Dokumentation zu Flächennutzungsregeln

Gruppe InMa

FABIAN PFLUG | PETER ZILZ

January 15, 2015

Leibniz Universität Hannover

Zusammenfassung

Um Flächen effektiv und möglichst genau zu einem Verbotsschild und dessen ungefährer Position zuzuweisen wird ein Algorithmus erdacht, welcher zu der Menge der Verbote eine Regelmenge definiert und mithilfe dieser das im Umkreis bestmögliche Polygon aus dem Daten von OpenStreetMap auswählt oder eins erstellt, welches dem Geltungsbereich der Nutzungsregel entsprechen soll.

Inhaltsverzeichnis

Kapitel 1

Bedienungsanleitung

Man braucht folgende Daten: Data.txt, rules.xml. Außerdem ist eine Internetverbindung zu openstreetmap.org nötig. Eine Manpage lässt sich mit `man -l inma.1` aufrufen.

1.1 Eingabedaten

Die Data.txt enthält die wichtigsten Eingabedaten. Sie ist genauso aufgebaut wie die im Testdatensatz zum Wettbewerb bereitgestellte Datei. In der ersten Zeile steht die Anzahl der folgenden Zeilen. Jede Zeile enthält am Ende eine Flächennutzungsregel. Am Beginn der Zeile steht die Nummer des Testdatensatzes, zu dem die Regel gehört. Gefolgt wird diese Nummer von einem Koordinatenpaar, das den Standort der entsprechneden Verbotsschilder beschreibt.
Die Rules.xml enthält die Bewertungsregeln, die das Programm benötigt. (Für deren Aufbau siehe unten.)

1.2 Ausführung

Die Standardausführung sieht folgendermaßen aus:
`java -jar InMa.jar --data Data.txt --rules Rules.xml`

1.3 Optionen

-h, –help
: Shows the help text.

-d, –data FILE
: Specifies the file containing the space usage rules and their locations.

-r, –rules FILE
: Specifies the file containing rules for evaluating the environment of the location.

-t, –threads NUMBER
: Specifies the number of threads.

-u, –overlap FILE
: Specifies the file containing overlap values, that should be reached to be successfull.

-o, –outputDir PATH
: Specifies the folder where output files will be stored. Can be overwritten for the images by -i.

-p, –path PATH
: Specifies the folder where the following files are: Data.txt, Rules.xml and optional Overlap.txt. -d, -r and -u are not needed and can overwrite -p.

-i, –image [PATH]
: Images are created to visualise the processed data. An output directory is optional.

-l, –height NUMBER

Specifies the height of the images.

-w, –width NUMBER

Specifies the width of the images.

1.4 Ausgabe

Als Ausgabe erhält man kml-Dateien, die die errechneten Gebiete enthalten. Sie können mit Google-Earth betrachtet werden.

Kapitel 2

Anforderungen

2.1 Lastenheft

Was der “Kunde” ausgeschrieben hat, das er haben will.
Aus der Aufgabenstellung des Wettbewerbs und aus den Kriterien zur Auswertung, die der Veranstalter online gestellt hat, kann man verschiedene geforderte Bestandteile herauslesen.
Aus der Aufgabenstellung für die erste Runde:

- bereitgestellten Testdaten herunterladen
- intuitives Regelwerk erstellen
- eine Software implementieren, die
  - einen Geltungsbereich errechnet
  - als Eingabe SURs und Geokoordinaten hat
  - Polygone als Ausgabe hat
  - erweiterbar für zusätzliche SURs
- die Testdaten mit der implementierten Software auswerten
- die Ergebnisse im beschriebenen KML-Format speichern
- Ergebnisse plausibilisieren

Aus der Aufgabenstellung für die zweite Runde:

- selbst Datensätze erstellen
  - die mindestens 10 verschiedene SURs enthalten
  - sie in unserer Umgebung erstellen
  - spezifiziertes Format für die IDs der SURs benutzen
  - Fotos der zugehörigen Informationsschilder beifügen
  - einen intuitiven Geltungsbereich angeben
  - KML-Format und Endung *[ID].truth.kml* benutzen
- eigene Datensätze mit der eigenen Software auswerten
- Korrektheit und Präzision der Ergebnisse diskutieren
  - von den eigenen Datensätzen und vom Testdatensatz
  - brechnete und intuitive Geltungsbereiche vergleichen

- intuitiv erstellte Geltungsbereiche dürfen nicht zur Berechnung benutzt werden

Weiteres aus der Aufgabenstellung:

- Dokumentationen
  - Bedienungsanleitung
  - Installationsanleitung
  - Entscheidungen in der Software-Entwicklung und über die Auswahl von Algorithmen und Datenstrukturen dokumentieren
- Verwendung von Open Street Map
- die Software soll mit kniffligen Fällen umgehen können

Aus den Kriterien zur Auswertung:

- Theoretischer Ansatz
  - Recherche
  - Modellierung
  - Komplexitätsanalyse
  - Aufwandsabschätzung
- Dokumentation
  - Konsitenz, Vollständigkeit und Nachvollziehbarkeit des Berichts
  - Gliederung und Layout des Berichts
  - Codedokumentation
- Softwaretechnik
  - klare Beschreibung der Anforderungen, Entwurfsüberlegungen und -entscheidungen
  - klare Beschreibung der Tests, Ergebnisse und Schlussfolgerungen daraus
  - Güte der Vorgehensweise, der Entwurfsentscheidungen und des Testkonzepts
- Benutzerschnittstelle
  - Verständlichkeit und Übersichtlichkeit (usability)
  - spannende GUI-Ideen (visibility)
- Funktionalität
  - funktionale Korrektheit und Güte der Lösung (im Sinne der Aufgabenstellung)
  - Nachprüfbarkeit der Testergebnisse

Kapitel 3

Fachspezifikation

3.1 Ziel des Projekts

3.1.1 Erläuterung des zu lösenden Problems

Die Software wird zur Teilnahme am Wettbewerb “Informaticup 2015”, der von der Gesellschaft für Informatik ausgreichtet wird, hergestellt.
Die Hauptaufgabe der Software ist es, für eine Position, an der sich ein Informationsschild befindet, einen Geltungsbereich zu berechnen. Hierzu können die öffentlich zugänglichen Daten aus Open Street Map benutzt werden.

3.1.2 Wünsche und Prioritäten des Kunden

Die Veranstalter des Wettbewerbs legen Wert auf eine gute Nachvollziehbarkeit und Dokumentation der Software. Das abgegebene Programm muss funktionieren, d.h. nicht abstürzen. Es muss aber nicht zwingend eine Erfolgsquote von hundert Prozent einfahren. Die Dokumentation der Software und des Entstehungsprozesses sind von höherer Priorität.

3.1.3 Domänenbeschreibung

Da es sich bei diesem Projekt um eine Teilnahme an einem Wettbewerb handelt, kann man keinen Einsatzbereich für die Software angeben. Insofern kommt das Programm hauptsächlich bei der Bewertung durch die Jury zum Einsatz.
Ein denkbares Szenario für den Einsatz ist das Kartieren von Flächennutzungsregeln. So muss zum Beispiel ein Nutzer nicht per Hand den Geltungsbereich eines Rauchverbots in die Karte eintragen. Das Programm errechnet automatisch aus der einer Position einen Bereich, den der Nutzer bestätigen oder verbessern kann.

3.1.4 Maßnahmen zur Anforderungsanalyse

Unser erster Schritt, um uns mit dem Problem vertraut zu machen, war es den bereitgestellten Testdatensatz zu betrachten. (Genaueres siehe unten.)

3.2 Rahmenbedingungen und Umfeld

3.2.1 Einschränkungen und Vorgaben

Bei Eingabedaten ist das selbstdefinierte Format der GI zu übernehmen. Bei Ausgabedaten ist das KML-Format vorgeschrieben. Nutzung von Open Street Map als Kartenmaterial und Java als primäre Programmiersprache.

3.2.2 Anwender

Die Jury ist nach derzeitigem Stand die einzige Anwendergruppe. Darüber hinaus könnte das Programm auf mobilen Geräten Anwendung finden.

3.2.3 Schnittstellen und angrenzende Systeme

Die Software baut auf dem Kartendienst Open Street Map auf.
Die Kriterien zur Berechnung des Geltungsbereichs werden in einer Datei gespeichert. Diese ließe sich auch von anderer Software nutzen. Zum Beispiel, um die darin beschriebenen Regeln automatisch zu erzeugen.

3.3 Funktionale Anforderungen

3.3.1 Use Case-Beschreibungen

1: Einfaches Benutzen der Software in der Komandozeile, d.h. mit den Parametern –data, –rules und –outputDir.
1.1: Zusätzlich mit dem Parameter –threads
1.2: Nur mit dem Parameter –path
2: Einfaches Benutzen mit Fehlervisualisierung. 2.1: Zusätzlich mit dem Parameter –overlap 3: Nur mit dem Parameter –help

Use Case Nr. 1	Standardausführung
Umfeld	Komandozeile
Systemgrenzen	Komandozeile
Ebene	komplette Programmbenutzung
Hauptakteur	Benutzer
Stakeholder u. Interesse	Benutzer will einen Datensatz berechnen lassen.
Voraussetzung	Internetverbindung zu OSM. Angegebene Verzeichnisse existieren. Data.txt und Rules.xml befinden sich an angegebener Stelle.
Garantien	Sind OSM-Daten unvollständig, wird ein kreisförmiger Geltungsbereich angegeben. Sind SURs unbekannt, gibt es dennoch eine Ausgabe.
Erfolgsfall	Es wird eine kml-Datei mit dem berechneten Lösungspolygon im angegebenen Verzeichnis erstellt.
Auslöser	Der Befehl wird abgeschickt.

Beschreibung	Ein normaler Programmablauf. Folgender Befehl wird abgeschickt.
	`java Start --data <file> --rules <file>`
	`--ouputDir <path>`
	Daten werden von OSM herunter geladen.
	Regeln werden angewendet.
	Lösungen werden im KML-Format gespeichert.

Erweiterungen	Mit zusätzlichen oder anderen Parametern. Siehe Use Cases 1.1 bis 1.6
Technologie	-

Use Case Nr. 1.1	Multithreading
Umfeld	Komandozeile
Systemgrenzen	Komandozeile
Ebene	komplette Programmbenutzung
Hauptakteur	Benutzer
Stakeholder u. Interesse	Benutzer will einen Datensatz berechnen lassen und Zeit sparen, indem die Arbeit auf mehrere Threads verteilt wird.
Voraussetzung	Internetverbindung zu OSM. Angegebene Verzeichnisse existieren. Data.txt und Rules.xml befinden sich an angegebener Stelle.
Garantien	Sind OSM-Daten unvollständig, wird ein kreisförmiger Geltungsbereich angegeben. Sind SURs unbekannt, gibt es dennoch eine Ausgabe.
Erfolgsfall	Es wird eine kml-Datei mit dem berechneten Lösungspolygon im angegebenen Verzeichnis erstellt. Alles in kürzerer Zeit.
Auslöser	Abschicken des Befehls

Beschreibung	`java Start --data <file> --rules <file>`
	`--ouputDir <path> --threads <int>`
	Threads werden erstellt und arbeiten.

Erweiterungen	-
Technologie	-

Use Case Nr. 1.2	verkürzte Standardausführung
Umfeld	Komandozeile
Systemgrenzen	Komandozeile
Ebene	komplette Programmbenutzung
Hauptakteur	Benutzer
Stakeholder u. Interesse	Benutzer will einen Datensatz berechnen lassen.
Voraussetzung	Internetverbindung zu OSM. Angegebene Verzeichnisse existieren. Data.txt und Rules.xml befinden sich an angegebener Stelle.
Garantien	Sind OSM-Daten unvollständig, wird ein kreisförmiger Geltungsbereich angegeben. Sind SURs unbekannt, gibt es dennoch eine Ausgabe.
Erfolgsfall	Es wird eine kml-Datei mit dem berechneten Lösungspolygon im angegebenen Verzeichnis erstellt.
Auslöser	Der Befehl wird abgeschickt.
Beschreibung	Folgender Befehl wird abgeschickt. `java Start --path <path>` Daten werden von OSM herunter geladen. Regeln werden angewendet. Lösungen werden im KML-Format gespeichert.
Erweiterungen	-
Technologie	-

Use Case Nr. 2	Regelüberprüfung
Umfeld	Komandozeile
Systemgrenzen	Komandozeile
Ebene	komplette Programmbenutzung
Hauptakteur	Benutzer
Stakeholder u. Interesse	Benutzer will die bisherigen Regeln anhand von Grundwahrheiten überprüfen.
Voraussetzung	Internetverbindung zu OSM. Angegebene Verzeichnisse existieren. Data.txt und Rules.xml befinden sich an angegebener Stelle. Es befinden sich ID.truth.kml Dateien im gleichen Verzeichnig wie Data.txt
Garantien	Sind OSM-Daten unvollständig, wird ein kreisförmiger Geltungsbereich angegeben. Sind SURs unbekannt, gibt es dennoch eine Ausgabe.
Erfolgsfall	Es wird eine kml-Datei mit dem berechneten Lösungspolygon im angegebenen Verzeichnis erstellt. Für jedes Polygon, keine Überlappung von mindestens 95% mit dem Lösungspolygon aufweist werden 2 Bilder erstellt, welche den Fehler visualisieren.
Auslöser	Der Befehl wird abgeschickt.
Beschreibung	Folgender Befehl wird abgeschickt. `java Start --data <file> --rules <file> --ouputDir <path>` Daten werden von OSM herunter geladen. Regeln werden angewendet. Bilder zu den unzureichenden Lösungen werden gespeichert. Lösungen werden im KML-Format gespeichert.
Erweiterungen	siehe 2.1
Technologie	-

Use Case Nr. 2.1	Regelüberprüfung mit varierender Genauigkeit.
Umfeld	Komandozeile
Systemgrenzen	Komandozeile
Ebene	komplette Programmbenutzung
Hauptakteur	Benutzer
Stakeholder u. Interesse	Benutzer will die bisherigen Regeln anhand von Grundwahrheiten überprüfen.
Voraussetzung	Internetverbindung zu OSM. Angegebene Verzeichnisse existieren. Data.txt, Rules.xml und overlap.txt befinden sich an angegebener Stelle. Es befinden sich ID.truth.kml Dateien im gleichen Verzeichnig wie Data.txt
Garantien	Sind OSM-Daten unvollständig, wird ein kreisförmiger Geltungsbereich angegeben. Sind SURs unbekannt, gibt es dennoch eine Ausgabe.
Erfolgsfall	Es wird eine kml-Datei mit dem berechneten Lösungspolygon im angegebenen Verzeichnis erstellt. Für jedes Polygon, keine Überlappung von mindestens 95% oder in overlap.txt angegebenem Wert mit dem Lösungspolygon aufweist werden 2 Bilder erstellt, welche den Fehler visualisieren.
Auslöser	Der Befehl wird abgeschickt.
Beschreibung	Folgender Befehl wird abgeschickt. `java Start --data <file> --rules <file> --ouputDir <path>` Daten werden von OSM herunter geladen. Regeln werden angewendet. Bilder zu den unzureichenden Lösungen werden gespeichert. Lösungen werden im KML-Format gespeichert.
Erweiterungen	-
Technologie	-

Use Case Nr. 3	Hilfetext
Umfeld	Kommandozeile
Systemgrenzen	Kommandozeile
Ebene	komplette Programmausführung
Hauptakteur	Benutzer
Stakeholder u. Interesse	Der Benutzer möchte sich den Hilfetext ausgeben lassen.
Voraussetzung	-
Garantien	Der Hilfetext wird angezeigt.
Erfolgsfall	Der Hilfetext wird angezeigt.
Auslöser	Abschicken des Befehls
Beschreibung	Folgender Befehl wird abgeschickt. `java Start --help` Der Hilfetext wird angezeigt.
Erweiterungen	-
Technologie	-

3.3.2 Technische Anforderungen

Besondere technische Anforderungen gibt es nicht.

