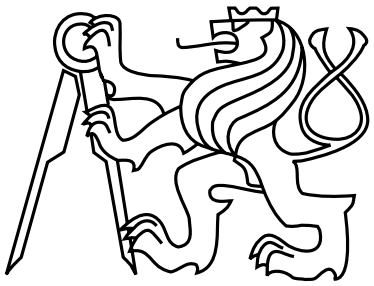
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra Počítačů



Diplomová práce

Systém pro testování kvality dat pro dopravní plánování

Simon Slováček

Vedoucí práce: Ing. Miroslav Bureš, Ph.D.

25. května 2016

Poděkování

Rád byl poděkoval panu Ing. Miroslavu Burešovi, Ph.D za obětavé odborné vedení, podnětnou zpětnou vazbu a veškerou pomoc při vypracování této diplomové práce. Dále děkuji své přítelkyni za podporu během mého studia a především při psaní této práce. V neposlední řadě také děkuji panu Mgr. Janu Hrnčířovi za jeho ochotu a pomoc při tvoření a ladění samotného programu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 25. 5. 2016 .....

Abstrakt

Cílem diplomové práce je vytvořit systém pro testování kvality dat pro dopravní plánování. Vstupní data pro testování jsou dané struktury a pochází z projektu SUPERHUB, který je používá pro plánování tras. Po rozboru vstupních dat je vytvořen návrh rešení pro daný problém. Následně je model realizován a jsou implementovány jednotlivé testovací metody. Pro ověření testovacích metod jsou také vyvinuty generátory chyb, které vytváří umělé defekty v datech pro testovací metody. Následně jsou analyzovány výsledky testů. Implementace jednotlivých metod je realizována za použití programovacího jazyka Java. Pro zobrazení výsledků a snazší ovládání testů je vytvořeno webové rozhraní. Finální testování prokázalo funkčnost implementovaného řešení.

Klíčová slova: SUPERHUB, testování, GTD graf, aplikace, Mongo DB, Google API, kozistence dat

Abstract

The aim of the thesis is to create a system for testing the quality of data for traffic planning. Input data for testing has specific structure and comes from SUPERHUB project, which is used for route planning. After analyzing input data is created a solution for this problem. Then the model is carried out and various testing methods are implemented. To validate the testing methods there are developed generators of errors which create artificial defects in data for testing. Then are test results analysed. Implementation of each method is made by using the Java programming language. For viewing the results and easier tests control is designed web interface. Final testing demonstrated functionality of implemented solution.

Key words: SUPERHUB, testing, GTD graph, application, Mongo DB, Google API, data consistency

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačů

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Simon Slováček**

Studijní program: Otevřená informatika
Obor: Softwarové inženýrství

Název tématu: **Systém pro testování kvality dat pro dopravní plánování**

Pokyny pro vypracování:

Podle instrukcí školitele navrhněte a implementujte systém, který bude testovat kvalitu vstupních dat použitých pro úlohy dopravního plánování. Systém bude používat sadu definovaných technik, které budou zjišťovat potenciální chyby nebo nekonzistence v datech. Formát vstupních dat bude definovaný vedoucím práce. Systém se bude skládat z komponent pro načtení a uložení testovaných dat, modulů pro jednotlivé techniky na kontrolu dat a modulu pro zobrazení výsledků testů. V rámci ověření a vyhodnocení systému proveďte experiment na vzorku reálných dopravních dat s pomocí techniky vložení umělých chyb.

Seznam odborné literatury:

Koomen T, van der Aalst B, Brokeman M, Vroon M. TMap Next, for result-driven testing. UTN Publishers, 2006.

Hrncir J, Jakob M. Generalised Time-Dependent Graphs for Fully Multimodal Journey Planning. In IEEE Intelligent Transportation Systems Conference (ITSC), 2013, pp.2138-2145.

Vedoucí: Ing. Miroslav Bureš, Ph.D.

Platnost zadání: do konce zimního semestru 2016/2017

doc. Ing. Filip Železný, Ph.D.
vedoucí katedry

L.S.

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 8. 10. 2015

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Plánování nejideálnější trasy se stalo každodenní záležitostí mnoha lidí, převážně ve velkých městech. Lidé si vyhledávají spojení hromadné dopravy, ideální trasu autem do práce, či na výlet. Ať už uživatel vyhledává trasu pomocí jakékoliv aplikace, či rozhraní, je nutné, aby podklady pro daný výpočet byly správné a co možná nejúplnější. Pokud vstupní data nebudou správná, cestování po naplánované trase může být v lepším případě časově náročnější, v horším může cestovatel dorazit na úplně jiné místo, než požadoval.

Je tedy vhodné vytvořit nástroj, který bude analyzovat data a poukazovat na chyby, či nesrovnalosti, které by mohly vést ke špatnému naplánování trasy. Pod chybami si lze představit například chybějící informace, nesprávné údaje o poloze nebo jízdních řádech a jiné.

Cílem této práce je vytvoření systému, který bude umožňovat testovat a ověřovat výše popsané skutečnosti. Systém bude schopen testovat data, která slouží jako vstup do plánovače tras. Rozhraní bude poskytovat informace o chybách, či nesrovnalostech v datech pro odborného uživatele, kterému umožní opravit, či dokonce vylepšit vstupní a poskytované informace pro plánovač tras.

1.1 Cíle práce

Cílem této práce je vytvořit systém pro testování kvality dat pro plánovače. K dosažení tohoto cíle je zapotřebí splnit jednotlivé podcíle:

1. Seznámit se s dostupnými daty, která budou analyzována. Je důležité znát jejich strukturu a možnosti, aby navržená struktura aplikace byla pro tyto data použitelná.

2. Vytvořit návrh architektury aplikace s přihlédnutím na analyzovaná vstupní data. Jednotlivé části aplikace budou nezávislé a modulární, tudíž přidání nových komponent, či testovacích metod bude možné i v budoucnu.

3. Samotná implementace, která vytvoří základní strukturu programu se všemi potřebnými moduly. Následně se tyto moduly budou rozšiřovat. Mezi rozšiřování patří například implementace nových testovacích metod, nových generátorů chyb, a podobně.

4. Provedení testů se sadou dostupných dat nad dokončeným systémem. Pomocí zanášení chybných dat budou jednotlivé metody analyzovat vstupní data. Výsledkem testování bude, s jakou přesností jsme schopni pomocí aplikace nalézt a rozlišit určité typy chyb v datech.

1.2 Superhub

Projekt SUPERHUB (SUstainable and PERsuasive Human Users moBility in future cities), který je financován převážně Evropskou komisí, vyvinul otevřenou platformu pro řízení a optimalizaci městské mobility. Klade si za úkol v reálném čase spojit všechny cestovní možnosti se zúčastněnými stranami zároveň s nástrojem umožňujícím mobilní služby, které umí řešit individuální potřeby pasažérů a navrhnou ideální dopravní trasu. SUPERHUB zahrnuje komplexní přístup v oblasti plánování cest v prostředí města. Zahrnuje pohled koncového uživatele a zohledňuje rovněž různé strategie města. Projekt úspěšně doručil platformu plánování cest nové generace, která byla otestována v Barceloně, Miláně, Helsinkách a v Brně v průběhu podzimu 2014. SUPERHUB je nyní open source platformou využívající dostupná data, města tedy nemusí vynakládat žádné prostředky na vývoj aplikací, ale jen do zprostředkování dat z různých systémů. [9]

1.3 Testování softwarových systémů

Při vytváření jakékoliv aplikace je důležité vždy provádět testy, které mohou odhalit chyby v kódu, v implementaci, v navrženém postupu, a podobně.

1.3.1 Testovací principy

Testování je široký pojem a zahrnuje v sobě velké množství technik a možných postupů. Pro přehlednost byl vytvořen výčet základních principů testování, na které by se měl brát ohled v rámci každého testování.

Principy přeloženy z [13]

Princip 1. - Testování dokazuje přítomnost chyb

Testování může dokázat, že chyby existují, ale nemůže dokázat, že žádné chyby neexistují. Testování snižuje pravděpodobnost nenalezených chyb v softwaru. Pokud ale žádné chyby nejsou nalezeny, není to důkaz, že software je bez chyby.

Princip 2. - Kompletní testování není možné

Testování všeho (všech kombinací vstupů a předpokladů) není možné kromě triviálních případů. Na místo kompletního testování by měly být použity analýzy a priority, které řeknou, jakým směrem se má testování soustředit.

Princip 3. - Včasné testování

Chyby by měly být nalezeny co nejdříve. Testování by mělo začít co nejdříve je možné v rámci vývojového životního cyklu a soustředit se na definované cíle.

Princip 4. - Třídění chyb

Testovací úsilí by mělo být zaměřeno úměrně k místům v modulech, kde se předpokládá či později nalezne velké množství chyb. Malé množství modulů často obsahuje nejvíce chyb nalezených při testování během vývoje nebo jsou příčinou nejvíce operačních selhání.

Princip 5. - Pesticidní paradox

Pokud jsou stejné testy prováděny znovu a znovu, nemusí tyto stejné testovací sety už najít žádnou novou chybu. Pro vyvarování se tohoto "pesticidního paradoxu"musí být testovací scénáře obnovovány a vyměňovány a nové a odlišné scénáře vytvořeny pro zvýšení pravděpodobnosti nalezení nových chyb.

Princip 6. - Testování je závislé na kontextu

Testování se provádí různě v souvislosti s odlišným kontextem. Například software, pro který je důležité zabezpečení, bude testován odlišně než komerční software bez nutnosti zabezpečení.

Princip 7. - Klam absence chyb

Nalezení a oprava chyb nepomůže, pokud systémové řešení je nepoužitelné a neodpovídá zadavatelovým potřebám a požadavkům.

1.3.2 Testovací fáze

Testování může probíhat v kterékoliv fázi softwarového projektu. Vždy záleží na zvoleném postupu a především na časových možnostech, které lze přidělit testování. Optimální množství testů totiž nelze stanovit, neboť nemůžeme obsáhnout všechny kombinace kritérií, které mohou nastat za běhu programu.

Testování lze rozdělit podle fáze, ve které se projekt nachází:

- **předimplementační fáze** - v této fázi projektu se zejména zkoumá a testuje navrhované řešení. Testuje se smysl a proveditelnost funkcí aplikace, scénářů, návrh struktury, a podobně.
- **implementační fáze** - v této fázi se již vytváří samotná aplikace. Probíhají testy správnosti jednotlivých funkcí, či komplikovanějších celků. Cílem těchto testů je především zamezit chybám v kódu.
- **dokončovací fáze** - aplikace je testována jako celek. Provádí se testovací scénáře, kontrola s požadavky zadavatele a podobně. Při těchto testech jsou často přítomni i zadavatelé projektu.

1.4 Testování programu

Cílem každého testování programu je jeho zdokonalení v oblasti dané problematiky. Testy jsou prováděny za účelem nalezení problémů a chyb, po jejichž odstranění dojde ke zdokonalení programu. Proto je testování směřováno na ty oblasti, kde je předpokládáno nalezení velkého množství chyb. Do testů by mělo být zahrnuto co možná nejvíce situací, které mohou při chodu programu nastat. Proto se doporučuje, aby se na testování podílel někdo, kdo není zainteresovaný do realizace programu. Pro programátora je zpravidla obtížné získat nad problémem potřebný nadhled.

Shrnutí

Shrnutí přeloženo z [19]

- Testování je proces průchodu programem s úmyslem nalézt chyby.
- Testování je více úspěšné, když není prováděno vývojáři.
- Dobrý test je takový, který má vysokou pravděpodobnost nalezení dosud neznámé chyby.
- Úspěšný test je takový, který nalezl dosud neznámé chyby.
- Úspěšné testování zahrnuje pečlivé definování vstupů, stejně jako výstupů.
- Úspěšné testování zahrnuje pečlivé studování výsledků.

1.5 Testování dat

Pokud program využívá k výpočtům zdrojová data, je vhodné je také testovat. Informace, které program zpracovává, mohou způsobit nesprávný výstup, či dokonce chod aplikace. Samotné testování lze rozdělit na dvě kategorie, testování konzistence dat a testování kvality dat.

Konzistence dat představuje logickou strukturu dat. Testování se tedy zaměřuje na propojení dat definovaných ve specifikacích. Popis a specifikace dat spolu s potřebami aplikace pro dané struktury vytváří sadu scénářů a testů, které je možné použít pro samotné testování.

Po testování struktury dat je žádoucí také testovat i správnost informací obsažených v datech. Pod kvalitou dat si lze představit mnoho druhů testů. Podstatné je vždy zaměřovat testy na požadavky aplikace a uživatelů, aby výstupy programů byly dle očekávání. Čím více je brán ohled na testování a ověření například geografických dat, tím více se může uživatel spolehnout na program, který tyto informace zpracovává.

Kapitola 2

Popis problému

Pro plánovač tras je důležité mít vstupní data úplná, správná a co nejpřesnější. Proto nástroj musí testovat co možná nejširší spektrum možných chyb a nesrovnalostí, které by mohly způsobit problémy při výpočtu nebo nesprávný výsledek plánovače.

2.1 Generalizovaný časově závislý graf (GTD) graf

Následující sekce je převzata ze zdroje [14].

Generalizovaný časově závislý graf (Generalised Time-Dependent Graph), dále jen GTD graf, je grafová struktura spojující grafy pro silniční síť a PT (Public Transport) typy hromadné dopravy. GTD graf využívá konstantní časy (čas potřebný pro přesun mezi dvěma zastávkami je konstanta na každé stanici) pro reprezentaci časově závislého grafu.

GTD graf se skládá ze 3 struktur:

- časově závislý graf G^T pro síť veřejné dopravy PT
- graf silniční sítě G^N obsahující chodníky, cyklostezky a silnice
- spojnice grafů D, která spojuje G^T a G^N

2.1.1 Časově závislý graf

Pro model sítě hromadné dopravy (jako jsou například autobusy, tramvaje, metro) se využívá časově závislý graf $G^T = (V^T, E^T, \rho^T)$ s konstantními časy přesunu. Je dáno S jako set vrcholů zastávek, které korespondují s existujícími vrcholy v síti hromadné dopravy. Jedna stanice může být spojena jednou či více cestami (linkami) dopravy. Linka je definována souborem PT tras, které jsou známy ve veřejné dopravě pod stejným označením. Mějme n jako počet linek spojených s danou stanicí $u \in S$, pak máme n linkových vrcholů $R_u = r_1^u, ..., r_n^u$, tady pro každou linku jeden vrchol. Vrcholy jsou virtuální vrcholy bez přímého spojení s reálnou dopravní sítí. Tyto vrcholy jsou využity k modelu konstantních časů pro přesun. Bez těchto vrcholů by nebylo možné vytvořit nenulový čas přesunu mezi různými linkami na stejné stanici. Množina všech vrcholů tohoto typu je definována jako $R = \cup_{u \in S} R_u$. Množina vrcholů V^T časově závislého grafu G^T je poté definována jako $V^T = S \cup R$.

Množina hran E^T grafu G^T je definována jako $E^T = A \cup B \cup C$ kde A je množina hran mezi linkou a vrcholem stanice, B je množina hran mezi vrcholy stanic v dané lince a C je množina hran mezi vrcholy stejné linky. Hrany $(v, w) \in A \cup B$ se nazývají transportní hrany. Formální zápis množin:

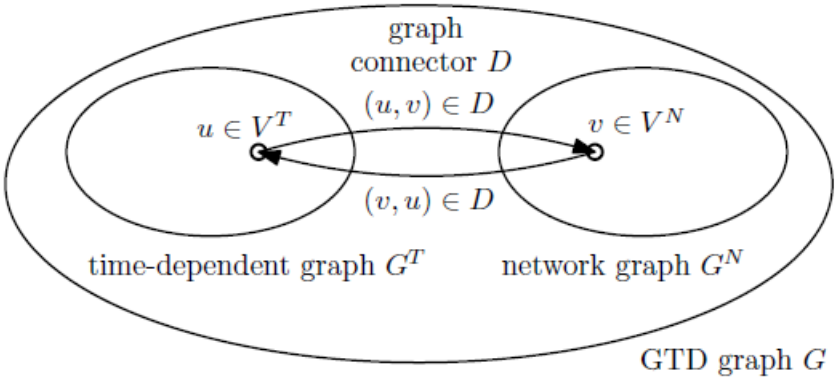
$A = \cup_{u \in S}\{(r^u, u) | r^u \in R_u\}$

$B = \cup_{u \in S}\{(u, r^u) | r^u \in R_u\}$

$C = \cup_{u,v \in S}\{(r^u, r^v) | r^u \in R_u \wedge r^v \in R_v\}$ kde r^u a r^v jsou součástí jedné linky (trasy)

Funkce pro získání jednotlivých časů pro danou linku je definována $f^{'}_{(v,w)} : N- > N$ pro každou hranu $(v, w) \in C$ jako $f^{'}_{(v,w)} := t^{'}$, kde t je odjezdový čas z vrcholu v a $t^{'} \geq t$ je nejbližší možný příjezd na vrcholu w. Předpokládá se, že předjíždění dopravních prostředků stejné linky není na stejném úseku linky možné. To znamená, že nejbližší čas příjezdu na daný vrchol r^w_j koresponduje s nejbližším možným příjezdem z nejbližšího možného výchozího vrcholu r^v_i. Mějme funkci g_v, která vrací transportní čas jako konstantu na stanici v. Pro příklad v Obrázku 2.1, přesun z linkového vrcholu r^v_0 do linkového vrcholu r^v_1 a obráceně trvá čas g_v. Potom trvání transportu $\rho^T_{(v,w)} : N- > N$ překročení hrany $(v, w) \in E^T$ z v v odjezdový čas t je definován jako

$$\rho^T_{(v,w)}(t) := \begin{cases} 0 & if (v, w) \in A \\ g_v & if (v, w) \in B \\ f^{'}_{(v,w)}(t) - t & if (v, w) \in C \end{cases}$$



Obrázek 2.1: Množiny GTD grafu

2.1.2 Graf silniční sítě

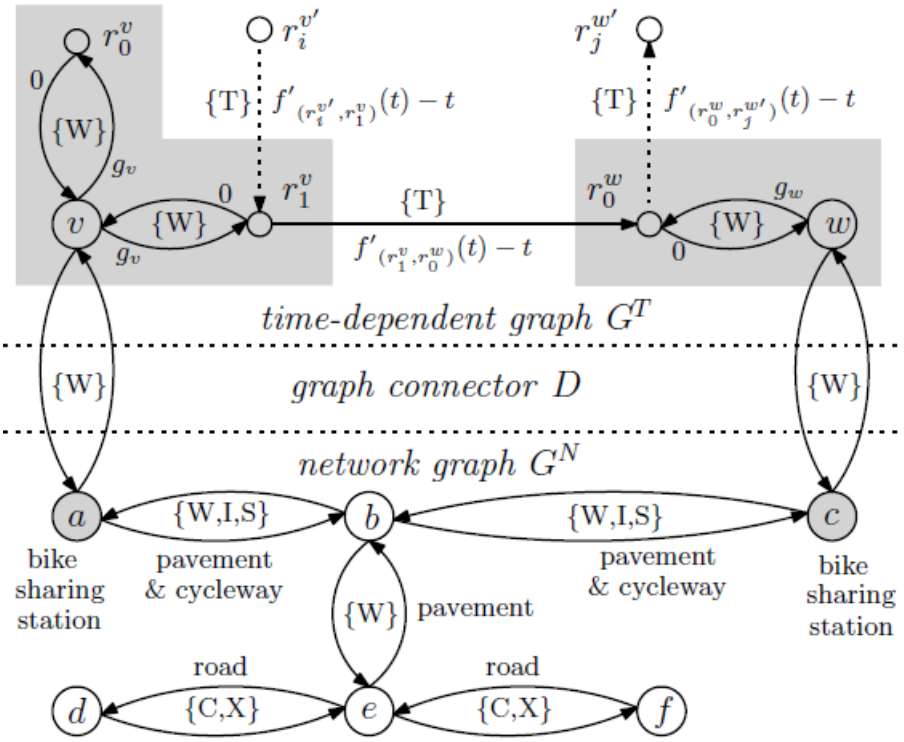
Model sítě pro individuální módy dopravy (chůze, kolo, půjčené kolo, auto) a transportu na požádání (taxi) definovaný jako vážený orientovaný graf, kde množina vrcholů V^N reprezentuje uzly a množina hran E^N představuje cesty, chodníky a cyklostezky. Délka každé hrany $(v, w) \in E^N$ je dána váženou funkcí $\rho^N : E^N - > R^+_0$.

