Bozza Tesi

Roberta Benassi

Parole Chiave:
Intelligent Agent
Web Search
Ontology Matching
Lexical Chain
Relevance Scoring

Ringraziamenti

Desidero ringraziare la Prof. Sonia Bergamaschi e tutto il gruppo di Basi di Dati dell'Università di Modena e Reggio Emilia per i preziosi consigli e l'aiuto fornito durante la realizzazione di questa tesi.

Un ringraziamento speciale va alla mia famiglia che ha reso possibile tutto questo, e a tutte quelle persone che mi hanno sostenuto ed incoraggiato in questi anni, in particolare Matteo e i ragazzi dell'auletta di Fisica: Elisa, Marzia, Giulio, Emanuele, Daniele, Stefano, Michele, Francesco, Silvia, Patrizia, Morgan, Alberto, Carlo e Luca.

Indice

Introduzione **vii**

I L'integrazione intelligente delle informazioni 1

1 Il programma I^3 **2**

1.1 Architettura di riferimento 4

1.1.1 Servizi di Coordinamento 6

1.1.2 Servizi di Amministrazione 8

1.1.3 Servizi di Integrazione e Trasformazione Semantica . . 9

1.1.4 Servizi di wrapping 10

1.1.5 Servizi Ausiliari 10

1.2 Il mediatore 11

1.2.1 L'approccio semantico all'integrazione 14

1.2.2 Trattamento dei MetaDati e delle Ontologie 15

2 Il sistema MOMIS **19**

2.1 L'approccio adottato 19

2.2 L'architettura generale di MOMIS 21

2.3 WordNet, un componente fondamentale di MOMIS 24

2.4 Generazione del Common Thesaurus 27

2.5 Generazione delle classi globali 31

2.6 Il sistema MIKS 32

II La scoperta delle informazioni 35

3 Topologia del Web: Il *Web Graph* 36

4 Strumenti di ricerca generali 42
4.1 Web Directories 42
4.1.1 *Yahoo!* 44
4.1.2 *Open Directory Project* 45
4.1.3 Tools automatici o semiautomatici 47
4.2 Motori di ricerca veri e propri 51
4.2.1 *Google* 65
4.3 Meta Motori di ricerca 70
4.3.1 *MetaCrawler* 74
4.3.2 *SavvySearch* 75
4.3.3 *ProFusion* 77
4.4 Conclusioni 79

5 Strumenti di ricerca specializzati 80
5.1 Motori di ricerca specializzati 80
5.1.1 *CiteSeer* 80
5.2 Strumenti personali per la ricerca di informazioni nel Web . . 83

6 Un nuovo approccio alla scoperta di informazioni 86
6.1 Strategie di esplorazione del *Web graph* 87
6.2 Focused Crawlers 97
6.3 Intelligent Focused Crawlers 104

III Estrazione delle informazioni: un approccio linguistico 109

7 Catene Lessicali - Stato dell'arte 110
7.1 Generazione automatica di sommari 110
7.2 Coerenza e Coesione: definizioni 116
7.3 Algoritmi per l'estrazione di catene lessicali 119
7.3.1 Algoritmo di Morris e Hirst 119

7.3.2 Algoritmo di Okamura e Honda 123
7.3.3 Algoritmo di Hirst e St-Onge 124
7.3.4 Algoritmo di Barzilay e Elhadad 136
7.3.5 Algoritmo di Silber e McCoy 141
7.3.6 Algoritmo di Fuentes e Rodriguez 142
7.3.7 Conclusioni 143
7.4 Sviluppi delle catene lessicali 144
7.4.1 Costruzione di link ipertestuali 144

IV Progetto e realizzazione di un Agente Hunter 148

8 TUCUXI: un approccio linguistico alla ricerca di informazioni 149
8.1 Tecnologia ad agenti per un crawling efficiente 151
8.1.1 Una strategia di crawling basata sulla semantica . . . 152
8.1.2 Etica per un crawler 157
8.2 Estrazione della semantica 162
8.2.1 Perchè le catene lessicali? 162
8.2.2 Estrazione di mappe concettuali 164
8.3 Relevance Scoring 166
8.4 Esperimenti 170
8.4.1 Quattro strategie a confronto 170
8.4.2 Confronto con Google 177
8.4.3 Gli Errori di TUCUXI 185
8.5 Il Manuale Utente 185

