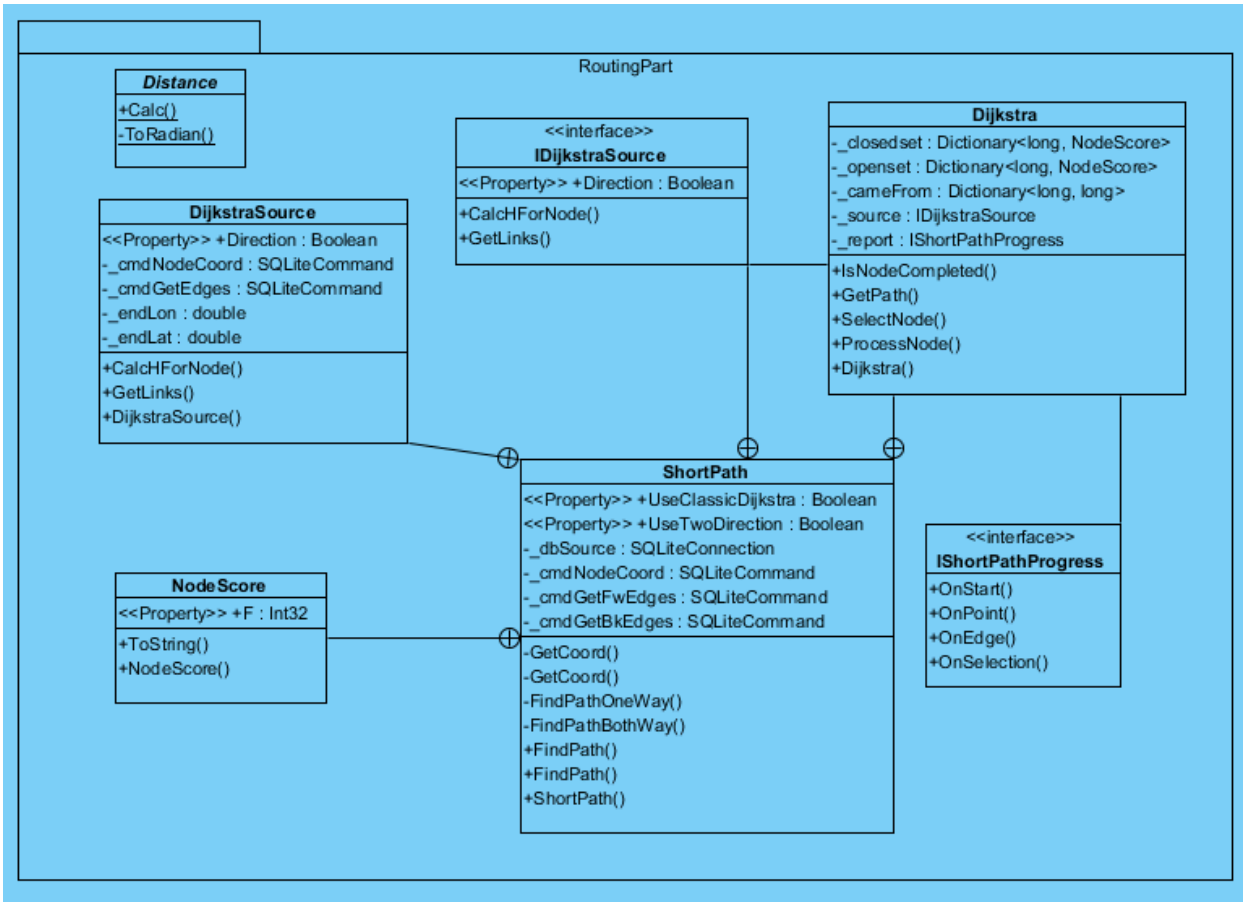
6.5.2.3.a SymNav – obsluha



Obrázek 25: Class diagram – SymNav (obsluha)

Jde o část, která se zabývá obsluhou a zpracování dat pro výpočet optimální trasy. Základem je třída *Navigation*, která uchovává informaci o nalezené optimální trase a umožňuje její přepočet.

6.5.2.3.b SymNav – RoutingPart



Obrázek 26: Class diagram - SymNav (RoutingPart)

Jde o část, která vypočítá optimální trasu k hledanému bodu za užití optimalizovaného Dijkstrova algoritmu.