2.1.3 Spojení grafů

Pro plánování cest za využití kombinace dopravy individuální, na požádání a hromadné dopravy, musí být časově závislý graf G^T a grafu silniční sítě G^N propojeny. Je dána $\theta : S -> P(V^N)$ představující mapování, které spojuje každý vrchol stanice $v \in S$ a množinu vrcholů $\theta(v) \in P(V^N)$ ze silniční sítě. Pro stanice metra a velké PT stanice, mapování přiřadí množinu korespondujících vstupů ze silniční sítě grafu G^N. Pro ostatní PT stanice mapování přiřadí nejbližší chodníkový vrchol ze silniční dopravy grafu G^N. Poté spojnice grafů D grafů G^T a G^N je definována jako množina spojujících hran:

$D = \{(v, w)|(v \in S \wedge w \in \theta(v)) \vee (v \in \theta(w) \wedge w \in S)\}$

Délka v metrech $\rho_d((v, w)) = |v, w|$ je přiřazena každé hraně $(v, w) \in D$ (použita Euklidovská vzdálenost mezi v a w).



Obrázek 2.2: Ukázka GTD grafu

2.2 Typy Chyb

2.2.1 Chybějící informace

Jedna ze základních chyb, která se může nacházet ve vstupních datech, jsou chybějící informace. Pokud v datech chybí spoj hromadné dopravy, či dopravní uzel, plánovač naplánuje cestu mnohem delší, či s časovým posunem.

2.2.2 Nesprávné informace

Pokud jsou všechny informace v datech úplné, nemusí být vždy správné a aktuální. Nesprávné GPS souřadnice, či nezanesení informace o jednosměrné cestě, také způsobí potíže pro uživatele, který pojede po naplánované trase. Nesprávné jízdní řády mohou například způsobit, že plánovač vyhodnotí přesun z jednoho konce města na druhý za pár sekund a podobně.

2.3 Formalizace problému

Aplikace obsahuje dané vstupy a preferované výstupy v závislosti na požadavcích jednotlivých testovacích metod.

2.3.1 Popis problému

Ke zpracování a kontrole dat dostaneme na vstupu graf $G = (V, E)$, kde V jsou vrcholy daného grafu a $E \subseteq \{(u, v)|(u, v \in V) \wedge (u \neq v)\}$ je množina hran tohoto grafu. Nad vstupním grafem G spustíme testovací metody $M = \{M_1, M_2, .., M_n\}$, kde M je množina všech testovacích metod, které se mají spustit. Každá metoda M_i obsahuje všechny potřebné parametry k dané metodě. n je počet testovacích metod, které mají být spuštěny programem. Spolu se všemi metodami jsou součástí vstupních informací ještě doplňující informace jako datum, popis testu a podobně. Tyto informace jsou zahrnuty v množině $P = \{a, b, c, .., x\}$, kde a až x představují ony dodatečné informace.

Problém, který má být vyřešen, je tedy dvojice $P = (G, N)$, kde G je vstupní graf a $N = (P, M)$ je dvojice vstupních parametrů, která obsahuje 2 množiny P a M popsané výše.

Řešením tohoto problému je pětice $M = (I, E, W, O, P)$, kde jednotlivé množiny představují:

- I - množina obecných informací (popis, název vstupního grafu, shrnutí a podobně)
- E - množina nalezených chyb (přehled chyb v textové podobě, který program objevil)
- W - množina nalezených nesrovnalostí (přehled vad v textové podobě, které program objevil a nedokáže s jistotou určit, zda se jedná o chybu - přenecháno na posouzení odborníka)
- O - množina nalezených požadovaných informací (přehled úspěšně zkontrolovaných vlastností, či částí vstupních dat)
- P - množina parametrů pro případné další využití (přehled parametrů, které jsou důležité pro případně další porovnávání, či výpočty)

2.3.2 Vnášení chyb

Pro testování samotných metod zavádíme do dat umělé chyby a testujeme, zda chyby byly programem nalezeny. Pro tento případ musíme mít připravená data s chybami. Mezi chyby, které jsou vnášeny do dat patří například odstranění uzlů či hran, změna GPS souřadnic, změna spojení mezi dvěma částmi grafu, změna maximální povolené rychlosti, a jiné.

- **Vstup:** graf G, $G = (V, E)$, kde V jsou vrcholy a E hrany daného grafu.(Součástí hran a vrcholů mohou být dodatečné informace v závislosti na typu dat)
- **Výstup:** graf G^1, $G^1 = (V^1, E^1), V \neq V^1 \vee E \neq E^1$, kde V^1 jsou vrcholy a E^1 hrany daného grafu. Hrany a vrcholu nejsou totožné s původními vstupními, neboť obsahují zanesené chyby.

Kapitola 3

Návrh řešení

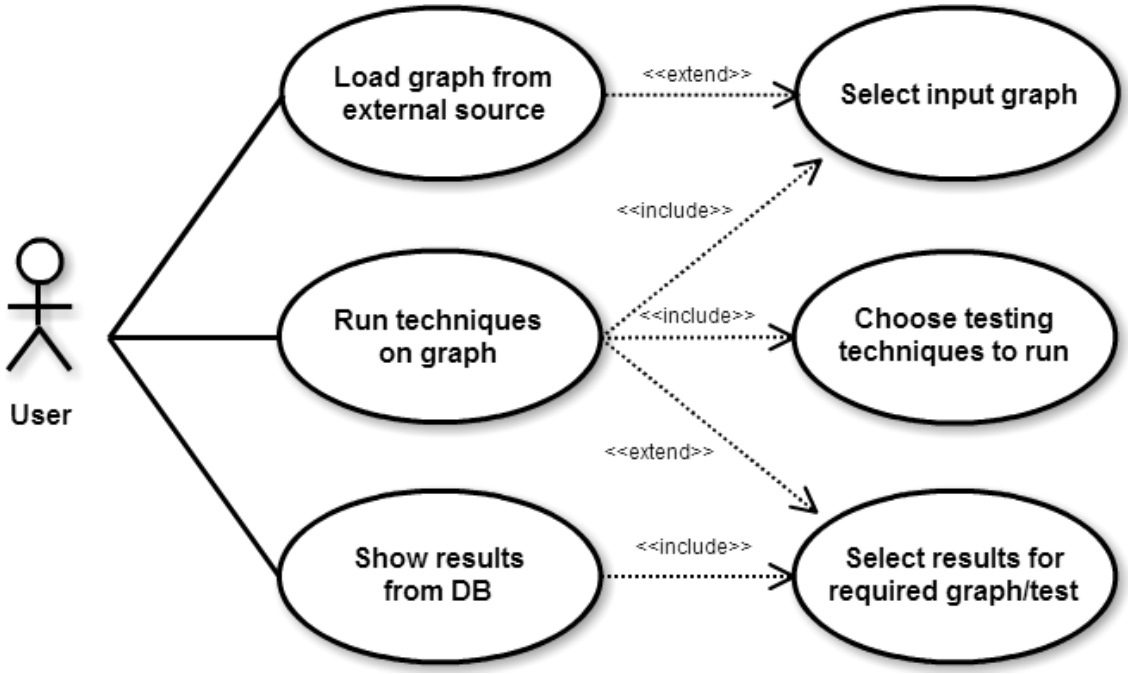
Před implementací byl vytvořen návrh postupu pro vytvoření požadovaného nástroje. Tento náhled na danou problematiku má sloužit jako plán a kostra pro samotnou implementaci. Má také umožnit, aby jednotlivé části byly logicky odděleny a mohly být snáze rozvíjeny v budoucnu.

3.1 Popis systému

Testovací nástroj má sloužit ke kontrole dat pro plánovače tras. Základním vstupem jsou data obsahující síť dopravy spolu s informacemi o ní. Uživatel si zvolí testovací metody, které chce spustit nad danými daty. Pokud testovací techniky vyžadují vstupní parametry, uživatel je zadá a spustí testování. Po proběhnutí požadovaných testů program na výstupu zobrazí informace o průběhu testovaní a nalezených chybách, či nesrovnalostech. Jednotlivé výsledky testů se uloží do databáze.

Pro základní přehled funkcionality slouží use case 3.1 zobrazující proces z pohledu uživatele, který znázorňuje základní akce:

- **Load graph from external source** - stáhnutí datového setu pro testování na souborový systém
- **Run techniques on graph** - spuštění testovacích technik nad danými daty
  - Select input graph - uživatel před testováním zvolí data, která chce testovat. Předpokladem je, že datový set má již připravený a stažený na souborovém systému
  - Choose testing techniques to run - uživatel určí, které testovací metody mají být spuštěny a s jakými parametry
- **Show results from DB** - uživatel je schopen zobrazit si výsledky a informace o proběhlých testech. Tyto informace jsou uloženy v databázi
  - Select results for required graph/test - pro zobrazení dat si uživatel musí zvolit, které informace chce z databáze vidět



Obrázek 3.1: Use case znázorňující základní funkcionalitu programu

3.2 Architektura systému

Systém byl rozdělen na jednotlivé celky,viz obrázek 3.2, které jsou níže popsány ve větším detailu.

3.2.1 Server s daty pro testování

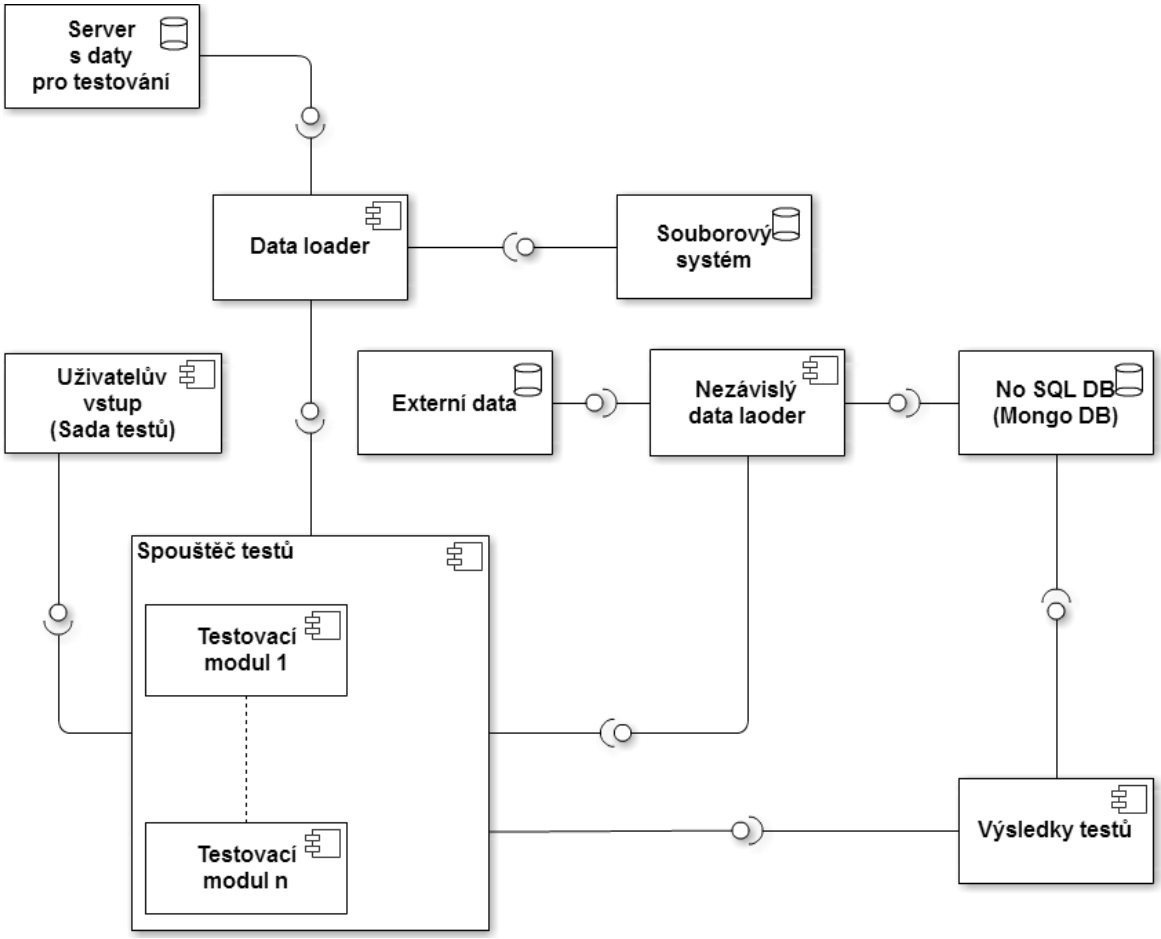
Server, který poskytuje data, která budou zpracovávána systémem. Z tohoto serveru se stahují data na lokální adresář systému, odkud se dále zpracovávají.

3.2.2 Data loader

Modul sloužící ke správě vstupních dat pro plánovač tras. Tato část aplikace zajišťuje stahování dat z předem nastaveného zdroje. Následně takto získaná data označí danou verzí, datem pořízení a uloží je na souborový systém k ostatním uloženým datovým souborům. Tyto uložené soubory také dodává do jiných částí systému pro jejich zpracování.

3.2.3 Spouštěč testů

Spouštěč testů je hlavní část celého programu. Jeho podstata je v tom, že přijímá strukturu testovacích scénářů (Sada testů). Po přijmutí všech potřebných informací se dotáže o data z data loaderu a s jejich pomocí spustí požadované operace popsané ve vstupní struktuře.



Obrázek 3.2: Schéma architektury systému

Výstupem daného modulu je objekt popisující výsledek celého testování spolu s dílčími výsledky jednotlivých testovacích modulů. Tento objekt je dále posílán do Výsledkového logu k následnému uložení do databáze.

3.2.4 Sada Testů

Struktura popisující průběh testování. Obsahuje informace o datech, která mají být použita k testování. Dále seznam všech testovacích modulů, které se mají spustit.

3.2.5 Testovací modul

Modul obsahuje všechny informace potřebné k tomu, aby se daná testovací technika spustila. Součástí je název třídy, která má být spuštěna, spolu se všemi potřebnými parametry a údaji. Výstupem modulu je souhrn poznatků a výsledků daného testu spolu s parametry, které mohou mít význam pro uživatele, či případné další použití v nástroji.

3.2.6 Nezávislý data loader

Pro porovnávání dat z externích zdrojů slouží nezávislý nahrávač dat. Pokud testovací modul využívá ve svých výpočtech informace z jiných systémů, dotáže se tohoto modulu. Potřebné informace pro dosažení dat jsou umístěny v parametrech daného testovacího modulu.

3.2.7 Výsledky testů

Tento modul sloužící pro komunikaci s Mongo databází, kam se ukládají výsledky testů, informace o vygenerovaných chybách, atd.

3.2.8 No SQL Database

Vzhledem k tomu, že výsledky testů budou definované datové struktury, které je žádoucí ukládat jako celé objekty, používá se struktura na ukládání dat ve formátu JSON. Pro tento účel je zvolena databáze MongoDB, která je schopná ukládat data například ve strukturách podobným souborovému systému.

Hlavní kolekce v databázi:

- Graphs - seznam grafů, které byly aplikací použity. Seznam slouží pro přehled dat v databázi. Pokud je v seznamu záznam o grafu, existuje k němu i kolekce s údaji o testech
- <ID grafu> - jednotlivé výsledky a údaje k danému grafu se ukládají v kolekci, která se označuje ID grafu
- Stations<City> - načtené informace o GPS souřadnicích ke stanicím hromadné dopravy z externího zdroje
- Suites - uložené sady testů uživatele, pro budoucí použití

3.2.9 Souborový systém

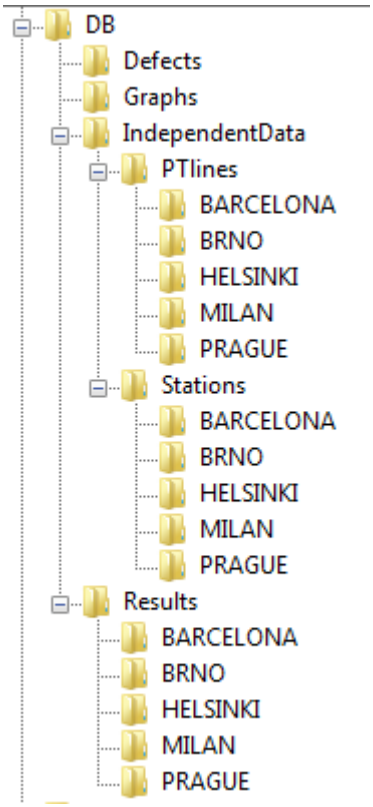
Pro ukládání grafů a serializovaných objektů slouží struktura na souborovém systému. Toto řešení je zvoleno kvůli struktuře a velikosti dat (vstupní grafy mají velikost i v řádech stovek MB). Struktura adresářů, se kterou aplikace počítá, viz obrázek 3.3.