9 Conclusioni e Sviluppi Futuri 198

A Appendice 200
A.1 KAON 200
A.2 GATE 201
A.3 KeyConcept 202

Elenco delle figure

1.1 Diagramma dei servizi di I^3. 5
1.2 Servizi I^3 presenti nel mediatore. 13

2.1 L'architettura di MOMIS. 21
2.2 Il sistema MIKS. 33

3.1 Il bowtie secondo Broder. 38
3.2 Average change interval over all web page. Fonte: Cho e Garcia-Molina: The Evolution of the Web and Implications for an Incremental Crawler, 26th VLDB Conference, 2000. . . 40
3.3 Average change interval for each domain. Fonte: Cho e Garcia-Molina: The Evolution of the Web and Implications for an Incremental Crawler, 26th VLDB Conference, 2000. 40
3.4 Età media delle pagine indicizzate da Google, così come indicato dai Web Servers. Fonte: www.Google.com 41

4.1 Architettura del tool *"Categorisation by Context"* 49
4.2 Architettura di un motore di ricerca vero e proprio 52
4.3 Architettura di Mercator 54
4.4 Copertura di Google 66
4.5 Architettura di Google, così come illustrata in [BRI 98]. 67
4.6 Il numero di pagine indicizzate dai più noti motori di ricerca . 70
4.7 Architettura di un semplice metasearch engine. 71
4.8 Architettura di riferimento di un semplice meta motore di ricerca 73
4.9 Architettura di MetaCrawler 74
4.10 Flusso di controllo di MetaCrawler 76

4.11 Architettura di ProFusion 78

5.1 CiteSeer Architecture 82

6.1 Esplorazione del Web realizzata da un crawler standard (in azzurro le pagine visitate, in blu i targets, in rosso i link seguiti). 98
6.2 Esplorazione del Web realizzata da un focused crawler (in azzurro le pagine visitate, in bianco quelle non visitate, in blu i targets, in rosso i link seguiti, in nero i link scartati 99
6.3 Architettura di un Focused Crawler. 100
6.4 Architettura del Focused Crawler di Chakrabarti, Van Der Berg e Dom [CHA 99]. 102

7.1 Organizzazione del Roget's Thesaurus. 121
7.2 Struttura di WordNet. 126
7.3 Strong Relation: synset in comune 127
7.4 Strong Relation: un link orizzontale 128
7.5 Strong Relation: compound word 129
7.6 Medium-Strong Relation: path ammissibili 129
7.7 Medium-Strong Relation: path non ammissibili 130
7.8 Esempio di relazione fra due parole 130
7.9 Esempio di costruzione dinamica di una catena lessicale. . . . 132
7.10 Creazione di una nuova catena. 133
7.11 Inserimento di una relazione extra-strong 134
7.12 Inserimento di una relazione strong 135
7.13 Risultato finale. 136
7.14 Split di una catena 137
7.15 Inserimento di un nuovo termine in relazione con quelli esistenti 138
7.16 Inserimento di un termine con più significati 139
7.17 Strong Chain (a) 139
7.18 Strong Chain (b) 140

8.1 Due ontologie a confronto 167
8.2 Esplorazioni page mode da www.yahoo.com 171
8.3 Esplorazioni page mode da www.berkeley.com 172

8.4 Esplorazioni site mode da “www.yahoo.com” 174
8.5 Esplorazioni site mode da www.berkeledy.edu 175
8.6 I parametri di learning a confronto 176
8.7 Un errore di TUCUXI?? 186
8.8 Setting Panels 187
8.9 Finestra per la Google Search Interface 189
8.10 Finestra per l’interazione con Google tramite protocollo SOAP 190
8.11 Scelta del Common Thesaurus 191
8.12 Scelta delle estensioni a WordNet. 191
8.13 Il pannello Start. 192
8.14 Il pannello Graphics - pagina di presentazione. 193
8.15 Il pannello Graphics - una pagina già visitata. 195
8.16 Il pannello View XML - il file XML generato. 196
8.17 Il pannello Results - Punteggi globali. 197

A.1 Ricerca di Persone all’interno di *www.dbgroup.unimo.it / Momis / partecipants.html* 203
A.2 Processo di indicizzazione di KeyConcept 204
A.3 Information Retrieval in KeyConcept 205

Introduzione

Il *World Wide Web* è stato definito come una rete di sorgenti di informazioni [BER 94]. In origine il Web si proponeva di facilitare lo scambio di documenti e di idee all'interno della comunità scientifica. Ora, anche altre e ben più numerose comunità accedono liberamente alla rete e pubblicano i propri documenti *on-line*. L'abbondanza delle sorgenti disponibili (siano esse pagine Web statiche o dinamiche, immagini, filmati, DBMS relazionali o ad oggetti, file xml) ha determinato l'esigenza di individuare ed integrare le risorse interessanti rispetto ai desideri di un utente.