6.6. Použité technologie

Zde je obsažen seznam použitých technologií. Jejich detailní funkce a využití je popsané v kapitole, která danou technologii přímo využívá.

- Windows Mobile, .Net CF – základní platforma pro terminálovou aplikaci. Operační systém a Framework, který umožňuje běh programů v technologii .Net a ve své upravené a redukované podobě v technologii .Net compact Framework.
- Visual Studio 2008 (2012), C# (https://www.microsoft.com/cze/msdn/vstudio/) – vývojové prostředí pro jazyk C#. Pro použití se systémem .Net CF lze využít pouze VS

verze 2008, případně starší. Pro vývoj desktopové části aplikace byla využita verze VS 2012.

- SQLite, SQL Server CE (http://www.sqlite.org/) – databázové nástroje využité v rámci aplikace. Jde o databázové systémy, které nepotřebují ke svému provozu speciální serverovou aplikaci. Systém SQLite je pod licencí public domain a je podporovaný v mnoha platformách pro velké množství jazyků. Výhodou je také obecnost a široká škála postupů, které se dají využít. Oba nástroje se obsluhují za pomoci jazyka SQL.

- OpenStreetMaps.org (http://www.openstreetmap.org) – mapa světa pod otevřenou licencí tvořená komunitně přímo jednotlivými uživateli.

- OSMSharp (http://www.osmsharp.com/) – nástroj určený k řešení problémů souvisejících s optimalizací tras a navigací. Opět je dostupný pod otevřenou licencí. Implementuje algoritmy a postupy pro řešení různých navigačních problémů a spolupracuje s datovým podkladem z OpenStreetMaps.org

- Visual Paradigm (UML) (http://www.visual-paradigm.com), Enterprise Architect (SPEM) (http://www.sparxsystems.com/) – analytické vývojové nástroje použité ke zpracování potřebných diagramů a návrhu aplikace.

6.7. Popis instalace

6.7.1. Instalace aplikace pro seřazení dat (RoutingCalculator)

Tuto aplikaci není nutné instalovat, stačí rozbalit přiložený soubor RoutingCalculator.rar, který obsahuje spustitelný soubor aplikace a potřebná mapová data.

Pro provoz je potřeba načíst odpovídající zdrojový soubor s hledanými informacemi. Tento soubor lze načíst buď ve formátu inf (formát, který následně vstupuje do terminálové aplikace) nebo ve formátu pre (formát pro systémový import dat z datového zdroje SAP, jde o vnitrofiremní formát vstupních dat a jeho popis není součástí této práce).

Následně je po spuštění výpočtu vygenerován soubor s optimální trasou, který je možné nahrát do terminálové aplikace.

6.7.2. Instalace terminálové aplikace (SymGps)

Pro instalaci terminálové aplikace do mobilního zařízení obsahující systém Windows Mobile je připraven soubor SymGPSInstall.CAB, který obsahuje autoinstalátor pro toto zařízení. Tento soubor stačí (pomocí připojení se technologií ActiveSync) zkopírovat do mobilního zařízení. Systém Windows Mobile automaticky rozbalí a nainstaluje tuto aplikaci.

Aby aplikace fungovala, je potřeba vložit na správné místo všechny potřebné podkladové soubory přiložené v příloze Mapy.rar. Datové soubory jsou ve výchozím nastavení očekávány na adrese „Storage Card\Mapy\“.

6.8. Optimalizace výpočetních operací

V této kapitole budou rozebrány některé možnosti optimalizace výpočetních částí aplikace.

6.8.1. Technologické optimalizace

První možností jsou optimalizace technologické, o kterých již byla řeč v předchozích kapitolách. Jde o využití databáze jako zdroje dat, důsledné doplnění indexů ke sloupcům, přes něž probíhá hledání, nebo výpočty a využití vhodných technologií pro organizaci těchto dat (blíže popsáno v 5.2.2.3.b).

Další optimalizací je optimalizace výpočtů, tedy snaha, aby se prováděl výpočet pouze v okamžiku, kdy je to nutné, a pouze na oblast, pro kterou to je nutné.

Příkladem druhého typu optimalizace je vykreslování dat ve třídě *SymNav.Wrappers.FinalPath,* metodě *GetPath*, které se provede pouze pro aktuálně zobrazovaný obrázek, není nutné vykreslovat trasu pro obrázky, které se nezobrazují.