3.4 Qualitätsanforderungen

3.4.1 Qualitätsziele des Projekts

3.4.1.1 Funktionalität

Bedeutet hier, dass die funktionalen Anforderungen entsprechend der Aufgabenstellung vollständig und korrekt implementiert werden.

3.4.1.2 Zuverlässigkeit

Es wird garantiert, dass eine fehlende Internetverbindung, ein nicht vorhanden Sein der Eingabedateien oder fehlerhafte Eingabedateien nicht zu unbestimmtem Verhalten führen.

3.4.1.3 Benutzbarkeit

Das Programm soll für unerfahrene Benutzer durch wenige Komandozeilenparamenter bedienbar sein. Für erfahrene Benutzer soll es verschiedene Einstellmöglichkeiten bieten.
Durch selbsterklärende Ausgaben und Hilfetexte soll das Programm leicht benutzbar sein.

3.4.1.4 Effizienz

Daten, welche von Open Street Map herunter geladen werden, können lokal zwischengespeichert werden, um so die Belastung der Server und das Datenvolumen zu reduzieren.
Die Laufzeit kann durch mehrere Threads reduziert werden.

3.4.1.5 Wartbarkeit

Designentscheidungen sollen dokumentiert, APIs zur Verfügung gestellt werden.

3.4.1.6 Portabilität

Das Programm sollte auf verschiedenen Plattformen laufen, aber es ist nicht erforderlich, dass dies außergewöhnliche Plattformen einschließt.

3.4.1.7 Robustheit

Neue oder unerwartete Space Usage Rules führen nicht zum Absturz, da es einen Fallbackalgorithmus gibt. Wenn Kombinationen von SURs nicht explizit behandelt werden, so wird die größtmögliche Teilmenge abgedeckt.

3.4.2 Qualitäts-Prioritäten des Kunden

Es wird Wert auf eine umfangreiche Dokumentation gelegt.

3.4.3 Wie Qualitätsziele erreicht werden sollen

Wir benutzen JUnit, um die Funktionalität zu gewährleisten.
Um die Qualität der Dokumentation zu sichern, orientieren wir uns an anderen gut dokumentierten Softwareprojekten.

3.5 Probleme und Risiken

Bei Zeitproblemen werden fehlerhafte Eingaben nicht mehr abgefangen. Oder auf die Teilnahme am Wettbewerb wird verzichtet.

3.6 Optionen zur Aufwandsreduktion

3.6.1 Inkrementelle Arbeit

Mögliche Erweiterungen werden nur bei genügend Zeit implementiert:

- visualisierung der Metadaten
- automatisierte Regelerstellung aus truth-Polygonen
- weitere Kartendienste außer Open Street Map

Kapitel 4

Erste Runde

4.1 Grundüberlegungen

Aus den Beispielen im Text lässt sich ableiten, dass es sinnvoll ist 'nicht rauchen' Gebiete eher in Bereiche zu legen, welche Gebäude representieren, oder alternativ Parkplätze bei der Berechnung auszublenden.

4.1.1 Testdatensatz

Zunächst wurde der Testdatensatz mithilfe von Google Maps, Google Street View, Google Earth und OpenStreetMap auf Richtigkeit, Machbarkeit der Lösungen und eventuelle Fehler überprüft. Im Testdatensatz lassen sich etliche Fehler und Unvollständigkeiten finden. So sind für die Datensätze:

- 33 (nur teilweise)
- 34 (nur teilweise)
- 36
- 38
- 39
- 41
- 43 - 45
- 49 (nur teilweise)
- 50 (nur teilweise)
- 53
- 55 - 62

keine Representationen der beabsichtigten Geltungsbereiche in OpenStreetMap vorhanden.

Weitere Fehler sind:

- 16 Der Bereich müsste größer sein.
- 33 Sollte den selben Bereich abdecken wie 24.
- 40 Es ist kein Häuserblock gemeint, sonder nur ein einzelnes Haus auf der gegenüberliegenden Strassenseite.
- 43 Der reale Bereich ist etwas verschoben.
- 51 Der reale Bereich liegt auf der gegenüberliegenden Strassenseite.
- 55 Das gewählte Gebäude ist falsch. Das reale befindet sich weiter westlich.
- 57 Das Bild zeigt das Gebäude, welches bei Bild 41 abfotografiert wurde.

- 59 Das Bild suggestiert, dass das Parkverbot in dem Strassenabschnitt vor dem Gebäude gilt und nicht in diesem, da es sich um ein Wohnhaus handelt.
- 84 Das Bild suggestiert einen anderen Bereich, als auf Google Maps angezeigt wird. (in Maps ist kein See erkennbar.)
- 85 Das Bild zeigt ein 'Betreten Verboten' Schild mit Schienen im Hintergrund und sollte dementsprechend für den Schienenbereich gelten und nicht für das Bahnhofsgebäude.
- 90 Das Verbot sollte nur für ein einzelnen Haus gelten.
- 93 Das Verbot gilt sinnvollerweise für den Zoo und nicht ein einzelnes Gebäude.
- 96 Das 'Betreten Verboten im Brandfall' gilt nur für den Fahrstuhl und nicht für das ganze Gebäude.

4.1.2 Fehlerbehebung

Nach Rücksprache mit Herrn Porada, sowie Herrn Schöning haben sich einige Fehler klären lassen.

- Die Fehler in 16, 33 und 59 wurden berichtigt.
- Bei 93 wurde geklärt, dass das Schild nur für ein einzelnes Zoogebäude gelten darf/kann, da im Zoo das teilweise füttern der Tiere erlaubt ist.
- Für 96 wurde es dabei belassen, dass man ein ganzes Haus nimmt.
- Für 40 und 85 hat sich keine Übereinstimmung der Interpretation finden lassen.
- Für die Restlichen wurden die Fehler als Aufnahmeunschärfe erklärt und werden nicht weiter hinterfragt sondern als richtig angenommen.

4.1.3 Erweiterung der Testdaten

Da in dem Testdatensatz so viele Fehler vorhanden sind, ist geplant diesen zu erweitern. Dazu wird eine Android-App programmiert, in welcher Nutzer SpaceUsageRules abhängig von ihrer Position angezeigt bekommen und sie neue hinzufügen können. Um die Qualität unserer bisherige Regeln zu überprüfen, und dem Benutzer die Eingabe zu vereinfachen, wird ein Vorschlag gemacht[1], welchen er annehmen oder abändern kann. Auch soll es ihm möglich sein einen komplett neuen Bereich einzeichnen zu können. Die angelegten Daten werden anschließend mit OpenStreetMap synchronisiert, sodass sich für die Nutzer auch ein eigener Nutzen ergibt und sie ein Ergebnis ihres Beitrages sehen. Für jede angelegte Regel werden Nutzungsdaten an uns übertragen. Wir erhoffen uns daraus einen Testdatensatz zu bekommen, welcher um den Faktor 5 bis 10 größer ist und es uns erlaubt die erstellten Regeln effizienter zu überprüfen. Für weitere Informationen siehe Abschnitt 4.5 auf Seite 20

4.2 Regeln

Um "Space Usage Rules" und "Gewichtungsregeln" nicht zu verwechseln, benutzen wir den Begriff "Verbot", wenn wir von Space Usage Rules reden, und den Begriff "Regel", wenn wir Gewichtungsregeln meinen.

4.2.1 Regelbeschreibung

Da in fast allen Fällen ein sehr nah gelegenes Polygon gemeint ist, wird zunächst auf den Abstand aller vorhandenen Polygone etwas aufaddiert, um den Abstand 0 zu vermeiden, und diese danach nach unseren Regeln zu gewichten um anschließend das Polygon auszuwählen, welches nach der Gewichtung den kürzesten Abstand hat.

[1] Dieses wurde in der Implementierung fallen gelassen, da die Berechnung des beabsichtigten Geltungsbereiches auf Mobilgeräten bis zu 10 Sekunden dauern kann und somit die Nutzerfreundlichkeit erheblich stört.

4.2.2 Formale Beschreibung

Sei ID die die Bezeichnung eines Testdatensatzes.
Sei t ein Tag $(key \rightarrow value)$ aus OpenStreetMap.
Sei $r(t)$ eine Regel $t \mapsto [0..2]$, die einem Tag t ein Gewicht zuordnet.
Sei Z_X die Menge der Tags t, die in X vorkommen.
Sei V eine Menge von Verboten v.
Sei V_{ID} die Menge der Verbote v, die im Testdatensatz ID gelten.
Sei R_V eine Menge von Funktionen r für die Verbote V.
Sei C_{ID} die Geokoordinate, an welcher das Bild zu ID aufgenommen wurde.
Sei $D_{X,ID}$ der Abstand eines Polygons X zu C_{ID}.
Sei O_{ID} die Menge der Polygone aus OpenStreetMap in der Umgebung von C_{ID}.
Sei A_X die Fläche eines Polygons X.
Sei $s(V) : V \mapsto \{r(\cdot)\}$ eine Abbildung von einer Menge von Verboten V auf eine Menge von Regeln $\{r(\cdot)\}$.
Sei P_{ID} das gegebene Lösungspolygon zu ID.
Sei $I(X,Y)$ das Schnittpolygon der Polygone X und Y.
Sei $o \in \mathbb{R}_{>0}$ ein Offset.
Sollte es für ein t keine Regel $r(t)$ geben, so wird die Standartregel: $r(t) = 1$ angewendet.
Sei Gew die Gewichtungsfunktion für ein Polygon X aus dem Testdatensatz ID, die definiert ist durch

$$\mathrm{Gew}(ID,X) := (D_{X,ID} + o) * \prod_{r \in s(V_{ID})} \prod_{t \in Z_{ID}} r(t). \tag{4.1}$$

G sei die Funktion, die aus einem Testdatensatz ID das Polygon X mit der kleinsten gewichteten Ditanz und dem kleinsten Flächeninhalt als Lösungspolygon auswählt.

$$\mathrm{G}(ID) := X \text{ wobei } X \in O_{ID} \text{ mit } \mathrm{Gew}(ID,X) < \mathrm{Gew}(ID,Y) \vee \Big(\mathrm{Gew}(ID,X) = \mathrm{Gew}(ID,Y) \wedge A_X < A_Y\Big) \forall Y \in O_{ID} \tag{4.2}$$

4.2.3 Beispiel

Eine Regel kann z.B. so aussehen:
Für das Verbot Nichtrauchen: wenn ein Tag den Key 'building' enhält dann halbiere den Abstand.
$R_{nichtrauchen} = [r('building') = 0.5]$

Sei der Offset als 1 angenommen und in der Umgebung zu ID nur zwei Flächen A, B in OpenStreetMap vorhanden.

- A enthält die Tags: 'building -> yes', 'addr:housenumber -> 34' und hat den Abstand 0.2;
- B enthält die Tags: 'landuse->residential' und hat den Abstand 0;

Daraus errechnet sich:

- $\mathrm{Gew}(ID,A) = (0.2+1) * 0.5 * 1 = 0.6$
- $\mathrm{Gew}(ID,B) = (0.0+1) * 1 = 1$

$\mathrm{G}(ID) = A$

4.2.4 Maschinelle Überprüfung der Regelsätze

Um die Güte der verschiedenen Regeln zu überprüfen wird ein Programm geschrieben, welche für jede Regel die Schnittpolygone von unseren Lösungspolygonen mit den Musterlösungen bildet und anschließend jeweils durch die Fläche unseres Polygons, sowie die des Musterlösungspolygons teilt. Von beiden Zahlen wird das Minimun genommen und als Indikator der Lösungsgüte angesehen. Die folgende Funktion f berechnet diese Güte. Dabei sind R ein Regelsatz, v ein Verbot (oder eine Verbotsmenge?), S_v die Menge der Datensätze, in denen v gilt, $q \in S_v$

Abbildung 4.1: Beispielhafte Ausgabe für Nummer 27

ein Datensatz, P_q die Musterlösung des Datensatzes q, I ? und $G(q,R) := \mathrm{G}(q)$ abhängig vom Regelsatz R.

$$f(v,R) := \sum_{q \in S_v} \min\left(\frac{I(P_q, G(q,R))}{P_q}, \frac{I(P_q, G(q,R))}{G(q,R)}\right) \tag{4.3}$$

Um die Güte des Regelsatzes für ein einzelnes Verbot vergleichbar zu anderen Verboten zu machen, wird abschließend der Indikator durch die Anzahl der IDs, welche ihn beeinflussen, geteilt.

$$Güte(R) := \sum_{v \in V} \frac{f(v,R)}{\#S_v} \tag{4.4}$$

Wir suchen also einen Regelsatz R für den $Güte(R)$ maximal ist.

4.2.5 Visuelle Überprüfung

Um die KML-Dateien nicht immer manuell in Google-Earth laden zu müssen, wurde eine weitere Klasse geschrieben, welche zur Visualisierung der Daten gedacht ist. Dabei werden Bilder erstellt, in welchen die Daten aus OSM als schwarze Linien bzw. Polygone eingezeichnet werden. Zusätzlich werden die Tags, welche die Objekte enthalten zu jedem Objekt eingezeichnet, um schnell eine Übersicht zu bekommen, was wozu gehört. Um die Güte des Regeldatensatzes bzw. des Algorithmus subjektiv beurteilen zu können, werden die truth bzw. die von unserem Algorithmus gefundenen Polygone farblich eingezeichnet. So ist eine schnelle Beurteilung, ob man das richtige Poylgon gewählt hat bzw. welche Tags man evtl. noch beachten sollte, möglich.

In Abbildung 4.1 ist beispielhaft eine Ausgabe gezeigt. Das Bild hat eine Größe von mehreren Megapixeln, da es sonst zu Überlappungen der Tags kommen würde wenn man sie größer schreibt. Auch wenn man jetzt noch reinzoomen muss um diese zu lesen, so ist es doch wesentlich bequemer, als die Überprüfung anhand der OpenStreetMap XML-Dateien. Wie in Abbildung 4.2 auf der nächsten Seite zu erkennen ist, ist das Lösungspolygon in Grün, das Ergebnispolygon unseres Algorithmus in Pink und die Geokoordinate als schwarzer Punkt eingezeichnet.

Als weiterer Hinweis wird zusätzlich zu jedem Polygon die reale und die gewichtete Distanz zur Geokoordinate, an welcher das Bild gemacht wurde, angegeben.