Il problema dell'integrazione di sorgenti eterogenee è affrontato e risolto da **MOMIS** (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources), un sistema semiautomatico sviluppato dall'Università di Modena e Reggio Emilia in collaborazione con l'Università di Milano. MOMIS fa parte del progetto nazionale di ricerca denominato INTERDATA.

La ricerca di informazioni rimane comunque un problema ancora aperto. Districarsi fra le innumerevoli fonti può rivelarsi, infatti, un processo tedioso e complicato. Per questo motivo i motori di ricerca hanno fornito un servizio sicuramente utile ai fini dell'individuazione di sorgenti potenzialmente interessanti.

Nemmeno i motori di ricerca sono immuni al problema dell'*Information Overload*, anzi, spesso l'eccesso di dati rende di poco valore se non inutilizzabili i risultati ottenuti tramite questa tipologia di strumenti. Inoltre, i motori di ricerca sono carenti per quanto riguarda la modalità con cui un utente esprime i propri interessi. Infatti, la maggior parte dei motori di ricerca affida il compito di recuperare i documenti sulla base delle *keywords* in essi contenute. Purtroppo, la ricerca per keywords soffre dei problemi legati alla sinonimia e alla polisemia. Un operatore umano è in grado di inviduare

i sinonimi di un termine, così come è in grado di stabilire il significato corretto di una parola, ma l'operatore macchina non è in grado di fare queste distinzioni. L'HTML stesso non è nato come linguaggio di rappresentazione della conoscenza, ma piuttosto come linguaggio per specificare il layout di un documento. Nasce in questo ambito l'idea del **Semantic Web** [HEN 01], in cui le risorse possono essere annotate con metadati *machine-readable*.

SEWASIE (**SE**mantic **W**ebs and **A**gent**S** in **I**ntegrated **E**conomies) è un progetto europeo che ha come obiettivo quello di ideare ed implementare un motore di ricerca avanzato che soddisfi le necessità delle SMEs. L'Università di Modena e Reggio Emilia è coordinatrice del progetto. Gli altri partner sono:

- CNA Servizi Modena s.c.a.r.l,
- Università di Roma La Sapienza,
- Rheinisch Westfaelische Technische Hochschule Aachen (Germania),
- The Victoria University of Manchester (UK),
- IBM Italia SPA,
- Thinking Networks AG (Germania),
- Fraunhofer-Gesellschaft zur Frderung der angewandten Forschung eingetragener Verein (Germania).

Lo scopo di questa tesi all'interno del progetto SEWASIE è progettare e realizzare un agente hunter in grado di individuare risorse di informazione rilevanti rispetto al Common Thesaurus di MOMIS. Pertanto l'agente deve implementare una strategia intelligente di esplorazione del Web, in modo da recuperare il maggior numero di pagine rilevanti visitando il numero minore possibile di documenti non interessanti.

L'agente hunter realizzato è stato denominato TUCUXI, acronimo di In**T**elligent H**U**nter Agent for **C**oncept **U**nderstanding and Le**X**ical Cha**I**ning.

TUCUXI (pronuncia "tookooshee") è in grado di utilizzare una tecnica di *Natural Language Processing* per estrarre dalle pagine HTML l'insieme dei

concetti in esse rappresentati, nonchè assegnare ai vari documenti un punteggio di rilevanza. Le sorgenti con un punteggio di rilevanza sufficientemente elevato sono candidate ad essere intergrate in MOMIS.

La tesi è organizzata nel seguente modo:

Parte I, in cui viene illustrato come MOMIS realizza l'integrazione di sorgenti di dati eterogenee. In particolare, l'attenzione sarà concentrata sulla costruzione del Common Thesaurus e su uno dei componenti fondamentali del sistema: il database lessicale WordNet.

Parte II, in cui vengono presentati gli strumenti attualmente a disposizione per effettuare ricerche di informazioni in rete. Sono state individuate due categorie fondamentali di motori di ricerca, quelli general-purpose e quelli special-purpose. Per quanto riguarda la prima categoria sono stati descritti dal punto di vista architetturale le Web Directories, i motori di ricerca veri e propri e i meta motori di ricerca. Per la seconda categoria si è evidenziato come il termine special-purpose indichi in realtà due diversi tipi di strumenti: i motori specializzati su un argomento o gli agenti *Personal Assistants*. L'analisi realizzata è volta ad evidenziare i limiti dei suddetti strumenti e a presentare un paradigma per la ricerca di informazioni in rete recentemente descritto in letteratura: il *Focused Crawling*.