Dále se zde ukládá i hash poslední množiny vyhledávaných dat (kombinace aktuální pozice a hledané pozice), který se při každém přepočtu kontroluje, a pokud nedojde k jeho změně, překreslení trasy se neprovádí.

6.8.2. Omezení prohledávané části

Další možností optimalizací, které již byly zmíněny v předchozích kapitolách, jsou optimalizace vycházející z možnosti omezení počtu bodů určených ke zpracovávání; příkladem je omezení prohledávané oblasti jen na určitý okruh od hledaného místa. Tento okruh lze také omezit díky faktu, že odečítači se pohybují pravidelně po přibližně stejných trasách. Další možná optimalizace se týká rozhodovacího algoritmu, který ukončí výpočet v okamžiku, kdy už byla nalezena trasa, jež splňuje zadané parametry, takže již není nutné prohledávat zbylou množinu uzlů.

Příkladem použití je vyhledávání bodu blízkého trase ve třídě *SymNav.Wrappers.FinalPath.* Metodě *FindCloseId.*

Tabulka 4: Doba nalezení blízké GPS pozice v závislosti na provedených optimalizacích

Omezující podmínka	Technologická úprava	Čas zpracování [s]	Počet mezivýsledků
limit = 0.1		106	87542
limit = 0.01		46	9779
limit = 0.01	doplněn Rtree	7	9777
limit = 0.01	zúžení pomocí Y koordinátů	2	128
limit = 0.01	seřazení výstupu a LIMIT 1	1	128

Jak ukazuje předchozí tabulka, zúžení prohledávané oblasti na odchylku v oblasti několika kilometrů od reálné pozice nám pomůže zmenšit čas potřebný pro vyhledávání o jeden řád. Příklad bez využití limitu zde neuvádíme, protože se při testování čas potřebný k výpočtu pohyboval v řádech desítek minut.

Další optimalizace byla dosažena pomocí technologické optimalizace, kdy byla zavedena technologie R-stromu – zúžení omezované oblasti i do dalšího směru a omezení dotazu do databáze tak, aby vracel pouze jeden nejbližší výsledek místo množiny výsledků.

6.8.3. Optimalizované hledání v SQL datech

Při vlastním hledání optimální trasy v terminálové aplikaci byla provedena optimalizace výchozího Dijkstrova algoritmu dle příkladu [18], který představuje implementaci klasického Dijkstrova algoritmu.

Optimalizace byla provedena omezením vstupních dat pro výpočet a dále pak optimalizací vlastního SQL dotazu, který získává GPS souřadnice pro každý zpracovávaný bod tak, aby bylo možné spočítat jeho vzdálenost a určit výhodnost.

Testování bylo provedeno na trase mezi body 49.001939N, 14.0067E a 49.16959N, 14.37887E. Celková vzdálenost bodů je 41km.

Původní dotaz používal technologii inner join:

select minX,minY,avg(rd.maxspeed) avgs from Road rd inner join WayNode wn on rd.wayId=wn.wayId inner join nodes n on wn.nodeId=n.id where wn.nodeId=(select nodeId from intersection where id=@id)

Provedení jednoho dotazu v původní podobě trvalo na mobilním zařízení cca 2s , na PC 0,06 s. Po zrušení výpočtu průměrné rychlosti tentýž jeden příkaz trval 1 s.

Vzhledem k tomu, že tento příkaz se provádí opakovaně v tisících volání, došlo k tomu, že původní výpočet trval na mobilním zařízení řádově desítky minut.

Proto byl dotaz upraven tak, aby nedocházelo ke zbytečnému napojování dat za pomocí technologie INNER JOIN, která vytváří velké množství kombinací a za určitých podmínek způsobuje vysokou náročnost výpočtu

Upravený dotaz:

select minX,minY from nodes n where n.Id=(select nodeId from intersection where id=@id) LIMIT 1

Touto úpravou bylo dosaženo řádového zrychlení celého výpočtu z desítek minut na cca 30 s.

Po následné úpravě a zúžení hledané trasy došlo k dalšímu zkrácení výpočetní doby, kdy celý se výpočet pohyboval lehce nad 10 s. Vzhledem k tomu, že toto měření bylo prováděno v emulátoru, lze uvažovat, a testy v terénu toto potvrzují, že reálné doby odezvy na odpovídajících zařízení jsou cca poloviční, tedy cca 5 s. Délka a složitost ověřované trasy patří k náročnějším případům, protože se dá předpokládat, že pohyb odečítače bude v jedné ucelené oblasti a mezi místy odečtu by měly být vzdálenosti řádově v kilometrech.

