

INSTITUT FÜR INFORMATIK

der Ludwig-Maximilians-Universität München



Projektarbeit

Implementierung und Evaluation eines fokussierten Context Graph Crawlers

Aufgabensteller: Dr. Matthias Schubert

Betreuer: Dr. Matthias Schubert

Bearbeiter: Andreas Hartl

Inhaltsverzeichnis

1	Einführung	3
2	Fokussiertes Context Graph Crawling	5
2.1	Fokussiertes Crawling	5
2.2	Der Context Graph	6
2.3	Crawling und Klassifikation	7
3	Implementierung des Context Graph Crawlers	9
3.1	Übersicht	9
3.1.1	Ziele	9
3.1.2	Verwendete Technologien	9
3.2	Wichtige Klassen	10
3.2.1	cgcrawler.cgcrawlerProject	10
3.2.2	crawler	10
3.2.3	cgcrawler.ContextGraphBuilder	10
3.2.4	cgcrawler.CLFTrainer	11
3.2.5	cgcrawler.CGrawler	11
3.2.6	cgcrawler.util.Stemmer	12
4	Bedienung von cgcrawler	13
4.1	Voraussetzungen	13
4.1.1	WordNet	13
4.2	cgcrawler.cfg	13
4.3	Das Programm-Menü	14
4.3.1	Load a different configuration file	15
4.3.2	Load a different project	15
4.3.3	Create a new project	15
4.3.4	Delete project	15
4.3.5	Enter seedpages	16
4.3.6	Build Context Graph	16
4.3.7	Train Classifiers	16
4.3.8	Start crawling	16
4.3.9	Output results	17
4.4	Kurzanleitung	17
5	Evaluation	18

1 Einführung

Eine der am häufigsten durchgeführten Tätigkeiten im Internet, und damit auch eine der wichtigsten, ist die Suche nach Daten und Informationen.

Nicht umsonst sind Suchmaschinen wie Google oder Yahoo aus der täglichen Arbeit im Web heute nicht mehr wegzudenken. Diese Suchmaschinen machen alle Gebrauch der Technik des Web-Crawlings, um Seiten im Internet für die Aufnahme in den Suchmaschinenindex zu finden. Die Essenz des Web-Crawlings besteht aus der Extraktion von Links aus allen Websites, die der Crawler besucht, und deren Aufnahme in eine Liste, die sog. Frontier, zur späteren Verarbeitung. Die Crawler der großen Suchmaschinen nehmen im Allgemeinen jeden gefundenen Link auf, um so viele Seiten wie möglich indexieren zu können. [Brin98]

So nützlich diese Suchmaschinen aber auch sein mögen, so haben sie doch einige Unzulänglichkeiten, die es einerseits interessant machen, selbst das Web zu crawlen, und die dazu anregen, dies mit anderen, ausgefeilteren Techniken zu tun, als einfach jeden gefundenen Link zu verfolgen:

Ein Aspekt, der es interessant macht, selbst das Web zu crawlen, anstatt eine Suchmaschine zu benutzen, ist die Größe des Internets: Lawrence und Giles kamen 2000 zu dem Schluss, dass keine Suchmaschine mehr als 16 Prozent des Webs indiziert.[Law00]Diese Zahl mag heute nicht mehr exakt stimmen, aber allein der Blick auf das Ausmaß des dynamischen Webs, die Menge an Daten, die jeden Tag durch Foren, Blogs und ähnliches hinzukommen, machen klar, dass niemals das gesamte Internet von einer einzigen Suchmaschine aus durchsuchbar sein wird.

Neben dieser Tatsache, dass die großen Suchmaschinen auch nur einen Teil des Internets kennen, besteht ein weiteres Problem darin, dass die Suche nach speziellen Dateitypen mit ihnen nicht einfach ist. Während heutzutage zwar Bilder und Videos gezielt gesucht werden können, so gibt es doch meist keine Möglichkeit, gezielt nach Links zu Dateien anderen Typs zu suchen. Eine Eingabe der Dateiendung in das Suchfeld einer Suchmaschine führt oft nur zu Seiten die sich auf etwas beziehen, dessen Kürzel identisch mit der Dateiendung ist, oder Seiten, die sich auf Programme beziehen, die mit dem gesuchten Dateityp umgehen können, aber nur selten zu den tatsächlich gesuchten Dateien.

Ein weiteres Problem der Suchmaschinen ist die Reihenfolge, in der die Suchergebnisse präsentiert werden. Techniken wie Google's PageRank [Brin98] basieren darauf, dass eine Seite einen umso höheren Rang hat, umso mehr andere Seiten zu ihr linken. Dies führt zwar oft zu guten Ergebnissen, da man davon ausgehen kann, dass eine Seite, zu der viele Links führen, wertvoll ist, aber auch oft zu nutzlosen Ergebnissen, da z.B. populäre Seiten die gesuchten Begriffe in einem andern Kontext benutzen, oder die gesuchten Begriffe nur erwähnen, ohne weitere Informationen darüber zu geben.

Eine spezielle Aufgabe, die mit Hilfe vorhandener Suchmaschinen recht schwer ist,

1 Einführung

ist die Suche nach 3d Modellen im VRML Format. Um diese Suche zu vereinfachen und automatisieren wurde daher im Rahmen dieser Arbeit ein Crawler erstellt, der die Technik des fokussierten Context Graph Crawlings nutzt, die es ermöglicht, Seiten im Internet aufgrund ihrer Ähnlichkeit zu vorhandenen Seiten zu suchen und finden.

Der Crawler wurde mit den Zielen entwickelt, problemlos von Laien genutzt werden zu können, als auch sehr einfach anpassbar zu sein um z.B. nach anderen Dateitypen zu suchen.

Die folgenden Kapitel befassen sich kurz mit der Theorie des fokussierten Context Graph Crawlings, als auch mit den Implementationsdetails und der Bedienung des Crawlers. Das letzte Kapitel gibt einen kurzen Überblick über die Ergebnisse des Einsatzes des Crawlers in der Praxis.