Popis hlavních adresářů:

- Defects - uložené informace o uměle zavedených chybách (včetně konkrétních změn pro případné porovnávání)
- Graphs - vstupní grafy nad kterými se pouští testovací metody
- IndependentData - data, které slouží k porovnání v příslušných testovacích metodách. Tyto informace pochází z externích zdrojů
  - PTLines - seznam linek hromadné dopravy v příslušném městě

  - Stations - seznam stanic hromadné dopravy v příslušném městě

- Results - pokud výstupní JSON je větší než maximální povolená velikost pro objekt v Mongo DB (16 MB), uloží se výsledky do tohoto adresáře, aby nebyly ztraceny. Do tohoto adresáře se také uloží, pokud se výsledek nepodaří uložit do Mongo DB i z jiného důvodu (nedostupnost a jiné důvody)



Obrázek 3.3: Struktura dat v souborovém systému

3.2.10 Generátor chyb

Pro testování našich implementovaných testovacích metod je zapotřebí mít zkreslená, či špatná vstupní data. Pro tento účel slouží generátor chyb. Generátor si načte vstupní data přes data loader a provede změny jako například vymazání, přidání, či úpravu údajů a podobně. Takto upravená data uloží opět pomocí data loaderu do souborového systému. Spolu s tím uloží také záznam o provedených chybách do databáze.

Nad upravenými daty se poté mohou spouštět testovací moduly a jednotlivé výstupy budou porovnávány se záznamem provedených chyb. Chyby by měly být generovány náhodně s přihlédnutím ke specifickým požadavkům testovacích metod (testujeme-li počet módů dopravy, je žádoucí mazat celé módy dopravy, atd.).

Tímto způsobem je možné simulovat případné chyby a zajistit, aby testovací techniky byly schopné tyto chyby rozlišit a upozornit na ně.

3.3 Metody pro hledání datových chyb

Hlavní funkce programu je testování dat. Testovat data lze různými způsoby. Způsoby testování, které jsou využity v programu, jsou popsány v této sekci.

3.3.1 Hlavní rozdělení testovacích scénářů

Testování správnosti dat lze provést mnoha způsoby. V této práci jsou zvoleny dva postupy pro testování. Jedním z nich je testování samotných dat oproti známým faktům a skutečnostem, které musí platit, a druhý způsob využívá porovnání různých verzí (zdrojů) stejných dat.

Testování s jednou datovou strukturou

Jednotlivé testovací moduly provedou výpočty nad vstupními daty. Výsledné hodnoty mohou být napočítány a uloženy pro případné budoucí použití. Pokud jsou známy vstupní parametry ať uživatelem zadané, či obecně známé, validují testovací metody vstupní data a na výstupu označí nalezené chyby, či potvrdí správnost testu.

Porovnání dvou datových struktur

Některé testovací techniky porovnávají dva datové sety. Často se jedná o porovnání dvou verzí dat. Při těchto technikách se určuje hlavní datový set, u kterého se předpokládá, že je správný (s přihlédnutím ke známým chybám) a porovnává se s druhým datovým vstupem. Nesrovnalosti, které nelze jednoznačně určit jako chyby, jsou označovány jako varování a odborník poté sám může posoudit, zda se jedná o chybu či nikoliv.

3.3.2 Testovací metody

Jednotlivé testovací metody lze rozdělit na základě způsobu vyhodnocování výsledků a vstupních parametrů.

Testy konzistence dat

Tyto testy využívají jedna vstupní data, buď bez uživatelových vstupních parametrů, či s nimi, pokud jsou vyžadovány.

Jednotlivé metody:

- **Silně souvislé komponenty** - tato metoda vypočítá počet silně souvislých komponent v grafu a případně porovná s požadovaným počtem, který uživatel zadal. Výslednou hodnotu uloží pro případné jiné techniky testování.
- **Kontrola jednosměrných chodníků** - chodníky musí být vždy obousměrné. Tato metoda zkoumá, zda data odpovídají požadavkům.

- **Kontrola času cesty hromadné dopravy** - metoda zkoumá, zda nejsou v datech nesmyslné informace o rychlosti (například přesun jedné části města na druhý za jednu sekundu).
- **Kladný čas cesty hromadné dopravy** - zkoumá, zda se nelze přemístit v grafu v záporném čase. Tento fakt je zkoumán především spolu s časovými plány dopravních prostředků (jízdní řády a podobně).
- **Nulový čas cesty hromadné dopravy** - tento test zjišťuje, zda se nelze přemístit v grafu na nenulovou vzdálenost za nulový čas.
- **Typy hromadné dopravy** - metoda porovnává módy dopravy a zjišťuje, zda nějaký nechybí, či nepřebývá.
- **Počet hran a vrcholů** - metoda porovnává počet hran a uzlů v grafu.
- **Počet hran a vrcholů dle typu** - metoda porovnává počet hran a uzlů dle typu v grafu.
- **Maximální povolená rychlost** - metoda zkoumá, zda maximální povolená rychlost v datech není mimo daný rozsah.
- **Kontrola spojení stanic s pěšími cestami** - metoda zkoumá zda jsou stanice hromadné dopravy spojené s pěší částí grafu.

Porovnání plánovacích dat

Všechny vstupní parametry nemusí uživatel zadávat sám, ale mohou být stahovány z nezávislých zdrojů informací. U tohoto testování musí být brán zřetel na to, jak moc důvěřujeme těmto externím vstupním datům.

Jednotlivé metody:

- **Kontrola GPS souřadnic stanic hromadné dopravy** - metoda porovnává GPS souřadnice stanic hromadné dopravy s externím zdrojem.
- **Kontrola linek hromadné dopravy** - porovnání seznam linek hromadné dopravy s externím zdrojem.
- **Kontrola názvu stanic hromadné dopravy** - porovnání názvů stanic hromadné dopravy s externím zdrojem.

Porovnání dvou verzí plánovacích dat

Všechny vypočtené hodnoty z výše uvedených metod mohou být využity při porovnání dvou verzí dat. Tato technika porovnává napočítané sumy, jména stanic, atd. Výstupem je poté seznam nalezených chyb a varování, které by měl uživatel ověřit. Pro tyto účely musí zvýše zmiňované metody ukládat dostatek informací do databáze, aby složité výpočty nemusely být vypočítávány znovu.

Kapitola 4

Implementace

Kapitola implementace popisuje výsledné řešení problému popsaném v sekci 2. Dále popisuje implementaci jednotlivých metod a použitých technik z 3 s přihlédnutím k zajímavostem a komplikacím, které se při implementaci objevily.

4.1 Vstupní a výstupní rozhraní

Pro vstupy a výstupy našeho systému jsou definovány struktury ve formátu JSON. Základní údaje o těchto strukturách jsou popsány v následujících sekcích.

4.1.1 Testovací modul - vstup a výstup

Vstup pro testovací modul má název třídy, která se má spustit, neboť sada testů je obecná a volá metody dle jejich názvů tříd. Dále obsahuje vstupní parametry potřebné pro danou testovací metodu. (obr. 4.1)

Výstup je poté objekt obsahují název metody, status s výsledkem vyhodnocení, shrnutí a jednotlivé množiny napočítaných hodnot, nalezených chyb, atd. (Obrázek 4.2)

Možný výstupní status:

- SUCCESS - metoda skončí dle očekávání bez nalezení chyb
- EXCEPTION - metoda nemohla proběhnout až do konce kvůli špatným vstupným parametrům, nestandardním vstupním grafem, neexistující externím zdrojům dat a podobně
- WARNING - status při nalezení nesrovnalostí, které nelze jednoznačně určit jako chyby
- FAILED - metoda nalezla chyby či nesrovnalosti nutné k bližšímu prozkoumání

Rozdělení výstupních informací a parametrů:

- resultParams - hlavní parametry a údaje spočtené danou metodou. Tyto hodnoty mohou sloužit jak pro uživatele pro přehled, tak především pro možnost porovnat různé verze dat aniž by metoda musela být spouštěna znovu nad stejnými daty.

- resultOK - hlavní informace o porovnávaných hodnotách, které proběhli úspěšně
- resultWARN - informace o nesrovnalostech v datech, které je žádoucí blíže prozkoumat, ale z podstaty věci nelze s určitostí označit nesrovnalost za chybu
- resultERROR - informace o nalezených chybách

```
{
  "name": "cz.agents.cvut.testModule.modules.strongComponent.GraphStrongComponent",
  "parameters": {
    "name": "GraphStrongComponent",
    "params": {
      "componentsN": "600"
    }
  }
}
```

Obrázek 4.1: Vstupní JSON pro metodu porovnávání počtu silně souvislých komponent

```
{
  "name": "cz.agents.cvut.testModule.modules.strongComponent.GraphStrongComponent",
  "status": "FAILED",
  "summary": "Graph Strong Component module successfullly completed /n Components: 2492",
  "resultParams": {
    "components": "2492"
  },
  "resultOK": {},
  "resultWARN": {},
  "resultERROR": {
    "components": "Components number not as requested: 600 actual: 2492"
  }
}
```

Obrázek 4.2: Výstupní JSON pro metodu porovnávání počtu silně souvislých komponent

4.1.2 Sada testů

Sada testů obsahuje všechny vstupní informace k testovacím modulům, které se mají spustit. Dále obsahuje informace o testovacích datech.(obr. 4.3)

Výstupem je poté souhrn jednotlivých výstupů z modulů. (obr. 4.4)

4.1.3 Popis vygenerovaných chyb

Pro uložení informací k vygenerovaným chybám nad daty slouží JSON, který je uložen v databázi pro případné porovnávání, zda byly nalezeny zanesené chyby. (obr. 4.5). Dále je uložen objekt na souborový systém, který obsahuje detaily o konkrétních provedených změnách pro případné porovnávání.

```
{ "name": "Simo test suite",
  "graphID": "Brno_1.0.0_GOOD_2016-1-13_12-50",
  "testModules": [
    { "name": "cz.agents.cvut.testModule.modules.strongComponent.GraphStrongComponent",
      "parameters": {
        "name": "GraphStrongComponent",
        "params": {
          "componentsN": "600"
        }}},
    { "name": "cz.agents.cvut.testModule.modules.onewayPavements.GraphConstraintOnewayPavements",
      "parameters": {
        "params": {
        }}},
    { "name": "cz.agents.cvut.testModule.modules.checkZeroTravelTime.ZeroTravelTime",
      "parameters": {
        "params": {
          "detailInfo": "Y"
        }},
    { "name": "cz.agents.cvut.testModule.modules.numberOfNaE.NumberOfNodesAndEdges",
      "parameters": {
        "name": "NumberOfNodesAndEdges",
        "params": {
          "edgesN": "30000",
          "nodesN": "20000",
          "tolerance": "10"
        }},
    { "name": "cz.agents.cvut.testModule.modules.numberOfNaEModes.NumberOfNodesAndEdgesModes",
      "parameters": {
        "params": {
          "comparedGraphID": "Praha_1.0.0_GOOD_2016-1-13_12-59"
        }},
    { "name": "cz.agents.cvut.testModule.modules.travelSpeed.GraphConstraintTravelSpeed",
      "parameters": {
        "name": "GraphConstraintTravelSpeed",
        "params": {
          "KMPHfrom": "0",
          "KMPHto": "50"
        }}}
  ]
}
```


Obrázek 4.3: Ukázka vstupního JSON pro sadu testů

```
{ "name": "Suite result example",
  "date": 1460303085518,
  "modulesResults": [
    { "name": "cz.agents.cvut.testModule.modules.strongComponent.GraphStrongComponent",
      "status": "FAILED",
      "summary": "Graph Strong Component module successfullly completed /n Components: 2492",
      "resultParams": {
        "components": "2492"
      },
      "resultOK": {},
      "resultWARN": {},
      "resultERROR": {
        "components": "Components number not as requested: 600 actual: 2492"
      }},
    { "name": "cz.agents.cvut.testModule.modules.checkTransportTime.TransportTimesCheck",
      "status": "FAILED",
      "summary": "GBus travel time check processed with exception. Data has bad travel time records.",
      "resultParams": {
        "satisfied": "false"
      },
      "resultOK": {},
      "resultWARN": {},
      "resultERROR": {
        "comparedGraph": "graph for compare not found",
        "busTravelTimes": "Bad travel times found. Number of them: 342"
      }},
    { "name": "cz.agents.cvut.testModule.modules.numberOfNaE.NumberOfNodesAndEdges",
      "status": "FAILED",
      "summary": "Test failed. For more details please see errors section",
      "resultParams": {
        "edgesN": "194004",
        "nodesNp": "63029",
        "edgesNp": "150000",
        "nodesN": "63029",
        "tolerance": "10"
      },
      "resultOK": {
        "nodesN": "Number of nodes found: 63029 required: 63029"
      },
      "resultWARN": {},
      "resultERROR": {
        "edgesN": "Number of edges found: 194004 required: 150000"
      }}
  ]
}
```


Obrázek 4.4: Ukázka výstupního JSON pro sadu testů

```
{
  "localGraphID": "Praha_1.0.0_BAD_2015-8-9_2-8",
  "defects": [
    {
      "name": "Remove random 300 nodes.",
      "params": {
        "deletedNodes": "300"
      },
      "type": "DELETE",
      "summary": "Sum check success. All nodes was deleted."
    }
  ]
}
```


Obrázek 4.5: Ukázka JSON pro vygenerovane chyby

4.2 Testovací metody

Následující sekce popisují implementované testovací metody.

4.2.1 Silně souvislé komponenty

Vstupními daty pro testovací techniky jsou vždy grafy. Tato metoda spočítá pomocí algoritmu počet silně souvislých komponent v grafu a porovná ho s předpokládaným počtem, který je součástí vstupních parametrů spolu s procentuální tolerancí, která může být akceptována pro uživatele. Popis logiky metody viz algoritmus 1.

GTDgraf := vstupní graf s daty;
predpokladanyPocetKomponent := vstupní údaj;
tolerance := vstupní údaj;
errorLog := část výstupního logu pro nalezené chyby;
okLog := část výstupního logu pro nalezené informace, které nejsou označeny jako chyby;
pocetKomponent := počet silně souvislých komponent ve vstupním grafu;
if $|pocetKomponent - predpokladanyPocetKomponent| < (predpokladanyPocetKomponent * tolerance)$ **then**
  okLog := okLog + "Počet komponent je v toleranci $predpokladanyPocetKomponent/pocetKomponent$";
else
  errorLog := errorLog + "Počet komponent není v toleranci $predpokladanyPocetKomponent/pocetKomponent$";
end
Result: Předpokládaný a skutečný počet silně souvislých komponent

Algoritmus 1: Kontrola počtu silně souvislých komponent

4.2.2 Kontrola jednosměrných chodníků

Vrcholy a hrany vstupního grafu jsou různého typu a obsahují různé informace. Tato metoda kontroluje síť chodníků v grafu. U pěší sítě se předpokládá, že člověk se může

pohybovat obousměrně. Vzhledem k tomu, že hrany v grafu jsou orientované, neměla by nastat situace, že v grafu bude 'pěší' hrana z vrcholu A do vrcholu B a zároveň nebude existovat 'pěší' hrana z B do A. Algoritmus projde všechny hrany pro pěší chůzi a hledá k nim hranu v opačném směru. Popis logiky metody viz algoritmus 2.

GTDgraf := vstupní graf s daty;
errorLog := část výstupního logu pro nalezené chyby;
for *všechny hrany typu WALK h z GTDgraf* **do**
  pocatecniVrchol := vrchol odkud směřuje hraha h;
  koncovyVrchol := vrchol kam směřuje hrana h;
  opacnaHrana := hrana vedoucí z koncovyVrchol do pocatecniVrchol;
  if *opacnaHrana není 'pěší' hrana* **then**
    errorLog := errorLog + "Jednosměrná cesta h nalezena.";
  end
end
Result: Počet nalezených jednosměrných cest a jejich výpis

Algoritmus 2: Jednosměrné WALK cesty

4.2.3 Typy hromadné dopravy

Vstupní graf obsahuje více typů hromadné dopravy. Metoda dostane na vstupu počet očekávaných typů dopravy. Pokud jsou známé všechny požadované typy dopravy, může být pro přesnější porovnání dodán seznam těchto typů. Následně se projdou hrany a vrcholy grafu pro nalezení všech typů dopravy. Nalezené módy dopravy jsou porovnány se vstupními parametry a výsledek porovnání je zaznamenán do výstupního logu. Popis logiky metody viz algoritmus 3.

GTDgraf := vstupní graf s daty;
modyDopravyPredp := předpokládané typy hromadné dopravy;
modyDopravy := mody dopravy v grafu GTDgraf;
errorLog := část výstupního logu pro nalezené chyby;
for *každý mód* $m \in modyDopravyPredp$ **do**
  if $m \notin modyDopravy$ **then**
    errorLog := errorLog + "Mód dopravy m není v grafu";
  end
end
Result: Report o nenalezených módech dopravy

Algoritmus 3: Typy hromadné dopravy

4.2.4 Kladný čas cesty hromadné dopravy

Čas potřebný k cestě nesmí vracet při výpočtu záporné hodnoty. Metoda prochází záznamy o časech jednotlivých cest a kontroluje, zda hodnoty jsou kladné.

Výpočet je rozdělen podle dvou hlavních typů hran. Pro hrany hromadné dopravy, které obsahují jízdní řády, se počítají časy na hranách přes všechny dostupné jízdní řády. Pro nalezení všech dostupných řešení je tedy nutné posouvat také čas, kde se má daná hodnota na hraně počítat. Druhým typem hran jsou všechny ostatní, u kterých stačí výpočet pouze

s jedním časem, neboť výpočet využívá pouze délku dané hrany a neobsahuje žádné časově závislé informace. Popis logiky metody viz algoritmus 4.

```
GTDgraf := vstupní graf s daty;
errorLog := část výstupního logu pro nalezené chyby;
for všechny hrany hromadné dopravy hD z GTDgraf do
    while existuje odjezd v jízdním řádu pro hD do
        cas := příjezd - odjezd;
        if cas < 0 then
            errorLog := errorLog + "Negativní čas spočítán pro hranu hD pro příjezd
             , odjezd";
        end
    end
end
for všechny ostatní hrany (ne hromadné dopravy) hO z GTDgraf do
    cas := čas potřebný pro překonání hrany hO;
    if cas < 0 then
        errorLog := errorLog + "Negativní čas spočítán pro hranu h0 ";
    end
end
Result: Soupis hran se záporným časem pro přesun
```

Algoritmus 4: Kladný čas pro přesun

4.2.5 Kontrola času cesty hromadné dopravy

Pomocí předchozích metod se zkontroluje konzistence dat pro výpočty plánovače. Pomocí této metody se kontroluje čas potřebný k přesunu s ohledem na rychlost přepravy a vzdálenosti, která musí být dosažena.

Pomocí jízdních řádů je vypočten průměrný čas pro cestu potřebnou k překonání dané hrany (pomocí mediánu pro všechny cesty v jízdním řádu). Podle tohoto času a délky hrany je následně zjištěna průměrná rychlost dopravního prostředku. Pokud je tato rychlost větší jak 100 km/h, je tato skutečnost oznámena ve výstupních informacích jako upozornění, neboť se nepředpokládá, že hromadné prostředky jezdí takovými rychlostmi.