6.8.4. Načítání a zobrazení dat v systému

Tomuto problému se již věnovala kapitola 5.2.3.2.c. V aplikaci tento problém řeší metoda *DrawNodes* ve třídě *System.Wrappers.FinalPath.*

Pro dodatečnou optimalizaci bylo postupováno pomocí optimalizačních postupů popsaných ve zdrojích: [19],[20].

Testování průměrného času potřebného pro vykreslení všech bodů plánované trasy a jeho optimalizace ukazuje následující tabulka.

Tabulka 5: Doba vykreslení bodů plánované trasy v závislosti na optimalizacích

Technologická úprava	**Čas zpracování [s]**
	32
PRAGMA synch = OFF, journal_mode= memory	33
PRAGMA locking_mode = normal	34
doplněny chybějící indexy	2

Z Tabulky jsou patrné otestované způsoby optimalizace pomocí uvedených zdrojů a jejich výsledky. Optimalizace vnitřního nastavení SQLite se neukázala v tomto případě jako efektivní. Nejvýraznější výsledek mělo doplnění chybějícího indexu na pole, přes něž probíhá napojení dat.

Další provedenou optimalizací je způsob ukládání těchto mezivýsledků (do proměnné *FinalPath.m_PathInRows*), kdy pokud nedojde ke změně aktuální trasy, nebo vyvolání přepočtu, tak si systém uchovává nalezený seznam bodů připravených k vykreslení,

a v okamžiku přepnutí na odpovídající obrázek dojde pouze k vykreslení této množiny. Tedy se již opakovaně neprovádí dotaz do databáze, pokud to není nezbytně nutné z důvodu změny hledaných souřadnic.

6.8.5. Závěr

Závěrem lze říci, že díky provedeným optimalizacím terminálové aplikace se doba mezi zahájením výpočtu a vykreslením výsledku uživateli v náročnějších případech zkrátila z původních několika desítek minut na hodnotu cca 7s. V reálném použití jsou tyto doby ještě kratší díky menší vzdálenosti odečtových míst.

7. Testování

Testy rychlosti odezvy terminálové aplikace, na které nejvíce záleží rychlost odezvy aplikace a která provádí hledání optimální trasy k následujícímu odečtovému místu a zobrazování dat, byly popsány v předchozí kapitole. Z výsledků vyplývá, že zobrazení a nalezení trasy k příštímu odečtovému místu je v reálných situacích, které budou při provozu aplikace nastávat v řádech sekund (v závislosti na délce a složitosti trasy).