2 Fokussiertes Context Graph Crawling

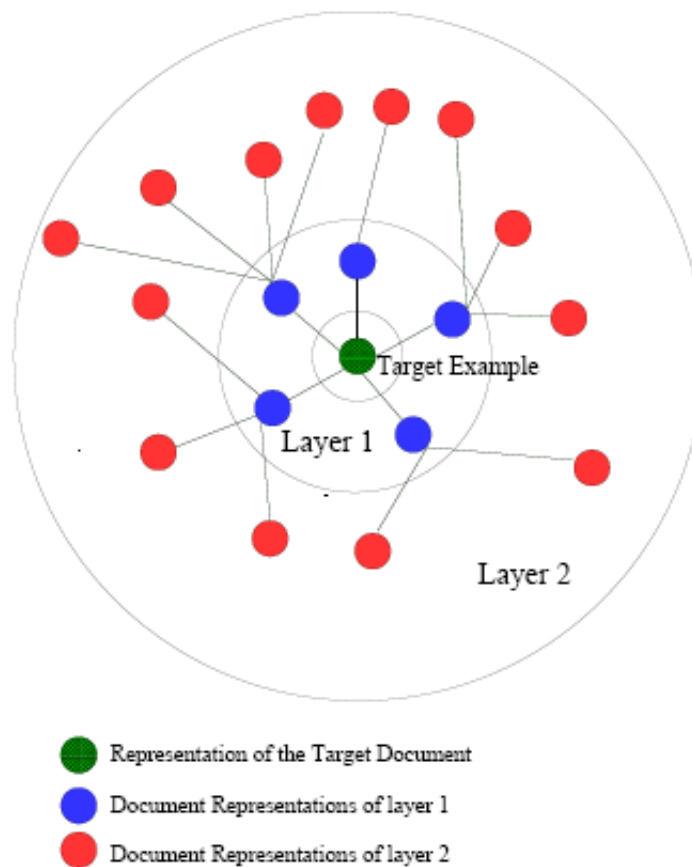
2.1 Fokussiertes Crawling

Das Ziel des fokussierten Crawlings ist es, die durch die Größe und Dynamik des Internets entstehenden Probleme rekursiver Crawler, wie sie von den meisten Suchmaschinen benutzt werden, zu lösen, indem nur ein Teil des Webs gecrawlt wird, der sich auf ein bestimmtes Thema bezieht. Der ideale fokussierte Crawler findet die maximale Menge relevanter Seiten, während er eine minimale Menge irrelevanter Seiten besucht. [Dil00]

Das Problem, das sich bei fokussiertem Crawling stellt, ist es, die Seiten auf dem Crawl-Weg zu einer relevanten Seite angemessen zu gewichten. Frühere fokussierte Crawler hatten vorwiegend Probleme, wenn thematisch relevante Seiten verlässlich durch Links auf off-topic Seiten gefunden wurden. Ein weiteres Problem für fokussiertes Crawling ist die einseitige Linkstruktur des Internets: Links können nur in eine Richtung verfolgt werden, was es ohne Hilfsmittel unmöglich macht, thematisch wahrscheinlich relevante Seiten zu erreichen, die sich in der Baumstruktur einer Website weiter oben befinden, wenn der Crawler über einen externen Link auf eine Webpage kommt, die sich weiter unten im Baum befindet.

Um diese Probleme zu lösen haben Diligenti et al. das Prinzip des Context Graphs eingeführt [Dil00], um Hierarchien in Webseiten zu modellieren und auszunutzen.

2.2 Der Context Graph



Darstellung des Context Graphs aus [Dil00]

Der Context Graph soll eine Repräsentation des Kontextes sein, in dem relevante Seiten üblicherweise gefunden werden. Seine Konstruktion beginnt mit einer vom Benutzer gegebenen relevanten Seite, der sog. Targetpage oder Seedpage. Von dieser Seite ausgehend werden dann die Ebenen (Layer) des Graphen über Backlinks konstruiert: In Layer 1 befinden sich die Seiten, die Links zur Targetpage haben. In Layer 2 befinden sich Seiten, die Links zu den Seiten in Layer 1 haben. Dies wird so fortgeführt für eine vom User bestimmbare Anzahl Ebenen.

Dieser Prozess wird für jede Seedpage wiederholt, sofern mehrere gegeben sind. Der endgültige Context Graph für alle gegebenen Seedpages ist dann die Kombination der Graphen der einzelnen Seiten. Damit der Graph bei einer großen Menge Seedpages und Ebenen nicht zu groß wird (da die Anzahl Seiten im Graph linear mit der Anzahl der Seedpages und exponentiell in der Anzahl der Ebenen steigt), beschränkt man die Anzahl

Seiten, die pro Ebene gespeichert werden, auf einen festen Wert, und wählt dann nur so viele Seiten zufällig aus der Gesamtmenge der Seiten dieser Ebene aus. Außerdem gibt es noch die Konvention dass, wenn zwei Seiten einer Ebene i von der selben Seite aus erreicht werden können, diese Seite zweimal in Ebene $i+1$ aufgenommen wird.

Zur Veranschaulichung ein Beispiel: Gesucht werden sollen Wissenschaftliche Arbeiten der Informatik. Relevante Webpages sind also solche, die Links zu Arbeiten in pdf Format haben. Dies werden hauptsächlich die Seiten von Forschern an Universitäten sein. Diese Seiten wiederum werden sehr wahrscheinlich von Seiten von Informatik-Instituten aus verlinkt sein. Und die Seiten der Institute werden sicherlich von den Websites von Universitäten aus verlinkt sein. Ein fertiger Context Graph wird also mit guter Wahrscheinlichkeit so aussehen: die Seedpages sind die Seiten der Forscher, in Ebene 1 befinden sich Seiten von Informatik-Instituten, und in Ebene 2 die Seiten von Universitäten. Hier zeigt sich schon gut das Ziel des fokussierten Context Graph Crawlings: ein anderer fokussierter Crawler, der nach Wissenschaftlichen Arbeiten der Informatik sucht, und die Website einer Universität besucht, wird dieser wenig Gewicht geben, da so eine Seite wahrscheinlich keine der Begriffe enthält, die zum Kontext der gesuchten Arbeiten gehören. Wenn aber der Context Graph Crawler solch eine Seite besucht, weiß er, dass er über eine solche Seite eine Targetpage in eventuell nur zwei Schritten erreichen kann. Somit bekommen auch eigentliche off-topic Seiten ein hohes Gewicht, wenn aufgrund des Context Graphs bekannt ist, dass solche Seiten verlässlich zu guten Ergebnissen führen.