Dále se kontrolují jednotlivé časy z jízdních řádů. Kontroluje se, zda odchylka od mediánu je v rozmezí 50% až 150%. Větší odchylky mohou být podezřelé, neboť se předpokládá, že časy v jízdních řádech mezi stejnými stanicemi budou přibližně stejné. Popis logiky metody

viz algoritmus 5.

```
GTDgraf := vstupní graf s daty;
warningLog := část výstupního logu pro nalezené nesrovnalosti;
for všechny hrany hromadné dopravy hD z GTDgraf do
    medianCas := průměrný čas pro danou hranu (pomocí jízdních řádů);
    delkaHrany := délka hrany;
    casPri100 := čas přesunu přes hranu při rychlosti 100 km/h;
    for všechny časy v jízdním řádu do
        casPrejezdu := čas pro přesun z jízdního řádu (příjezd - odjezd);
        if casPrejezdu > casPro100 then
            warningLog := warningLog + "Rychlost hromadného prostředku je větší
             jak 100 km/h pro hranu hD ";
            continue;
        end
        if casPrejezdu < (medianCas * 0.5) || casPrejezdu > (medianCas * 1.5)
         then
            warningLog := warningLog + "Rychlost hromadného prostředku je mimo
             daný rozsah pro hranu hD ";
        end
    end
end
```

Result: Soupis hran podezřelými časy pro přesun

Algoritmus 5: Čas potřebný k přesunu dopravního prostředku

4.2.6 Nulový čas cesty hromadné dopravy

Čas potřebný k přemístění z jednoho místa na druhé nesmí být nulový. Člověk ani dopravní prostředek se nemohou přesunout za nulový čas. Všechny hrany hromadné dopravy jsou kontrolovány na čas potřebný k přesunu.

Implementace je podobná jako u zkoumání kladného času. Zde se kontroluje vypočítaný čas, zda není nulový. Pokud čas potřebný pro přesun je nulový a délka hrany není nulová,

označí program hranu jako chybnou. Popis logiky metody viz algoritmus 6.

```
GTDgraf := vstupní graf s daty;
errorLog := část výstupního logu pro nalezené chyby;
for všechny hrany hromadné dopravy hD z GTDgraf do
    while existuje odjezd v jízdním řádu pro hD do
        cas := příjezd - odjezd;
        delkaHrany := délka hrany hD;
        if cas == 0 && delkaHrany! = 0 then
            errorLog := errorLog + "Nulový čas spočítán pro hranu hD pro příjezd ,
              odjezd";
        end
    end
end
for všechny ostatní hrany (ne hromadné dopravy) hO z GTDgraf do
    cas := čas potřebný pro překonání hrany hO;
    delkaHrany := délka hrany hD;
    if cas == 0 && delkaHrany! = 0 then
        errorLog := errorLog + "Nulový čas spočítán pro hranu h0 ";
    end
end
Result: Soupis hran s nulovým časem pro přesun
```

Algoritmus 6: Nulový čas pro přesun

4.2.7 Kontrola GPS souřadnic stanic hromadné dopravy

Každý vrchol označující stanici hromadné dopravy obsahuje také GPS souřadnice polohy stanice. Souřadnice jsou za pomocí této metody porovnávány s nezávislým zdrojem informací. Vstupem metody je graf obsahující stanice hromadné dopravy a nezávislý zdroj informací, který obsahuje informace o stanicích hromadné dopravy a jejich souřadnicích. Program vypočte vzdálenost souřadnic nalezených v grafu a v nezávislém zdroji. Pokud vzdálenost je větší než povolená odchylka, která je součástí vstupních parametrů metody, je tato stanice zaznamenána jako chyba ve výstupním logu.

Jako externí zdroj informací pro tuto metodu byla zvolena služba od Google. Tato internetová služba vrátí na požadavek, obsahující GPS souřadnice, název stanice, zvolený radius a typy hledaných hromadných doprav, odpověď, zda je stanice ve zvoleném rozsahu. Vzhledem k tomu, že tato služba je zpoplatněná při překročení množství požadavků přes 1000 za den, jsou odpovědi ukládány do databáze. Program nejdříve zkontroluje, zda se již potřebné informace nenachází v databázi a až poté se ptá internetové služby. Popis logiky

metody viz algoritmus 7.

GTDgraf := vstupní graf s daty;
vrcholyHromadneDopravy := všechny vrcholy hromadné dopravy v GTDgraf;
tolerance := tolerance odchylky vzdálenosti v metrech;
errorLog := výstupní log pro nalezené chyby;
ulozeneData := uložené GPS souřadnice v databázi z předešlých testů;
for $v \in vrcholyHromadneDopravy$ **do**
  stanice := název stanice vrcholu v;
  staniceGPS := GPS souřadnice stanice;
  staniceGPSext := GPS souřadnice z externího zdroje;
  if $stanice \in ulozeneData$ **then**
    staniceGPSext := ulozeneData[stanice];
  else
    staniceGPSext := pozadavek na Google;
    ulozeneData := ulozeneData + staniceGPSext;
  end
  vzdalenost := vzdalenost mezi staniceGPS a staniceGPSext;
  if $vzdalenost > tolerance$ **then**
    errorLog := errorLog + stanice;
  end
end
Result: Výstupní JSON obsahující důležité informace a nalezené chyby

Algoritmus 7: Kontrola GPS souřadnic stanic hromadné dopravy

4.2.8 Počet hran a vrcholů

Testovací technika porovnávající počet hran a vrcholů vůči předpokládaným hodnotám. Na vstupu metody je graf, předpokládaný počet hran a uzlů a procentuální tolerance. Vstupní hodnoty jsou porovnány vůči celkovému počtu hran a uzlů grafu. Pokud rozdíly v hodnotách přesahují toleranci, je zaznamenána chyba ve výstupním logu. Popis logiky metody viz algoritmus 8.

GTDgraf := vstupní graf s daty;
predpokladanyPocetVrcholu := vstupní údaj;
predpokladanyPocetHran := vstupní údaj;
tolerance := vstupní údaj;
errorLog := část výstupního logu pro nalezené chyby;
okLog := část výstupního logu pro nalezené informace, které nejsou označeny jako chyby;
pocetVrcholu := počet vrcholů v grafu;
pocetHran := počet hran v grafu;
if $|predpokladanyPocetVrcholu - pocetVrcholu| < (pocetVrcholu * tolerance))$ **then**
  okLog := okLog + "Počet vrcholů je v toleranci";
else
  errorLog := errorLog + "Počet vrcholů není v toleranci";
end
Result: Počet hran a vrcholů

Algoritmus 8: Kontrola názvů stanic hromadné dopravy

4.2.9 Počet hran a vrcholů dle typu

Podobný proces jako metoda Počet hran a vrcholů. V této technice jsou vrcholy a hrany rozděleny na jednotlivé typy. Jsou to typy jako hrany a vrcholy jednotlivých typů hromadné dopravy a ostatních nedopravních kategorií (chodníky a podobné). Pokud rozdíly překročí danou toleranci, jsou nalezené chyby zaznamenány ve výstupním logu.

Nepředpokládá se, že počty všech typů hran a vrcholů budou známé. Proto je tato metoda používána jako porovnání dvou datových souborů (grafů). Pro každý graf jsou vypočítány dané hodnoty a následně porovnány. Popis logiky metody viz algoritmus 9.

GTDgraf := vstupní graf s daty;
n := počet typů hran a vrcholů;
predpokladanyPocetVrcholu[n] := vstupní údaje;
predpokladanyPocetHran[n] := vstupní údaje;
tolerance := vstupní údaj;
errorLog := část výstupního logu pro nalezené chyby;
okLog := část výstupního logu pro nalezené informace, které nejsou označeny jako chyby;
pocetVrcholu[n] := počet vrcholů v grafu dle typu;
pocetHran[n] := počet hran v grafu dle typu;
for $i \in n$ **do**
  if $|predpokladanyPocetVrcholu[i] - pocetVrcholu[i]| < (predpokladanyPocetVrcholu[i] * tolerance))$ **then**
    okLog := okLog + "Počet vrcholů je v toleranci pro typ i";
  else
    errorLog := errorLog + "Počet vrcholů není v toleranci pro typ i";
  end
end
Result: Předpokládaný a skutečný počet hran a vrcholů dle typu

Algoritmus 9: Počet hran a vrcholů dle typu

4.2.10 Kontrola linek hromadné dopravy

Metoda kontroluje označení linek hromadné dopravy nalezené v grafu oproti nezávislému seznamu. Pro zpracování testovací techniky je nutné mít graf s informacemi o hromadné dopravě a nezávislý seznam linek v daném městě. Název linky obsahuje hrana, která spojuje vrcholy typu "GraphStopNode", tedy vrcholy hromadné dopravy. Program prochází skrz vrcholy tohoto typu a hrany, které je spojují. Označení linky z dané hrany je porovnáno vůči nezávislému seznamu. Pokud linka není v seznamu, je tato nenalezená linka zaznamenána do výstupního logu. Během procesu se všechny linky nalezené v grafu ukládají do unikátního seznamu. Tento seznam se porovná ke konci programu s nezávislým seznamem. Linky, které přebývají v nezávislém seznamu (a nejsou obsaženy v grafu), jsou také vypsány ve výstupních informacích.

Vstupní graf obsahuje data vždy pro jedno město a jeho nejbližší okolí. Pro danou oblast je vytvořen seznam linek hromadné dopravy, který je uložen v souborovém systému, odkud

se pro dané město vždy načítá. Popis logiky metody viz algoritmus 10.

GTDgraf := vstupní graf s daty;
seznamLinekExt := nezávislý seznam linek z externího zdroje;
seznamLinekInt := seznam linek z grafu;
errorLog := výstupní log pro nalezené chyby;
for *každý vrchol hromadné dopravy* $v \in GTDgraf$ **do**
  linka := číslo linky vrcholu v;
  if $v \notin seznamLinekExt$ **then**
    errorLog := errorLog + linka;
  end
  seznamLinekInt := seznamLinekInt + linka;
end
for *každá linka* $l \in seznamLinekExt$ **do**
  if $l \notin seznamLinekInt$ **then**
    errorLog := errorLog + l;
  end
end
Result: Výstupní JSON obsahující důležité informace a nalezené chyby

Algoritmus 10: Kontrola linek hromadné doprav

4.2.11 Kontrola názvů stanic hromadné dopravy

Metoda porovnává názvy stanic hromadné dopravy z grafu oproti nezávislému seznamu stanic. Potřebné vstupy metody jsou graf (obsahující názvy stanic) a seznam stanic z nezávislého zdroje. Informaci o názvu stanice obsahuje vrchol grafu type "GraphStopNode", tedy vrchol hromadné dopravy. Program prochází všechny vrcholy tohoto typu a porovnává název stanice oproti nezávislému seznamu. Pokud stanice není v nezávislých datech, je zaznamenána do výstupního logu jako chyba. Během procesu se názvy stanic ukládají do unikátního seznamu, který je opět porovnám oproti externím informacím. Tímto způsobem jsou nalezeny případné stanice, které nejsou v grafu, ale nalézají se v nezávislém seznamu.

Vstupní graf obsahuje data vždy pro jedno město a jeho nejbližší okolí. Pro danou oblast je vytvořen seznam stanic hromadné dopravy, který je uložen v souborovém systému, odkud

se pro dané město vždy načítá. Popis logiky metody viz algoritmus 11.

GTDgraf := vstupní graf s daty;
seznamStanicExt := nezávislý seznam linek z externího zdroje;
seznamStanicInt := seznam linek z grafu;
vrcholyHromadneDopravy := všechny vrcholy hromadné dopravy v GTDgraf;
errorLog := výstupní log pro nalezené chyby;
for $v \in vrcholyHromadneDopravy$ **do**
  stanice := název stanice vrcholu v;
  if $v \notin seznamStanicExt$ **then**
    errorLog := errorLog + stanice;
  end
  seznamStanicInt := seznamStanicInt + stanice;
end
for *každá stanice* $s \in seznamStanicExt$ **do**
  if $s \notin seznamStanicInt$ **then**
    errorLog := errorLog + s;
  end
end
Result: Výstupní JSON obsahující důležité informace a nalezené chyby

Algoritmus 11: Kontrola názvů stanic hromadné dopravy

4.2.12 Maximální povolená rychlost

Součástí hran v grafu je mimo jiné i maximální povolená rychlost pro danou cestu. Uživatel si na vstupu zvolí rozmezí, ve kterém očekává, že se rychlosti mají pohybovat. Metoda projde všechny informace o maximálních rychlostech v grafu a pokud tato rychlost je mimo zvolené rozmezí, zaznamená skutečnost do výstupního logu. Popis logiky metody viz algoritmus 12.

GTDgraf := vstupní graf s daty;
minRychlost := vstupní údaj - minimální možná rychlost;
maxRycholst := vstupní údaj - maximální možná rychlost;
errorLog := část výstupního logu pro nalezené chyby;
okLog := část výstupního logu pro nalezené informace, které nejsou označeny jako chyby;
for *všechny transportní hrany h z GTDgraf* **do**
  rychlostHrany := maximální povolená rychlost hrany h;
  if $rychlostHrany < minRychlost || rychlostHrany > maxRycholst$ **then**
    errorLog := errorLog + "Rychlost není v toleranci $h - rychlostHrany$";
  end
end
Result: Hrany obsahující maximální rychlost mimo daný rozsah

Algoritmus 12: Kontrola maximálních rychlostí

4.2.13 Kontrola spojení stanic s pěšími cestami

Pro plánování tras je zapotřebí správného napojení jednotlivých módů dopravy. Pokud někdo cestuje například tramvají a chce pokračovat dále metrem, předpokládá se, že

z tramvaje vystoupí na chodník a půjde k metru. Pro výpočty podobných i složitějších tras a kombinací, je nutné, aby stanice hromadné dopravy byly dostupné z pěší trasy. Metoda kontroluje, zda se na každou stanici lze v grafu dostat pomocí hrany představující pěší cestu. Popis logiky metody viz algoritmus 13.

```
GTDgraf := vstupní graf s daty;
errorLog := část výstupního logu pro nalezené chyby;
for všechny vrcholy hromadné dopravy v z GTDgraf do
    odchoziHrany := odchozí hrany pro vrchol v;
    prichoziHrany := příchozí hrany pro vrchol v;
    vystupniHrany := hrany vedoucí z koncovyVrchol;
    vrcholSpojenSwalk := ne;
    for hO ∈ odchoziHrany do
        if hO je 'pěší' hrana then
            vrcholSpojenSwalk := ano;
            break
        end
    end
    for hP ∈ prichoziHrany do
        if hP je 'pěší' hrana then
            vrcholSpojenSwalk := ano;
            break
        end
    end
    if vrcholSpojenSwalk == ne then
        errorLog := errorLog + "Vrchol hromadné dopravy v není spojen s pěší
         cestou";
    end
end
Result: Soupis stanic hromadné dopravy nespojených s pěší cestou
```

Algoritmus 13: Propojení stanic s pěší sítí

4.3 Techniky pro generování umělých chyb

Zanášení chyb do grafů simuluje případné chyby, které se mohou objevit v datech. Na vstupu generátoru chyb je zapotřebí graf, který bude upravován. Po zanesení všech požadovaných chyb se nový graf uloží na souborový systém a může být použit pro výpočty stejně jako graf původní. Jednotlivé implementované techniky jsou popsány níže.

4.3.1 Odstranění uzlů hromadné dopravy

Proces odstraní vrcholy hromadné dopravy spolu se souvisejícími hranami, aby graf zůstal konzistentní. Odstraněny budou pouze vrcholy typu "GraphStopNode", tedy vrcholy

pro hromadnou dopravu. Popis logiky techniky viz algoritmus 14.

GTDgraf := vstupní graf s daty;
procento := procento uzlů k vymazání (vstupní parametr);
pocetKvymazani := počet vrcholů určených k vymazaní (procento * celkový počet uzlů hromadné dopravy);
vymazatIDs := unikátní seznam vrcholů určených k vymazání;
while */vymazatIDs/ != pocetKvymazani* **do**
  id := náhodný uzel hromadné dopravy;
  vymazatIDs := vymazatIDs + id;
end
for *každý* $id \in vymazatIDs$ **do**
  hrany := všechny hrany spojené s vrcholem *id*;
  v := vrchol identifikovaný *id*;
  GTDgraf := GTDgraf - hrany;
  GTDgraf := GTDgraf -v;
end
Result: GTD graf bez daného procenta vrcholů

Algoritmus 14: Odstranění uzlů hromadné dopravy

4.3.2 Odstranění běžných uzlů

Obdobný proces jako u odstranění vrcholů hromadné dopravy. Tentokrát se odstraňují vrcholy, které nejsou spojeny s hromadnou dopravou. Popis logiky techniky viz algoritmus 15.

GTDgraf := vstupní graf s daty;
procento := procento uzlů k vymazání (vstupní parametr);
pocetKvymazani := počet vrcholů určených k vymazaní (procento * celkový počet uzlů nepatřící k hromadné dopravě);
vymazatIDs := unikátní seznam vrcholů určených k vymazání;
while */vymazatIDs/ != pocetKvymazani* **do**
  id := náhodný uzel, který nepatří k hromadné dopravě;
  vymazatIDs := vymazatIDs + id);
end
for *každý* $id \in vymazatIDs$ **do**
  hrany := všechny hrany spojené s vrcholem *id*;
  v := vrchol identifikovaný *id*;
  GTDgraf := GTDgraf - hrany;
  GTDgraf := GTDgraf - v;
end
Result: GTD graf bez daného procenta vrcholů

Algoritmus 15: Odstranění běžných uzlů

4.3.3 Odstranění hran hromadné dopravy

Proces odstraní hrany spojující vrcholy hromadné dopravy. Počet odstraněných hran je dán procentem hran na vstupu. Popis logiky techniky viz algoritmus 16.

GTDgraf := vstupní graf s daty;
procento := procento hran k vymazání (vstupní parametr);
pocetKvymazani := počet hran určených k vymazaní (procento * celkový počet hran hromadné dopravy);
vymazatIDs := unikátní seznam hran určených k vymazání;
while */vymazatIDs/ != pocetKvymazani* **do**
  id := náhodná hrana, která patří k hromadné dopravě;
  vymazatIDs := vymazatIDs + id);
end
for *každý id* $\in$ $vymazatIDs$ **do**
  h := hrana identifikovaný *id*;
  GTDgraf := GTDgraf - h;
end
Result: GTD graf bez daného procenta hran

Algoritmus 16: Odstranění hran hromadné dopravy

4.3.4 Odstranění běžných hran

Metoda odstraní hrany, které nejsou hranami hromadné dopravy (chodníky a podobně). Popis logiky techniky viz algoritmus 17.

GTDgraf := vstupní graf s daty;
procento := procento hran k vymazání (vstupní parametr);
pocetKvymazani := počet hran určených k vymazaní (procento * celkový počet hran);
vymazatIDs := unikátní seznam vrcholů určených k vymazání;
while */vymazatIDs/ != pocetKvymazani* **do**
  id := náhodná hrana;
  vymazatIDs := vymazatIDs + id);
end
for *každý id* $\in$ $vymazatIDs$ **do**
  h := hrana identifikovaný *id*;
  GTDgraf := GTDgraf - h;
end
Result: GTD graf bez daného procenta hran

Algoritmus 17: Odstranění běžných hran

4.3.5 Změna GPS souřadnic u vrcholů hromadné dopravy

Ve vstupním grafu změní GPS souřadnice u daného procenta vrcholů, které jsou typu "GraphStopNode". GPS souřadnice jsou posunuty maximálně o 500 metrů od původní po-

lohy. Popis logiky techniky viz algoritmus 18.