4.2.6 Inkrementelle Regelerstellung

Aus dem Testdatensatz werden zunächst alle Datensätze herausgefiltert, welche nicht einfach lösbar sind. (Vgl. Abschnitt 4.1.1 auf Seite 11) Es wird mit einem leeren Regelsatz angefangen und dieser anschließend inkrementell erweitert. Dazu wird wie folgt vorgegangen:

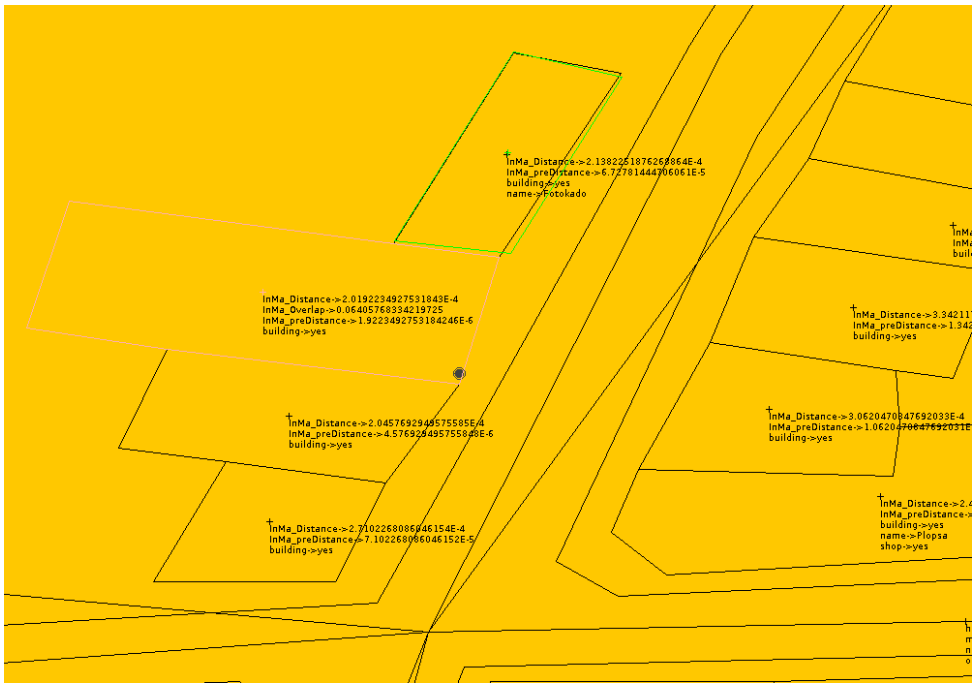


Abbildung 4.2: Reingezoomte Ausgabe für Nummer 27

1. Definiere für jeden Datensatz ein optimales Polygon, bzw einen Overlap-Wert, welcher erreicht werden muss.
2. Der Regelsatz wird auf alle Testdatensätze angewendet.
3. Für jeden Datensatz, bei welchem die Regeln nicht das optimale Polygon finden, wird ein Bild wie in Abschnitt 4.2.5 auf der vorherigen Seite erstellt.
4. Dem Benutzer werden alle Bilder, angezeigt und dieser kann nun weitere Regeln hinzufügen oder bestehende ändern.
5. gehe zu 2

Nach dem Abdecken der einfachen Fälle kann mit den schwierigeren genauso vorgegangen werden.

4.2.6.1 Beispiel 1

Nachdem der Algorithmus mit einem leerem Regelsatz durchgelaufen ist zeigt Abbildung 4.2 einen Ausschnitt des Bildes, wie es in Schritt 3 erstellt wurde. Wir können nun einfach sehen, dass es offensichtlich nicht das richtige Polygon ist. Der einzige Unterschied der bestmöglichen Lösung im Vergleich zu dem nächstbesten Polygon ist, dass das Lösungspolygon noch den Namen “Fotokado” hat. Desweiteren sehen wir, dass kein weiteres Polygon in der Nähe einen Namen hat. “Fotokado” an sich ist jetzt etwas, dass man verwenden könnte, sinnvoller ist es aber, generell zu sagen, dass, wenn Namen vorkommen, diese Gebäude bevorzugt werden sollen. Für Nr. 27 gilt nur: parking="no". Es ist also wahrscheinlich sinnvoll, folgende Regel zu erstellen[2]:

$$R_{parking=''no''} = [r('name') = 0.8]$$

4.2.6.2 Beispiel 2

In Abbildung 4.3 auf der nächsten Seite ist zu erkennen, dass es keinen Unterschied bei den Tags für das richtige und das nächstgelegene Polygon gibt. Hier kann man nicht mehr viel machen und wird wahrscheinlich immer das falsche Polygon ausgeben.

Abbildung 4.3: Reingezoomte Ausgabe für Nummer 31

Abbildung 4.4: Ausgabe mit Achteck Erweiterung.

4.2.7 Erweiterung

Um den Fall abzudecken, in welchem in OpenStreetMap keine ausreichenden Daten vorhanden sind, wird um den Punkt C_{ID} ein Achteck[3] mit Radius b gelegt und dieses mit der gewichtet nächsten Großfläche geschnitten. Eine Großfläche X definiert sich dadurch, dass ihre Fläche A_X einen Schwellwert u überschreitet.

Da z.B. ein Verbot für offenem Feuer in einem ganzen Parkgebiet gelten wird, in OpenStreetMap jedoch oftmals einzelne Häuser fehlen, für welche einzelne Regeln gelten ist dieses nicht generell anzuwenden, sondern an Verbote und Eigenschaften der Großfläche zu binden.

Eine beispielhafte Ausgabe der Erweiterung ist is Abbildung 4.4 auf der vorherigen Seite zu sehen. Hier wurde es sinnvoll genutzt um ein Industriegebiet zu schneiden und so auf die ungefähre Fläche der gewünschten Lösung zu kommen.

4.3 Softwareimplementierung

Auch wenn Anfangs eine deutliche Tendenz zu C/C++ als Programmiersprache der Wahl war, fiel durch das spätere Interesse für eine Android Applikation die Wahl auf Java. So ist es einfach möglich einen Kern mit Algorithmen zu schreiben und diesen von den verschiedenen Anwendungen als Bibliothek nutzen zu lassen.

4.3.1 Entwurf

4.3.2 Eingabeformat

4.3.2.1 Regeln

Die benutzten Regeln liegen in einer Datei im XML-Format vor. Darin werden vier von uns definierte Tags benutzt: 'ruleset', 'rule', 'restriction' und 'OSMTag'. Gewicht, Schwellwert und Radius sind Attribute des Tags 'OSMTag'.
'ruleset' umfasst die Gesamtheit aller Regeln. Eine Regel wird durch 'rule' repräsentiert. 'rule' enthält wiederum beliebig viele restrictions und OSMTags. 'restriction' stellt eine Space Usage Rule dar. Der Name dieser SUR wird als Inhalt (nicht als Attribut) von 'restriction' gepeichert. Der Inhalt von 'OSMTag' ist der Name eines Tags, der in OSM einem Objekt zugeordnet ist. Key und Value dieses Tags werden dabei nicht getrennt. Das Gewicht, der Schwellwert und der Radius, die diesem Tag zugeordnet werden, werden in XML in den Attributen 'weight', 'threshold' und 'radius' gespeichert.
Eine Gültige Eingabe kann folgendermaßen aussehen:

```
<ruleset>
  <rule>
    <restriction>smoking="no"</restriction>
    <OSMTag weight="0.5" threshold="1e-5" radius="3.5e-5">addr:housenumber</OSMTag>
    <OSMTag weight="1.5" threshold="3e-4" radius="5.7e-3">landuse-forest</OSMTag>
  </rule>
  <rule>
    <restriction>fishing="no"</restriction>
    <restriction>smoking="no"</restriction>
  </rule>
</ruleset>
```

Dieses Regelwerk sagt aus, dass ein Nichtrauchen Verbot wahrscheinlich in Flächen gilt, welche eine Hausnummer zugewiesen haben und sehr unwahrscheinlich in Gebieten, die als Waldgebiete ausgeschildert sind. Sollte die so ermittelte Fläche größer sein als einer der Schwellwerte, so wird aus allen Radien der überschrittenen Schwellwerte ein neuer Radius berechnet. Ein reguläres Polygon wird erzeugt und mit der ermittelten Fläche geschnitten.

[2] 0.8 ist willkürlich gewählt. 0.0 oder 0.5 würden für genau dieses Datensatz auch zum richtigen Ergebnis führen, können aber wiederrum in anderen Datensätzen hinderlich sein, da sie eine wesentlich größere Veränderung bewirken.
[3] Relativ geringer Rechenaufwand bei gleichzeitig bester Annäherung an einen Kreis.

Für das Angel- und Rauchverbot sind keine Regeln definiert. Unser Algorithmus wird sich also nur die nächstegelegene Fläche als wahrscheinlich nehmen, oder sollten der Punkt innerhalb von 2 Flächen liegen[4], so wird die kleinere von beiden ausgewählt.

Wichtig: Es wird der Regelsatz angewendet, welche die größtmögliche Überlappung bei dem Verbotsmengen hat. Sollte es also eine Verbotsmenge in den Daten geben, welche aus

```
smoking="no", food="no"
```

besteht, so wird das Regelwerk für smoking=”no” angewendet. Um das explizit auszuschließen sind Verbotmengen mit leeren Regelmengen anzugeben wie im Beispiel für Angel- und Rauchverbot getan.

4.3.2.2 Daten

Hier werden die aus der Aufgabenstellung übernommenen Regeln beachtet und das Eingabeformat wie beschrieben umgesetzt.

> Die erste Zeile enthält die Anzahl c der Space Usage Rules. Es folgen zeilenweise die Informationen über die c Regeldefinitionen. Eine Zeile beginnt mit der ID der Space Usage Rule gefolgt von deren Position in geographischen Koordinaten in der Form $Breitengrad, Längengrad$. Als Eingabewerte sind für $Breitengrad$ und $Längengrad$ Fließkommazahlen mit einem ’.’ als Dezimaltrennzeichen erlaubt. Die einzelnen Informationen sind durch Kommas und optionale Leerzeichen getrennt.
>
> (GI Aufgabenstellung)

4.3.3 Test

Um die Qualität hoch zu halten werden für alle wichtigen Klassen und Methoden JUnit Tests entworfen. Dabei ist das jeweils andere Teammitglied angehalten anhand der Dokumentation Blackbox-Test zu schreiben um bei den Test unvoreingenommen zu sein und engagiert möglichst viele Fehler zu finden.

4.4 Automatisierte Regelerstellung

Da uns die Grundaufgabe nicht informatiklastig genug ist haben wir beschlossen unser selbst definiertes Regelwerk abschließend gegen einen genetischen Algorithmus antreten zu lassen.

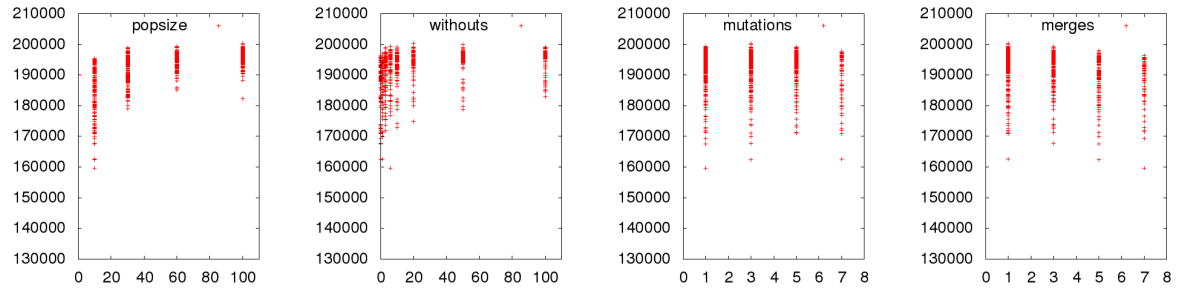
4.4.1 Eingabedaten

- Eine Liste mit SpaceUsageRules wie in Abschnitt 4.3.2.2 definiert.
- Die dazugehörigen truth.kml Dateien.
- Verschiedene Parameter zum Steuern des genetischen Algorithmus
  - Populationsgröße
  - Anzahl der Verbesserungsversuche
  - Anzahl der Populationen, welche unverändert in die nächste Generation gehen.
  - Anzahl der Populationen, welche mutiert in die nächste Generation gehen.
  - Anzahl der Populationen, welche mit anderen gemischt in die nächste Generation gehen.

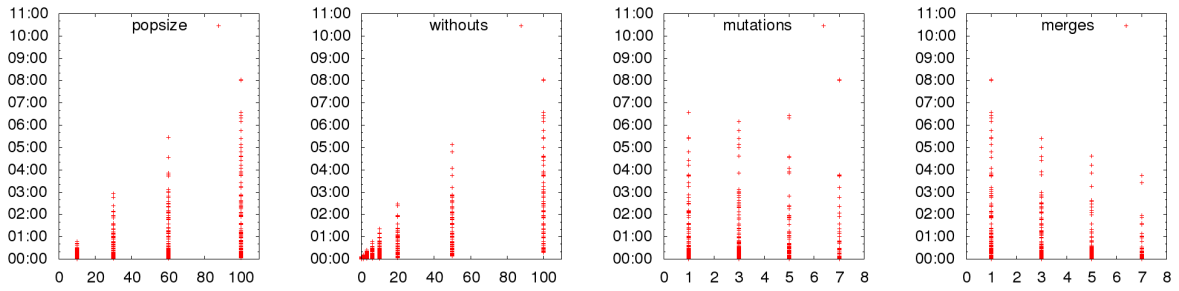
4.4.2 Ausgabedaten

Das System ist so entworfen, dass es als Ausgabe Daten liefert wie sie in Abschnitt 4.3.2.1 auf der vorherigen Seite von unserem Algortihmus verlangt werden.

[4] Es könnte z.B. eine Fläche ’Niedersachsen’ und eine Fläche ’Waldgebiet’ definiert sein.



(a) Qualität



(b) Laufzeit

Abbildung 4.5: in Abhängigkeit verschiedener Paramter.

4.4.3 Vorgehensweise

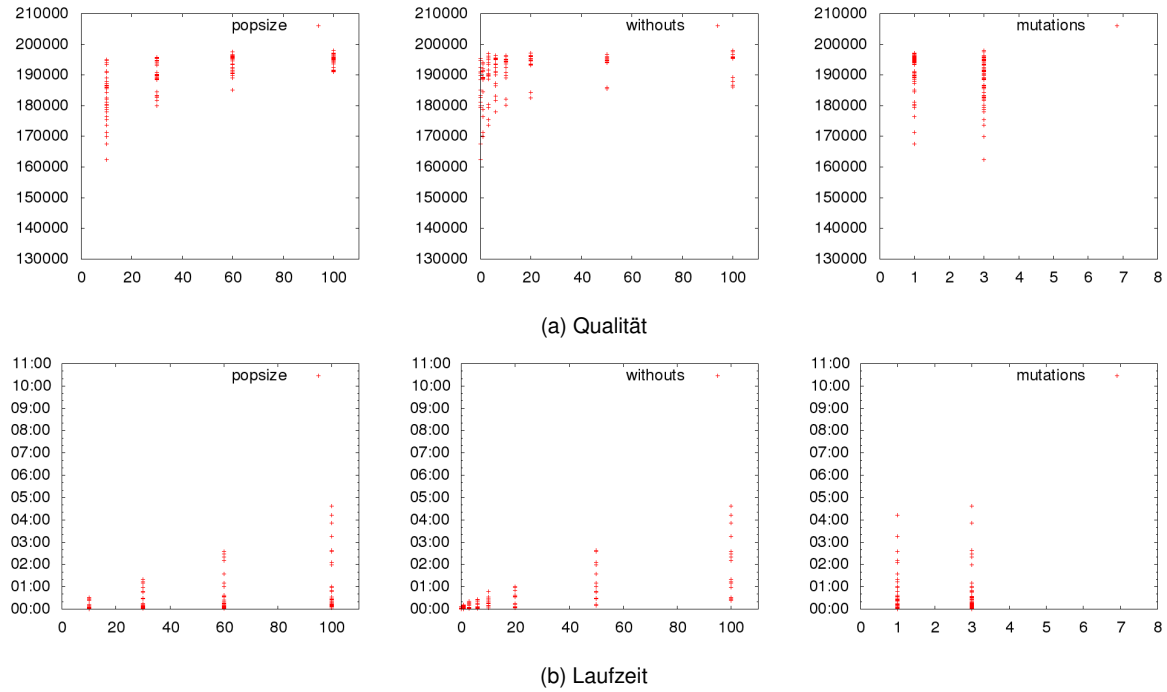
Es werden zunächst für alle Eingabedaten die Daten im Umkreis der Positionsdaten aus Openstreetmap heruntergeladen. Aus diesen werden alle möglichen Tags extrahiert. Tags, welche nur einmalig vorkommen, werden ignoriert, um zum einen die Menge zu verkleinern und zum anderen die Regeln allgemeiner zu halten und sie so einfacher und besser auf andere Datensätze übertragen zu können. Anschließend wird für jede Verbotsmenge ein Genetischer Algorithmus erstellt, welcher die Verbote für diesen optimiert und in einem eigenem Thread läuft. Nachdem alle Algorithmen durchgelaufen sind, werden die Optimalen eingesammelt, alle Regeln, welche keinen Einfluss auf das Lösungspolygon haben entfernt, und aus ihnen ein Regelwerk erstellt und in eine Datei geschrieben.