Das sich noch stellende Problem ist nun das Finden der Seiten in der jeweils höheren Ebene, was ja aufgrund der oben schon erwähnten Einseitigkeit der Links im Internet in diese Richtung nicht direkt geht. Hierzu muss man sich mit bestehenden Suchmaschinen behelfen, welche die Suche nach Backlinks anbieten. Da aber inzwischen die großen Suchmaschinen wie Google und Yahoo diese Funktionalität bereitstellen, stellt das kein Problem mehr da, und der Context Graph kann automatisch erstellt werden; der User muss nur einige Seedpages zur Verfügung stellen.

2.3 Crawling und Klassifikation

Das eigentliche Crawling ist recht einfach wenn der Context Graph erstellt wurde: Der Crawler besitzt für jede Ebene des Context Graphs eine Queue (Frontier). Besucht der Crawler eine Webpage, so klassifiziert er sie, um herauszufinden, welcher Ebene im Context Graph die Page zugeordnet werden kann, ob sie nicht relevant ist, oder ob es sich schon um eine Targetpage handelt. Wird sie als relevant, d.h. einer Ebene im Context Graph zugehörig, klassifiziert, so werden alle Links extrahiert und der entsprechenden Frontier hinzugefügt. Der Crawler besucht nacheinander alle Seiten in den Frontiers, angefangen immer bei der niedrigsten Frontier, d.h. der, die Links zu Seiten enthält, die mit höchster Wahrscheinlichkeit direkte Nachbarn von Targetpages sind. Wird eine Seite als irrelevant klassifiziert, werden ihre Links nicht weiter verfolgt. Außerdem werden die URLs aller besuchten Seiten gespeichert, um keine Seite wiederholt zu besuchen, falls ein weiterer Link zu ihr gefunden wird.

Um die besuchten Seiten korrekt klassifizieren zu können, braucht der Crawler noch

2 Fokussiertes Context Graph Crawling

einen Classifier, der aus dem Context Graph erstellt wird. Für die Art des Classifiers gibt es mehrere Optionen, die zum selben Ergebnis führen, daher wird hier nicht weiter darauf eingegangen. Entscheidend ist, dass der Classifier bei Eingabe des Textes einer Webpage diese einer Ebene im Context Graph zuordnen kann, bzw. sie als nicht-relevant klassifizieren kann.

3 Implementierung des Context Graph Crawlers

3.1 Übersicht

3.1.1 Ziele

Die Aufgabe des Crawlers ist es, mit Hilfe des Context Graph Prinzips 3d Modelle im VRML Format (.wrl Dateien) im Internet zu finden, oder genauer gesagt, Webseiten, die direkte Links zu VRML Dateien enthalten.

Die zwei großen Ziele bei der Implementation waren

1. Die Bedienung des Crawlers soll einfach sein, so dass auch ein Laie damit zurechtkommt
2. Der Crawler soll nicht ausschliesslich nach VRML Dateien suchen können, sondern in der Lage sein, sowohl nach Webseiten zu suchen, deren Thema vom Benutzer vorgegeben wird, als auch nach Dateien mit vom Benutzer bestimmtem Format

3.1.2 Verwendete Technologien

Der Context Graph Crawler *cgcrawler* wurde in Java mit der JDK 1.6.0 geschrieben. Zur Speicherung der Daten greift er mittels JDBC auf eine Oracle Datenbank zu, die auf der Maschine, auf der er ausgeführt wird, laufen muss.

Der Code setzt auf dem Crawler-Framework von Dr. Matthias Schubert auf. Dazu gehören die Pakete *dm* und *smpro*, deren Funktionalität für die Erstellung und Benutzung des Klassifikators genutzt wird, mit dem Webseiten den Context Graph Ebenen zugeordnet werden. Das Paket *dm* enthält Datentypen und Algorithmen zum Datamining auf Text-Dokumenten. Das Paket *smpro* enthält diversen Code zum Datamining mit Hilfe der WEKA Bibliothek.[Wit05]

Zum Parsen von Webseiten und der Umwandlung in das TextDoc Format des *dm* Paketes wird der Code verwendet, der im Rahmen der Projektarbeit von Sarah Blume erstellt wurde. Dieser verwendet die Tidy Bibliothek [Tidy] zum Parsen von HTML.

Die zur Erstellung des Context Graphs notwendigen Backlinks werden über die Suchmaschine Yahoo gesucht, dazu wird die Yahoo! Search API [Yahoo] verwendet.

Der Crawler führt Stemming auf allen Webseiten durch, dazu verwendet er WordNet 2.0 [Word], mit dem er über die JWNL API [JWNL] kommuniziert.

3.2 Wichtige Klassen

3.2.1 `cgcrawler.cgcrawlerProject`

Der *cgcrawler* wurde so designed, dass mehrere "Projekte" unabhängig voneinander existieren können. Ein Projekt wird über seinen Namen identifiziert, und enthält mehrere Eigenschaften. Dies sind vor allem die Parameter des Context Graphs: die Anzahl der Ebenen und die Anzahl der Webseiten, die maximal pro Ebene gespeichert werden. Alle Daten, die der Crawler erhält oder erstellt, wie der Context Graph selber, der Klassifikator, die Listen von gefundenen Targetpages, bekannten Seiten oder Seiten in der Frontier, sowie diverse Hilfs- und Sicherungs-Daten, sind immer einem Projekt zugeordnet.

Die Klasse *cgcrawlerProject* stellt die Repräsentation eines Projektes dar, und stellt die Funktionalität zur Verfügung, Projekte zu erstellen, laden und löschen, und vor allem zum Datenaustausch zwischen Crawler und Datenbank.

Wird ein neues Projekt erstellt, so werden alle Datenbank-Tabellen erstellt, die der Crawler verwenden kann. Dabei wird dem Tabellennamen jeweils der Projektname vorangestellt (z.b. `Testcrawl_Targetpages`, `Testcrawl_Frontiers`, etc.).

Die Klasse stellt Methoden zum Schreiben und Lesen aller Daten, die der Crawler verwendet, zur Verfügung, so dass die anderen Klassen des Crawlers keine Information über die Datenbankverbindung brauchen, und, unabhängig vom geladenen Projekt, transparent in die Datenbank schreiben bzw. aus ihr lesen können.

3.2.2 `crawler`

Die Klasse *crawler* verbindet die verschiedenen Teile des Crawlers, stellt ein User Interface per Kommandozeile zur Verfügung, und liest die Konfigurationsdatei ein.