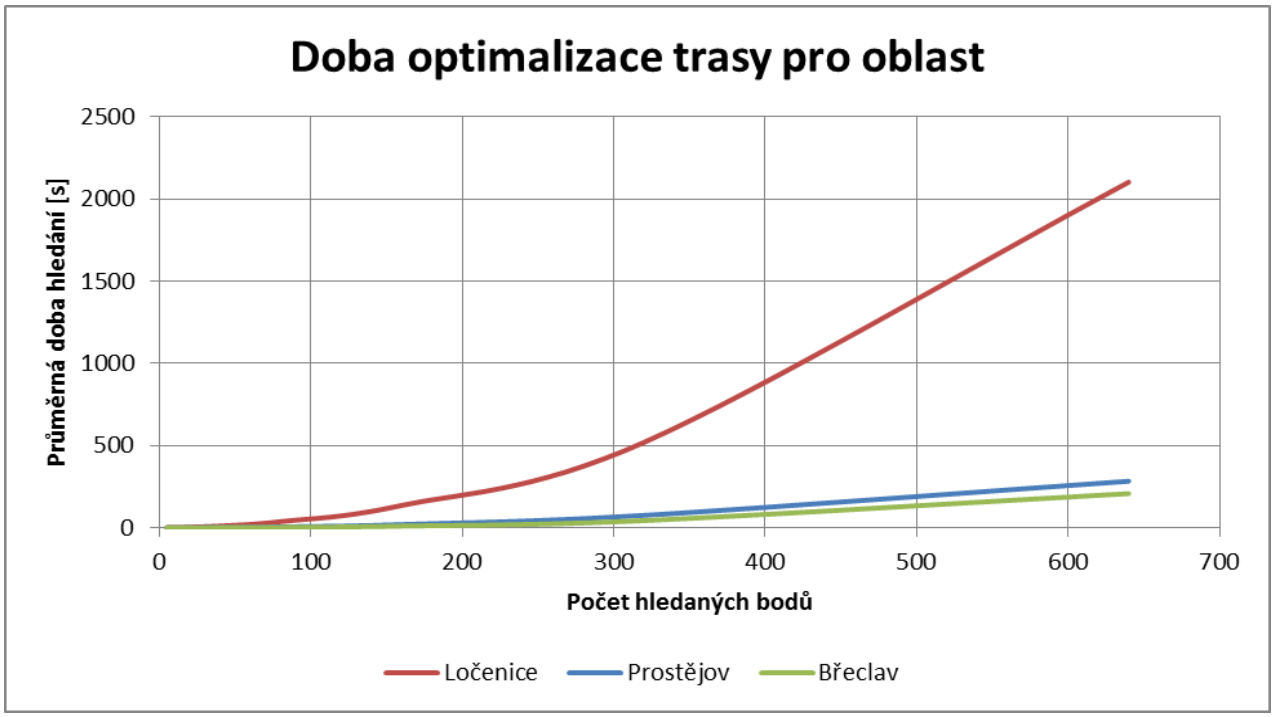
Další testování je rozděleno do dvou částí. První částí je rychlostní testování optimalizačního algoritmu prováděného v aplikaci *RoutingCalculator* (která, jak již bylo uvedeno, provádí výpočet trasy obchodního cestujícího a připravuje data pro navigaci v terminálové aplikaci), druhou částí je test terminálové aplikace podle zvolených testovacích scénářů.

7.1. Rychlostní test *RoutingCalculator*

Testování probíhalo měřením doby odezvy aplikace potřebné pro výpočet optimální trasy, tedy doby potřebné pro vyřešení problému obchodního cestujícího v závislosti na množství hledaných bodů. Další parametr, který výrazně ovlivňuje rychlost výpočtu, je velikost prohledávané oblasti, tedy i vzájemná vzdálenost jednotlivých hledaných bodů.

Vstupní body pro vyhledávání vychází z reálných dat a jsou uvedeny v přílohách. Podkladem pro vyhledávání jsou kompletní data pro Jihočeský a Jihomoravský kraj.

Ukázky výsledků jsou v níže uvedeném grafu (Obrázek 27: Graf doby výpočtu optimalizace trasy) a v tabulkách: Tabulka 6 a Tabulka 7.



Obrázek 27: Graf doby výpočtu optimalizace trasy

Z testování vyplynula polynomiální závislost na množině vstupních bodů. Nejedná se o závislost exponenciální, růst je zde pozvolnější (především díky využití genetického algoritmu), ale také nejde o čistě lineární závislost.

Další informace, která vyplynula z testování, je, že doba výpočtu je velmi výrazně závislá na velikosti prohledávané oblasti. Například pro oblast „Ločenice“, která představuje oblast vesnic a kde jsou obsaženy body i desítky kilometrů od sebe, jsou doby odezvy o řád vyšší než v městské oblasti „Prostějov“, kde se vzdálenost mezi body pohybuje maximálně v řadech kilometrů.

Důvodem tohoto časového rozdílu je nutnost vypočítat vzájemné vzdálenosti mezi jednotlivými body za pomocí Dijkstrova preprocesoru, blíže viz 5.2.3.1.a.

Tabulka 6: Doba výpočtu optimální trasy obchodního cestujícího Ločenice (oblast 680 km^2)

Počet hledaných bodů	Velikost oblasti[km^2]	Čas průměr [s]
5	671	0,87
10	671	1,8
20	671	4,08
40	671	11,75
80	674	37,6
160	677	136,18
320	681	519,4
640	681	2101,96

Tabulka 7: Doba výpočtu optimální trasy obchodního cestujícího Prostějov (oblast 7 km^2)

Počet hledaných bodů	Velikost oblasti[km^2]	Čas průměr [s]
5	3	0,05
10	3	0,2
20	3	0,5
40	5	1,8
80	7	5,3
160	7	19,7
320	7	76
640	8	283,27