GTDgraf := vstupní graf s daty;
procento := procento vrcholů ke změně (vstupní parametr);
pocetKzmene := počet uzlů určených ke změně (procento * celkový počet uzlů hromadné dopravy);
zmenitIDs := unikátní seznam vrcholů určených ke změně;
while */zmenitIDs/ != pocetKzmene* **do**
  id := náhodný vrchol hromadné dopravy;
  zmenitIDs := zmenitIDs + id);
end
for *každý* $id \in zmenitIDs$ **do**
  v := vrchol identifikovaný *id*;
  vUpraveny := upravené GPS souřadnice;
  GTDgraf := GTDgraf - v;
  GTDgraf := GTDgraf + vUpraveny;
end
Result: GTD graf s upravenými GPS souřadnicemi

Algoritmus 18: Změna GPS souřadnic u vrcholů hromadné dopravy

4.3.6 Změna GPS souřadnic u běžných vrcholů

Ve vstupním grafu změní GPS souřadnice u daného procenta vrcholů, které nejsou typu "GraphStopNode". GPS souřadnice jsou posunuty maximálně o 500 metrů od původní polohy. Popis logiky techniky viz algoritmus 19.

GTDgraf := vstupní graf s daty;
procento := procento vrcholů ke změně (vstupní parametr);
pocetKzmene := počet uzlů určených ke změně (procento * celkový počet uzlů nepatřící k hromadné dopravě);
zmenitIDs := unikátní seznam vrcholů určených ke změně;
while */zmenitIDs/ != pocetKzmene* **do**
  id := náhodný vrchol nepatřící k hromadné dopravě;
  zmenitIDs := zmenitIDs + id);
end
for *každý* $id \in zmenitIDs$ **do**
  v := vrchol identifikovaný *id*;
  vUpraveny := upravené GPS souřadnice;
  GTDgraf := GTDgraf - v;
  GTDgraf := GTDgraf + vUpraveny;
end
Result: GTD graf s upravenými GPS souřadnicemi

Algoritmus 19: Změna GPS souřadnic u vrcholů nepatřících k hromadné dopravě

4.3.7 Změna GPS souřadnic u náhodných vrcholů bez rozlišování typu

Ve vstupním grafu změní GPS souřadnice u daného procenta vrcholů bez přihlížení k typu daných vrcholů. GPS souřadnice jsou posunuty maximálně o 500 metrů od původní

polohy. Popis logiky techniky viz algoritmus 20.

GTDgraf := vstupní graf s daty;
procento := procento vrcholů ke změně (vstupní parametr);
pocetKzmene := počet uzlů určených ke změně (procento * celkový počet uzlů);
zmenitIDs := unikátní seznam vrcholů určených ke změně;
while *|zmenitIDs| != pocetKzmene* **do**
  id := náhodný vrchol;
  zmenitIDs := zmenitIDs + id);
end
for *každý id* $\in$ *zmenitIDs* **do**
  v := vrchol identifikovaný *id*;
  vUpraveny := upravené GPS souřadnice;
  GTDgraf := GTDgraf - v;
  GTDgraf := GTDgraf + vUpraveny;
end
Result: GTD graf s upravenými GPS souřadnicemi

Algoritmus 20: Změna GPS souřadnic u vrcholů bez odlišování typu

4.3.8 Změna délky hrany

Proces změní délku hrany. Počet změněných hran je dán procentem hran na vstupu. Pro zvýšení účinnosti testu je délka hrany zvýšena na desetinásobek až stonásobek své původní velikosti. Popis logiky techniky viz algoritmus 21.

GTDgraf := vstupní graf s daty;
procento := procento hran ke změně (vstupní parametr);
pocetKzmene := počet hran určených ke změně (procento * celkový počet hran);
zmenitIDs := unikátní seznam hran určených ke změně;
while *|zmenitIDs| != pocetKzmene* **do**
  id := náhodná hrana;
  zmenitIDs := zmenitIDs + id);
end
for *každý id* $\in$ *zmenitIDs* **do**
  h := hrana identifikovaná *id*;
  hUpravena := upravená délka hrany;
  GTDgraf := GTDgraf - h;
  GTDgraf := GTDgraf + hUpravena;
end
Result: GTD graf s upravenými délkami hran

Algoritmus 21: Změna délky hrany

4.3.9 Odstranění hromadné dopravy

Z původního grafu jsou odstraněny všechny hrany a vrcholy spojené s daným typem dopravy. Na vstupu je dán počet typů dopravy, které mají být odstraněny. Náhodným výběrem

se poté tento počet typů dopravy odstraní z grafu. Popis logiky techniky viz algoritmus 22.

```
GTDgraf := vstupní graf s daty;
pocetKvymazani := počet módů k odstranění (vstupní parametr);
modyKvymazani := unikátní seznam módů určených ke vymazání;
while /modyKvymazani/ != pocetKvymazani do
    mod := náhodný mód hromadné dopravy;
    modyKvymazani := modyKvymazani + mod;
end
for každý mod ∈ vymazatIDs do
    vrcholyKvymazani := vrcholy daného módu dopravy;
    for každý vrchol v ∈ vrcholyKvymazani do
        hrany := všechny hrany spojené s vrcholem v;
        GTDgraf := GTDgraf - hrany;
        GTDgraf := GTDgraf -v;
    end
end
Result: GTD graf s odstraněnými módy dopravy
```

Algoritmus 22: Odstranění hromadné dopravy

4.3.10 Odstranění linek hromadné dopravy

Ze vstupních dat jsou odstraněny hrany a vrcholy pro linky hromadné dopravy. Počet odstraněných linek je dán procentem všech linek v grafu. Popis logiky techniky viz algoritmus 23.

```
GTDgraf := vstupní graf s daty;
procento := procento linek k vymazání (vstupní parametr);
pocetKvymazani := počet linek určených k vymazaní (procento * celkový počet
 linek);
vymazatIDs := unikátní seznam linek určených k vymazání;
while /vymazatIDs/ != pocetKvymazani do
    id := náhodná linka;
    vymazatIDs := vymazatIDs + id;
end
for každá linka l ∈ vymazatIDs do
    for každý hrana h patřící k lince l do
        GTDgraf := GTDgraf - h;
    end
end
Result: GTD graf bez daného procenta linek hromadné dopravy
```

Algoritmus 23: Odstranění linek hromadné dopravy

4.3.11 Změna maximální rychlosti hrany

Maximální povolená rychlost je informace na hraně grafu. Tato informace je změněna pro dané procento hran na vstupu. Maximální povolená rychlost je změněna na hodnotu

z možností: 30,40,50,60,70,80,90,100,110,120,130,140,150 Popis logiky techniky viz algoritmus 24.

GTDgraf := vstupní graf s daty;
procento := procento hran ke změně (vstupní parametr);
pocetKzmene := počet hran určených ke změně (procento * celkový počet hran);
zmenitIDs := unikátní seznam hran určených ke změně;
while */zmenitIDs/ != pocetKzmene* **do**
  id := náhodná hrana;
  zmenitIDs := zmenitIDs + id;
end
for *každý id* ∈ *zmenitIDs* **do**
  h := hrana identifikovaná *id*;
  hUpravena := upravená maximální rychlost;
  GTDgraf := GTDgraf - h;
  GTDgraf := GTDgraf + hUpravena;
end
Result: GTD graf s upravenými maximálními rychlostmi

Algoritmus 24: Změna maximální rychlosti hrany

4.3.12 Změna spojení sítě hromadné dopravy s ostatní dopravou

Vstupní GTD graf obsahuje 2 subgrafy, graf hromadné dopravy a ostatní dopravy, které jsou spolu spojené specifickými hranami. Vždy jeden vrchol pro danou stanici hromadné dopravy je 'hlavní' a spojuje stanici se sítí ostatní dopravy. Metoda mění tyto napojení na jiné vrcholy v okolí 500 m od původního vrcholu. Popis logiky techniky viz algoritmus 25.

GTDgraf := vstupní graf s daty;
procento := procento vrcholů ke změně (vstupní parametr);
pocetKzmene := počet vrcholů určených ke změně (procento * celkový počet vrcholů hromadné dopravy);
zmenitIDs := unikátní seznam vrcholů určených ke změně;
while */zmenitIDs/ != pocetKzmene* **do**
  id := náhodný vrchol;
  zmenitIDs := zmenitIDs + id;
end
for *každý id* ∈ *zmenitIDs* **do**
  v := vrchol identifikovaný *id*;
  if *v není hlavní vrchol* **then**
    v := vyhledej hlavní vrchol pro *v*;
  end
  hStare := aktuální hrany pro vrchol *v*;
  hNove := hrany pro nové spojení;
  GTDgraf := GTDgraf - hStare;
  GTDgraf := GTDgraf + hNove;
end
Result: GTD graf se změněnými hranami spojující dopravu hromadnou s osobní

Algoritmus 25: Změna spojeni sítě hromadné dopravy s ostatní dopravou

4.3.13 Odstranění spojení sítě hromadné dopravy s ostatní dopravou

Vstupní GTD graf obsahuje 2 subgrafy, graf hromadné dopravy a ostatní dopravy, které jsou spolu spojené specifickými hranami. Vždy jeden vrchol pro danou stanici hromadné dopravy je 'hlavní' a spojuje stanici se sítí ostatní dopravy. Metoda odstraní tyto napojení z grafu. Popis logiky techniky viz algoritmus 26.

```
GTDgraf := vstupní graf s daty;
procento := procento vrcholů k odstranění (vstupní parametr);
pocetKodstraneni := počet vrcholů určených k odstranění (procento * celkový počet
 vrcholů hromadné dopravy);
zmenitIDs := unikátní seznam vrcholů určených k odstranění;
while /zmenitIDs/ != pocetKodstraneni do
    id := náhodný vrchol;
    zmenitIDs := zmenitIDs + id;
end
for každý id ∈ zmenitIDs do
    v := vrchol identifikovaný id;
    if v není hlavní vrchol then
        v := vyhledej hlavní vrchol pro v;
    end
    hStare := aktuální hrany pro vrchol v;
    GTDgraf := GTDgraf - hStare;
end
```

Result: GTD graf s odstraněnými hranami spojující dopravu hromadnou s osobní
Algoritmus 26: Odstranění spojení sítě hromadné dopravy s ostatní dopravou

4.3.14 Změna názvů stanic hromadné dopravy

Názvy jednotlivých stanic jsou důležité při vyhledávání trasy plánovačem. Tento generátor simuluje chyby v datech. Simuluje překlepy a rozdílné názvy od původních dat. U procenta náhodných vstupních stanic změní název. Změna názvu probíhá pomocí nahrazení

jednoho znaku za jiný náhodně vybraný. Popis logiky techniky viz algoritmus 27.

GTDgraf := vstupní graf s daty;
procento := procento vrcholů ke změně (vstupní parametr);
pocetKzmene := počet vrcholů určených ke změně (procento * celkový počet vrcholů hromadné dopravy);
zmenitIDs := unikátní seznam vrcholů určených ke změně;
while *|zmenitIDs| != pocetKzmene* **do**
  id := náhodný vrchol;
  zmenitIDs := zmenitIDs + id;
end
for *každý id* $\in$ *zmenitIDs* **do**
  v := vrchol identifikovaný *id*;
  vNovy := *v* se změněným názvem stanice;
  GTDgraf := GTDgraf - *v*;
  GTDgraf := GTDgraf + *vNovy*;
end
Result: GTD graf se změněnými názvy stanic hromadné dopravy

Algoritmus 27: Změna názvů stanic hromadné dopravy

4.4 Uživatelské rozhraní

Samotný kód testovacího rozhraní je vytvořen tak, aby mohl být použit a volán z kódu jiné aplikace, neboť se předpokládá, že tento nástroj může sloužit jako část případného plánovače. Pro testování dat mimo jinou strukturu slouží webové rozhraní. Toto rozhraní umožňuje odbornému uživateli spouštět testovací techniky, či načítat nové grafy. Další funkcí rozhraní je zobrazení výsledných výstupních dat jednotlivých testů, které jsou uloženy v mongo DB.

Shrnutí hlavních funkcí webového rozhraní:

- **Grafy a výsledky** - zobrazení výsledků uložených v mongo DB
- **Načtení nového grafu** - načte nový datový set ze SUPERHUB a uloží ho na lokální DB
- **Suite konfigurátor** - spouštění testů nad daty
- **Generátor defektů** - vytvoření nových datových souborů se zavedenými umělými defekty

Pro bližší popis funkcí webového rozhraní, včetně instalace a spuštění, je určen přiložený manuál A

Kapitola 5

Testování a vyhodnocení testů

5.1 Evaluace vlastního programu

Při vývoji testovacích technik byl vždy zvolen jeden datový set pro testování. Pouze jeden z důvodů toho, že testovacích dat nebylo k dipozici velké množství. Také při odlaďování pouze na jednom datovém setu minimalizujeme možnost úpravy kódy 'na míru' vstupním datům. Pokud by se ladilo na všech dostupných grafech, postupem času by kód nemusel fungovat obecně.

Po vyladění kódu se metoda spustila i nad ostatními grafy pouze pro ověření, že metoda funguje správně. Pokud se nalezla dodatečná chyba, která musela být odstraněna, chyba se zanalyzovala a opět na původních datech kontrolovala. Po zjištění chyby v kódu a odladění na původním testovacím setu, kdy se potvrdilo, že metoda funguje stále správně, jako fungovala před dodatečnou změnou, byly spuštěny testy opět nad všemi grafy. Při vývoji programu došlo k takto dodatečným úpravám po testování nad všemi daty pouze v ojedinělých případech. Tohoto nízkého počtu dodatečných zásahů do kódů bylo docíleno také proto, protože při ladění kódů byly spouštěny desítky či dokonce stovky testů s náhodně generovanými chybami, aby spektrum případných chyb a různorodost vstupních dat byla simulována již ve vývojové fázi.

5.2 Vstupní datové sety

Jednotlivé testovací metody byly spuštěny nad dostupnými daty. Pro simulaci chyb byla vstupní data také změněna generátorem chyb, který upravil původní data sety. Jednotlivé vstupní grafy jsou tedy rozděleny na následující typy:

- **Correct** - Popis dat, které v rámci testování považujeme jako správná. Jednotlivé metody mohou napočítat hodnoty, které se poté budou využívat v příštím testovaní s dalšími verzemi dat.
- **Wrong1** - data se zanesenými chybami.
- **Wrong2** - data se zanesenými chybami jako Wrong1. Za vstupní graf je použit Wrong1 data set, přičemž z původních chyb je 50% odstraněno a dalších 30% nových chyb přidáno.

Data set ID	Region	Number of nodes	Number of edges	Number of PT stops	Area size (km^2)	Transport modes
Brno_1.0.0_GOOD_2016-1-13_12-50	Brno	63029	194004	27083	230,22	5
Milan_1.0.0_GOOD_2016-1-13_13-23	Milan	213188	808085	17571	181,8	4
Barcelona_1.0.0_GOOD_2016-1-13_13-20	Barcelona	228850	1117812	17655	101,9	4
Helsinki_1.0.0_GOOD_2016-1-13_13-29	Helsinki	288351	986373	43185	184,5	5
Praha_1.0.0_GOOD_2016-1-13_12-59	Praha	124501	407655	19156	496,0	5

Tabulka 5.1: Data sety 'Correct'

Data set ID	testing technique	# defects detected	# false alarms
Praha_1.0.0_GOOD_2016-1-13_12-59	TransportTimesCheck	191	
	ZeroTravelTime	118	
	TransportStopStationsGSPCheck (20 m)		4258
	StationWalkConnectionCheck	1687	

Tabulka 5.2: Vyhodnocení data setu 'Correct' pro Prahu

5.2.1 Data set 'Correct'

Při testování jsou takto označená data považována za správná a vůči nim se poté kontrolují jiné datové sety. Pro testování byla dostupná data pro následující města: Brno, Praha, Milan, Barcelona, Helsinki. V příloze Testing results v záložce Correct jsou pak zaznamenány také základní informace o daných datech (viz také tabulka 5.1, která je ukázkou z přílohy Testing results)

5.2.2 Výsledky nad data sety Correct

Základní data k dispozici pro testování jsou reálná data, a proto samotná obsahují chyby, či nesrovnalosti. Po provedení některých testovacích technik, jako například metody kontrolující GPS souřadnice stanic, či metody zpracovávající jízdní řády, dostáváme na výstupu upozornění na možné chyby. Všechny tyto nalezené chyby jsou zaznamenány v příloze Testing results v sekci 'Correct_results'. Pro ukázku je níže popsán seznam nalezených chyb pro data set Praha viz tabulka 5.2. Tyto napočítané hodnoty jsou důležité pro budoucí porovnávání nad grafy se zanesenými chybami, abychom dokázali rozlišit, které chyby přišly již v původním data setu, a které jsou nově objevené.

Zero Travel Time

Metoda zpracovává informace grafu, které pochází z jízdních řádů. Vzhledem k tomu, že v datech jsou spojeny nejbližší příjezd dopravního prostředku spolu s odjezdem z předchozí stanice, může se v datech vyskytnout situace, kdy čas potřebný pro přesun je nulový, i když vzdálenost mezi stanicemi nulová není.