Parte III, in cui si illustra lo stato dell'arte nell'ambito dell'*Information Extraction*, con particolare riferimento alla tecnica di *Natural Language Processing* detta delle "catene lessicali"(*Lexical Chains*). La tecnica delle catene lessicali è un'interessante metodologia di *text clustering* basata sulla proprietà fondamentale di un testo in quanto tale: la coesione lessicale.

Parte IV, in cui viene descritto come l'agente TUCUXI è stato progettato ed implementato. Vengono anche presentati importanti risultati ottenuti tramite una serie di valutazioni empiriche dell'attività di crawling e viene dimostrato come TUCUXI abbia, rispetto a Google, notevoli capacità di discriminare fra pagine rilevanti e pagine non rilevanti.

Quest'ultimo aspetto, unito alla validità delle catene lessicali anche in ambito multilingua, consente di pensare ad un futuro sviluppo in termini di sistema multiagente (MAS).

La piattaforma ad agenti utilizzata in questa tesi è JADE 2.61 [1], ampiamente studiata nella tesi di Enrico Natalini [NAT 02]. TUCUXI si avvale dello studio di fattibilità svolto nella suddetta tesi.

[1] http://jade.cselt.it

Parte I

L'integrazione intelligente delle informazioni

Capitolo 1

Il programma I^3

La crescita delle sorgenti di dati sia all'interno dell'azienda che sulla rete ha reso possibile l'accesso ad una quantità molto ampia di informazioni. La probabilità di reperire un dato di interesse è di conseguenza aumentata vertiginosamente, allo stesso tempo si registra una diminuzione altrettanto importante della probabilità di venirne in possesso in tempi e soprattutto nei modi desiderati. La causa principale di questo fenomeno è una grande eterogeneità dei dati disponibili, sia per quanto riguarda la natura (testi, immagini, audio, video, etc.), sia il modo in cui vengono descritti.

Gli standard esistenti (TCP/IP, ODBC, OLE, CORBA, SQL, etc.) risolvono parzialmente i problemi relativi alle diversità hardware e software dei protocolli di rete e di comunicazone tra i moduli; rimangono però irrisolti quelli relativi alla modellazione delle informazioni. I modelli e gli schemi dei dati sono infatti differenti e questo crea una eterogeneità semantica (o logica) non risolvibile da questi standard.

Un'importante area, sia di ricerca che di applicazione, riguarda l'integrazione di DataBase eterogenei e dei *datawarehouse* (magazzino dei dati). Questi lavori studiano la possibilità di materializzare presso l'utente finale delle viste, ovvero porzioni di sorgenti, replicando fisicamente i dati e affidandosi a complicati algoritmi di mantenimento ai fini di garantirne la consistenza.

In letteratura il termine Integrazione delle Informazioni (I^2) indica il processo adottato da quei sistemi che, al contrario di quelli sopra citati, combinano informazioni provenienti da diverse sorgenti (o parti selezionate

di esse) senza dover ricorrere alla duplicazione fisica [WIE 96].

L'integrazione delle informazioni deve essere distinta da quella dei dati e dei DataBase. La scelta delle sorgenti e dei dati richiede *conoscenza* ed *intelligenza* e capacità di *fusione* e *sintesi*. Quando l'Integrazione delle Informazioni fa uso di tecniche di Intelligenza Artificiale si parla allora di Integrazione Intelligente delle Informazioni (*Intelligent Integration of Information* - I^3). Al contrario di quanto accade con l'uso di tecniche tradizionali (che si limitano ad una semplice aggregazione), questa forma di integrazione si prefigge lo scopo di accrescere il valore delle informazioni gestite (ottenendone anche di nuove dai dati utilizzati).

Il programma I^3 è un'ambiziosa ricerca finalizzata ad indicare un'architettura di riferimento che realizzi l'integrazione di sorgenti eterogenee in maniera automatica. Il programma I^3 è sviluppato dall'ARPA (Advanced Research Projects Agency)[HUL 95], l'agenzia che fa capo al Dipartimento di Difesa statunitense. In quel contesto si è potuto osservare che l'integrazione aumenta il valore dell'informazione ma richiede una forte adattabilità realizzativa: si devono infatti riuscire a gestire i casi di aggiornamento e sostituzione delle sorgenti, dei loro ambienti e/o piattaforme, della loro ontologia e della semantica. Le tecniche sviluppate dall'*Intelligenza Artificiale*, potendo efficacemente dedurre informazioni utili dagli schemi delle sorgenti, diventano pertanto uno strumento prezioso per la costruzione automatica di soluzioni integrate flessibili e riusabili.