7.2. Protokol z testování aplikace

Tabulka 8: Ukázka protokolu z testování aplikace

č.	Stručný popis testovacího případu	Popis nebo odkaz na přílohu č…	Očekávaný výsledek testu	Kategorie chyby
1	Vytvoření plánu trasy a převod dat do terminálu	Provede se vložení dat a přepočet dle dokumentů 1, 2	Provede se převod a optimalizace dat do inf souboru pro vložení do terminálu	OK=Bez chyby
2	Vložení dat do terminálu	Na SD kartu se zkopíruje vytvořený inf soubor a otevře se z terminálové aplikace	Data se načtou v podobě inf souboru a umožní výběr bodu pro navigování	4=Malá chyba
3	Navigace k vybranému odečtovému místu	Provede se výběr požadovaného bodu a spustí se navigace, dle dokumentu 3	Zobrazí se hledané místo a nalezená trasa od aktuální pozice	OK=Bez chyby
4	Výběr dalšího odečtového místa a navigace k němu	Provede se výběr jiného bodu v pořadí a spustí se navigace, dle dokumentu 3	Zobrazí se hledané místo a nalezená trasa od aktuální pozice	OK=Bez chyby
5	Zobrazení nalezené trasy na mapě	Uživatel zadá hledaný bod (TC 3,4) a bude se šipkami pohybovat po mapovém podkladu, dokument 5, 6	Na mapě se vykreslí nalezená trasa	4=Malá chyba
6	Nalezení místa dle názvu obce	Uživatel zadá hledaný název obce pomocí stisku klávesy "F" a potvrdí stiskem klávesy "enter", dokument 4	Zobrazí se hledané místo	OK=Bez chyby
7	Hledání místa mimo mapový podklad	Uživatel zadá hledané místo pomocí GPS souřadnic, nebo názvu obce, které je mimo hranice podkladu	Zobrazení informace o neexistenci podkladu	OK=Bez chyby
8	Změna aktuální pozice	Uživatel změní svou aktuální pozici a vyvolá přepočet	Vyvolá se přepočet optimální trasy k cíli	OK=Bez chyby

Z výsledků testování vyplynulo, že aplikace je funkční, většina testů dopadla úspěšně a bez chyb. Pouze ve dvou případech se objevila malá chyba.

V případě 2 jde o fakt, že v situaci pádu aplikace se ztratí reference na zdrojový soubor a je nutné jej v aplikaci znovu vybrat.

V případě 5 jde o problém, kdy za určité situace se křivka trasy vykreslí až za hledaný bod. Tento problém je způsoben tím, že trasa se aproximuje obecným způsobem vždy mezi GPS body nalezené trasy. Pokud cílový bod leží mezi dvěma těmito body, vykreslí se křivka až k dalšímu nalezenému bodu, tedy za hledaný bod.

Žádné zásadní chyby nebo problémy nebyly nalezeny. Celý protokol testování je obsažen v příloze pod názvem Test_protocol_SymGPS_MMR_2014_04_17_tested.xlsx.

8. Závěr

Závěrem lze říci a z výsledků testování rovněž vyplývá, že aplikace je funkční a splňuje zadaná kritéria. Podařilo se najít optimální algoritmus, respektive kombinaci algoritmů, postupů a vhodných datových zdrojů, aby bylo možné realizovat aplikaci, která splní zadaná kritéria.

Dále byl proveden návrh a implementace vlastní aplikace. Kritéria jsou dále popsána a vyhodnocena jednotlivě.

8.1. Vstup pro více uzlů

Vstupem aplikace je importovaný datový soubor, který může obsahovat libovolný počet uzlů v požadovaném formátu.

8.2. Dynamický přepočet

Tento bod je splněn díky možnosti přepočtu hledané trasy v terminálové aplikaci. Z důvodů efektivity nelze v této verzi aplikace v terminálové aplikaci přepočíst celou plánovanou trasu obchodního cestujícího, ale jen trasu k příštímu odečtovému bodu.

Výpočet trasy obchodního cestujícího se provádí v desktopové aplikaci, která provede seřazení a optimalizaci těchto bodů před jejich nahráním do terminálu.