Data set ID	CORRECT data set ID	defect type	count of these defects
Praha_1.0.0_BAD_2016-1-29_9-13	Praha_1.0.0_GOOD_2016-1-13_12-59	ChangePTedgeConnection - changed 1 procent of route nodes	nodes: 80 , edges: 160
Praha_1.0.0_BAD_2016-1-29_9-43	Praha_1.0.0_GOOD_2016-1-13_12-59	Change GPS position of route nodes - 2 procent	382
Praha_1.0.0_BAD_2016-1-29_10-14	Praha_1.0.0_GOOD_2016-1-13_12-59	Change GPS position of road nodes - 3 procent	3159
Praha_1.0.0_BAD_2016-1-29_10-45	Praha_1.0.0_GOOD_2016-1-13_12-59	Change length of edge - 1 procent	4076
Praha_1.0.0_BAD_2016-1-29_13-42	Praha_1.0.0_GOOD_2016-1-13_12-59	Change max speed of edge - percent	3591
Praha_1.0.0_BAD_2016-1-29_14-6	Praha_1.0.0_GOOD_2016-1-13_12-59	Remove 2 procent of roate edges	968
Praha_1.0.0_BAD_2016-1-29_14-30	Praha_1.0.0_GOOD_2016-1-13_12-59	Remove 3 procent of road edges	10773
Praha_1.0.0_BAD_2016-1-29_14-57	Praha_1.0.0_GOOD_2016-1-13_12-59	Remove 3 procent of cross road nodes.	nodes 3159, edges 17644
Praha_1.0.0_BAD_2016-1-29_15-26	Praha_1.0.0_GOOD_2016-1-13_12-59	Remove PT stop 2 procent of nodes.	nodes 382 edges 4312
Praha_1.0.0_BAD_2016-1-31_17-20	Praha_1.0.0_GOOD_2016-1-13_12-59	Remove 3 procent of pt lines edges.	9
Praha_1.0.0_BAD_2016-1-31_17-40	Praha_1.0.0_GOOD_2016-1-13_12-59	Remove PT mode	TRAM
Praha_1.0.0_BAD_2016-2-12_0-53	Praha_1.0.0_GOOD_2016-1-13_12-59	RemovePTedgeConnection - removed 1 procent of route nodes	nodes: 32, edges: 64
Praha_1.0.0_BAD_2016-2-12_0-28	Praha_1.0.0_GOOD_2016-1-13_12-59	Change length of edge - 1 procent	484
Praha_1.0.0_BAD_2016-2-12_1-36	Praha_1.0.0_GOOD_2016-1-13_12-59	Change GPS position of PT and Road nodes - 3 procent	3735
Praha_1.0.0_BAD_2016-3-11_9-24	Praha_1.0.0_GOOD_2016-1-13_12-59	Change Random Stations Name - 15 procent	2865

Tabulka 5.3: Ukázka tabulky z přílohy, ze sekce Wrong1 pro Prahu

Transport Times Check

Data zpracovávaná touto metodou také pochází z jízdních řádů a obdobně jako v metodě Zero Travel Time se porovnávají časy potřebné k přesunutí z jedné stanice na druhou. Nyní však nehledáme pouze nulové hodnoty, ale kontrolujeme, zda časy v datech jsou reálně zvládnutelné pro hromadnou dopravu. Vzhledem k nastavený parametrům metody tedy výsledné hodnoty mohou být buď chybami v datech, či krajními hodnotami, které je třeba zkontrolovat.

Transport Stop Stations GSP Check

Nalezené hodnoty v tabulce pro tuto metody jsou označené jako false alarm z toho důvodu, že pro kontrolu GPS souřadnic využívá metoda externí zdroj, který nemusí obsahovat vždy stejné informace o všech stanicích.

Station Walk Connection Check

Nalezené hodnoty touto metodou jsou i v correct grafu z toho důvodu, že data obsahují informace i o části okolí daných měst. Data s údaji o hromadné dopravě mnohdy zasahují daleko za hranice města. Vzhledem k tomu, že spoj mezi stanicí hromadné dopravy a silniční sítí nesmí být delší než 500 metrů, jsou nalezené hodnoty i pro případ správného data setu korektní. Při analýze dat bylo potvrzeno, že tyto 'nespojené' vrcholy grafu leží mimo dané město a daleko od silniční sítě, kterou data obsahují.

5.2.3 Data set 'Wrong1'

Sekce Wrong1 přílohy Testing results představuje vytvořená vstupní data se zavedenými chybami, které mohou nastat. Z tabulky 5.4 lze vyčíst, že v příloze je zaznamenáno ID daného grafu spolu s ID původním grafem, do kterého byly zavedeny chyby. Dále obsahuje informace o typu zavedených defektů a jejich počtu.

5.2.4 Data set 'Wrong2'

Sekce Wrong2 přílohy Testing results představuje vytvořená vstupní data se zavedenými chybami, které mohou nastat. Jako vstupní data sety jsou zde použity již dříve upravená data se zanesenými chybami. Z tabulky 5.5 lze vyčíst, že v příloze je zaznamenáno ID daného grafu spolu s ID původního grafu, se zavedenými chybami a také s úplně původním grafem, ze kterého tyto data sety vznikly. Dále obsahuje informace o typu zavedených defektů. V jednotlivých sloupcích je poté znázorněno, kolik chyb bylo v původním Wrong1 data setu, kolik chyb bylo odstraněno (napraveno) a kolik přidáno do nového data setu Wrong2.

Data set ID	CORRECT data set ID	defect type	count of these defects
Praha_1.0.0_BAD_2016-1-29_9-13	Praha_1.0.0_GOOD_2016-1-13_12-59	ChangePTedgeConnection - changed 1 procent of route nodes	nodes: 80 , edges: 160
Praha_1.0.0_BAD_2016-1-29_9-43	Praha_1.0.0_GOOD_2016-1-13_12-59	Change GPS position of route nodes - 2 procent	382
Praha_1.0.0_BAD_2016-1-29_10-14	Praha_1.0.0_GOOD_2016-1-13_12-59	Change GPS position of road nodes - 3 procent	3159
Praha_1.0.0_BAD_2016-1-29_10-45	Praha_1.0.0_GOOD_2016-1-13_12-59	Change length of edge - 1 procent	4076
Praha_1.0.0_BAD_2016-1-29_13-42	Praha_1.0.0_GOOD_2016-1-13_12-59	Change max speed of edge - percent	3591
Praha_1.0.0_BAD_2016-1-29_14-6	Praha_1.0.0_GOOD_2016-1-13_12-59	Remove 2 procent of roate edges	968
Praha_1.0.0_BAD_2016-1-29_14-30	Praha_1.0.0_GOOD_2016-1-13_12-59	Remove 3 procent of road edges	10773
Praha_1.0.0_BAD_2016-1-29_14-57	Praha_1.0.0_GOOD_2016-1-13_12-59	Remove 3 procent of cross road nodes.	nodes 3159, edges 17644
Praha_1.0.0_BAD_2016-1-29_15-26	Praha_1.0.0_GOOD_2016-1-13_12-59	Remove PT stop 2 procent of nodes.	nodes 382 edges 4312
Praha_1.0.0_BAD_2016-1-31_17-20	Praha_1.0.0_GOOD_2016-1-13_12-59	Remove 3 procent of pt lines edges.	9
Praha_1.0.0_BAD_2016-1-31_17-40	Praha_1.0.0_GOOD_2016-1-13_12-59	Remove PT mode	TRAM
Praha_1.0.0_BAD_2016-2-12_0-53	Praha_1.0.0_GOOD_2016-1-13_12-59	RemovePTedgeConnection - removed 1 procent of route nodes	nodes: 32, edges: 64
Praha_1.0.0_BAD_2016-2-12_0-28	Praha_1.0.0_GOOD_2016-1-13_12-59	Change length of edge - 1 procent	484
Praha_1.0.0_BAD_2016-2-12_1-36	Praha_1.0.0_GOOD_2016-1-13_12-59	Change GPS position of PT and Road nodes - 3 procent	3735
Praha_1.0.0_BAD_2016-3-11_9-24	Praha_1.0.0_GOOD_2016-1-13_12-59	Change Random Stations Name - 15 procent	2865

Tabulka 5.4: Ukázka tabulky z přílohy, ze sekce Wrong1 pro Prahu

Data set ID	Data set before ID	CORRECT data set ID	defect type	count of these defects before	count of these defects after	count of removed defects	count of new inserted defects
Praha_1.0.2_BAD_	Praha_1.0.0_BAD_	Praha_1.0.0_GOOD_20	ChangePTedgeConnection - changed 1 procent of route node	nodes: 80 , edges: 160	nodes: 13 , edges: 26	??	??
Praha_1.0.2_BAD_	Praha_1.0.0_BAD_	Praha_1.0.0_GOOD_20	Change GPS position of route nodes - 2 procent	382	305	141	164
Praha_1.0.2_BAD_	Praha_1.0.0_BAD_	Praha_1.0.0_GOOD_20	Change GPS position of road nodes - 3 procent	3159	2527	1579	948
Praha_1.0.2_BAD_	Praha_1.0.0_BAD_	Praha_1.0.0_GOOD_20	Change length of edge	4076	3261	2038	1223
Praha_1.0.2_BAD_	Praha_1.0.0_BAD_	Praha_1.0.0_GOOD_20	Change max speed of edge	3591	2873	1795	1078
Praha_1.0.2_BAD_	Praha_1.0.0_BAD_	Praha_1.0.0_GOOD_20	Remove 2 procent of roate edges	968	774	484	290
Praha_1.0.2_BAD_	Praha_1.0.0_BAD_	Praha_1.0.0_GOOD_20	Remove 3 procent of road edges	10773	8618	5386	3232
Praha_1.0.2_BAD_	Praha_1.0.0_BAD_	Praha_1.0.0_GOOD_20	Remove 3 procent of cross road nodes.	nodes 3159, edges 1764	nodes 2527, edges 1415	1579	948
Praha_1.0.2_BAD_	Praha_1.0.0_BAD_	Praha_1.0.0_GOOD_20	Remove PT stop 2 procent of nodes.	nodes 382 edges 4312	nodes 305 edges 3621	191	114
Praha_1.0.2_BAD_	Praha_1.0.2_BAD_	Praha_1.0.0_BAD_201	Remove 3 procent of pt lines edges.	9	7	4	3
Praha_1.0.2_BAD_	Praha_1.0.0_BAD_	Praha_1.0.0_GOOD_20	Remove PT mode	TRAM	UNDERGROUND	TRAM	UNDERGROUND
Praha_1.0.2_BAD_	Praha_1.0.0_BAD_	Praha_1.0.0_GOOD_20	RemovePTedgeConnection - removed 1 procent of route node	nodes: 32, edges: 64	nodes: 17, edges: 34	16	1
Praha_1.0.2_BAD_	Praha_1.0.0_BAD_	Praha_1.0.0_GOOD_20	Change length of edge - 1 procent	484	388	242	146
Praha_1.0.2_BAD_	Praha_1.0.0_BAD_	Praha_1.0.0_GOOD_20	Change GPS position of PT and Road nodes - 3 procent	3735	2988	1867	1121
Praha_1.0.2_BAD_	Praha_1.0.0_BAD_	Praha_1.0.0_GOOD_20	Change Random Stations Name - 15 procent	2865	2292	1432	860

Tabulka 5.5: Ukázka tabulky z přílohy, ze sekce Wrong2 pro Prahu - zmenšená verze sloužící jen jako náhled

5.3 Testování řešení implementovaných metod

Při kontrole řešení je nutné důkladně projít a porovnat nalezené chyby, či proč metoda nenalezla zavedené defekty. Při vývojové fázi se kontrolují jednotlivé nalezené vadné hrany, či vrcholy, u kterých se nalezla/nenalezla chyba. Při nalezených chybách v 'CORRECT' datech musely být všechny označené defekty prozkoumány, aby mohlo být řečeno, že metoda funguje správně.

Při zavádění defektů máme tu výhodu, že víme, které informace byly změněny, i když defekty jsou generované převážně náhodně. Informace o vygenerovaných defektech jsou uloženy na souborovém systému jako serializované objekty. Jednotlivé testovací metody poté mají přidanou funkci kontroly. Na konci každé testovací metody (odděleně od samotného vyhodnocení) se načtou informace o vygenerovaných defektech a porovnají se s nalezenými chybami. Program je tedy schopen zanalyzovat výstup metody za předpokladu, že víme o zanesených vadách v datech. V reálném používání ovšem nevíme, zda a kde jsou jaké chyby obsaženy. Přidaná kontrola slouží převážně pro testování samotných testovacích metod a zvyšuje tím kontrolu pro vývojáře. Vzhledem k tomu, že program je vytvořen tak, aby nové metody mohly být přidávány, je tato struktura ukládání a načítání zavedených chyb připravena i pro případné rozšíření programu v budoucnu.

5.4 Vyhodnocení účinnosti testovacích metod

Nad danými data sety byly spuštěny všechny testovací metody. Jejich výstup je přehledně zobrazen v tabulce v příloze Testing results v sekci Experimnet1_results A Experimnet2_results. Pro daný graf je vždy znázorněno jeho ID. Dále na jednotlivých řádcích jsou vypsány jednotlivé metody a jejich výstupy. V tabulce jsou znázorněny tyto hodnoty/sloupce:

- **Data Set ID** - ID daného grafu, nad kterým byly spuštěny testovací metody
- **Testovací technika** - Název testovací techniky
- **Typ aplikovaného defektu** - popis defektu, který byl do dat zaveden
- **Status metody** - výstupní status metody (SUCCESS, FAILED, a podobně)
- **# nalezených chyb v CORRECT grafu** - počet nalezených 'vad' v correct grafu (viz sekce 5.2.2)
- **# celkových nalezených chyb** - celkový počet nalezených 'vad' v daném grafu
- **# nalezených vytvořených chyb** - počet nalezených zavedených defektů
- **# nenalezených zavedených chyb** - počet nenalezených zavedených defektů
- **# falešných chyb** - počet nalezených defektů nad rámec zavedených chyb
- **rozmezí/tolerance** - informace o nastavených parametrech pro danou metodu

5.4.1 Silně souvislé komponenty

Metoda kontroly silně souvislých komponent (viz kapitola 4.2.1) je citlivá na změnu struktury v datech. I jedna hrana navíc může změnit počet silně souvislých komponent, pokud spojí zrovna 2 nezávislé komponenty. U této metody se předpokládá, že 2 různé grafy nebudou mít vždy zcela totožné hodnoty. Proto jeden ze vstupních parametrů je i tolerance, se kterou si uživatel rozhodne, kdy již lze předpokládat, že v datech může existovat chyba. Metoda zareagovala na následující vygenerované typy chyb (jedná se o zavedené chyby, které z podstaty věci i mění strukturu grafu):

- **Odstranění běžných hran (viz kapitola 4.3.4)**
- **Odstranění hran hromadné dopravy (viz kapitola 4.3.3)**
- **Odstranění běžných uzlů (viz kapitola 4.3.2)**
- **Odstranění uzlů hromadné dopravy (viz kapitola 4.3.1)**
- **Odstranění hromadné dopravy (viz kapitola 4.3.9)**
- **Odstranění linek hromadné dopravy (viz kapitola 4.3.10)**

Nelze však spoléhat na to, že všechny výše popsané chyby budou touto metodou vždy zachyceny. Ve výsledné tabulce (příloha Testing results) lze vyčíst, že ne pro všechny testy nad generovanými chybami tato metoda zafungovala. Je to způsobeno tím, že odstraněním hrany vždy nemusíme rozpojit silně souvislé komponenty a podobně. Dále stojí za zmínku, že v příloze vidíme označené chyby, avšak status metody je 'SUCCESS'. To je způsobeno již zmíněnou tolerancí, kterou si nastaví uživatel. Pokud je rozdíl porovnávaných počtů v toleranci, metoda neoznačí rozdíl jako chybu (pro vyhodnocení testů byly výstupy z metod porovnávány důkladněji pro větší přehlednost, a proto jsou hodnoty vyznačeny a popsány i když metoda byla úspěšná dle statusu).

5.4.2 Kontrola jednosměrných chodníků

Metoda kontroluje existenci jednosměrných chodníků (viz kapitola 4.2.2). Pro nalezení chyby je zapotřebí mít nekonzistentní data jako například chybějící hrana, či hrana, která je nesprávně určena a není chodníkem. Dle simulovaných testů metoda nalezla chyby datech s následujícími vygenerovanými chybami:

- **Odstranění běžných hran (viz kapitola 4.3.4)** - náhodné odstranění hran způsobí chybějící spoje a metoda nalezne jednosměrné chodníky

Ve výsledkové tabulce lze vyčíst, že ne všechny zavedené defekty byly nalezeny touto metodou. Tyto nenalezené chyby byly prozkoumány a rozděleny do dvou kategorií. První kategorií jsou odstranění hrany, které nejsou chodníky. Další skupinou jsou situace, kdy mezi smazanými chybami chybí obě hrany pro chodník určující oba směry spoje.

5.4.3 Typy hromadné dopravy

Aby graf neobsahoval typ hromadné dopravy, musí v datech chybět všechny informace (na hranách i vrcholech), které jsou spojené s daným typem hromadné dopravy (viz popis metody 4.2.3). Nalezené chyby touto metodou:

- **Odstranění hromadné dopravy (viz kapitola 4.3.9)** - metoda určí, že graf neobsahuje požadovaný typ dopravy
- **Odstranění linek hromadné dopravy (viz kapitola 4.3.10)** - při vyšším procentu odstraněných linek hromadné dopravy může být zároveň odstraněn celý mód dopravy

Při testování také nastala situace, kdy při odstranění velkého počtu vrcholů byl zároveň odstraněn mód dopravy. Pro označení chyby tedy nesmí být v grafu žádný vrchol daného módu dopravy oproti požadovaným parametrům. Metoda také upozorní na módy dopravy, které v grafu přebývají oproti očekávanému seznamu.

5.4.4 Kladný čas cesty hromadné dopravy

Vzhledem k tomu, že struktura a pravidla vstupních dat neumožňují obsahovat záporné časy pro přesun po hraně, neobjevila metoda žádné nesrovnalosti napříč daty. Avšak i toto zjištění není nežádoucí, neb potvrzuje správnost jednoho předpokladu, který o datech máme. Pokud by data byla například poškozena přenosem, či dokonce byla špatné a obsahovala záporné časy, tato metoda (Kladný čas cesty hromadné dopravy 4.2.4) by je odhalila. Časové informace o přesunech pochází z dat o jízdních řádech, která mohou být také poškozená.

5.4.5 Kontrola času cesty hromadné dopravy

Metoda kontroluje, zda čas potřebný pro přesun hromadné dopravy, není nerealistický, ať už pomalý, či příliš rychlý (viz popis metody 4.2.5). Nalezené chyby touto metodou:

- **Změna délky hrany (viz kapitola 4.3.8)**

Při změně délky hrany se vypočtená průměrná rychlost výrazně zvýší, či sníží a tím pádem je vyhodnocena jako nepřípustná či podezřelá.

Nastavení rozsahu určujícího, kdy je hrana označena za špatnou, musí být prováděno se znalostí daných dat. Při moc velkém rozsahu nezachytíme menší nesrovnalosti a při malém zachytíme zase i hrany, které nemusí být kontrolovány. Tento rozsah byl zvolen, aby pokryl větší část středních hodnot, ale také aby upozornil i na krajní hodnoty. Proto již při spuštění nad daty 'CORRECT' máme počet nalezených nesrovnalostí a nezačínáme další porovnání s nulovým číslem. Toto nastavení nám umožní zachytit jiné typy defektů, jako například odstranění hran (viz popis metody 4.3.3), neboť nalezený počet chyb je menší než původní očekávaný.

5.4.6 Nulový čas cesty hromadné dopravy

Při vytváření vstupního grafu může dojít k chybné struktuře na základě jízdních řádů. Tato metoda, na rozdíl od kontroly kladných časů (Kladný čas cesty hromadné dopravy 4.2.4), které by neměly nastat nikdy, kontroluje, zda čas pro přesun mezi stanicemi není nulový (viz metoda 4.2.6). Na základě testování a analýzy dat bylo zjištěno, že již ve vstupních datech se nachází tyto typy chyb.

Pokud tyto chyby nepotřebujeme odstranit a počítáme s nimi pro příští testování, můžeme tím nalézt jiné typy defektů, které ovlivňují část grafu s hromadnou dopravou.