Progettare la costruzione di supersistemi ad hoc che interessino una grande quantità di sorgenti, non correlate semanticamente, è faticoso ed il risultato è un sistema scarsamente manutenibile o adattabile, strettamente finalizzato alla risoluzione dei problemi per cui è stato implementato. Secondo il programma I^3 una soluzione a questi problemi può essere trovata mediante l'introduzione di architetture modulari sviluppabili secondo i principi proposti da uno standard.

Tale standard deve porre le basi dei servizi che devono essere realizzati attraverso l'integrazione e deve cercare di abbassare i costi di sviluppo e di mantenimento. Pertanto costruire nuovi sistemi diventa realizzabile con minor difficoltà e minor tempo (e quindi costi) se si riesce a supportare lo sviluppo delle applicazioni riutilizzando la tecnologia già sviluppata. Per la

riusabilità è fondamentale l'esistenza di interfacce ed architetture standard. Il paradigma suggerito per la suddivisione dei servizi e delle risorse nei diversi moduli, si articola su due dimensioni:

- *orizzontalmente* in tre livelli: il livello utente, il livello intermedio che fa uso di tecniche di IA e il livello delle sorgenti di dati;
- *verticalmente*: diversi domini in cui raggruppare le sorgenti.

I domini dei vari livelli non sono strettamente connessi, ma si scambiano dati ed informazioni la cui combinazione avviene a livello dell'utilizzatore, riducendo la complessità totale del sistema e permettendo lo sviluppo di applicazioni con finalità diverse.

I^3 si concentra sul livello intermedio della partizione, quello che media tra gli utenti e le sorgenti. In questo livello sono presenti vari moduli, tra i quali si possono evidenziare:

- **Facilitator e Mediator** (le differenze tra i due sono sottili ed ancora ambigue in letteratura): ricercano le fonti interessanti e combinano i dati da esse ricevuti;
- **Query Processor:** riformula le query aumentando le probabilità di successo;
- **Data Miner:** analizza i dati per estrarre informazioni intensionali implicite.

1.1 Architettura di riferimento

L'obiettivo del programma I^3 è quello di ridurre il tempo necessario alla realizzazione di un integratore di informazioni, fornendo una raccolta e una formalizzazione delle soluzioni prevalenti nel campo della ricerca.

In particolare due sono le ipotesi che rappresentano la base del progetto I^3 . La prima è connessa con la difficoltà che si ha nel ricercare delle informazioni all'interno della molteplicità delle sorgenti di informazione che in questo momento è possibile individuare. Il secondo aspetto è legato al fatto che le fonti di informazione e i sistemi informativi, pur essendo spesso