4.4.4 Fitnessfunktion

Die Fitness berechnet sich in erster Linie aus der Überlappung von Lösungspolygon und dem errechnetem Polygon. In zweiter Linie aus der Anzahl der dafür benutzen Regeln, da wir eine minimale Anzahl als optimal ansehen.

4.4.5 Laufzeitanalyse[5]

- calculate fitness 54%
  - calculate the weighted distance 33%
    - calculate the distance 16%
    - apply our weights 14%
  - consider a threshold value 18%
- generate next generation 39%



(a) Qualität

(b) Laufzeit

Abbildung 4.6: des Genetischen Algorithmus für fixen Merge-Wert von 5.

4.4.6 Evaluation[6]

Wie in Abbildung 4.5 auf der vorherigen Seite zu sehen ist, wird die Laufzeit und die Ergebnisse des Genetischen Algorithmus positiv beeinflusst, wenn man den Merge-Wert möglichst hoch setzt. Kein anderer Faktor hat so einen positiven Einfluss auf das Ergebniss. (Bei höheren Populationsgrößen steigt zwar auch die Qualität, aber gleichzeitig auch die Laufzeit im Gegensatz zum Merge-Wert.)

Wenn wir festlegen, dass 50% der Populationen in der neuen Generation aus den Besten der alten zusammen gepaart werden sollen, dann erhalten wir Ergebnisse (Siehe Abbildung 4.6), die in der Qualität sehr homogen sind und für Populationsgröße und Wiederholungen in der Laufzeit ansteigt. Es ist also zu vermuten, dass bereits für eine kleine Anzahl von Wiederholungen und Populationsgrößen ein brauchbares Ergebnis erzielt werden kann. Des weiteren ist Ersichtlich, dass die Anzahl der Populationen, welche mutiert übernommen werden sollen keinen nennenswerten Einfluss auf die Qualität oder Laufzeit hat.

Sollte das Programm später mit einem größeren Datensatz genutzt werden, schlagen wir also folgende Parameter vor:

- Populationsgröße: 200 (500, wenn man die Rechenzeit hat.)
- Anzahl der Verbesserungsversuche: 100 (500, wenn man die Rechenzeit hat.)
- Anzahl der unveränderten Populationen: 20
- Anzahl der mutierten Populationen: 40
- Anzahl der gemergten Populationen: 100

4.5 Softwareimplementierung zur Testdatensatzerweiterung[7]

Um die Testdatensätze zu erweitern wird eine Android App geschrieben, welche es dem Benutzer erlaubt weitere SpaceUsageRules anhand seines aktuellen Standorts einzugeben.

[5] Fehlende Angaben zu 100% durch kleine Funktionsaufrufe

[6] Alle Durchläufe fanden mit einem Thread auf dem selben Rechner statt (Core i5-3570K - 16GB RAM)

[7] Die App hat es leider nicht bis über die Alpha Version hinaus geschafft, und konnte somit nicht produktiv für uns und den Wettbewerb eingesetzt werden.

4.5.1 Nutzerbasis

Am Anfang wurde noch geplant, die App nur an Freunde zu verteilen und auf deren Mitarbeit zu hoffen. Da wir im Freundeskreis einige Sonder- und Sozialpädagogen haben, wurde überlegt, für diese einen Mehrwert zu bieten, indem wir die Möglichkeit geben, die App in den Einstellungen umzukonfigurieren, dass nur Daten zu Barrierefreiheit angezeigt und bearbeitet werden können.

Dies sorgt dafür, dass einige Klicks weniger notwendig sind und auch weniger Daten von den OSM-Servern heruntergeladen werden müssen und sorgt hoffentlich dafür, dass die Nutzer einen Sinn für sich in der App sehen und diese nicht nur aus Freundlichkeit benutzen.

In späteren Gesprächen mit anderen potentiellen Nutzern stellte sich heraus, dass diese bereit wären die App zu nutzen, wenn sie ihnen nur Daten zu Rauchverboten anzeigen würden. Also wurde sie noch einmal verändert, sodass sie einem in den Einstellungen erlaubt die Suche und hinzufügen auf einen beliebigen Tag zu beschränken.

4.5.2 Standortbestimmung

Da Android Geräte generell eine Möglichkeit bieten zur Standortbestimmung, z.B. Anhand von GPS oder Netzwerkdaten, wird diese ausgenutzt und als Startwert benutzt. In einer späteren Phase des Projekts wurde dem Nutzer die Möglichkeit gegeben, sich Informationen zu einem beliebigen Punkt auf der Karte geben zu lassen, indem er einfach länger auf die Karte drückt. In diesem Fall wird auch der Standort auf die gewählte Kartenposition gesetzt.

4.5.3 OpenStreetMap

Es werden Daten von OSM genutzt um den Nutzer bereits bekannte Nutzungsregeln anzuzeigen. Damit dieser bei Änderungen keinen eigenen Account anlegen muss, wird von uns ein Account für die Nutzung mit der App angelegt, in welchem alle Änderungen, welche in Zusammenhang mit der App stehen gemacht werden. Aktuell erscheint es uns als die beste Möglichkeit um nicht zu viele Leute vom hinzufügen dadurch abzuschrecken, dass sie sich irgendwo anmelden müssen, oder aber ihre Nutzerdaten an uns weiter geben zu müssen um etwas hinzuzufügen. Nach Beendigung des Wettbewerbs soll die App auf OAuth umgestellt werden und den Nutzern auch weiterhin zur Verfügung stehen.

Um Änderungen der Regeln zu speichern wird die API Version 0.6 benutzt und für jede Änderung ein Changeset erstellt und geschlossen. Dabei wird darauf geachtet, dass ausser einer vorgegebenen Menge von Tags keine Daten geändert werden.

4.5.4 Unser Nutzen

Wenn der Nutzer neue Daten zu OSM hinzufügt, dann wird nicht nur der Datensatz an OSM gesendet, sondern auch an uns, zusammen mit der Position des Nutzers.

Um genau zu sein, bekommen wir folgende Daten:

- Die aktuelle Position des Nutzers
- Die OSM-ID des/der ausgewählten Polygons/Polygone
- ~~Eine Information, ob das Polygon neu erstellt wurde~~[8]
- Sämtliche Nutzungsregeln, welche benutzt wurden

Sollte der Wettbewerb vorbei sein, oder wir genug Daten gesammelt haben wird die Datenübertragung an uns durch ein Update der App beendet.

[8] Erkennt man daran, dass die Version des Polygons 1 ist.

4.5.5 Hinderlichkeiten

Durch diese Vorgehensweise haben wir leider keine Bilder, welche Helfen würden die Angaben der Benutzer zu verifizieren, also müssen wir davon ausgehen, dass alles richtig ist.

4.5.6 Veröffentlichung

Mitlerweile ist eine erste Version der App fertig und unter der URL `https://play.google.com/store/apps/details?id=de.uni_hannover.inma` im Google Play Store veröffentlicht.

4.6 Analyse

Gestartet wird das Programm mit `Start.main(String[])`. Die Kernmethode, die die für diesen Wettbewerb interessante Arbeit macht, ist `Rules.calculateBest(Collection,Point)`.

4.6.1 Zeitkomplexität

Die Laufzeit der Methode calculateBest ist im wesentlichen Abhängig von der Anzahl an Polygonen, die heruntergeladen werden. Von dieser Eingabe abhängig ergibt sich eine lineare Laufzeit. Denn jedes Polygon wird einmal und unabhängig von den anderen bewertet.
Daneben gibt es noch andere Eingabeparameter, deren größe variiert. Zum Beispiel hat die Anzahl der Space Usage Rules, die in einem Bereich gelten soll, einfluss auf die Laufzeit. Die Anzahl der Ecken eines Polygons beeinflusst die Komplexität ebenso. Solche Faktoren verändern Laufzeit aber nicht so stark.
Im folgenden eine tabellarische Übersicht über wichtige Methoden und deren Zeitkomplexität.

Methode	Eingabe	Laufzeit
`calculateBest()`	n=Anz. der Ways	n+O(`considerThresholds()`)
`considerThresholds()`	m=Anz. der SUR	m+O(m)+O(`getArea()`)+O(`createNgon()`)+O(`intersection()`)
`createNgon()`		konstante Zeit, da die Methode immer exakt gleich ausgeführt wird → O(1)
`getArea()`	n=Anz. der Ecken	keine Angaben gefunden
`intersection()`	n=Anz. der Ecken	keine Angaben gefunden

Dabei wurde das Auslesen der Rules und das Herunterladen der Daten nicht beachtet.

4.6.2 Metriken

Die folgende Tablle listet ein paar Metriken auf, die unser Projekt beschreiben.

Metrik	Wert
McCabe Cyclomatic Complexity	30
Number of Classes	16
Number of Packages	5
Depth of Inheritance Tree (max.)	3

Kapitel 5

Zweite Runde

5.1 Datenerstellung

Die Bilder wurden am Nachmittag des 15.Januar mit einem Android Tablet erstellt. Dieses enthält einen GPS-Sensor, sowie eine einfache Kamera.

5.2 Probleme

Beim Erstellen der Truth-Datensätze stellte sich heraus, dass der GPS-Sensor teilweise ungenaue Positionen, welche um etliche Meter, wenn nicht sogar hunderte Meter von der eigentlichen Position abweichen. Dieses ist im genauen Gegensatz zu den Testdatensätzen, bei welchen die Bilder eine genauere Position lieferten, als es aus der Data.txt herauszulesen war.

Als Lösung dieses Problems, wurde eine Überprüfung des Abstandes zwischen GPS-Koordinate und Data.txt Koordinate implementiert, welche bei überschreiben eines Schwellwertes die Koordinate im Bild verwirft. Es kann davon ausgegangen werden, dass Daten, welche in die Data.txt eingetragen sind von Menschen überprüft wurden.

5.3 Regelwerk

Das von uns bis jetzt definierte Regelwerk liefert nach einem ersten Durchlauf zufriedenstellende Ergebnisse. Die Überlappungen sind meistens gegeben, jediglich in 2 Fällen lag er komplett daneben. Dieses ist auf ein unvollständiges Regelwerk aufgrund zu wenig Datansätzen zurück zu führen.

So gab es bis jetzt keine Definition für die Regel ”Zugang erst ab 21 Jahren”. Dieses führte dazu, dass die Backup-Regel genommen wurde, das nächstebeste Polygon zu nehmen. Da jedoch wie in Abbildung 5.1 auf der nächsten Seite zu sehen ist, das Bild vor dem gültigen Gebäude aufgenommen wurde ist dem bisherigem Regelsatz nur noch die Regel:

```
<rule>
  <restriction>access:age="21+"</restriction>
  <OSMTag weight="0.7" >building</OSMTag>
</rule>
```

hinzuzufügen um auch diesen Datensatz vollständig zu bearbeiten.

Im zweiten Fall ist wie in Abbildung 5.2 auf der nächsten Seite zu sehen keine Representation beabsichtigten Geltungsbereiches in OpenStreetMap vorhanden. Und auch wenn man davon ausgehen kann, dass niemand gerne Hundkot vor der Nase hätte ist der beabsichtigte Geltungsbereich für das Hinweisschild nur die Grünfläche, welche an den Parkplatz und Strasse angrenzt. Das Regelwerk ist also dahingehend zu ändern, dass folgende Regel hinzugefügt wird:

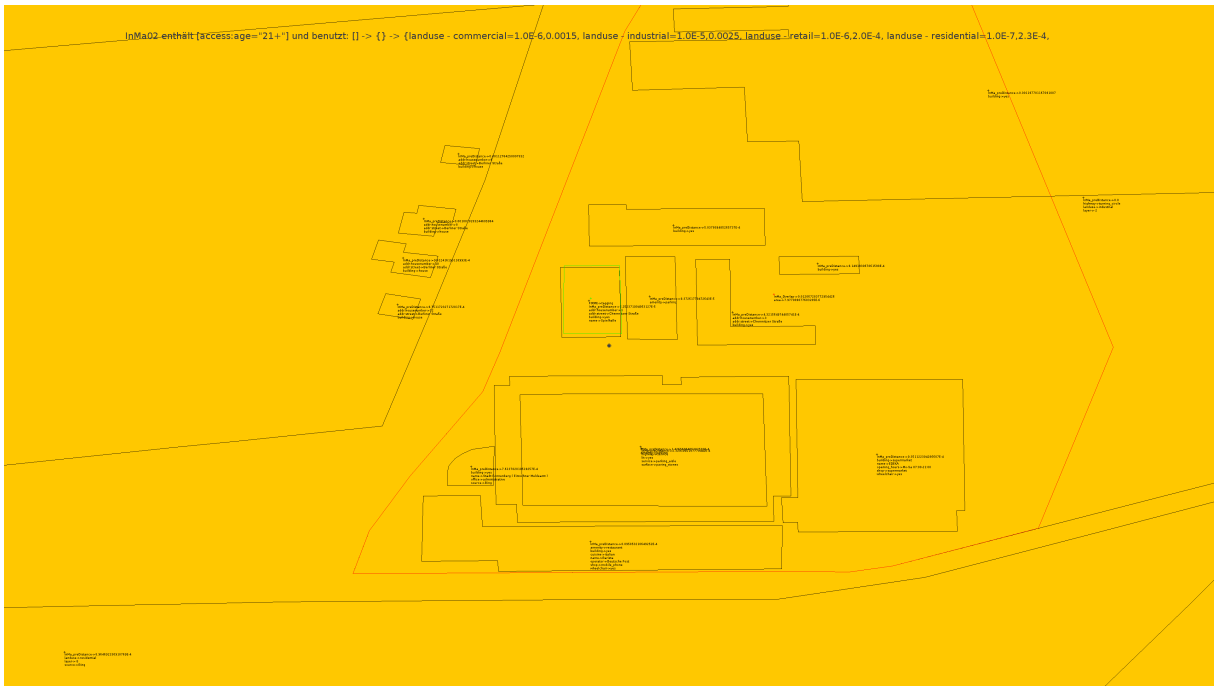


Abbildung 5.1: Unvollständige Rules Definition.