5.4.7 Kontrola GPS souřadnic stanic hromadné dopravy

Kontrola GPS souřadnic probíhá oproti Google API (viz popis metody 4.2.7). Nalezené chyby touto metodou:

- **Změna GPS souřadnic u vrcholů hromadné dopravy (viz kapitola 4.3.5)**
- **Změna GPS souřadnic u náhodných vrcholů bez odlišování typu (viz kapitola 4.3.7)**
- **Změna názvů stanic hromadné dopravy (viz kapitola 4.3.14)** - nově vygenerované názvy nenalezeny v externím datovém zdroji

Dle výsledné tabulky (příloha Testing results) je patrné, že všechny testy této metody skončily statusem 'FAILED' v důsledku toho, že použité Google API dle testování na některé stanice nevrací pozitivní výsledek. Může to být způsobeno rozdílnými datovými strukturami pro dané API, neboť při běžném hledání na mapách Google mají testovací stanice často totožné názvy. Jako výchozí počet chyb se napočítaly všechny nesrovnalosti v příslušném 'CORRECT' grafu a pomocí tohoto čísla se porovnávaly výsledky z následných grafů. Tato situace může vést k tomu, že pokud některé chyby přibudou a jiné zmizí, může být výsledný počet chyb zkreslený a je zapotřebí důkladnější analýza. Na druhou stranu mít počáteční počet předpokládaných chyb také může odhalit chybějící data, neboť například při odstranění uzlů hromadné dopravy (viz popis metody 4.3.1) pozorujeme nižší počet celkově nalezených chyb.

Při použití této metody se musí s rozvahou volit také rozsah, při kterém říkáme, v jakém okruhu od GPS souřadnice může stanice v externím zdroji být, aby to ještě nebylo nepovažováno za chybu. V datech vstupních i externích máme stanice se stejným názvem, ale na různých souřadnicích (stanice v jedno, či druhém směru nebo stanice se stejnými názvy v okruhu jedné křižovatky). Proto při zvolení většího rozsahu nemusíme zachytit tu správnou očekávanou stanici. Generátor chyb přesouvá GPS souřadnice o 500 metrů. Proto při testech, kdy byl povolený rozsah schválně zvolen na 499 metrů, nezachytil všechny vygenerované defekty. Nenalezeny byly v důsledku toho, že v externím zdroji dat již byly například posunuty o 10 m a tedy posun o 500 m v datech znamenal novou odchylku pouze 490 od externích dat.

Použité Google API vrací souřadnice pro názvy stanic a proto metoda nezachytí změny GPS souřadnic uzlů dopravní sítě, jak je možno vidět u defektů změny GSP souřadnic u běžných uzlů (viz popis metody 4.3.6) a změny GPS souřadnic bez odlišení typu (viz popis metody 4.3.7).

5.4.8 Počet hran a vrcholů

Kontrola počtu hran a vrcholů porovnává také strukturu dat, tentokrát se však porovnávají zvlášť vrcholy a hrany (viz popis metody 4.2.8). I u tohoto porovnávání se počítá s tolerancí změn, které mohou nastat u jiného grafu, aniž by vzniklo podezření chyb v datech. Označení chyby touto metodou může odhalit například chybějící úseky dat, či nově přidaná data (nová linka hromadné dopravy a podobně). Nalezené chyby touto metodou:

- **Odstranění běžných hran (viz kapitola 4.3.4)**
- **Odstranění hran hromadné dopravy (viz kapitola 4.3.3)**
- **Odstranění běžných uzlů (viz kapitola 4.3.2)**
- **Odstranění uzlů hromadné dopravy (viz kapitola 4.3.1)**
- **Odstranění hromadné dopravy (viz kapitola 4.3.9)**
- **Odstranění linek hromadné dopravy (viz kapitola 4.3.10)**
- **Odstranění spojeni sítě hromadné dopravy s ostatní dopravou (viz kapitola 4.3.13)**

Opět musíme vhodně volit toleranci, neboť počet všech hran a uzlů jsou velké čísla. I 1% tolerance může znamenat změnu tisíců hran pro vyhodnocení potencionální chyby v datech.

Data set ID	Testovací technika	Typ aplikovaného defekt	Status metod	# nalezených chyb v CORRECT grafu	# celkový počet nalezených chyb	# nalezených vytvořených chyb	# nenalezených zavedených chyb	# falešných chyb
Brno_1.0.2_BAD_2016-1-30_21-33	GraphStrongComponer	edges deleted	FAILED	2492	2966	474	-----	-----
Brno_1.0.2_BAD_2016-1-30_21-41	GraphStrongComponer	nodes and edges delete	FAILED	2492	2678	186	-----	-----
Brno_1.0.2_BAD_2016-1-30_21-50	GraphStrongComponer	nodes and edges delete	FAILED	2492	2821	329	-----	-----
Praha_1.0.2_BAD_2016-1-29_14-18	GraphStrongComponer	removed route edges	FAILED	510	698	188	-----	-----
Praha_1.0.2_BAD_2016-1-29_14-44	GraphStrongComponer	edges deleted	FAILED	510	1514	1004	-----	-----
Praha_1.0.2_BAD_2016-1-29_15-13	GraphStrongComponer	nodes and edges delete	FAILED	510	890	380	-----	-----
Praha_1.0.2_BAD_2016-1-29_15-37	GraphStrongComponer	nodes and edges delete	FAILED	510	671	161	-----	-----

Tabulka 5.6: Ukázka tabulky z přílohy, ze sekce Wrong2 result pro metodu silně souvislých komponent

Data set ID	Testovací technika	Typ aplikovaného defektu	Status metody	# nalezených chyb v CORRECT grafu	# celkový počet nalezených chyb	# nalezených vytvořených chyb	# nenalezených zavedených chyb	# falešných chyb	rozmezí/tolerance
Brno_1.0.2_BAD_2016-1-30_21-2	NumberOfNodesAndEdges	removed route edges	SUCCESS	194004 edges, 63029 nodes	192995 edges, 63029 nodes	1009 edges	0	0	10%
Brno_1.0.2_BAD_2016-1-30_21-3	NumberOfNodesAndEdges	edges deleted	SUCCESS	190865 edges, 63029 nodes	190080 edges, 63029 nodes	3139 edges	0	0	10%
Brno_1.0.2_BAD_2016-1-30_21-4	NumberOfNodesAndEdges	nodes and edges deleted	SUCCESS	194004 edges, 63029 nodes	189458 edges, 62168 nodes	4545 edges, 861 nodes	0	0	10%
Brno_1.0.2_BAD_2016-1-30_21-5	NumberOfNodesAndEdges	nodes and edges deleted	SUCCESS	194004 edges, 63029 nodes	189550 edges, 62489 nodes	3630 edges, 432 nodes	0	0	10%
Brno_1.0.2_BAD_2016-1-31_16-9	NumberOfNodesAndEdges	remove pt lines (edges)	SUCCESS	194004 edges, 63029 nodes	193525 edges, 63029 nodes	479 edges	0	0	10%
Brno_1.0.2_BAD_2016-1-31_16-1	NumberOfNodesAndEdges	remove PT mode	SUCCESS	194004 edges, 63029 nodes	193510 edges, 63029 nodes	494 edges	0	0	10%
Brno_1.0.2_BAD_2016-2-11_23-5	NumberOfNodesAndEdges	RemovePTedgeConnection	SUCCESS	194004 edges, 63029 nodes	193982 edges, 63029 nodes	22 edges	0	0	10%

Tabulka 5.7: Ukázka tabulky z přílohy, ze sekce Wrong2 result pro metodu počtu hran a uzlů

Data set ID	Testovací technika	Typ aplikovaného defektu	Status metody	# nalezených chyb v CORRECT grafu	# celkový počet nalezených chyb	# nalezených vytvořených chyb	# nenalezených zavedených chyb	# falešných chyb	rozmezí/ tolerance
Brno_1.0.2_BAD_2016-1-30_20-43	TransportStopStationsGSPCheck	ChangePTedgeConnection	FAILED	6996	6996	0	0	0	20 m
Brno_1.0.2_BAD_2016-1-30_20-51	TransportStopStationsGSPCheck	GPS changed for PT nodes	FAILED	6996	7328	431	1	0	20 m
Brno_1.0.2_BAD_2016-1-30_20-58	TransportStopStationsGSPCheck	GPS changed for road nodes	FAILED	6996	6996	0	0	0	20 m
Brno_1.0.2_BAD_2016-1-30_21-7	TransportStopStationsGSPCheck	changed length of edge	FAILED	6996	6996	0	0	0	20 m
Brno_1.0.2_BAD_2016-1-30_21-16	TransportStopStationsGSPCheck	Change max speed of edge	FAILED	6996	6996	0	0	0	20 m
Brno_1.0.2_BAD_2016-1-30_21-24	TransportStopStationsGSPCheck	removed route edges	FAILED	6996	6996	0	0	0	20 m
Brno_1.0.2_BAD_2016-1-30_21-33	TransportStopStationsGSPCheck	edges deleted	FAILED	6996	6996	0	0	0	20 m
Brno_1.0.2_BAD_2016-1-30_21-41	TransportStopStationsGSPCheck	nodes and edges deleted	FAILED	6996	6996	0	0	0	20 m
Brno_1.0.2_BAD_2016-1-30_21-50	TransportStopStationsGSPCheck	nodes and edges deleted	FAILED	6996	6882	-114	0	0	20 m
Brno_1.0.2_BAD_2016-1-31_16-9	TransportStopStationsGSPCheck	remove pt lines (edges)	FAILED	6996	6996	0	0	0	20 m
Brno_1.0.2_BAD_2016-1-31_16-16	TransportStopStationsGSPCheck	remove PT mode	FAILED	6996	6996	0	0	0	20 m
Brno_1.0.2_BAD_2016-2-11_23-54	TransportStopStationsGSPCheck	RemovePTedgeConnection	FAILED	6996	6996	0	0	0	20 m
Brno_1.0.2_BAD_2016-2-11_23-44	TransportStopStationsGSPCheck	changed length of edge	FAILED	6996	6996	0	0	0	20 m
Brno_1.0.2_BAD_2016-2-12_1-21	TransportStopStationsGSPCheck	GPS changed for PT & Road nodes	FAILED	6996	7245	202	302	0	20 m
Brno_1.0.2_BAD_2016-3-11_9-20	TransportStopStationsGSPCheck	Change Random Stations Name	FAILED	6996	9415	3240	0	0	20 m

Tabulka 5.8: Ukázka tabulky z přílohy, ze sekce Wrong2 result pro metodu GPS souřadnic stanic hroamdné dopravy

5.4.9 Počet hran a vrcholů dle typu

Předchozí metoda porovnává celkové množství hran a vrcholů. Při změně dat mohou vrcholy a hrany přibýt stejně tak jako jinde ubýt. Proto tato metoda zohledňuje i jednotlivé typy hran a vrcholů. Je tedy možné odhalit nejenom úbytek či příbytek vrcholů a hran, ale také jejich změnu. Nalezené chyby touto metodou:

- **Odstranění běžných hran (viz kapitola 4.3.4)**
- **Odstranění hran hromadné dopravy (viz kapitola 4.3.3)**
- **Odstranění běžných uzlů (viz kapitola 4.3.2)**
- **Odstranění uzlů hromadné dopravy (viz kapitola 4.3.1)**
- **Odstranění hromadné dopravy (viz kapitola 4.3.9)**
- **Odstranění linek hromadné dopravy (viz kapitola 4.3.10)**
- **Odstranění spojeni sítě hromadné dopravy s ostatní dopravou (viz kapitola 4.3.13)**

Změna typu hrany způsobí změnu na dvou místech, tudíž je větší šance odhalení chyby. Pro příklad si představme, že hrana patří autobusové dopravě a změní se v obyčejnou hranu (například chybí mód dopravy pro autobusy, či jen jedna linka). Počet hran pro autobusovou dopravu se sníží a zároveň počet obyčejných hran se zvýší. Takto detailnější rozdělení hran může nalézt více nesrovnalostí než metoda předchozí, či více popsat, proč předchozí metoda odhalila chybu a blíže napoví možný zdroj chyb. Vzhledem k vysokým počtům hran a uzlů byla zvolena tolerance 1% pro větší viditelnost nalezených hodnot. Vzhledem k tomu, že změny v linkách a dopravě jsou očekávány, nejsou nalezené nesrovnalosti označovány jako chyby (status 'FAILED'), ale jako varování (status 'WARNING')

5.4.10 Kontrola linek hromadné dopravy

Linky hromadné dopravy mohou přibývat a také ubývat v datech, proto tato metoda poukazuje na rozdíly dle dostupných informací o daných datech a poukazuje na linky, které v datech chybí, či jsou v nich navíc. Metoda (viz kapitola 4.2.10) zareagovala na následující defekty:

- **Odstranění uzlů hromadné dopravy (viz kapitola 4.3.1)** - náhodně odstraněné uzly tvořily celou linku hromadné dopravy
- **Odstranění linek hromadné dopravy (viz kapitola 4.3.10)** - odstraní přímo danou linku
- **Odstranění hromadné dopravy (viz kapitola 4.3.9)** - odstraní mód a tedy i všechny jeho linky

5.4.11 Kontrola názvů stanic hromadné dopravy

Chyby v názvech stanic mohou vést při plánování k nesprávnému výsledku. Proto metoda porovnává všechny názvy stanic s dostupnými informacemi a nalezne stanice mimo daný seznam, či stanice, které oproti seznamu chybí (viz popis metody 4.2.11). Nalezené chyby touto metodou:

- **Odstranění uzlů hromadné dopravy (viz kapitola 4.3.1)** - odstraní dané stanice
- **Změna názvů stanic hromadné dopravy (viz kapitola 4.3.14)** - změní název stanice

Defekty, které odstraňují linky hromadné dopravy, či celé módy dopravy, tato metoda nezachytila. Bylo to z důvodu, že odstranění informací o dopravě probíhá na hranách a informace o stanicích jsou spojeny s vrcholy. Jinými slovy linky, či mód dopravy již v datech neexistují, ale stanice zůstaly na svých místech stát pořád. Tento způsob zavádění defektů byl zvolen, protože na hraně bývá často více informací, než jen jeden mód, či linka dopravy.

5.4.12 Maximální povolená rychlost

Metoda vyhodnotí chybné hrany v případě, když maximální povolená rychlost dané hrany je mimo předpokládaný rozsah (viz popis metody4.2.12). Proto je nezbytné, aby uživatel, který spouští tuto metodu, měl přehled o datech a věděl, v jakém rozmezí se v datech mohou pohybovat rychlosti. Pokud víme, že v dané oblasti neexistují dálnice, či rychlostní silnice, na kterých je rychlost vyšší než 90 km/h, můžeme tuto rychlost zvolit jako maximální. Pro finální testování byl vzhledem k datům zvolen rozsah 0-130 km/h. Při tomto rozsahu neobsahují testovací data bez úprav žádné chyby a jednodušeji se poté porovnávají zavedené chyby při umělých defektech. Nalezené chyby touto metodou:

- **Změna maximální rychlosti hrany (viz kapitola 4.3.11)** - metoda nalezne všechny zavedené chyby mimo požadovaný rozsah

Musí být brát ohled na to, že ne všechny chyby metoda dokáže odhalit. Pokud se v datech změní maximální rychlost, avšak nová rychlost se pohybuje stále v požadovaném rozsahu, metoda není schopna takovouto odchylku odhalit.

5.4.13 Kontrola spojení stanic s pěšími cestami

Metoda označí jako chybu stanici hromadné dopravy, která není spojena s chodníky (viz popis metody 4.2.2). Nalezené chyby touto metodou:

- **Odstranění hran hromadné dopravy (viz kapitola 4.3.3)** - po odstranění hran, které spojují stanice s chodníky
- **Odstranění běžných uzlů (viz kapitola 4.3.2)** - uzly, na které se napojují spojovací hrany

- **Odstranění uzlů hromadné dopravy (viz kapitola 4.3.1)** - uzly pro spojení mezi sítěmi hromadné dopravy a cestami
- **Odstranění spojeni sítě hromadné dopravy s ostatní dopravou (viz kapitola 4.3.13)** - odstraní přímo dané spojení

Z vyhodnocení (příloha Testing results) je patrné, že pro všechny spuštěné testy je výsledný status metody neúspěšný ('FAILED'). Po detailní kontrole byl vyhodnoceno, že tento status je správný, neboť rozloha dat pro vstupní graf se liší pro hromadnou dopravu a pro silniční síť. Hromadná doprava často zasahuje mimo město a kvůli ucelenosti informací, nejsou tedy stanice mimo město spojeny se silniční sítí, tedy chodníky. Pro vyhodnocení výsledků je uživatel nucen přihlédnout ke skutečnostem známým z předešlých verzí dat, či při kontrole zohlednit polohu stanic, zda není stanice mimo město a nesrovnalost tím pádem není tak závažná, neboť hlavní kalkulace pro případný plánovač se budou konat jen na území města.

Kapitola 6

Závěr

Plánovač tras v dnešní době používá téměř každý. Důkladné testování vstupních dat pro plánovače zaručí vetší spolehlivost navrhovaných cest, a tím sníží případné potíže uživatelů. Před samotnou implementací byla nastudována struktura vstupních dat, aby implementované techniky kontrolovaly data na správných místech. Vstupní data ze SUPERHUB obsahují spoustu informací o dopravě a zahrnují tak velkou škálu možností dopravy. S velkou strukturou ovšem také přichází vyšší možnost chyb v datech.

Jednotlivé testovací techniky byly implementovány jako samostatné moduly, které lze spouštět jednotlivě, či všechny naráz v jednom testovacím setu. Toto řešení umožňuje snadné přidání nových testovacích technik, které otestují další informace v datech, a tím zvýší kvalitu testovaných dat ještě více. Pro budoucí implementaci a testování nových metod je vytvořen také systém pro generování chyb, který náhodně vygenerované defekty uloží. Takto uložené informace poté mohou být porovnávány s výstupy testovacích metod pro ověření, jak účinná testovací technika je. Pro usnadnění zobrazení výsledků a spouštění jednotlivých funkcí programu bylo vytvořeno webové rozhraní. Pomocí tohoto rozhraní uživatel může procházet výsledky z již proběhlých testů, spouštět nové testy a podobně.

Značnou část práce tvoří samotné vyhodnocení testů. Vzhledem k velikosti vstupních dat a počtu implementovaných metod a generátoru chyb trvalo spuštění jedné sady testů řádově hodiny. Následné zpracování výsledků bylo časově náročné, neboť kontrola musela být důkladná, aby metody mohly být označeny jako funkční. Dle výsledků testů lze říci, že implementované metody odhalí daný typ defektů, na který jsou navrženy. Zároveň lze prohlásit, že testovací techniky neoznačí žádné falešné chyby, které by označit neměly.