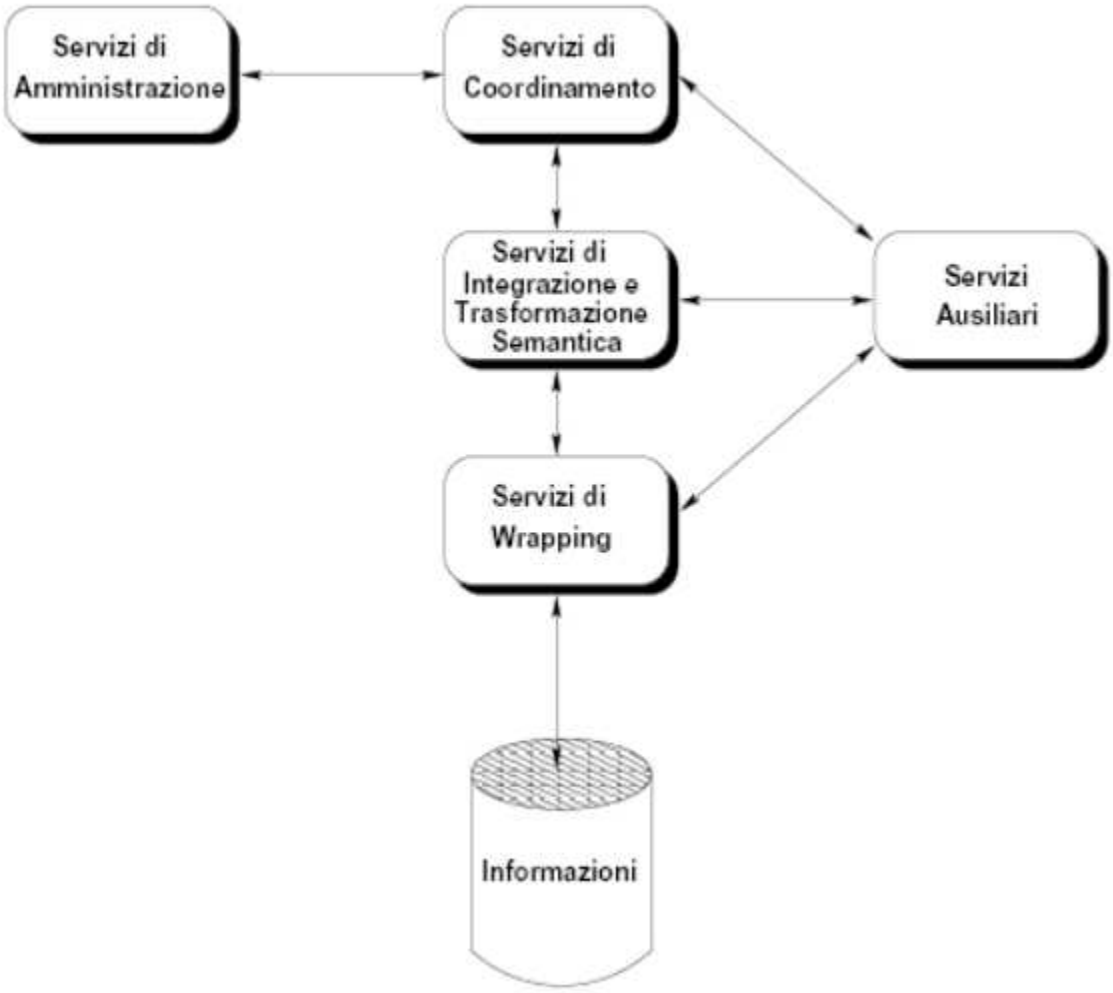

Figura 1.1: Diagramma dei servizi di I^3.

semanticamente correlati tra di loro, non lo sono in una forma semplice nè preordinata. Di conseguenza il processo di integrazione delle informazioni può risultare molto complesso.

Pertanto è necessario proporre una architettura di riferimento che rappresenti alcuni dei servizi che un integratore di informazioni deve contenere e le possibili interconnessioni fra di essi. Tale descrizione non vuole imporre nè delle soluzioni implementative, nè è da ritenersi esaustiva delle funzionalità che un sistema di integrazione deve includere.

In Figura 1.1 viene riportata l'architettura di un sistema I^3. L'architettura di riferimento dà grande rilevanza ai Servizi di Coordinamento. Questi servizi giocano infatti due ruoli: in primo luogo, possono localizzare altri servizi I^3 e fonti di informazioni che possono essere utilizzate per costruire il sistema stesso; secondariamente, sono responsabili di individuare ed invocare a run-time gli altri servizi necessari a dare risposta ad una specifica richiesta di dati.

Sono comunque in totale cinque le famiglie di servizi che sono state rappresentate in questa architettura. I servizi individuati, così come rappresentati nell'architettura proposta, possono essere soggetti a due differenti chiavi di lettura: la lettura seguendo l'asse orizzontale e quella seguendo l'asse verticale mettono in evidenza diversi aspetti e diversi compiti del sistema. Se si percorre l'asse verticale, si può intuire come avviene lo scambio di informazioni nel sistema: in particolare, i servizi di wrapping provvedono ad estrarre le informazioni dalle singole sorgenti. Tali informazioni vengono poi impacchettate ed integrate dai Servizi di Integrazione e Trasformazione Semantica, per poi essere passate ai servizi di Coordinamento che ne avevano fatto richiesta.

L'asse orizzontale mette invece in risalto il rapporto tra i servizi di Coordinamento e quelli di Amministrazione, ai quali spetta infatti il compito di mantenere informazioni sulle capacità delle varie sorgenti (che tipo di dati possono fornire ed in quale modo devono essere interrogate). Funzionalità di supporto, che verranno descritte successivamente, sono invece fornite dai Servizi Ausiliari, responsabili dei servizi di arricchimento semantico delle sorgenti. Vengono ora analizzate in dettaglio le funzionalità specifiche di ogni servizio e le problematiche che devono essere affrontate.

1.1.1 Servizi di Coordinamento

I servizi di Coordinamento sono quei servizi di alto livello che permettono l'individuazione delle sorgenti di dati interessanti, ovvero che probabilmente possono dare risposta ad una determinata richiesta dell'utente. A seconda delle possibilità dell'integratore che si vuole realizzare, possono essere rappresentati da meccanismi che includono dalla selezione dinamica delle sorgenti (o brokering, per Integratori Intelligenti) fino al semplice Matchmaking, in

cui il mappaggio tra informazioni integrate e locali è realizzato manualmente ed una volta per tutte. Vediamo alcuni esempi.