In dieser Klasse befindet sich die main Methode des Crawlers, d.h. sie ist der Eintrittspunkt, wenn der Crawler gestartet wird. Sie liest zuerst die Konfigurationsdatei (`cgcrawler.cfg`), in der sich die zur Verbindung zur Oracle Datenbank nötigen Daten befinden. Mit diesen wird dann versucht, eine Verbindung zur Datenbank herzustellen. Ist dies erfolgreich, so wird automatisch das zuletzt verwendete Projekt geladen, und das Benutzermenü angezeigt. Von diesem aus lassen sich alle Funktionen des Crawlers starten (siehe Kapitel 4 für Details).

3.2.3 `cgcrawler.ContextGraphBuilder`

Die Klasse *ContextGraphBuilder* ist für die Erstellung des Context Graphs verantwortlich.

Neben einiger Aktionen zur Zustands-Sicherung und -Wiederherstellung im Falle eines Absturzes arbeitet die Klasse folgendermaßen:

Zuerst liest sie die für den Graphen notwendigen Informationen aus dem geladenen Projekt: Anzahl Ebenen, Anzahl der maximal pro Ebene zu speichernden Seiten und URLs der Seedpages. Ein Thread (*LinkHandlerThread*) ist zuständig für die Suche nach Backlinks. Im ersten Durchlauf werden alle Backlinks für die Seedpages gesucht, in den weiteren dann die Backlinks zu den gefundenen Seiten der vorherigen Ebene. Werden

mehr Backlinks gefunden, als maximal pro Ebene gespeichert werden dürfen, so werden erst alle Backlinks gesucht, und anschließend die erlaubte Anzahl per Zufall daraus ausgewählt. Die URLs aller gefundenen Backlinks werden (gemeinsam mit der Ebene im Context Graph der sie angehören) in einer Queue gespeichert, aus der sich mehrere andere Threads bedienen. Diese *WebAccessThreads* laden jeweils die zur URL gehörende Webpage, parsen sie, und wandeln sie in das interne Format um, dass der Crawler für Webseiten benutzt. Die so entstandenen *TextDocs* werden dann in einer Queue gespeichert, aus der sie von wiederum einem eigenen Thread (*DocumentHandlerThread*) gelesen werden, der dafür verantwortlich ist, sie, zusammen mit der zugehörigen Ebene im Graph, in die Datenbank zu schreiben.

Auf diese Weise entsteht der Context Graph in der Datenbank, als Liste von *TextDocs* gepaart mit einer Zahl, die ihre Ebene im Graph repräsentiert.

3.2.4 cgcrawler.CLFTrainer

Die Klasse *CLFTrainer* erzeugt einen Klassifikator aus dem Context Graph, der beim Crawlen zur Klassifikation von unbekanntem Webseiten verwendet wird.

Vor dem Erstellen des Klassifikators werden die Daten im Context Graph noch vorbereitet: Es wird ein *Feature Selector* erstellt und auf alle Dokumente im Graph angewandt. Dadurch werden seltene (und damit unwichtige) Wörter aus den Dokumenten entfernt. Außerdem wird noch Stemming auf allen Dokumenten durchgeführt. Dabei werden alle Wörter auf ihren Stamm reduziert, so dass ein Wort, das in verschiedenen Formen vorkommt (z.B. 'run' und 'running'), die korrekte Gewichtung im Klassifikator zugewiesen bekommt. Beim Stemming wird außerdem festgestellt, wie viele korrekte englische Wörter in einem Dokument enthalten sind, so dass nicht-englische Seiten entfernt werden können.

Nach dieser Vorbereitung wird dann, mit Hilfe der *dm* und *smpro* Pakete, ein k-Nächste-Nachbarn Klassifikator mit einem k-Means Model Maker erstellt, und einer Datei gespeichert, die den Namen des aktuellen Projekts mit der Endung *.clf* erhält.

3.2.5 cgcrawler.CGcrawler

Die Klasse *CGCrawler* erfüllt einige einfache Aufgaben für den Crawler, wie z.B. das Erstellen einer Webseite mit Links zu allen gefundenen Targetpages - ihre Hauptfunktionalität liegt aber beim eigentlichen Crawlen.

Nach der Initialisierung (dem Einlesen des Klassifikators und der Frontier, dem Wiederherstellen des Zustandes des letzten Laufs, etc.) wird das Crawlen durch mehrere *CrawlThreads* ausgeführt. Diese holen sich zuerst jeweils eine URL aus der Frontier. Die zugehörige Webseite wird dann geladen und in ein *TextDoc* geparkt. Dann wird die Seite gestemmt, wobei auch wieder festgestellt wird ob es sich um eine englische Seite handelt oder nicht. Nicht-englische Seiten werden nicht weiter behandelt. Handelt es sich um eine korrekte Seite, so wird diese anschließend mit Hilfe des Klassifikators klassifiziert, also einer Ebene im Context Graph zugeordnet. Wird sie als nicht-relevant klassifiziert (d.h. der äußersten Ebene des Graphen zugeordnet) wird sie nicht weiter behandelt. Ist sie

dagegen relevant, so wird überprüft, ob es sich um eine Targetpage handelt, und wenn ja, so wird sie in der entsprechenden Tabelle in der Datenbank gespeichert. Aus allen relevanten Seiten werden die Links extrahiert und in die ihrer Ebene entsprechende Frontier geschrieben, solange es sich nicht um eine bereits bekannte URL handelt. Entsprechend wird auch die URL jeder Seite nach der Bearbeitung gespeichert, um niemals eine schon bekannte Seite zu besuchen.

Die Klasse *CGCrawler* ist auch noch verantwortlich für die Verwaltung der Frontier, der Liste aller zu besuchenden URLs. Dies geschieht aus Performance-Gründen zweistufig: Die Frontier befindet sich im Hauptspeicher, um schnellen Zugriff zu garantieren, und wird regelmäßig in die Datenbank geschrieben, um nur möglichst wenig Datenverlust zu haben wenn der Crawler beendet wird.

3.2.6 *cgcrawler.util.Stemmer*

Die Klasse *Stemmer* dient dazu, *TextDocs* zu stemmen, d.h. alle Wörter in einem Dokument auf ihren Stamm zu überführen. Hat ein *TextDoc* z.B. die Form {"run}=1, "running}=2} (die Zahl stellt dar, wie oft das Wort im Dokument vorkommt), so wird es beim Stemming in {"run}=3} überführt. Auf diese Weise wird sichergestellt, dass ein Wort, das häufig in anderen Formen vorkommt, nicht als irrelevant angesehen wird wenn es in einer selteneren Form auftritt.