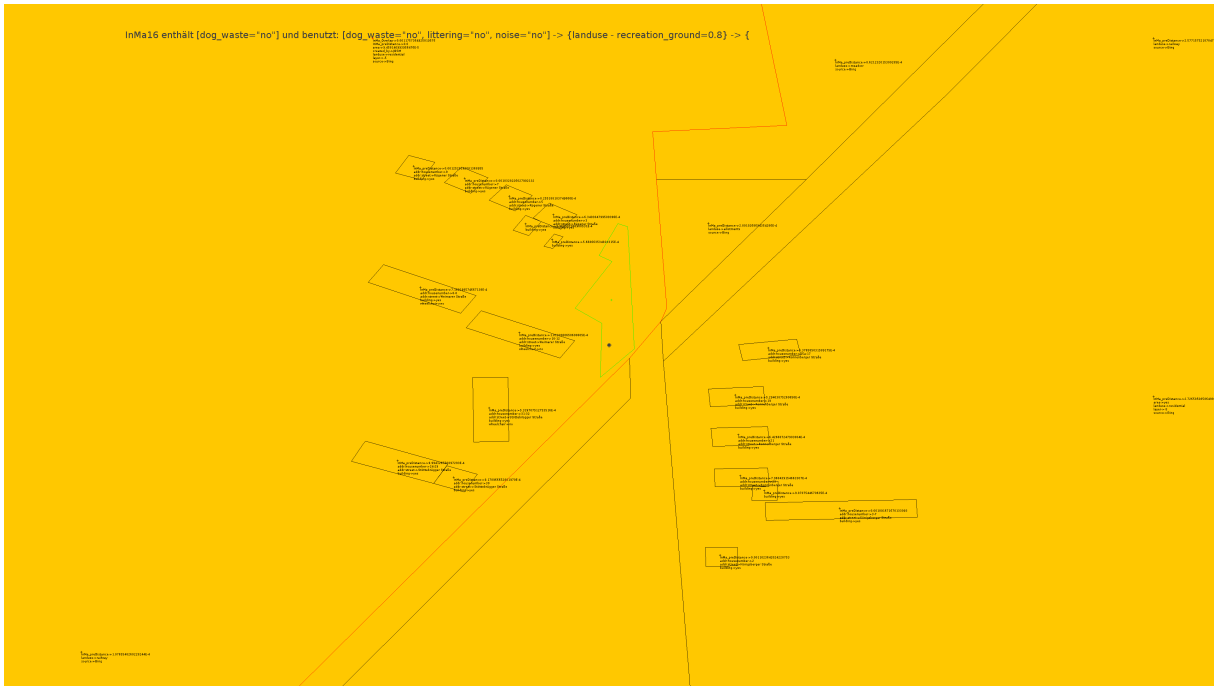


Abbildung 5.2: Unvollständige Rules Einschränkungen.

```
<rule>
  <restriction>dog_waste="no"</restriction>
  <OSMTag threshold="1e-8" radius="0.00023">landuse - residential</OSMTag>
</rule>
```

oder aber die bestehende Regel

```
<rule>
  <restriction>dog_waste="no"</restriction>
  <restriction>littering="no"</restriction>
  <restriction>noise="no"</restriction>
  <OSMTag weight="0.8" >landuse - recreation_ground</OSMTag>
</rule>
```

um den OSMTag erweitert wird. Beides führt in diesem Fall zum gleichen Ergebnis.

5.4 Gesamtergebnis

5.4.1 Testdatensatz

Unser Programm und unsere Regeln führen in 61 von 96 Fällen des Testdatensatzen zu der korrekten Lösung. In den verbleibenden 35 Datensätzen finden wir folgende Hinderlichkeiten:

- 9x ist das Unigebäude vorhanden, welches in OpenStreetMap nicht als einzelnes Gebäude gewertet wird. Einen Zusammenschluss von Flächen, welche die Eigentschaft Gebäude haben und aneinander angrenzen wäre möglich gewesen, hätte aber dazu geführt, dass viele andere Lösungen fehlerhaft werden.
- Nr. 24, 33 Hier ist es schwierig einen Teil des Gebäudes auszugrenzen, da dieses über keine besonderen Merkmale verfügt.
- Nr. 31 Hier ist es unmöglich für ein Programm zwischen der richtigen Lösung und falschen Lösung zu unterscheiden, da ist es keine Metadaten in OpenStreetMap gibt, welche die beiden Flächen unterscheiden.
- Nr. 34 Auch wenn der gegebene Lösungsbereich nicht genau getroffen wurde, so sind wir doch mit der von uns berechneten Lösung zufrieden. Hier ist zu erkennen, dass das hinzufügen von Umkreisbeschränkungen eine Sinnvolle Ergänzung darstellt. Der beabsichtigste Lösungsbereich wird Qualitativ getroffen.
- Nr. 36 Hier wurde vor dem Ergänzen des Regelmenge ein ähnliches Ergebnis wie in Nr. 34 erziehlt. Jetzt jedoch wird ein etwas falsches Gebäude ausgewählt. Wir können jedoch sagen, dass sollten die Daten von OpenStreetMap durch weitere Gebäude erfänzt werden, so sind wir zuversichtlich, dass eine optimale Lösungs gefunden wird.
- Nr. 38, 39, 41, 48, 57, 60 und 61 Siehe Nr. 34. Wir sind zufrieden mit dem Ergebnis und zuversichtlich, dass sollten mehr Daten vorhanden sein, ein optimaleres Ergebnis gefunden wird.
- Nr. 43. Hier können wir uns den sehr kleinen Lösungsbereich Aufgrund der gegebenen Polygone in OpenStreetMap nicht erklären. Wir empfinden unsere Lösung aber als richtig.
- Nr. 44 Ein äusserst schwieriger Datensatz. Hier sind keine Gebäude in OpenStreetMap vorhanden und selbst wenn, können wir aufgrund der GPS Position nicht ausschließen, dass ein Falsches Lösungspolygon gewählt wird.
- Nr. 45 Hier ist der mit der Fläche geschnittene Bereich etwas klein gewählt. Dieses lässt noch Verbesserungen offen, jedoch ist hier abzuwägen wie groß das durchschnittliche Geschäft ist, welches die Benutzung mit Fahrrädern verbietet. Diese Größe sollte berücksichtigt werden bei Bestimmung der Schnittfläche. Wäre die GPS Koordinate in der Mitte des Gebäudes platziert, so hätte sich eine fast optimale Lösung ergeben.
- Nr. 49 Hier hat unser Algorithmus komplett versagt. Auch nach etlichen Überlegungen sind wir zu keinem Ergebnis gekommen, das uns zufriedenstellen würde.

- Nr. 50 Hier hat der Algorithmus etwas daneben gegriffen, aber man kann erkennen, dass unsere Lösung nicht allzu weit entfernt ist von der Musterlösung. Eine zusammenfassung von Gebäuden hätte auch hier helfen können, hätte jedoch auch nicht das optimale Lösungspolygon ergeben.
- Nr. 51 Eine weiteres sehr schwierige Problem. Hier hätte eine bessere Lösung gefunden werden können, Wenn man den Gebäudekomplex mit einer quadratischen Fläche ähnlich den angrenzenden Gebäuden schneidet. Dieses ist jedoch eine höchst komplexe Aufgabe und würde auch nicht den Erfolg garantieren. Besser wäre es, wenn OpenStreetMap mehr Daten hätte.
- Nr. 53. Aufgrund der Tatsache, dass hier keine Daten in OpenStreetMap vorhanden sind sind wir sehr zufrieden mit dem Ergebnis. Es wurde keines der Angrenzenden Gebäude ausgewählt, sondern ein Bereich von ungefähr der Gesuchten Größe mit nur etwas verschobener Position.
- Nr. 55 Der Gewählte Bereich ist etwas zu groß gewählt, hier ist der Gegenpol zu Nr. 45.
- Nr. 56 Aufgrund der Fehlenden Daten in OpenStreetMap die nächstbeste und aufgrund der Daten auch sinnvolle Lösung. Wenn man nicht wüsste, dass es falsch ist, so würde ein Mensch dieses wahrscheinlich auch auswählen.
- Nr. 58 und 59 Hier ist zu sagen: Pech gehabt. Wäre die Koordinate etwas näher an der andere Fläche oder das Gebäude in OpenStreetMap vorhanden, wäre die Lösung besser gewesen.
- Nr. 62 Hier können wir Musterlösung und Datenlage nicht vernünftig interpretieren und bewerten.
- Nr. 79 Hier scheinen die Daten von OpenStreetMap und der Musterlösung zu divergieren. Unter den gegebenen Umständen ist die von uns gewählte Lösung als Richtig anzusehen.

5.4.2 Eigene Daten

Auch hier sind wir zufrieden mit den Ergebnissen. Auch wenn nicht immer die Optimalen Ergebnisse gefunden wurden, so sind diese doch brauchbar und es mussten nur einige wenige Anpassungen an den Regeln vorgenommen werden. Bedingt durch die vielen kleinen Geschäfte, welche sich in Teilbereichen von Häusern befinden ist es schwierig deren genauen Umriss zu erstellen. Es hat sich hier für uns gezeigt, dass einiges an Arbeit in OpenStreetMap gesteckt werden kann und uns ermuntert dieses auch zu tun.

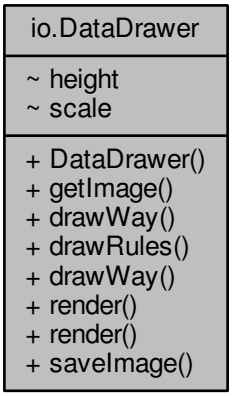
Zusammenfassend ist zu sagen, dass wir bis auf einige wenige Ausrutscher mit der Qualität der Lösungen zufrieden sind.

Kapitel 6

Class Documentation

6.1 io.DataDrawer Class Reference

Collaboration diagram for io.DataDrawer:



Public Member Functions

- DataDrawer (int width, int height, Coordinate middle, double scale)
- BufferedImage getImage ()
- void drawWay (Way w)
- void drawRules (String rules)
- void drawWay (Way w, Color color)
- void render (float radius)
- void render (Collection< Way > data)
- void saveImage (String name) throws IOException

6.1.1 Detailed Description

DataDrawer is a tool to visualize the polygons and their tags around any place. The two main purposes are to familiarize with the existing kinds of tags and to see how good (or bad) the implementation works.

Author

Peter Zilz

6.1.2 Constructor & Destructor Documentation

6.1.2.1 io.DataDrawer.DataDrawer (int width, int height, Coordinate middle, double scale)

Creates a DataDrawer for a specific location.

Parameters

width	width of the picture
height	height of the picture
middle	center of the area to be rendered

6.1.3 Member Function Documentation

6.1.3.1 void io.DataDrawer.drawRules (String rules)

draws the given string in with a bigger font then usual on top of the image.

Parameters

rules	

6.1.3.2 void io.DataDrawer.drawWay (Way w)

Draws a Way in the default Color Color#black. Just calls drawWay(Way, Color).

Parameters

w	the Way to be drawn

6.1.3.3 void io.DataDrawer.drawWay (Way w, Color color)

Draws a Way with a specific outline color.

Parameters

w	the Way to be drawn
color	color of the outline

6.1.3.4 BufferedImage io.DataDrawer.getImage ( )

Returns the rendered image. If it has not been rendered yet, it is blank.

Returns

rendered (or blank) image

6.1.3.5 void io.DataDrawer.render (float radius)

Downloads and renders the area around the given location.

Parameters

radius	radius of the area

6.1.3.6 void io.DataDrawer.render (Collection< Way > data)

Renders a specific set of data.

Parameters

data	List of Polygons with tags

6.1.3.7 void io.DataDrawer.saveImage (String name) throws IOException

Stores the rendered picture. The file extension is not automatically appended.

Parameters

name	name of the file incl. file extension

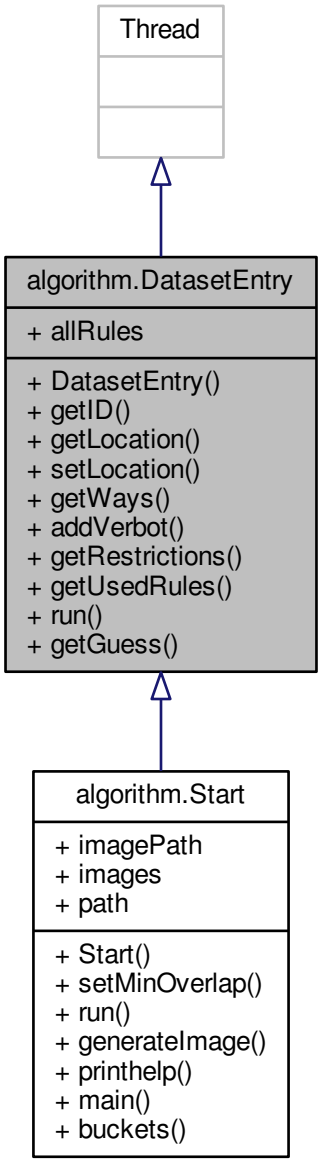
Exceptions

IOException	

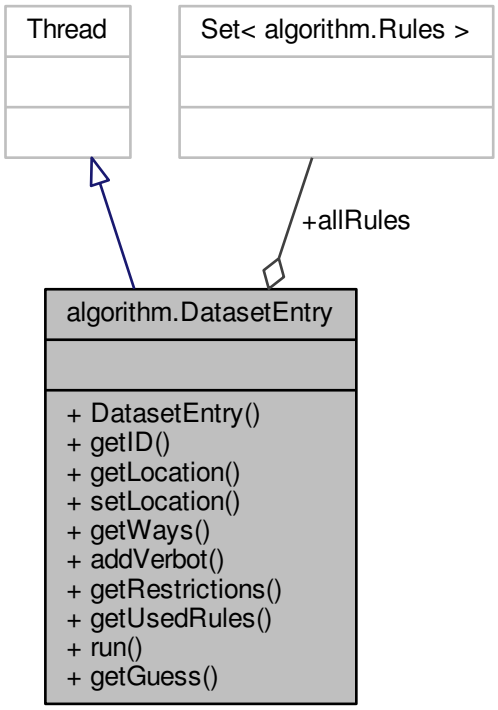
XXX create metadata

6.2 algorithm.DatasetEntry Class Reference

Inheritance diagram for algorithm.DatasetEntry:



Collaboration diagram for algorithm.DatasetEntry:



Public Member Functions

- DatasetEntry (Point location, String id)
- String getID ()
- Point getLocation ()
- void setLocation (Point p)
- Collection< Way > getWays ()
- void addVerbot (String restriction)
- Collection< String > getRestrictions ()
- Rules getUsedRules ()
- void run ()
- Way getGuess ()

Static Public Attributes

- static Set< Rules > allRules = new HashSet<Rules>()

6.2.1 Detailed Description

An entry in the dataset. this is used to compute the rules belonging to it and using this rules to compute the best polygon in the area around the given location.

Author

Fabian Pflug

6.2.2 Constructor & Destructor Documentation

6.2.2.1 algorithm.DatasetEntry.DatasetEntry (Point location, String id)

Initialises a datasetentry. has to provide a coordinate for the location and the ID of the Entry

Parameters

location	the location in the entry
id	the ID of the entry

6.2.3 Member Function Documentation

6.2.3.1 void algorithm.DatasetEntry.addVerbot (String restriction)

adds a restriction to the set of restrictions for this dataset.

Parameters

restriction	a restriction

6.2.3.2 Way algorithm.DatasetEntry.getGuess ( )

returns the Way guessed by our algorithm or null if the run-function did not run yet.

Returns

the way guessed

6.2.3.3 String algorithm.DatasetEntry.getID ( )

returns the id of this dataset (first colum in the input data)

Returns

the id of the dataset

6.2.3.4 Point algorithm.DatasetEntry.getLocation ( )

the location of this dataset

Returns

the location.

6.2.3.5 Collection<String> algorithm.DatasetEntry.getRestrictions ( )

retuns a Collection of restrictions which belong to this dataset

Returns

restrictions belonging to this dataset.

6.2.3.6 Rules algorithm.DatasetEntry.getUsedRules ( )

returns the rules used by this dataset to compute the best polygon. (this is only valid after the thread run)

Returns

a set of rules.

6.2.3.7 Collection<Way> algorithm.DatasetEntry.getWays ( )

returns a collection of ways around the given Location (this is only valid after the thread runs)

Returns

OSM-objects around the given Location or null

6.2.3.8 void algorithm.DatasetEntry.run ( )

computes the best ruleset to use and with it the best OSM-object to choose for the restrictions.

6.2.3.9 void algorithm.DatasetEntry.setLocation (Point p)

Sets the location for this dataset, so where to search from.