- **Facilitation e Brokering Services:** l'utente manda una richiesta al sistema e questo usa un deposito di metadati per ritrovare il modulo che può trattare la richiesta direttamente. I moduli interessati da questa richiesta possono essere o uno solo alla volta (nel qual caso si parla di Brokering) oppure più di uno (e in questo secondo caso si tratta di facilitatori e mediatori, attraverso i quali a partire da una richiesta ne viene generata pić di una da inviare singolarmente a differenti moduli che gestiscono sorgenti distinte, e reintegrando poi le risposte in modo da presentarle all'utente come se fossero state ricavate da un'unica fonte).

  In questo ultimo caso, in cui una query può essere decomposta in un insieme di sottoquery, si farà uso di servizi di Query Decomposition e di tecniche di Inferenza (mutuate dall'Intelligenza Artificiale) per una determinazione dinamica delle sorgenti da interrogare, a seconda delle condizioni poste nell'interrogazione. I vantaggi che questi servizi di Coordinamento portano stanno nel fatto che non è richiesta all'utente del sistema una conoscenza del contenuto delle diverse sorgenti, ma viene fornita un'unica rappresentazione delle sorgenti e in questo modo un sistema omogeneo che gestisce direttamente la sua richiesta. Pertanto l'utente non è obbligato a conoscere i domini con i quali i vari moduli I^3 si trovano ad interagire. Risulta quindi evidente la considerevole diminuzione di complessità di interazione col sistema che deriva da una architettura di questo tipo.

- **Matchmaking:** il sistema viene configurato manualmente da un operatore in fase di inizializzazione. Successivamente a quella fase, tutte le richieste verranno trattate allo stesso modo. Sono definiti gli anelli di collegamento tra tutti i moduli del sistema, e nessuna ottimizzazione è fatta a tempo di esecuzione.

1.1.2 Servizi di Amministrazione

Sono servizi usati dai Servizi di Coordinamento per localizzare le sorgenti utili, per determinare le loro capacità, e per creare ed interpretare TEMPLATE. I Template sono strutture dati che descrivono i servizi, le fonti ed i moduli da utilizzare per portare a termine un determinato task. Sono quindi utilizzati dai sistemi meno intelligenti, e consentono all'operatore di predefinire le azioni da eseguire a seguito di una determinata richiesta, limitando al minimo le possibilità di decisione del sistema.

In alternativa all'uso dei Template, possono essere utilizzate le **Yellow Pages**: si tratta di servizi di directory che memorizzano le informazioni sul contenuto delle varie sorgenti e sul loro stato (attiva, inattiva, occupata). Consultando queste Yellow Pages, il mediatore è in grado di spedire alla giusta sorgente la richiesta di informazioni, ed eventualmente di rimpiazzare questa sorgente con una equivalente nel caso non fosse disponibile. Fanno parte di questa categoria di servizi il Browsing: si permette all'utente di navigare attraverso le descrizioni degli schemi delle sorgenti, recuperando informazioni su queste. Il servizio si basa sulla premessa che queste descrizioni siano fornite esplicitamente tramite un linguaggio dichiarativo leggibile e comprensibile all'utente. Altri moduli interessanti sono:

- **Resource Discovery:** sono in grado di riconoscere e ritrovare gli strumenti che gestiscono una determinata richiesta a run time. Questi servizi sono in grado di acquisire ed aggiornare dinamicamente informazioni sui tool (nomi e servizi forniti) e sul loro stato. Inoltre acquisiscono e mantengono le informazioni sui domini informativi.

- **Iterative Query Formulation:** sono di supporto all'utente in fase di formulazione di query particolarmente complesse sullo schema integrato; specialmente qualora la query formulata non abbia prodotto informazioni interessanti, suggeriscono quali condizioni rilasciare, come rendere più specifica o pić generale la query.

- **Primitive di costruzione delle configurazioni:** servono a scegliere due cose: da una parte i servizi di tool e le sorgenti appropriate per

svolgere un opportuno task, dall'altra il giusto modo di collegarli tra loro per creare una configurazione.