Die Klasse wurde auf Geschwindigkeit ausgelegt, zu Lasten von Speichereffizienz: Jedes unbekannte Wort wird nur beim ersten Auftreten in WordNet gesucht (was relativ lange dauert), und anschließend in einer HashMap gespeichert, so dass der Stamm jedes Wortes in konstanter Zeit gefunden werden kann, sobald es einmal in WordNet gesucht wurde. Die *cgcrawlerProject* Klasse bietet deswegen auch die Möglichkeit, diese HashMap zu speichern und laden, so dass das Stemming (und der davon abhängige Test auf englische Sprache eines Dokuments) beim Crawlen effizient ist, da *CGCrawler* die HashMap in regelmäßigen Abständen speichert, und beim Neustart wieder lädt.

4 Bedienung von cgcrawler

4.1 Voraussetzungen

Um den *cgcrawler* ausführen zu können, müssen einige Voraussetzungen erfüllt sein:

1. Der Computer muss über eine aktuelle Java Installation verfügen
2. Auf dem Computer (oder im Netzwerk) muss eine Oracle Datenbank vorhanden sein; es wird ein entsprechender User Account vorausgesetzt
3. Es muss eine bestehende Internet-Verbindung geben, entweder direkt oder per Proxy

4.1.1 WordNet

Damit der Crawler korrekt arbeitet muss WordNet 2.0 installiert sein. WordNet ist Free-ware, und kann unter <http://wordnet.princeton.edu/oldversions> heruntergeladen werden. Zu Beachten ist, dass die korrekte Version 2.0 installiert wird, da andere Versionen inkompatibel sind.

Nach der Installation von WordNet muss noch die Datei *JWNLproperties.xml* (zu finden im *cgcrawler* Verzeichnis) angepasst werden: am Ende der Datei ist der Wert des Parameters *dictionary_path* auf den Pfad zu setzen, unter dem das WordNet Dictionary installiert wurde (die Zeile sieht dann z.B. so aus: `<param name="dictionary_path" value="c:\programme\wordnet\2.0\dict" />`).

4.2 cgcrawler.cfg

Die Konfigurationsdatei *cgcrawler.cfg* wird vom *cgcrawler* zu Programmstart geöffnet. Aus ihr werden die benötigten Parameter zur Verbindung mit der Datenbank, so wie einige andere Informationen gelesen.

Die folgenden drei Einträge müssen immer vorhanden sein:

dburl	Der Pfad, über den die Datenbank erreicht werden kann (z.B. dburl = JDBC:Oracle:oci8:@ wenn die Datenbank lokal vorhanden ist)
dbuser	Der Username des Datenbank Accounts
dbpass	Das Passwort des Datenbank Accounts

4 Bedienung von cgcrawler

Wenn der Computer nur über einen Proxy auf das Internet zugreifen kann, müssen außerdem die folgenden drei Einträge vorhanden sein:

useproxy Dieser Parameter muss auf true gesetzt sein (useproxy = true)

proxy Die URL des Proxy Servers

proxyport Der Port, über den der Proxy Server anzusprechen ist

Optional ist der Parameter *numcrawlthreads*, über den sich bestimmen lässt, wie viele Threads der Crawler benutzen soll, um Seiten aus dem Internet zu laden (diese Zahl entspricht dann auch der maximal gleichzeitig vorhandenen Web-Verbindungen, die der Crawler öffnet). Ein guter Wert für diesen Parameter ist 30.

Schließlich kann die Konfigurationsdatei noch einen Eintrag *lastproject* haben. Dieser wird automatisch vom Crawler gesetzt, und dient dazu, bei Programmstart automatisch das zuletzt geladene Projekt zu laden.

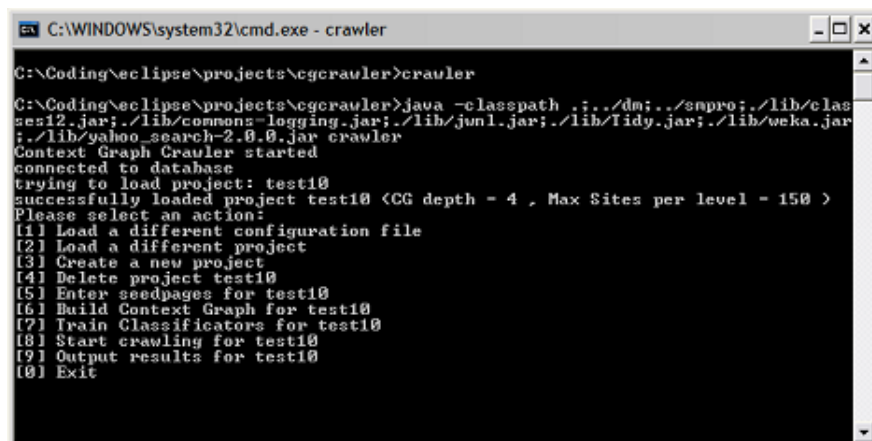
4.3 Das Programm-Menü

Der Crawler kann mit der Kommandozeile

```
java -classpath ../dm;../smpro;./lib/classes12.jar;./lib/commons-logging.jar;./lib/jwnl.jar;
./lib/Tidy.jar;./lib/weka.jar;./lib/yahoo_search-2.0.0.jar crawler
```

gestartet werden. Die Dateien *crawler.bat* (für Windows) und *crawler.sh* (für Linux) führen diese Kommandozeile aus, so dass es reicht, diese Dateien auszuführen, um den Crawler zu starten.

Sind die Voraussetzungen aus 4.1 erfüllt und die Parameter zur Datenbankverbindung in der Konfigurationsdatei korrekt, erscheint das Bedienungs-Menü, über das sich der Crawler steuern lässt:



```
C:\WINDOWS\system32\cmd.exe - crawler
C:\Coding\eclipse\projects\cgcrawler>crawler
C:\Coding\eclipse\projects\cgcrawler>java -classpath ../dm;../smpro;./lib/classes12.jar;./lib/commons-logging.jar;./lib/jwnl.jar;
./lib/Tidy.jar;./lib/weka.jar crawler
Context Graph Crawler started
connected to database
trying to load project: test10
successfully loaded project test10 <CG depth = 4 , Max Sites per level = 150 >
Please select an action:
[1] Load a different configuration file
[2] Load a different project
[3] Create a new project
[4] Delete project test10
[5] Enter seedpages for test10
[6] Build Context Graph for test10
[7] Train Classifiers for test10
[8] Start crawling for test10
[9] Output results for test10
[0] Exit
```

Konnte keine Verbindung zur Datenbank hergestellt werden, so steht nur die Option zum Laden einer anderen Konfigurationsdatei zur Verfügung. Existiert noch kein Projekt, oder kann das Projekt, das als *lastproject* in der Konfigurationsdatei angegeben ist, nicht geladen werden, so stehen nur die Optionen zum Laden oder Erstellen eines neuen Projekts zur Verfügung.