Parameters

p	the location where the picture is taken.

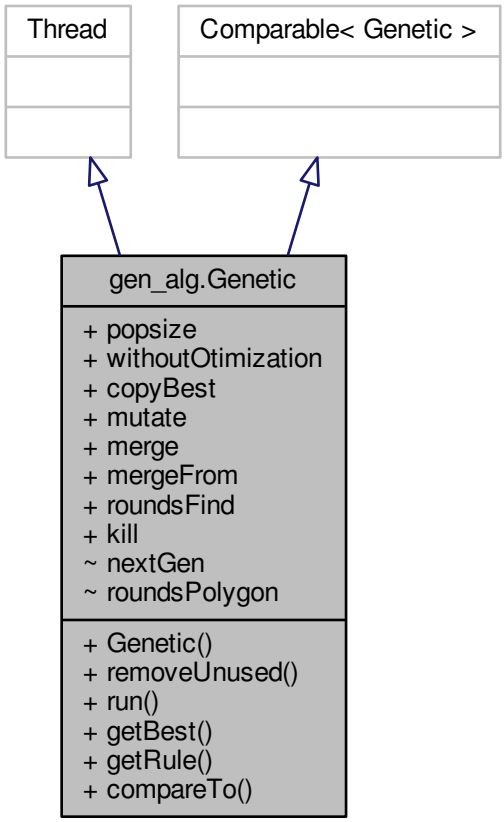
6.2.4 Member Data Documentation

6.2.4.1 Set<Rules> algorithm.DatasetEntry.allRules = new HashSet<Rules>() [static]

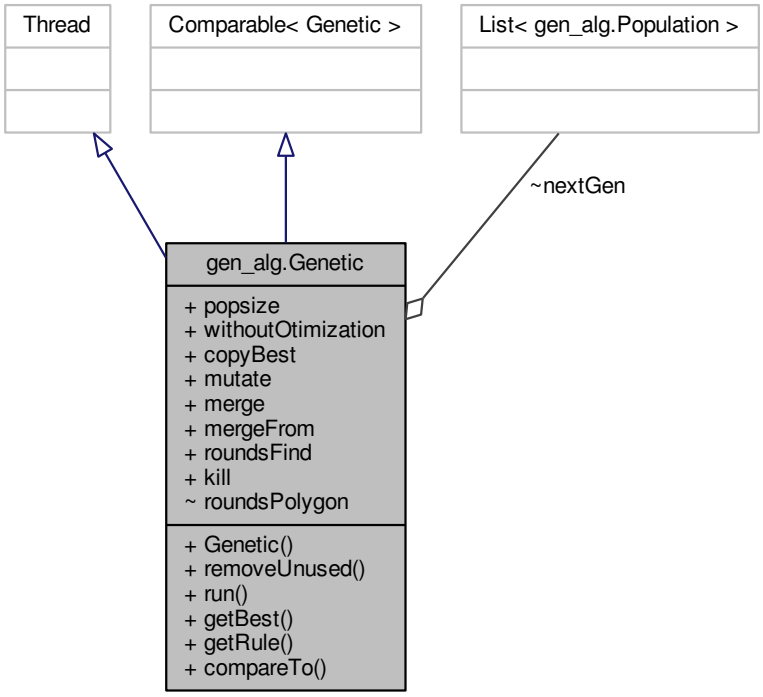
a the parsed rules from the input File

6.3 gen_alg.Genetic Class Reference

Inheritance diagram for gen_alg.Genetic:



Collaboration diagram for gen_alg.Genetic:



Public Member Functions

- **Genetic** (String signlist, Set< String > IDs, Collection< String > possible, int popsize, int without, int copyBest, int mutate, int merge, int mergeFrom) throws Exception
- void **removeUnused** ()
- void run ()
- Population getBest ()
- String getRule ()
- int compareTo (Genetic o)

Public Attributes

- int popsize = 200
- int withoutOtimization = 300
- int copyBest = popsize∗1/10
- int mutate = popsize∗4/10
- int merge = popsize∗3/10
- int mergeFrom = popsize/2
- int **roundsFind**

Static Public Attributes

- static boolean kill = false

6.3.1 Detailed Description

TODO die javadoc ins englische umschreiben. Klasse zum durchlaufen eines Genetischen Algorithmus.

6.3.2 Member Function Documentation

6.3.2.1 int gen_alg.Genetic.compareTo (Genetic o)

Vergleich um die Gentischen Algorithmen nach ihrer geschätzten Laufzeit zu sortieren.

6.3.2.2 Population gen_alg.Genetic.getBest ( )

Gibt die beste Population zurück.

6.3.2.3 String gen_alg.Genetic.getRule ( )

gibt die Namen der SpaceUsageRules zurück

6.3.2.4 void gen_alg.Genetic.run ( )

die Methode zum durchlaufen des Genetischen algorithmus mit der Steuerung aus den statischen Variablen.

6.3.3 Member Data Documentation

6.3.3.1 int gen_alg.Genetic.copyBest = popsize∗1/10

Die Anzahl der Populationen, welche unbearbeitet in die nächste generation übernommen werden sollen

6.3.3.2 boolean gen_alg.Genetic.kill = false [static]

wenn ein SigInt abgefangen wird, dann wird kill auf true gesetzt und beendet den Algorithmus vorzeitig.

6.3.3.3 int gen_alg.Genetic.merge = popsize∗3/10

Die Anzahl der Populationen pro Generation, welche aus anderen zusammen gesetzt werden sollen

6.3.3.4 int gen_alg.Genetic.mergeFrom = popsize/2

Die Menge der Poplationen, aus denen die Populationen zusammen gesetzt werden sollen.

6.3.3.5 int gen_alg.Genetic.mutate = popsize∗4/10

Die Anzahl der Populationen, welche mutiert in die nächste Generation übernommen werden solllen.

6.3.3.6 int gen_alg.Genetic.popsize = 200

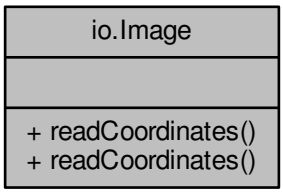
Die Anzahl der Populationen, welche den Algorithmus durchlaufen sollen

6.3.3.7 int gen_alg.Genetic.withoutOtimization = 300

Dei Anzahl der Runden, die ohne Optimierung durchlaufen werden, bevor der Algorithmus stoppt.

6.4 io.Image Class Reference

Collaboration diagram for io.Image:



Static Public Member Functions

- static Coordinate readCoordinates (String filename) throws ImageProcessingException, IOException
- static Coordinate readCoordinates (File f) throws ImageProcessingException, IOException

6.4.1 Detailed Description

Class to read in image-metadata and give the geocoordination, where the image was made.

Author

Fabian Pflug

6.4.2 Member Function Documentation

6.4.2.1 static Coordinate io.Image.readCoordinates (String filename) throws ImageProcessingException, IOException [static]

reads in the file and extracts the geocoordination if possible.

Parameters

filename	the filename of the image

Returns

the gps position where the image was taken.

Exceptions

ImageProcessingException	if the metadata can't be read.
IOException	if the file can't be read.

6.4.2.2 static Coordinate io.Image.readCoordinates (File f) throws ImageProcessingException, IOException [static]

reads in the file and extracts the geocoordination if possible.

Parameters

f	filepointer to the image

Returns

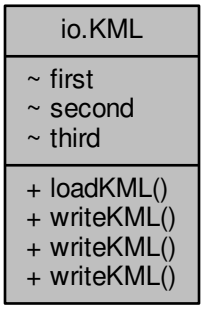
the gps position where the image was taken.

Exceptions

ImageProcessingException	if the metadata can't be read.
IOException	if the file can't be read.

6.5 io.KML Class Reference

Collaboration diagram for io.KML:



Static Public Member Functions

- static Geometry loadKML (File kml)
- static void writeKML (Geometry p, File f) throws UnsupportedEncodingException, IOException
- static void writeKML (Geometry p, String name, File f) throws IOException
- static String writeKML (Geometry p, String name)

6.5.1 Detailed Description

A class to read in and write out KML-data from files.

Author

Fabian Pflug

6.5.2 Member Function Documentation

6.5.2.1 static Geometry io.KML.loadKML (File kml) [static]

reads the coordinates from a kml-file an returns them in a polyline.

Parameters

kml	a filepointer to a file, which should have valid kml

Returns

a polyline containing all extracted coordinates.

6.5.2.2 static void io.KML.writeKML (Geometry p, File f) throws UnsupportedEncodingException, IOException [static]

Makes valid kml from the given Geometry and writes it to the given file.

Parameters

p	Geometry to write
f	the file to write to.

Exceptions

IOException	all exceptions encountered on trying to write the file.

6.5.2.3 static void io.KML.writeKML (Geometry p, String name, File f) throws IOException [static]

Makes valid kml from the given Geometry and writes it to the given file.

Parameters

p	Geometry to write
name	a name to be shown in google earth
f	the file to write to.

Exceptions

IOException	all exceptions encountered on trying to write the file.

6.5.2.4 static String io.KML.writeKML (Geometry p, String name) [static]

Makes valid kml from the given Geometry.

Parameters

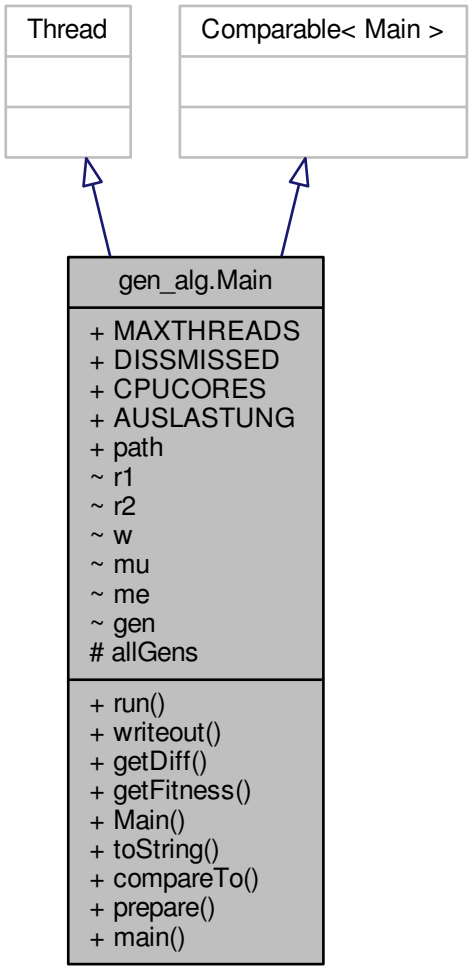
p	Geometry-object to be converted to a kml string.
name	a name to be shown in google earth

Returns

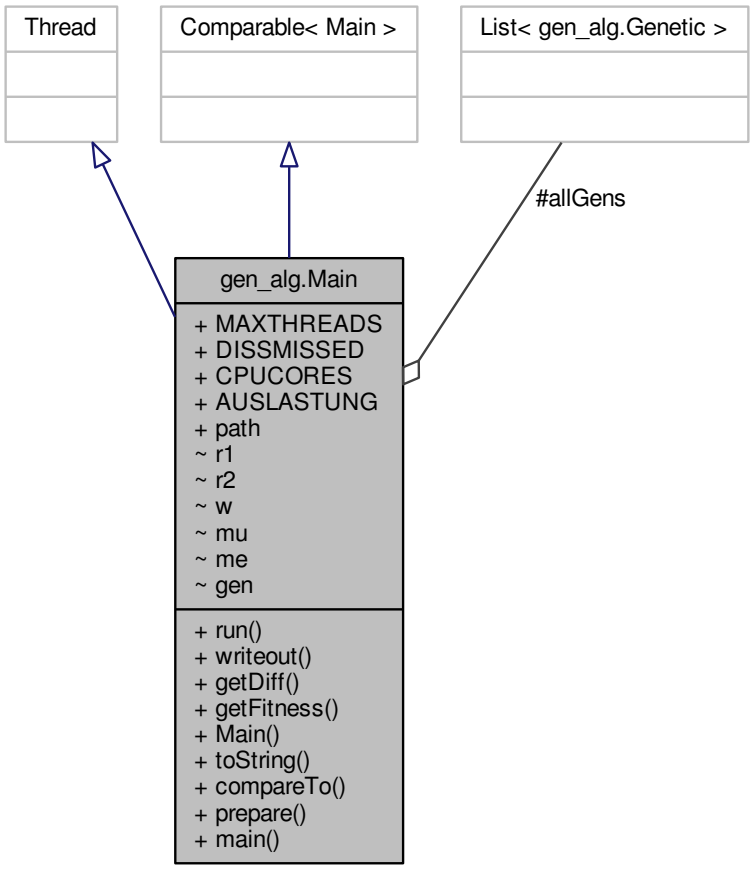
String containing valid kml

6.6 gen_alg.Main Class Reference

Inheritance diagram for gen_alg.Main:



Collaboration diagram for gen_alg.Main:



Public Member Functions

- void **run** ()
- void **writeout** () throws IOException
- int **getDiff** ()
- int **getFitness** ()
- **Main** (int p, int w, int mu, int me, int gen)
- String **toString** ()
- int **compareTo** (Main o)

Static Public Member Functions

- static void **prepare** () throws IOException, ImageProcessingException
- static void main (String[] args) throws Exception, IOException

Static Public Attributes

- static final int **MAXTHREADS** = 1

- static int **DISSMISSED** = 5
- static final int **CPUCORES** = 4
- static final double **AUSLASTUNG** = 1.0 - (1.0/(CPUCORES-MAXTHREADS))
- static final String **path** = "../Testdatensatz/"

Protected Attributes

- List< Genetic > **allGens**

6.6.1 Detailed Description

TODO die Javadoc schreiben. Klasse, welche für jede SpaceUsageRule aus dem Testdatensatz einen genetischen Algorithmus erstellt und diesen durchlaufen lässt.

6.6.2 Member Function Documentation

6.6.2.1 static void gen_alg.Main.main (String[] args) throws Exception, IOException [static]

Parameters

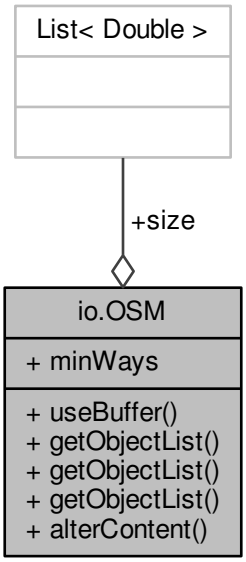
args	

Exceptions

IOException	
ImageProcessingException	

6.7 io.OSM Class Reference

Collaboration diagram for io.OSM:



Static Public Member Functions

- static void useBuffer (boolean use)
- static Collection< Way > getObjectList (Coordinate c)
- static Collection< Way > getObjectList (Coordinate c, float radius)
- static synchronized Collection
  < Way > getObjectList (Coordinate c, float radius, String tagname)
- static int alterContent (Collection< Way > ways, Coordinate location)

Static Public Attributes

- static final int **minWays** = 30
- static List< Double > **size** = new ArrayList<Double>()

6.7.1 Detailed Description

Handling communication with OpenStreetMap. Providing really easy to use functions.

Author

Fabian Pflug

6.7.2 Member Function Documentation

6.7.2.1 static int io.OSM.alterContent (Collection< Way > ways, Coordinate location) [static]

writes all the altert content in the collection of ways to Openstreetmap.

Parameters

ways	a list of ways which are altert
location	the coordinate of the user.

Returns

1 on success. a negative Value otherwise.

6.7.2.2 static Collection<Way> io.OSM.getObjectList (Coordinate c) [static]

the simplest way to get a Set of OSM-objects is by just giving a coordinate.

Parameters

c	the coordinate where data should be fetched around.

Returns

a collection of OSM-Objects.