8.3. Rychlost výpočtu, která umožní do 10 s zobrazit výsledek operace

Díky technologickému postupu, který předpočítá trasu, pomocí nástroje *RoutingCalculator*, ještě před jejím nahráním do terminálu, lze říci, že úkoly, které provádí terminálová aplikace (nalezení trasy k příštímu odečtovému místu, vykreslení mapy s touto trasou) a její odezvy jsou v přibližně konstantním čase (nezávislém na celkovém počtu procházených bodů), který se pohybuje v reálných situacích do 10 s (závisí na složitosti silniční sítě a vzdálenosti od cíle).

Doba trvání výpočtu trasy obchodního cestujícího v desktopové aplikace *RoutingCalculator* je popsaná v bodě 7.1 a bude v některých případech delší než požadovaných 10 s (záleží na množství zadaných bodů). Tento výpočet se ale provede dopředu v okamžiku, kdy je volná výpočetní kapacita PC a do terminálu vstupují tato data již seřazená, takže tato doba potřebná pro výpočet se již neprojeví při práci a zobrazování dat v aplikaci v terminálu.

8.4. Aplikace musí být schopna nalézt cestu, i pokud je odečítač mimo silnici (kombinace jízdy autem a ostatní dopravy)

Aplikace umožňuje nalézt trasu, i pokud je výchozí bod mimo silniční síť. Umožňuje tedy kombinovat dopravu autem a pěší dopravu mimo vozovku.

8.5. Možnost trajektorii dynamicky přepočítat během cesty

Trajektorie a výpočet trasy k příštímu odečtovému bodu je součástí terminálové aplikace, je v ní možné vyvolat dynamický přepočet.

8.6. Nalezení bodu bez nutnosti znát cílovou GPS pozici

Aplikace obsahuje databázi měst a jejich GPS pozic, na základě těchto informací je možné hledat i body, které nebyly obsaženy v původním plánu.

8.7. Všechny výpočty musí probíhat offline, bez nutnosti připojení k internetu

Všechny datové podklady aplikace jsou k dispozici offline, včetně grafických a navigačních mapových podkladů, stejně tak knihovny potřebné pro výpočet optimální trasy. K výpočtu tedy není nutné připojení k internetu.

8.8. Pozice v obrazovém podkladu se musí zobrazit s přesností „na dům“

Při řešení tohoto úkolu bylo zjištěno, že grafické mapové podklady obsahují nepřesnosti dané jejich linearizací do dvourozměrné podoby. Z tohoto důvodu byly zavedeny a vypočteny korekční koeficienty, které pro každý obrázek mapového podkladu vyjadřují

jeho zkreslení vůči realitě, respektive jaká část geografického stupně odpovídá jednomu pixelu v určeném směru. Díky těmto přepočtovým koeficientům se lze k této přesnosti výrazně přiblížit. Vždy však záleží na kvalitě mapového podkladu v daném místě.

8.9. Určit aktuální pozici z GPS modulu a pamatovat si poslední nalezenou pozici v případě ztráty signálu

Při práci s aplikací se uchovává poslední nalezená pozice, která pak slouží jako zdroj pro hledání v případě, že dojde ke ztrátě signálu GPS.

9. Seznam literatury

[1] KUČERA, Petr. Metodologie řešení okružního dopravního problému. Praha, 2009. Dostupné z: http://www.pef.czu.cz/cs/?dl=1&f=13035. Disertační práce. Česká zemědělská univerzita v Praze, Provozně ekonomická fakulta. Vedoucí práce Prof. Rndr. Jaroslav Havlíček, CSc.

[2] ŠEDA, Miloš. *Teorie grafů* [online]. Brno, 2003 [cit. 2012-04-09]. Dostupné z: http://www.uai.fme.vutbr.cz/~mseda/TG03_MS.pdf. VUT v Brně.