Im Folgenden wird eine kurze Beschreibung aller Optionen gegeben.

4.3.1 Load a different configuration file

Bei Auswahl dieses Menüpunktes kann anschließend der Name einer anderen Konfigurationsdatei eingegeben werden, die dann geladen wird. Das Programm versucht dann auch sofort, eine neue Datenbankverbindung mit den Parametern aus der Datei herzustellen.

4.3.2 Load a different project

Über diesen Menüpunkt lässt sich ein anderes Projekt laden. Es muss nur der Name angegeben werden, anschließend wird es geladen, sofern es existiert. Dieses Projekt wird dann auch als *lastproject* in die Konfigurationsdatei geschrieben, und beim nächsten Programmstart automatisch geladen.

4.3.3 Create a new project

Hiermit lässt sich ein neues Projekt erstellen. Nach Auswahl des Menüpunktes muss zuerst ein Name für das Projekt eingegeben werden. Anschließend fragt das Programm nach der Anzahl der Ebenen, die der Context Graph dieses Projektes bekommen soll. Gute Werte sind hier 3 bis 5. Anschließend muss die Anzahl der Seiten angegeben werden, die bei Erstellung des Context Graph pro Ebene maximal gespeichert werden sollen. Gute Werte dafür liegen zwischen 100 und 300.

Als nächstes kann angegeben werden, ob der Crawler für dieses Projekt nur nach Webseiten, die bestimmte Dateitypen enthalten, suchen soll. Dazu muss zuerst die Anfrage mit Eingabe von "yes" bestätigt werden, und anschließend der Dateityp (die Dateierdung, etwa "pdf"), nach dem gesucht werden soll, angegeben werden. Wird dies gemacht, so wird der Crawler später nur Webseiten, die einen oder mehrere Links zu Dateien des angegebenen Typs enthalten, in die Liste der gefundenen Seiten aufnehmen. Andernfalls wird er alle Seiten aufnehmen, die als Targetpage klassifiziert werden.

Entsprechen alle angegebenen Werte den Voraussetzungen (d.h. der Projektname existiert noch nicht, keine negativen Werte für die Anzahl Graph Ebenen, etc.), so wird das neue Projekt angelegt und geladen, und die anderen Menü-Optionen stehen ab sofort dafür zur Verfügung. Außerdem wird dieses Projekt ab sofort beim Programmstart automatisch geladen, solange bis ein neues Projekt geladen oder erstellt wird.

4.3.4 Delete project

Über diesen Menüpunkt kann das aktuell geladene Projekt gelöscht werden. Nach der Bestätigung mittels Eingabe von "yes" werden alle Daten des Projektes aus der Datenbank

gelöscht.

4.3.5 Enter seedpages

Hiermit können die Seedpages, d.h. die Webseiten der Art, nach der der Crawler suchen soll, für ein neues Projekt in die Datenbank aufgenommen werden. Die URLs der Seedpages müssen in einer Textdatei stehen. Der Name der Textdatei kann dann nach Auswahl des *Enter Seedpages* Menüpunkts angegeben werden, worauf die Datei gelesen wird und alle URLs darin dem Projekt hinzugefügt werden.

Die Anzahl und Güte der Seedpages ist ausschlaggebend für den Erfolg des Crawlers. Die Anzahl sollte möglichst hoch sein, und die Seiten sollten möglichst viel relevanten Text enthalten.

4.3.6 Build Context Graph

Nach Auswahl dieses Menüpunkts erstellt das Programm den Context Graph für die vorhandenen Seedpages.

4.3.7 Train Classificators

Über diesen Menüpunkt wird der für das Crawlen notwendige Klassifikator trainiert. Dazu muss die Anzahl der *Featurewords* angegeben werden. Aus allen Dokumenten im Context Graph werden nur die *Featurewords* häufigsten Wörter verwendet, um den Klassifikator zu trainieren. Eine zu kleine Anzahl führt dazu, dass eventuell wichtige, d.h. häufig vorkommende, Wörter nicht in den Klassifikator aufgenommen werden, eine zu hohe Zahl kann dazu führen dass sehr viele unwichtige, d.h. selten vorkommende, Wörter unnötigerweise aufgenommen werden. Ein guter Wert für die Anzahl der *Featurewords* liegt zwischen 2000 und 3000.

Nach Angabe dieses Wertes wird der Klassifikator erstellt und in einer Datei gespeichert, die den Namen des aktuellen Projektes mit der Endung *.clf* trägt.

4.3.8 Start crawling

Dieser Menüpunkt startet den eigentlichen Crawl Prozess, unter der Voraussetzung, dass Context Graph und Klassifikator existieren.

Wird das erste mal für das geladene Projekt gecrawlt, dann können nach Auswahl dieses Menüpunktes noch zusätzliche URLs von Webseiten angegeben werden, bei denen der Crawler startet. Die Eingabe von "cancel" beendet die Abfrage. Werden hier keine zusätzlichen Seiten angegeben, so werden nur die Seedpages als Startseiten benutzt. Anschließend startet der Crawl Prozess, der ohne Unterbrechung das Internet nach Zielseiten durchsucht.

Wurde schon einmal für dieses Projekt gecrawlt, so lädt der Crawler automatisch den Zustand, bei dem er beendet wurde, und setzt dann dort das Crawling fort.

4.3.9 Output results

Bei Auswahl dieses Menüpunktes wird eine Webseite erstellt (mit dem Namen des geladenen Projektes), die anklickbare Links auf alle Seiten enthält, die der Crawler bisher für dieses Projekt gefunden hat.