6.7.2.3 static Collection<Way> io.OSM.getObjectList (Coordinate c, float radius) [static]

you can also define the (bounding box) radius around the point to retrieve data.

Parameters

c	the coordinate where data should be fetched around.
radius	a radius around this point.

Returns

a collection of OSM-Objects.

6.7.2.4 static synchronized Collection<Way> io.OSM.getObjectList (Coordinate c, float radius, String tagname) [static]

fetches and parses data from OpenStreetMap. you can provide a tagname to reduce the output to. Then the OverpassApi server are used.

Parameters

c	the coordinate where data should be fetched around.
radius	a radius around this point.
f	the file to read or save data to
tagname	a tagname to reduce the Output to.

Returns

a collection of OSM-Objects.

6.7.2.5 static void io.OSM.useBuffer (boolean use) [static]

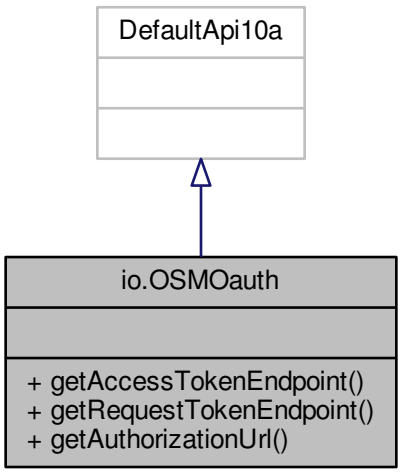
to reduce the load on the OSM-Server, it is possible to use a local filebuffer

Parameters

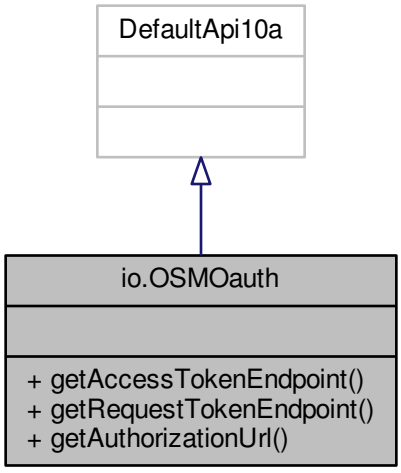
use	if the buffer is to be used.

6.8 io.OSMOauth Class Reference

Inheritance diagram for io.OSMOauth:



Collaboration diagram for io.OSMOauth:



Public Member Functions

- String **getAccessTokenEndpoint** ()
- String **getRequestTokenEndpoint** ()
- String **getAuthorizationUrl** (Token requestToken)

6.8.1 Detailed Description

Useful information for the OAuth process.

See Also

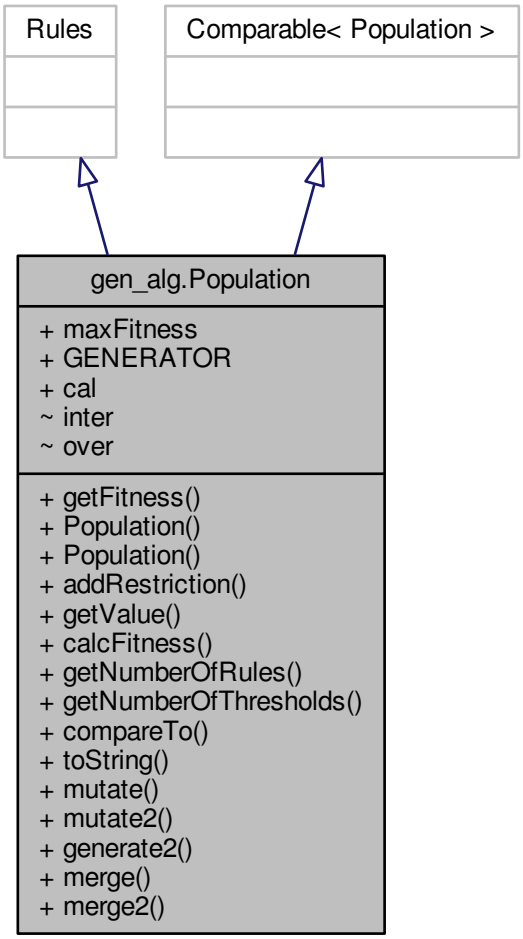
https://github.com/fernandezpablo85/scribe-java

Author

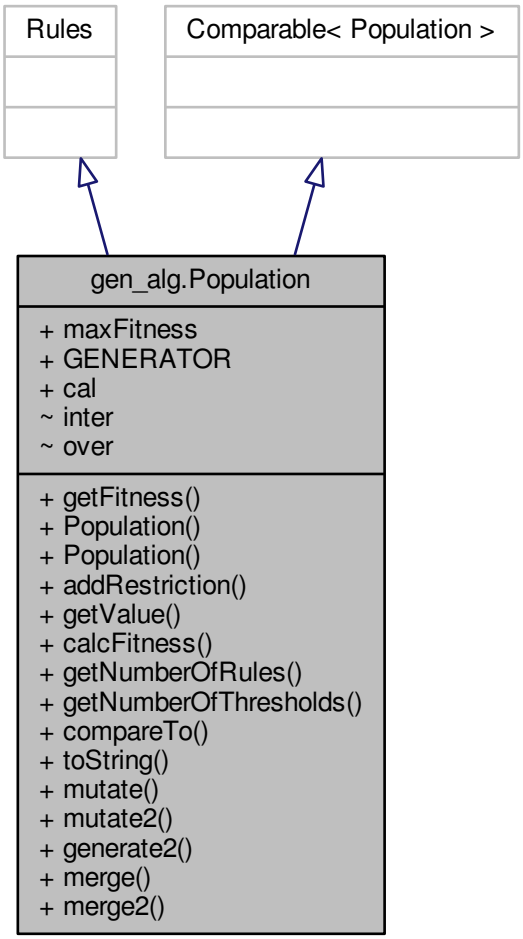
Fabian Pflug

6.9 gen_alg.Population Class Reference

Inheritance diagram for gen_alg.Population:



Collaboration diagram for gen_alg.Population:



Public Member Functions

- int getFitness ()
- **Population** (Population p)
- Population (Collection< String > possible)
- void **addRestriction** (String res)
- double getValue (String key)
- void calcFitness (List< Geometry > truths, List< Collection< Way >> possiblities, List< Point > locations)
- int getNumberOfRules ()
- int getNumberOfThresholds ()
- int compareTo (Population o)
- String toString ()

Static Public Member Functions

- static Population mutate (Population p, Collection< String > possible)

- static Population **mutate2** (Population p, Collection< String > possible)
- static Population generate2 (Population q, Collection< String > possible)
- static Population merge (Population p, Population q)
- static Population merge2 (Population p, Population q)

Static Public Attributes

- static final int maxFitness = 10000
- static int **GENERATOR** = 8
- static long **cal** =0

6.9.1 Detailed Description

Author

Fabian Pflug

6.9.2 Constructor & Destructor Documentation

6.9.2.1 gen_alg.Population.Population (Collection< String > possible)

Initializes a population an puts random tags and values for the rules.

Parameters

possible	a list of all possible tags.

6.9.3 Member Function Documentation

6.9.3.1 void gen_alg.Population.calcFitness (List< Geometry > truths, List< Collection< Way >> possiblities, List< Point > locations)

calculates the fitness value, based on the given lists of examples each list should have the same length and every i-th entry in the list should belong to the i-th entry in every other list. forming a list of datasets.

Parameters

truths	a list of ground truth polygons
possiblities	a list of a collection of possible polygons
locations	a list of locations where to start from.

6.9.3.2 int gen_alg.Population.compareTo (Population o)

compares this population to another, by comparing their fitness values, and on equality their number of rules. a best population would have the highest fitness and the lowest number of rules.

6.9.3.3 static Population gen_alg.Population.generate2 (Population q, Collection< String > possible) [static]

Initializes a population an puts random tags and values for the rules.

Parameters

possible	a list of all possible tags.

6.9.3.4 int gen_alg.Population.getFitness ( )

return the actual fitness value

Returns

the fitness value

6.9.3.5 int gen_alg.Population.getNumberOfRules ( )

Returns the number of rules used by this population

Returns

number of rules

6.9.3.6 int gen_alg.Population.getNumberOfThresholds ( )

returns the number of thresholds defined for this population.

Returns

number of thresholds

6.9.3.7 double gen_alg.Population.getValue (String key)

return the value for the given key in the entry set.

Parameters

key	a possible key

Returns

0 if it does not exist, otherwise a value between 0 and 2.

6.9.3.8 static Population gen_alg.Population.merge (Population p, Population q) [static]

combines this Population with another to form a new randomly merged population.

Parameters

p	the other Population

Returns

a new Population as randomly merged.

6.9.3.9 static Population gen_alg.Population.merge2 (Population p, Population q) [static]

combines this Population with another to form a new randomly merged population.

Parameters

p	the other Population

Returns

a new Population as randomly merged.

6.9.3.10 static Population gen_alg.Population.mutate (Population p, Collection< String > possible) [static]

Initializes a population with another population, by mutating the parameter.

Parameters

p	another Population to mutate
possible	a list of all possible tags.

6.9.3.11 String gen_alg.Population.toString ( )

makes a string representation of this population. which is basically a line as described in Abschnitt 4.3.2.1 auf Seite 17

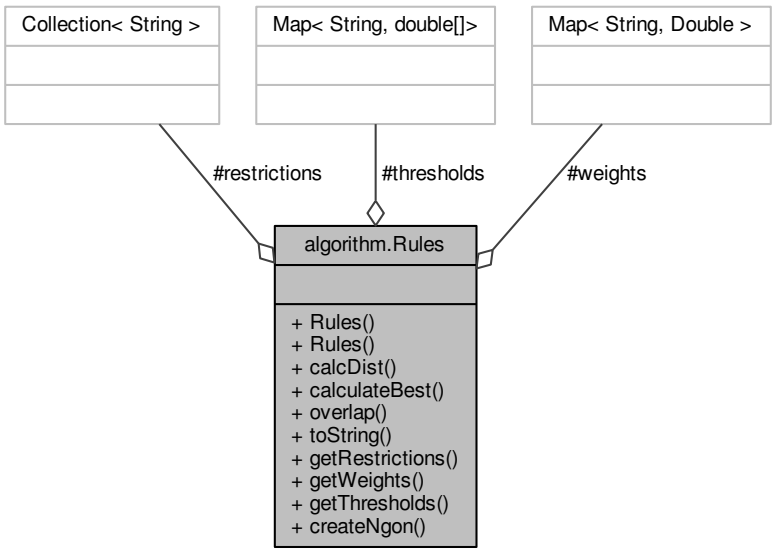
6.9.4 Member Data Documentation

6.9.4.1 final int gen_alg.Population.maxFitness = 10000 [static]

the maximum reachable fitness-value. the higher the better the results, to high an it might cause an overflow.

6.10 algorithm.Rules Class Reference

Collaboration diagram for algorithm.Rules:



Public Member Functions

- Rules ()
- Rules (Collection< String > restrictions, Map< String, Double > rules, Map< String, double[]> thresholds)
- double calcDist (Point c, Way w)
- Way calculateBest (Collection< Way > ways, Point location)
- float overlap (Collection< String > v)
- String toString ()
- Collection< String > **getRestrictions** ()
- Map< String, Double > **getWeights** ()
- Map< String, double[]> **getThresholds** ()

Static Public Member Functions

- static Geometry createNgon (int n, double radius, Point center)

Protected Attributes

- Collection< String > restrictions
- Map< String, Double > weights
- Map< String, double[]> thresholds

6.10.1 Detailed Description

represents our rules and the related restrictions.

(see also Abschnitt 4.2 auf Seite 12)

Author

Fabian Pflug and Peter Zilz

6.10.2 Constructor & Destructor Documentation

6.10.2.1 algorithm.Rules.Rules ( )

Initializes with an empty set of rules, restictions and thresholds.

6.10.2.2 algorithm.Rules.Rules (Collection< String > restrictions, Map< String, Double > rules, Map< String, double[]> thresholds)

Initializes with the given rules and restrictions.

Parameters

restrictions	a set of restrictions.
rules	a set of rules as "osm-key - osm-value to [0..2]"
thresholds	map from singel SUR names to pairs of threshold and radius

6.10.3 Member Function Documentation

6.10.3.1 double algorithm.Rules.calcDist (Point c, Way w)

calculates the weighted distance from a given coordinate to the given way.

Parameters

c	the coordinate to calculate the distance from
w	the OSM-object to calculate the distance to

Returns

the weighted distance by tag and rules

6.10.3.2 Way algorithm.Rules.calculateBest (Collection< Way > ways, Point location)

calculates the way with the nearest weighted distance to the given location.

Parameters

ways	a collection of ways to check.
location	the starting location

Returns

a way from ways

6.10.3.3 static Geometry algorithm.Rules.createNgon (int n, double radius, Point center) [static]

Creates a regular polygon with n vertices. Is is created by rotating, starting at (radius,0) relative to the given center.

Parameters

n	number of vertices
radius	distance from the vertices to the center
center	center around which the polygon is created

Returns

regular polygon

6.10.3.4 float algorithm.Rules.overlap (Collection< String > v)

calculates the overlap of this collection of restrictions with another.

Parameters

v	a collection of restrictions

Returns

a value between 0 and 1, defining the overlap or 0.1 if this Rules has no restrictions.

6.10.3.5 String algorithm.Rules.toString ( )

returns the rules to print out as defined in Abschnitt 4.3.2.1 auf Seite 17

6.10.4 Member Data Documentation

6.10.4.1 Collection<String> algorithm.Rules.restrictions [protected]

a set of all the restrictions.

6.10.4.2 Map<String,double[]> algorithm.Rules.thresholds [protected]

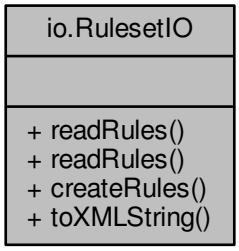
maps names of single OSM-tags to pairs of threshold and radius

6.10.4.3 Map<String,Double> algorithm.Rules.weights [protected]

the rules as "osm-key - osm-value to [0..2]"

6.11 io.RulesetIO Class Reference

Collaboration diagram for io.RulesetIO:



Static Public Member Functions

- static List< Rules > readRules (String filename) throws IOException
- static List< Rules > readRules (File filename) throws IOException
- static Rules createRules (Element ruleElement)
- static String toXMLString (List< Rules > ruleList)

6.11.1 Detailed Description

This class handles the storing of the ruleset. All rules are stored together in an XML file. For more information on the structure of this XML file, see the documentation.

The core functions of this class are readRules(File) and toXMLString(List). The first one reads the file and parses it via Jsoup. The second one parses a list of rules into an xml string, that can be written to a file.

To avoid typos, the names of the used rules are defined as static and final fields.

Author

Peter Zilz

6.11.2 Member Function Documentation

6.11.2.1 static Rules io.RulesetIO.createRules (Element ruleElement) [static]

Parses a Jsoup Element into an instance of Rules.

Parameters

ruleElement	Jsoup representation of the rule-tag with subtags

Returns

new Rules object

6.11.2.2 static List<Rules> io.RulesetIO.readRules (String filename) throws IOException [static]

Reads and parses an xml file that contains a set of rules.

Parameters

filename	name of the file to read and parse

Returns

list of rules, or `null` if the file doesn't exist

Exceptions

IOException	

6.11.2.3 static List<Rules> io.RulesetIO.readRules (File filename) throws IOException [static]

Reads and parses an xml file that contains a set of rules.

Parameters

filename	the file to read and parse

Returns

list of rules, or `null` if the file doesn't exist

Exceptions

IOException	

6.11.2.4 static String io.RulesetIO.toXMLString (List< Rules > ruleList) [static]

Parses a list of Rules into an XML string, that can be stored in a file.