[3] RNDR. TEDA, Jaroslav, Ph.D. PROGRAMUJTE.COM. *Genetické algoritmy a jejich aplikace v praxi* [online]. [cit. 2014-02-18]. Dostupné z: http://programujte.com/clanek/2005072601-geneticke-algoritmy-a-jejich-aplikace-v-praxi/

[4] Pelikán, J.: Diskrétní modely v operačním výzkumu, Professional Publishing, Praha, 2001

[5] Kučera, L.: Kombinatorické algoritmy, SNTL, Praha, 1983

[6] Karp, R.M.: Reducibility, among Combinatorial Problems, In: Complexity of Computes Computations, Plennum Pres, New York, 1972, s.85-104

[7] Kuan, M-K.: Graphic Programming Using Odd or Even Points, Chinese Math., 1, 1962, s.273-277

[8] Edmonds, J., Johnson, J., Ellis, L.: Matching, Euler Tours and the Chinese Postman, Math. Prog., 5, 1973, s.88-124

[9] Cook, S.A.: The Complexity of Theorem Proving Procedures, 3rd STOC, 1971, s.151- 158

[10] NÝDL, Václav. *Diskrétní matematika II.* Vyd. 1. České Budějovice: Jihočeská univerzita, Pedagogická fakulta. 86 s. ISBN 80-704-0492-2.

[11] JIROVSKÝ, Lukáš. Teorie grafů: Diplomová práce, MFF UK. [online]. [cit. 2014-03-14]. Dostupné z: http://teorie-grafu.cz/

[12] HLINĚNÝ, Petr. *Základy Teorie Grafů*. Brno, 2010. Dostupné z: http://is.muni.cz/do/1499/el/estud/fi/js10/grafy/Grafy-text10.pdf. Fakulta informatiky Masarykova Univerzita.

[13] [online]. [cit. 2014-03-30]. Dostupné z: http://www.dupuis.me/sites/default/files/OSMUpload.zip

[14] MIČKA, Pavel. Dijkstruv-algoritmus. *Algoritmy.net* [online]. [cit. 2014-04-17]. Dostupné z: http://www.algoritmy.net/article/5108/Dijkstruv-algoritmus

[15] KOLÁŘ, Josef. Teoretická informatika. 2. vyd. Praha : Česká informatická společnost, 2004. 205 s. ISBN 80-900853-8-5.

[16] OpenStreetMap Wiki contributors. Osmosis [online]. OpenStreetMap Wiki, [cit. 2014-4-21]. Dostupné z: http://wiki.openstreetmap.org/w/index.php?title=Osmosis&oldid=998482.

[17] OpenStreetMap Wiki contributors. OsmSharp [online]. OpenStreetMap Wiki, [cit. 2014-4-21]. Dostupné z: http://wiki.openstreetmap.org/w/index.php?title=OsmSharp&oldid=1013220.

[18] Map routing geospatial database [online]. [cit. 2014-4-21]. Dostupné z: http://www.dupuis.me/node/27.

[19] How do I improve the performance of SQLite?. In: *Stackoverflow.com* [online]. [cit. 2014-04-27]. Dostupné z: http://stackoverflow.com/questions/1711631/how-do-i-improve-the-performance-of-sqlite/

[20] LYON, Jim. SQLite Optimization FAQ. In: *The university of Tennesse Knoxville* [online]. [cit. 2014-04-27]. Dostupné z: http://web.utk.edu/~jplyon/sqlite/SQLite_optimization_FAQ.html

[21] MAREŠ, Amadeo. *1001 tipů a triků pro C# 2010*. Brno: Computer Press, 2011, 416 s. ISBN 978-80-251-3250-0.

[22] NÝDL, Václav. *Diskrétní matematika I.* Vyd. 1. České Budějovice: Jihočeská univerzita, Pedagogická fakulta, 1999, 71 s. ISBN 80-704-0359-4.

10. Seznam obrázků

11. Seznam Tabulek

12. Seznam příloh

- *MMR_optimalizace_cesty.pdf* – Text diplomové práce ve formátu pdf.

12.1. Zdroje_dat

Obsahuje datové zdroje potřebné pro provoz aplikace a její vstupy.

- *Mapy.rar* – zdrojová podkladová a navigační data potřebná pro provoz terminálové aplikace
- *map.osm* – navigační data ve formátu osm
- *breclav.inf, locenice.inf, prostejov.inf* – data obsahující převedené vstupní GPS body určené k hledání

12.2. Aplikace

Obsahuje vlastní aplikaci a související soubory.

12.2.1. Dokumentace

Obsahuje dokumentaci k aplikaci včetně použitých diagramů.

12.2.2. Test

Obsahuje testovací protokol k aplikaci a seznamy naměřených hodnot.

12.2.3. UC

Obsahuje jednotlivé use casy popsané v práci.

12.2.4. Zdrojove_kody

Obsahuje zdrojové kódy k aplikaci.