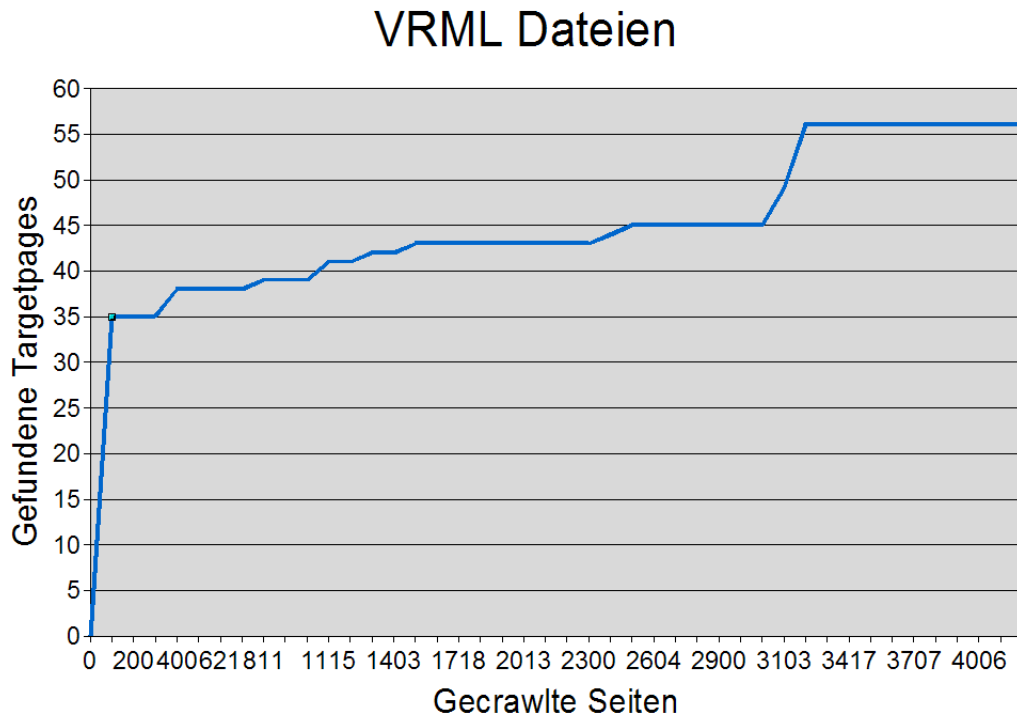
4.4 Kurzanleitung

Dieser Abschnitt gibt einen kurzen Überblick über die Schritte, um den Crawler das erste mal zu benutzen.

- Zuerst muss sichergestellt sein, dass alle Voraussetzungen aus Abschnitt 4.1 für den korrekten Ablauf erfüllt sind, vor allem dass WordNet installiert ist, und die Datei *JWNLproperties.xml* den korrekten Installationspfad enthält
- Anschließend sind die für die Verbindung zur Datenbank (und für die Verbindung zum Internet, falls ein Proxy dazu nötig ist) notwendigen Parameter in die Konfigurationsdatei *cgcrawler.cfg* einzutragen
- Nun kann das Programm über *crawler.bat* bzw. *crawler.sh* gestartet werden
- Anschließend muss zuerst ein neues Projekt mittels Menüpunkt 3 erstellt werden
- Für den Context Graph Crawler ist es wichtig, dass er weiß, wie die Seiten aussehen, die er finden soll. Dazu muss er einige solche Seiten, die *Seedpages*, kennen. Diese müssen 'per Hand', also z.B. mittels einer Suchmaschine, gesucht werden, und die URLs in eine Textdatei geschrieben werden, die anschließend über Menüpunkt 5 vom Crawler gelesen werden kann.
- Jetzt kann mittels Menüpunkt 6 der Context Graph erstellt werden
- und der Klassifikator mittels Punkt 7
- Nun sind alle notwendigen Daten vorhanden, und über Menüpunkt 8 kann das Crawlen gestartet werden

Der Crawler kann jederzeit beendet werden. Beim nächsten Start kann dann sofort Menüpunkt 8 ausgewählt werden, um dort weiter zu crawlen, wo der Crawler stehen geblieben ist.

5 Evaluation



Leider zeigte sich, dass fokussiertes Crawling mittels Context Graph nicht optimal für die Suche nach VRML Dateien im Internet ist. Bei längeren Läufen des Crawlers lag die Harvest-Rate nur bei rund einem Prozent, d.h. für jede gefundene Seite mit VRML Datei wurden ca. 100 nicht-relevante Seiten besucht.

Diese schlechte Erfolgsrate lässt sich leicht erklären, wenn man gefundene Seiten näher betrachtet, denn das Problem liegt darin, dass die VRML Dateien im Internet in stark unterschiedlichen Kontexten vorkommen:

- Modelle von Fahrzeugen, Flugzeugen, etc. auf Seiten von Hobby-Modellieren
- Modelle von Mathematischen Körpern auf Seiten zum Thema Mathematik
- Modelle von Organen und Ähnlichem auf medizinischen Seiten
- Modelle von Maschinen oder Gebäuden und Brücken auf diversen Seiten (z.B. Modelle des Mars Rovers auf einer NASA Seite)
- Modelle von diversen Logos völlig unterschiedlicher Webseiten

- usw.

Die VRML Dateien kommen also nur sehr selten in einem “VRML Kontext” vor. Es zeigte sich, dass in diesem Kontext, d.h. auf Seiten auf denen Wörter wie “VRML” selbst, “3d”, “modell”, etc. oft vorkommen, es meist nur um das VRML Format, oder um diverse VRML Viewer, Programme, die damit arbeiten, oder API Anleitungen geht. Die VRML Dateien dagegen finden sich viel mehr in dem Kontext des Objektes, das sie darstellen. Das bedeutet, dass Webseiten, die VRML Dateien enthalten, vorwiegend Text enthalten, der das dargestellte Objekt beschreibt, und es fast keine Begriffe gibt, die allen Seiten gemeinsam sind. Dies führt erstens dazu, dass der Context Graph aus sehr unterschiedlichen Seiten besteht, die nur wenige gemeinsame Begriffe haben (aber trotzdem sehr lückenhaft sein wird, da man nur einen Bruchteil der Kontexte, in denen VRML Dateien vorkommen, kennt und bei Auswahl der Seedpages berücksichtigen kann), so dass der daraus erstellte Klassifikator nicht optimal ist. Außerdem ist es leicht einsehbar, dass die meisten dieser Seiten, die VRML Dateien enthalten, viel mehr mit anderen Seiten des Kontextes verlinkt sind (z.B. mit diversen Seiten über den Mars, im Falle der Seite mit dem Modell des Mars Rovers), als mit Seiten zum Thema VRML. Dies führt dazu, dass der Crawler, wenn er eine Seite mit VRML Datei findet, sie und die direkt gelinkten Seiten oft als wenig relevant klassifizieren wird, und auch wenn er ihren Links folgt, von dort keine relevanten Seiten schnell erreichen kann.