Parameters

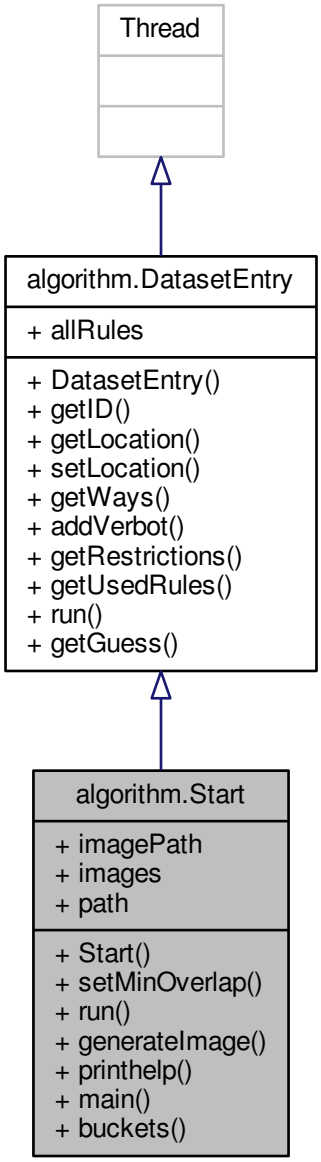
ruleList	list of rules to be stored.

Returns

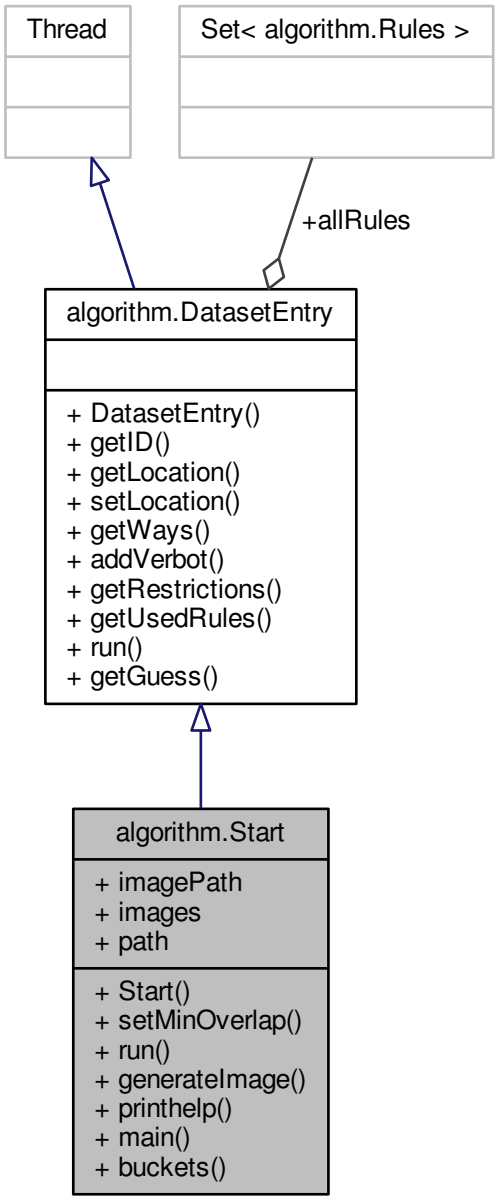
xml string "human readable"

6.12 algorithm.Start Class Reference

Inheritance diagram for algorithm.Start:



Collaboration diagram for algorithm.Start:



Public Member Functions

- Start (Point backup, String id)
- void setMinOverlap (double overlap)
- void run ()
- void generateImage ()

Static Public Member Functions

- static void printhelp ()
- static void main (String[] args) throws IOException
- static void **buckets** (int n, double min, double max, List< Double > list)

Static Public Attributes

- static String imagePath = null
- static boolean images = false
- static String path = null

6.12.1 Detailed Description

The main class for our application. An instance is responsible for one ID in the read in dataset.

Author

Fabian Pflug

6.12.2 Constructor & Destructor Documentation

6.12.2.1 algorithm.Start.Start (Point backup, String id)

creates an entry in the dataset and tries to read in the coordinates from the image belonging to it.

Parameters

backup	a backup coordinate if it is not possible to read metadata from the image.
id	the id of this dataset.

6.12.3 Member Function Documentation

6.12.3.1 void algorithm.Start.generateImage ( )

generates two images for this entry with colored lines for the truth and guess polygon. one is zoomed in, the other one shows a wider perspective

6.12.3.2 static void algorithm.Start.main (String[] args) throws IOException [static]

the main starting point for our application.

Parameters

args	the command line parameters.

Exceptions

IOException	the error thrown if there are problems reading the nesessary files.

6.12.3.3 static void algorithm.Start.printhelp ( ) [static]

prints out a help message

6.12.3.4 void algorithm.Start.run ( )

the worker method, which handels the IO and calculations for this datasetEntry

6.12.3.5 void algorithm.Start.setMinOverlap (double overlap)

Sets the minimum overlap to reach so that no image is created.

Parameters

overlap	the overlap value

6.12.4 Member Data Documentation

6.12.4.1 String algorithm.Start.imagePath = null [static]

the output path, to save the images to

6.12.4.2 boolean algorithm.Start.images = false [static]

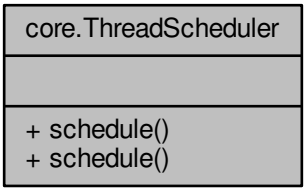
of it is set, then images will be created

6.12.4.3 String algorithm.Start.path = null [static]

The base path, where the input-data is located.

6.13 core.ThreadScheduler Class Reference

Collaboration diagram for core.ThreadScheduler:



Static Public Member Functions

- static void schedule (Collection<?extends Thread > threads, int maxparallel)
- static void schedule (Collection<?extends Thread > threads, double minLoad, int maxparallel)

6.13.1 Detailed Description

A Helper Class to efficiently schedule Threads and therefore have a maximal CPU usage.

Author

Fabian Pflug

6.13.2 Member Function Documentation

6.13.2.1 static void core.ThreadScheduler.schedule (Collection<?extends Thread > threads, int maxparallel) [static]

schedules a collection of Threads, by running maxparallel in parallel. returns if all threads finished running and joined.

Parameters

threads	a collections of runnable threads
maxparralel	the maximal number of threads to run in parallel

6.13.2.2 static void core.ThreadScheduler.schedule (Collection<?extends Thread > threads, double minLoad, int maxparallel) [static]

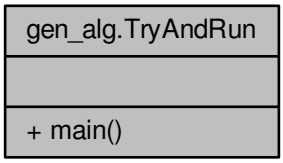
schedules a collection of threads, by starting one thread after another until minLoad is reached. an optimal value for minLoad would be: 1.0 - 1.0/CPUCORES so whenever one core is idle, another thread is started returns if all threads finished running and joined.

Parameters

threads	a collections of runnable threads
minLoad	a minimum number of CPU Load which should be reached.

6.14 gen_alg.TryAndRun Class Reference

Collaboration diagram for gen_alg.TryAndRun:

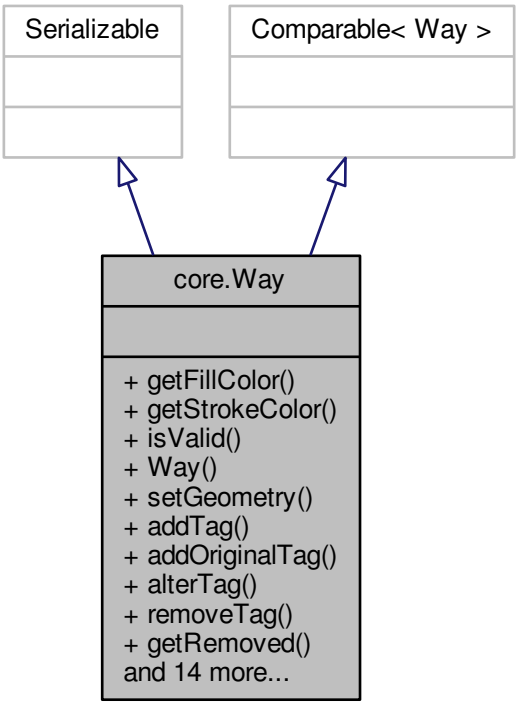


Static Public Member Functions

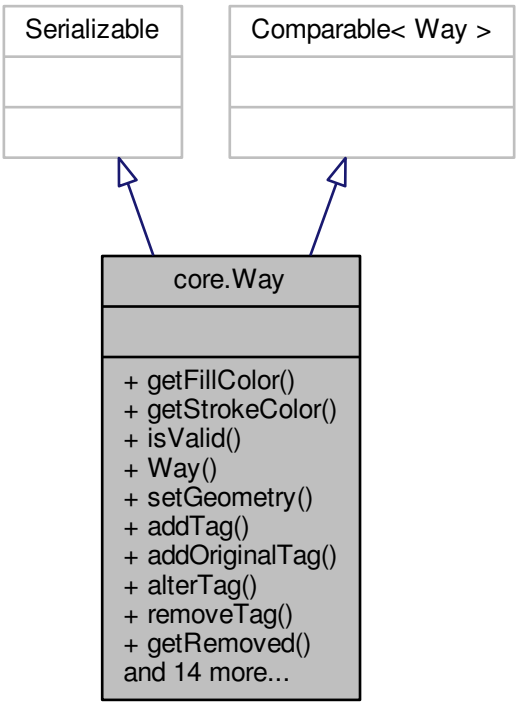
- static void **main** (String[] args) throws Exception, IOException

6.15 core.Way Class Reference

Inheritance diagram for core.Way:



Collaboration diagram for core.Way:



Public Member Functions

- int getFillColor (String key)
- int getStrokeColor (String key)
- boolean isValid ()
- Way (Geometry g)
- void **setGeometry** (Geometry g)
- void addTag (String key, String value)
- void **addOriginalTag** (String key, String value)
- void **alterTag** (String key, String value)
- void removeTag (String key)
- Set< String > **getRemoved** ()
- Map< String, String > getTags ()
- Map< String, String > **getChangedTags** ()
- String getValue (String key)
- boolean isPolygon ()
- boolean isLineString ()
- boolean isPoint ()
- Coordinate[] **getPoints** ()
- double **distanceTo** (Geometry d)
- double getArea ()
- long getId ()
- void setId (long id)

- Geometry getGeometry ()
- boolean **equals** (Object obj)
- int **compareTo** (Way o)

6.15.1 Detailed Description

Representation of an OSM-Object

6.15.2 Constructor & Destructor Documentation

6.15.2.1 core.Way.Way (Geometry g)

Standartkonstruktor, welcher alle nötigen Werte initialisiert.

6.15.3 Member Function Documentation

6.15.3.1 void core.Way.addTag (String key, String value)

fügt der Map ein weiteres key-Value pair hinzu. sollte der Key schon vorhanden sein, dann wird dieser überschrieben.

6.15.3.2 double core.Way.getArea ( )

Returns the area of the bounding box.

Returns

area of the bounding box

6.15.3.3 int core.Way.getFillColor (String key)

returns a fillColor depending of the value of a given key.

6.15.3.4 Geometry core.Way.getGeometry ( )

Returns the current shape.

Returns

current shape.

6.15.3.5 long core.Way.getId ( )

gibt die OSM-ID des Objektes zurück.

Returns

OSM-ID

6.15.3.6 int core.Way.getStrokeColor (String key)

returns a stroke color depending of the value of a given key.

6.15.3.7 Map<String,String> core.Way.getTags ( )

gibt eine Map mit allen Key-Value paaren zurück.

6.15.3.8 String core.Way.getValue (String key)

gibt die Value zu dem gegebenen Key zurück oder null, wenn dieser nicht vorhanden ist.

6.15.3.9 boolean core.Way.isLineString ( )

Checks if the shape of this is a series of lines (e.g. LineString) and not a closed polygon.

Returns

`true` if sequence of lines, `false` otherwise

6.15.3.10 boolean core.Way.isPoint ( )

Checks if the shape of this object is just a single point

Returns

`true` if point, `false` otherwise

6.15.3.11 boolean core.Way.isPolygon ( )

Checks if the shape of this way is closed polygon

Returns

`true` if polygon, `false` if not

6.15.3.12 boolean core.Way.isValid ( )

Checks if the shape of this object is valid.

Returns

`true` if valid, `false` if not

6.15.3.13 void core.Way.removeTag (String key)

Entfernt ein key-Value Paar aus der Map. Sollte es nicht vohanden sein, so wird kein Fehler geworfen.

6.15.3.14 void core.Way.setId (long id)

setzt die OSM-ID dieses Objektes.

Parameters

id	

Anhang A

Daten

A.1 Unser Regelsatz

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ruleset>
  <rule> <!--- the rules to use if none are applicable -->
    <OSMTag  threshold="1e-6" radius="0.0015">landuse - commercial</OSMTag>
    <OSMTag  threshold="1e-6" radius="0.0002">landuse - retail</OSMTag>
    <OSMTag  threshold="1e-7" radius="0.00023">landuse - residential</OSMTag>
    <OSMTag  threshold="1e-5" radius="0.0025">landuse - industrial</OSMTag>
  </rule>
  <rule>
    <restriction>access:dog="no"</restriction>
    <OSMTag weight="0.9" >amenity</OSMTag>
    <OSMTag weight="0.8" >building</OSMTag>
    <OSMTag  threshold="1e-8" radius="0.00023">landuse - residential</OSMTag>
  </rule>
  <rule>
    <restriction>smoking="no"</restriction>
    <OSMTag weight="0.8" >amenity - bank</OSMTag>
    <OSMTag weight="0.8" >building - dormitory</OSMTag>
    <OSMTag weight="0.8" >building - yes</OSMTag>
    <OSMTag weight="1.3" >amenity - college</OSMTag>
    <OSMTag weight="0.95" >name</OSMTag>
    <OSMTag weight="0.8" >leisure</OSMTag>
    <OSMTag weight="0.8" >shop</OSMTag>
    <OSMTag weight="0.95" >building</OSMTag>
    <OSMTag  threshold="1e-6" radius="0.0015">landuse - commercial</OSMTag>
    <OSMTag  threshold="1e-8" radius="0.0002">landuse - retail</OSMTag>
    <OSMTag  threshold="1e-8" radius="0.00023">landuse - residential</OSMTag>
    <OSMTag  threshold="1e-6" radius="0.0025">landuse - industrial</OSMTag>
  </rule>
  <rule>
    <restriction>dog_waste="no"</restriction>
    <restriction>littering="no"</restriction>
    <restriction>noise="no"</restriction>
    <OSMTag weight="0.8" >landuse - recreation_ground</OSMTag>
  </rule>
  <rule>
    <restriction>food="no"</restriction>
    <OSMTag weight="0.5" >building - yes</OSMTag>
  </rule>
```

```
  <rule>
    <restriction>swimming="no"</restriction>
    <OSMTag weight="0.5" >natural - water</OSMTag>
    <OSMTag weight="0.5" >water - pond</OSMTag>
  </rule>
  <rule>
    <restriction>parking="no"</restriction>
    <OSMTag weight="0.5" >name</OSMTag>
  </rule>
  <rule>
    <restriction>littering="no"</restriction>
    <OSMTag weight="0.8">amenity - grave_yard</OSMTag>
  </rule>

        <!-- Added for own Dataset -->
  <rule>
    <restriction>dog_waste="no"</restriction>
    <OSMTag threshold="1e-8" radius="0.00023">landuse - residential</OSMTag>
  </rule>
  <rule>
    <restriction>access:age="21+"</restriction>
    <OSMTag weight="0.7" >building</OSMTag>
  </rule>
</ruleset>
```