Byl vytvořen systém pro testování kvality dat pro plánovače tras, který obsahuje modulární testovací techniky a generátory umělých chyb. Samotný program, stejně jako i jeho výsledky, mohou být ovládány z implementovaného webového rozhraní pro usnadnění práce uživatele. Navržený model struktury aplikace se tedy podařilo realizovat a následně otestovat nad všemi dostupnými datovými sety.

Literatura

[1] *Apache Tomcat* [online]. Dostupné z: http://tomcat.apache.org/.

[2] *Ecplise - dokumentation and tutorials* [online]. Dostupné z: https://eclipse.org.

[3] *Google Places API Web Service* [online]., [cit. 10.2.2016]. Dostupné z:https://developers.google.com/places/web-service/?hl=cs.

[4] *Wiki Books LaTeX/Internationalization* [online]., [cit. 20.2.2016]. Dostupné z: http://en.wikibooks.org/wiki/LaTeX/Internationalization.

[5] *Směrnice děkana pro závěrečné prácel* [online]., [cit. 25.3.2016]. Dostupné z: https://www.fel.cvut.cz/cz/rozvoj/smerniceSZZ.pdf.

[6] *Latex - online manuál* [online]., [cit. 25.5.2016]. Dostupné z: http://www.cstug.cz/old/latex/lm/frames.html/.

[7] *Cacoo - nástroj pro vytvýření diagramů* [online]., [cit. 26.5.2016]. Dostupné na: https://cacoo.com/.

[8] *jQuery JSONView* [online]., [cit. 26.5.2016]. Dostupné z: https://github.com/yesmeck/jquery-jsonview/.

[9] *SUPERHUB* [online]., [cit. 26.5.2016]. Dostupné z: http://www.scmagazine.cz/media/W1siZiIsIjIwMTYvMDEvMDUvMTAvNTEvMDQvMGFiMGViN2UtZDA5Mi00OGVlLWE5N2YtZTE4MzcwZDUzNTIzL1NVUEVSSFVCLnBkZiJdXQ/0212052e8033aa13/SUPERHUB.pdf.

[10] I. Carreras, S. Gabrielli, D. Miorandi, A. Tamilin, F. Cartolano, M. Jakob, and S. Marzorati. SUPERHUB: A user-centric perspective on sustainable urban mobility. In *6th ACM workshop on Next generation mobile computing for dynamic personalised travel planning*, pages 9–10. ACM, 2012.

[11] Jason Brittain; Ian F. Darwin. *Programming JavaScript Applications*. O'Reilly Media, 2014.

[12] Eric Elliott. *Tomcat: The Definitive Guide*. O'Reilly Media, 2007.

[13] The Foundation Level Working Group. *Foundation Level Syllabus (2011)* [online]., [cit. 14.8.2015]. Dostupné z: http://www.istqb.org/downloads/finish/16/15.html.

[14] Jan Hrncir and Michal Jakob. Generalised time-dependent graphs for fully multimodal journey planning. In *IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2138–2145. IEEE, 2013.

[15] Michal Jakob, Jan Hrncir, Luis Oliva, Francesco Ronzano, Pavol Zilecky, and Jason Finnegan. Personalized Fully Multimodal Journey Planner. In *Prestigious Applications of Intelligent Systems (PAIS)*, pages 1225–1226, 2014.

[16] Josef Kolář. *Teoretická informatika.* Česká technika - nakladatelství ČVUT, 1st edition, 2009.

[17] Tim Koomen, Leo van der Aalst, Bart Broekman, and Michiel Vroon. *TMap Next, for result-driven testing.* UTN Publishers, 2006.

[18] Francesco Marchioni. *MongoDB for Java Developers.* Packt Publishing, 2015.

[19] Glenford J. Myers, Corey Sandler, and Tom Badgett. *The Art of Software Testing.* Wiley Publishing, 3rd edition, 2011.

[20] Poornachandra Sarang. *Java Programming (Oracle Press).* McGraw-Hill Education, 2011.

Příloha A

Manuál aplikace

Uživatelský manuál k uživatelskému rozhraní pro nástroj k testovaní dat pro dopravní plánování

Simon Slováček

Obsah

Kapitola 1

O aplikaci

MongoWebEditor je webové rozhraní sloužící jako nástroj pro odborné uživatele, kteří pracují se systémem pro testování kvality dat pro dopravní plánování. Webový editor má své rozhraní a nastavení oddělené od samotného testovacího programu a volá jen hlavní funkce programu. Samotný testovací program je přiložen jako knihovna, kterou webové rozhraní volá přes Servlety.

Webové rozhraní umožní uživateli prohlížet výsledky testů uložených v Mongo databázi. Dále přes něj uživatel může načíst nové data sety pro testování a spouštět nad nimi testovací techniky.

Kapitola 2

Instalace

Aplikace je primárně určena a testována za pomocí Tomcatu (konkrétně testováno pro Tomcat v 8). Žádné speciální nastavení pro Tomcat není nutné. Aplikace byla testována v systému Windows.

Pro správnou funkci programu je nutné mít následující předpoklady:

- Nainstalovaná Java - testováno pro v. 1.7 a 1.8
- Přístup k internetu - nutno pro připojení k Mongo DB

Doporučený postup pro instalaci programu je zkopírování celého souboru dat na souborový systém. V souboru projektu naleznete Tomcat obsahující již daný program a strukturu databáze na souborovém systému, kam aplikace ukládá data.

2.1 Nastavení parametrů

Pro fungování aplikace je důležité nastavit parametry v adresářích, které se nacházejí v cestě $TOMCAT\_DIRECTORY/webapps/MongoWebEditor/WEB-INF/classes/$. Pokud jste aplikaci dostali jen jako .war soubor, je nutné tyto soubory nahradit uvnitř tohoto souboru. Soubory s parametry jsou:

- applicationProp.properties
- mongodb.properties

2.1.1 Aplikační parametry

Soubor applicationProp.properties obsahuje informace o cestě k databázi na souborovém systému. Před puštěním aplikace je nutné tuto cestu nastavit. Cesta musí vést do kořenového adresáře *DB* a mít dvojitá lomítka v cestě (viz obr. 2.1).

Dále konfigurační soubor obsahuje klíč ke Google API, který je potřeba mít funkční pro správné fungování metody kontroly GPS souřadnic u stanic hromadné dopravy. Klíč uvedený v souboru

(obr. 2.1) je zaregistrovaný pod autorem této práce a je určen pro vývoj a testování aplikace a není zaručeno, že klíč bude fungovat v budoucnu.

```
path=C:\\Users\\sslovace\\Desktop\\MyW\\DP\\Kod\\DB\

googleKey=AIzaSyCFnWkaA7NbzLyxNcmgizPXK35Kb5vsKCM
```

Obrázek 2.1: applicationProp.properties

2.1.2 Mongo DB parametry

Soubor mongodb.properties obsahuje informace o připojení k Mongo DB databázi, kde se ukládají výsledky testů a jiné pomocné informace (obr. 2.2). Základní údaje obsahují připojení k databázi, kterou vlastní ČVUT a slouží pro účely diplomové práce autora tohoto manuálu.

```
host=its.felk.cvut.cz
port=27017
database=testingframework
user=testingframeworkuser
password=mnnjsbatfs8iiht3
```

Obrázek 2.2: applicationProp.properties

Kapitola 3

Start aplikace

V systému Windows se spustí Tomcat server pomocí souboru $TOMCAT\_DIRECTORY/bin/startup.bat$. Po spuštění server poběží v otevřeném okně, kde jsou také vidět logy aplikace.

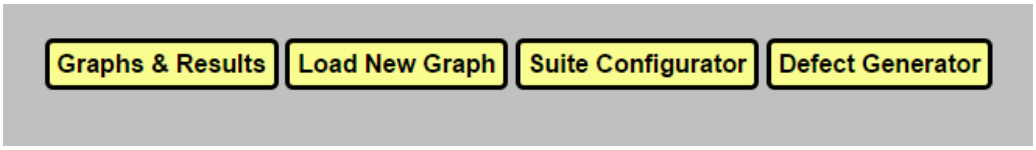
Pro používání webového rozhraní může uživatel použít libovolný browser (testováno převážně v Chrome). Pokud necháme základní nastavení tomcatu, stačí zadat url
http://localhost:8080/MongoWebEditor/

Kapitola 4

Funkce aplikace

Webové rozhraní umožňuje uživateli ovládat následující funkce (obr. 4.1):

- **Graphs & Results** - zobrazení výsledků uložených v mongo DB
- **Load New Graph** - načte nový datový set ze SUPERHUB a uloží ho na lokální DB
- **Suite Configurator** - spouštění testů nad daty
- **Defect Generator** - vytvoření nových datových souborů se zavedenými umělými defekty



Obrázek 4.1: Úvodní stránka

4.1 Graphs & Results

V této sekci aplikace nejprve vidíme seznam grafů v databázi (obr. 4.2). Po kliknutí na daný graf se nám zobrazí informace uložené k v databázi pro daný graf (obr. 4.3). Po vybrání konkrétních dat se zobrazí uložený JSON s detailními informacemi (obr. 4.4).

Used Graphs

ID	City	Version	Type	Date
Brno_1.0.0_GOOD_2016-1-13_12-50	Brno	1.0.0	GOOD	Wed Jan 13 2016 12:50:38
Praha_1.0.0_GOOD_2016-1-13_12-59	Praha	1.0.0	GOOD	Wed Jan 13 2016 12:59:20
Milan_1.0.0_GOOD_2016-1-13_13-23	Milan	1.0.0	GOOD	Wed Jan 13 2016 13:23:12
Barcelona_1.0.0_GOOD_2016-1-13_13-20	Barcelona	1.0.0	GOOD	Wed Jan 13 2016 13:20:02
Helsinki_1.0.0_GOOD_2016-1-13_13-29	Helsinki	1.0.0	GOOD	Wed Jan 13 2016 13:29:33
Brno_1.0.0_BAD_2016-1-27_0-1	Brno	1.0.0	BAD	Wed Jan 27 2016 00:01:49

Obrázek 4.2: Graphs & Results

Name	Date	Comments
Defect description	Sat Feb 27 2016 00:05:00	
Test Suite	Wed Jan 27 2016 00:09:52	

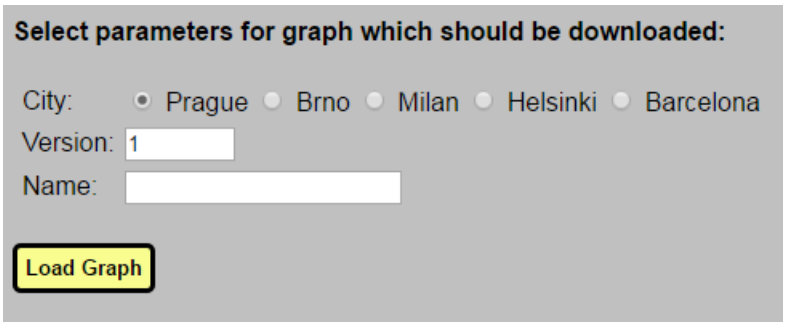
Obrázek 4.3: Informace v DB pro daný graf

```
{
  _id: {
      $oid: "56a7fcc04937af915889a936"
  },
  name: "Test Suite",
  date: 1453849792358,
  modulesResults: [
    {
        name: "cz.agents.cvut.testModule.modules.strongComponent.GraphStrongComponent",
        status: "SUCCESS",
        summary: "Graph Strong Component module successfullly completed /n Components: 2492",
      ▸ resultParams: {  2 items  },
      ▸ resultOK: {  1 item  },
        resultWARN: {},
        resultERROR: {}
    },
    {
        name: "cz.agents.cvut.testModule.modules.onewayPavements.GraphConstraintOnewayPavements",
        status: "SUCCESS",
        summary: "Number of found one way pavements: 0",
      ▸ resultParams: {  1 item  },
        resultOK: {},
        resultWARN: {},
        resultERROR: {}
    },
```


Obrázek 4.4: JSON s detailními informacemi

4.2 Load New Graph

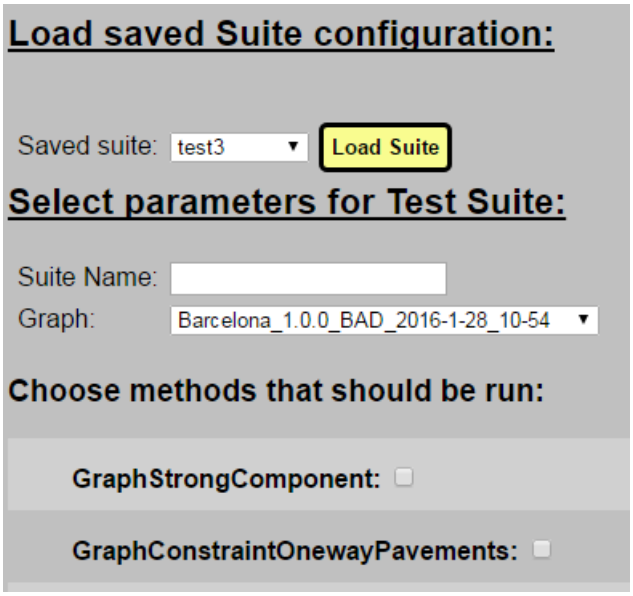
V této části aplikace může uživatel stáhnout aktuální verzi dat ze SUPERHUB a dát datovému setu název a verzi (obr. 4.5).



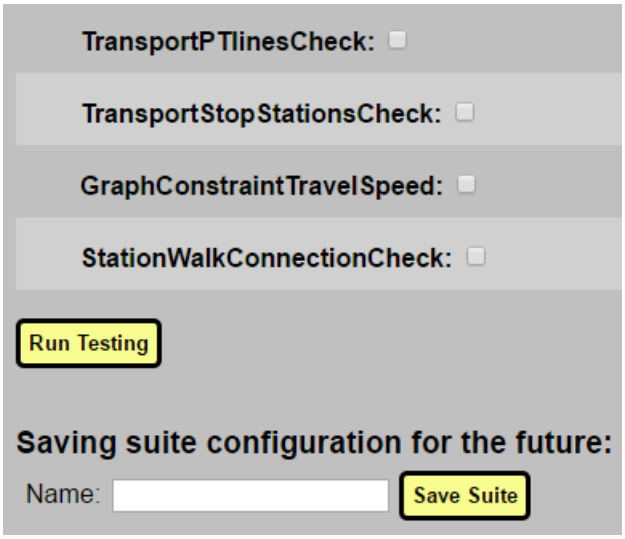
Obrázek 4.5: Informace v DB pro daný graf

4.3 Suite Configurator

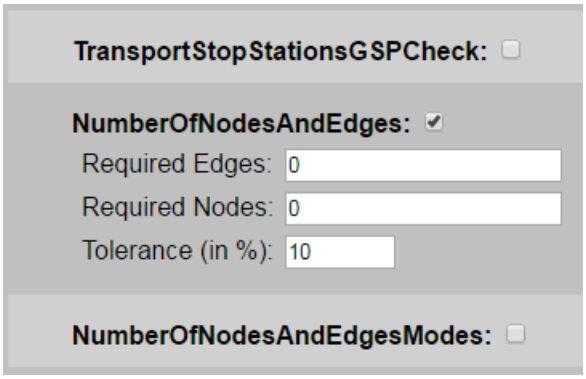
Uživatel zde vybere vstupní graf a testovací metody, které chce nad daty spustit (obr. 4.6 a 4.7). Dále lze nastavenou konfiguraci uložit (obr. 4.7) pro příští použití. Načtení uložené konfigurace lze také provést na této stránce (obr. 4.6). Pro vybrání testovací metody se musí zaškrtnout checkbox u dané metody. Pokud metoda vyžaduje vstupní parametry, objeví se a uživatel je musí vyplnit (obr. 4.8)



Obrázek 4.6: Suite konfigurator první část



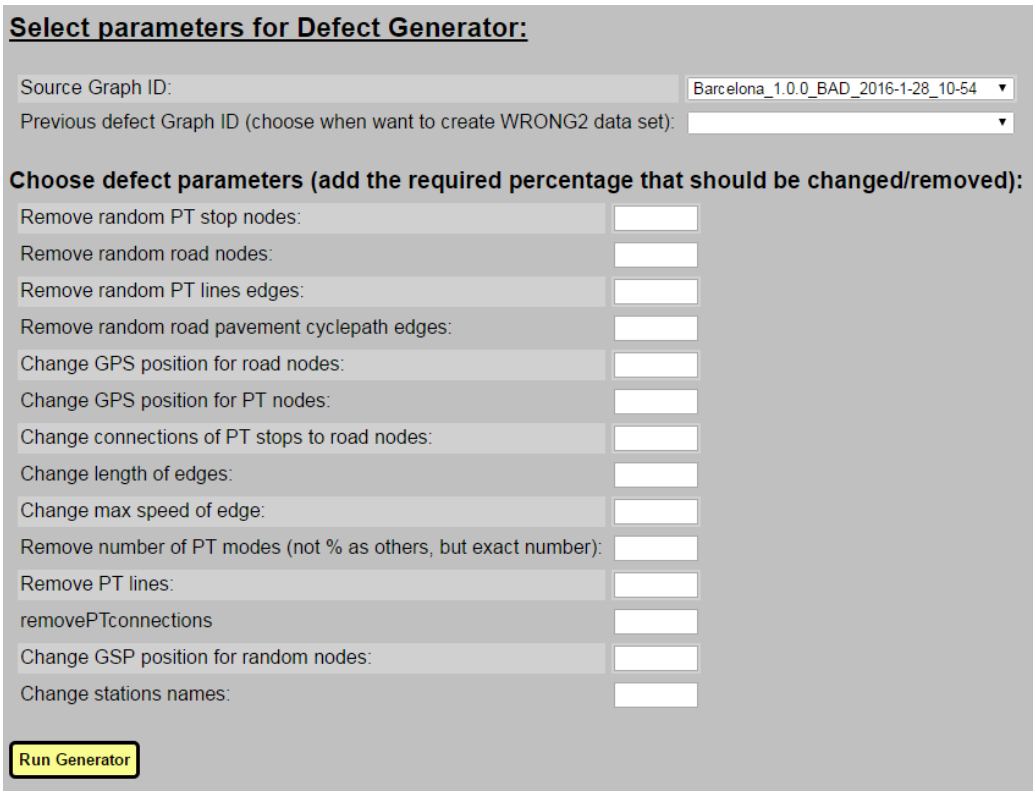
Obrázek 4.7: Suite konfigurator druhá část



Obrázek 4.8: Ukázka vstupních parametrů pro metodu počtu uzlů a hran

4.4 Defect Generator

Uživatel pomocí tohoto rozhraní může vygenerovat nové datové sety se zavedenými chybami, dle zvolených typů chyb (obr. 4.9).



Obrázek 4.9: Stránka pro generování umělých chyb

Příloha B

Obsah CD

- Eclipse projekt se zdrojovým kódem aplikace
- Tomcat se spustitelnou verzí aplikace
- Struktura souborového systému se základními daty
- Digitální podoba této práce ve formátu PDF
- Testing results - tabulky popisující nastavení a výsledky testů
- Manuál uživatelského rozhraní