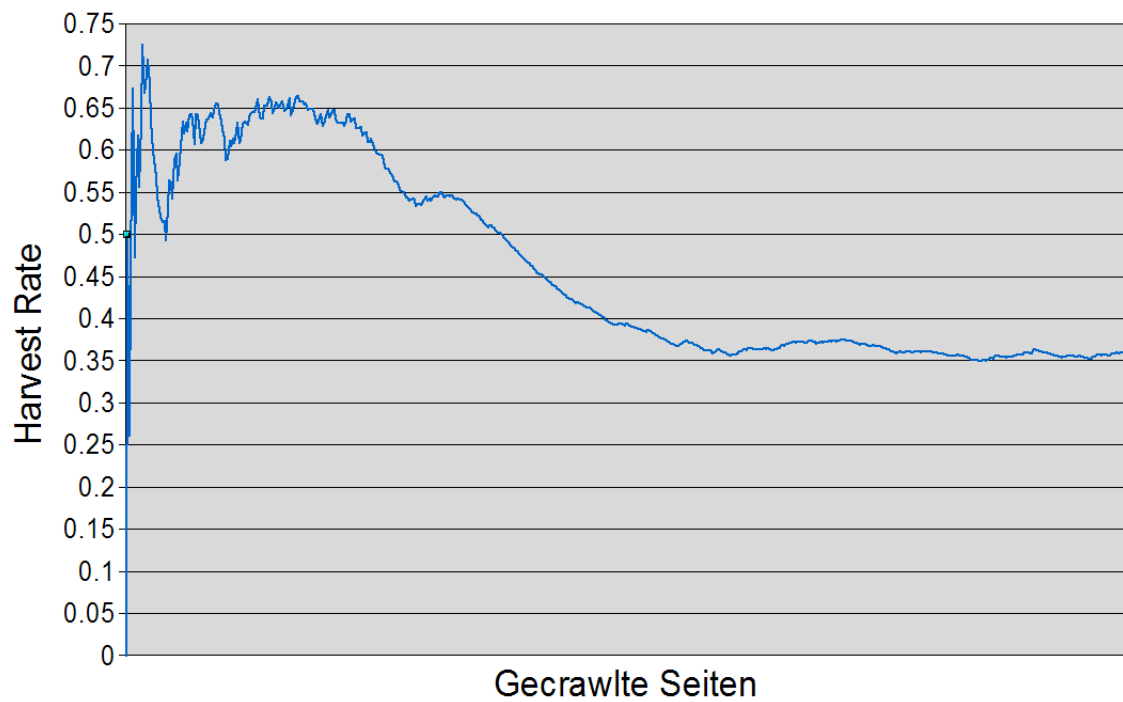
Es muss jedoch erwähnt werden, dass, auch wenn das Ergebnis nicht überzeugt, VRML Dateien immer noch viel schneller gefunden werden als mit einem rekursiven Crawler bzw. einer Suchmaschine.

Um die Vorzüge des Context Graph Crawlings besser zu zeigen wurden noch Experimente durchgeführt, bei denen nicht nach bestimmten Dateitypen gesucht wurde, sondern nach Webseiten mit Text zu einem bestimmten Thema.

In der Literatur zum Thema fokussiertem Crawling finden sich häufig Experimente, bei denen Seiten zum Thema Rad fahren (Biking, Cycling) gesucht werden (z.B. [Chak99]). Dies wurde auch mit dem cgcrawler versucht, und führte zu recht guten Ergebnissen, im Besten Fall mit einer Harvest-Rate um 50% oder höher. Das bedeutet, im Schnitt war jede zweite besuchte Seite relevant; Der Crawler hat sich dauerhaft nur auf Seiten zum Thema Rad fahren bewegt und ist Links auf Seiten, die nicht zum Thema passen, nicht gefolgt.

Bei den Experimenten mit dem Crawler hat sich auch gezeigt, wie wichtig die Auswahl der Seedpages ist. So hat z.B. ein Experiment zum Thema Rad fahren, bei dem mehrere Link-Listen zu Biking-Seiten als Seedpages verwendet wurden, anschließend zu einer Harvest-Rate unter 20% geführt. Wurde dagegen die Wikipedia Seite zu “Biking”, die extrem viele Begriffe zum Thema und aus dem entsprechenden Jargon enthält, als Seedpage verwendet, ergab sich das nachfolgend illustrierte Ergebnis für die Harvest-Rate für einen Crawl über einige Tausend Webseiten:

Webseiten zum Thema "Biking"



Diese unterschiedlichen Ergebnisse zeigen deutlich, dass es für effizientes fokussiertes Crawling von oberster Priorität ist, dass der Benutzer dem Crawler qualitativ hochwertige Seedpages zum Lernen bereitstellt, die möglichst viele thematisch wichtige und möglichst wenige nicht zum Thema passende Begriffe enthalten.

Literaturverzeichnis

- [Brin98] Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine (<http://www-db.stanford.edu/~backrub/google.html>). Computer Networks and ISDN Systems, 30(1-7):107-117.
- [Chak99] S. Chakrabarti, M. van der Berg, and B. Dom, "Focused crawling: a new approach to topic-specific web resource discovery," in Proceedings of 8th International World Wide Web Conference (WWW8), 1999.
- [Dil00] Diligenti, M., Coetzee, F., Lawrence, S., Giles, C. L., and Gori, M. (2000). Focused crawling using context graphs. In Proceedings of 26th International Conference on Very Large Databases (VLDB), pages 527-534, Cairo, Egypt.
- [Law00] Lawrence, S. and Giles, C. L. (2000). Accessibility of information on the web. Intelligence, 11(1), 32-39.
- [Wit05] Ian H. Witten and Eibe Frank (2005) "Data Mining: Practical machine learning tools and techniques", 2nd Edition, Morgan Kaufmann, San Francisco, 2005. (<http://www.cs.waikato.ac.nz/ml/weka/>)
- [Tidy] <http://sourceforge.net/projects/jtidy>
- [Yahoo] <http://developer.yahoo.com/search/>
- [Word] <http://wordnet.princeton.edu/oldversions>
- [JWNL] <http://jwordnet.sourceforge.net/>