1.1.3 Servizi di Integrazione e Trasformazione Semantica

Questi servizi supportano le manipolazioni semantiche necessarie per l'integrazione e la trasformazione delle informazioni. Il tipico input per questi servizi è rappresentato da una o più sorgenti di dati, e l'output è la vista integrata o trasformata di queste informazioni. Tra questi servizi si distinguono quelli relativi alla trasformazione degli schemi (ovvero di tutto ciò che va sotto il nome di metadati) e quelli relativi alla trasformazione dei dati. Sono spesso indicati come servizi di Mediazione, essendo tipici dei moduli mediatori.

- Servizi di **integrazione degli schemi**. Supportano la trasformazione e l'integrazione degli schemi e delle conoscenze derivanti da fonti di dati eterogenee. Ne fanno parte i servizi di trasformazione dei vocaboli e dell'ontologia, usati per arrivare alla definizione di un'ontologia unica che combini gli aspetti comuni alle singole ontologie usate nelle diverse fonti. Queste operazioni sono molto utili quando devono essere scambiate informazioni derivanti da ambienti differenti, dove molto probabilmente non si condivide un'unica ontologia. Fondamentale, per la fase di creazione dell'insieme dei vocaboli condivisi, è la fase di individuazione dei concetti presenti in diverse fonti, e la riconciliazione delle diversità presenti sia nelle strutture, sia nei significati dei dati.

- Servizi di **integrazione delle informazioni**. Provvedono alla traduzione dei termini da un contesto all'altro, ovvero dall'ontologia di partenza a quella di destinazione. Possono inoltre occuparsi di uniformare la "granularità" dei dati (come possono essere le discrepanze nelle unità di misura, o le discrepanze temporali).

- Servizi di **supporto al processo di integrazione**. Sono utilizzati nel momento in cui una query è scomposta in molte subquery, da inviare a

fonti differenti, e nel momento in cui i risultati provenienti dalle singole subquery devono essere integrati. Comprendono inoltre tecniche di caching, per supportare la materializzazione delle viste (problematica molto comune nei sistemi che vanno sotto il nome di datawarehouse).

1.1.4 Servizi di wrapping

Sono utilizzati per fare sì che le fonti di informazioni aderiscano ad uno standard, che può essere interno o proveniente dal mondo esterno con cui il sistema vuole interfacciarsi. Si comportano come traduttori dai sistemi locali ai servizi di alto livello dell'integratore e viceversa quando si interroga la sorgente di dati. In particolare, sono due gli obiettivi che si prefiggono:

1. permettere ai servizi di coordinamento e di mediazione di manipolare in modo uniforme il numero maggiore di sorgenti locali, anche se queste non sono state esplicitamente pensate come facenti parte del sistema di integrazione;

2. essere il più riusabili possibile. Per fare ciò, dovrebbero fornire interfacce che seguano gli standard più diffusi (e tra questi, si potrebbe citare il linguaggio SQL come linguaggio di interrogazione di basi di dati, e CORBA come protocollo di scambio di oggetti). Questo permetterebbe alle sorgenti estratte da questi wrapper universali di essere accedute dal numero maggiore possibile di moduli mediatori.

In pratica, compito di un wrapper è modificare l'interfaccia, i dati ed il comportamento di una sorgente, per facilitarne la comunicazione con il mondo esterno. Il vero obiettivo è quindi standardizzare il processo di wrapping delle sorgenti, permettendo la creazione di una libreria di fonti accessibili; inoltre, il processo stesso di realizzazione di un wrapper dovrebbe essere standardizzato, in modo da poter essere riutilizzato da altre fonti.

1.1.5 Servizi Ausiliari

Aumentano le funzionalità degli altri servizi descritti precedentemente: sono prevalentemente utilizzati dai moduli che agiscono direttamente sulle informazioni. Vanno dai semplici servizi di monitoraggio del sistema (un utente

vuole avere un segnale nel momento in cui avviene un determinato evento in un database, e conseguenti azioni devono essere attuate), ai servizi di propagazione degli aggiornamenti e di ottimizzazione.

1.2 Il mediatore

Secondo la definizione proposta da Wiederhold *"un mediatore è un modulo software che sfrutta la conoscenza su un certo insieme di dati per creare informazioni per una applicazione di livello superiore". . . . Dovrebbe essere piccolo e semplice, così da poter essere amministrato da uno, o al più pochi, esperti.* Compiti di un mediatore sono allora:

- assicurare un servizio stabile, anche nel caso di cambiamento delle risorse;
- amministrare e risolvere le eterogeneità delle diverse fonti;
- integrare le informazioni ricavate da pić risorse;
- presentare all'utente le informazioni attraverso un modello scelto dall'utente stesso.

Il progetto MOMIS ha tra i suoi obiettivi la realizzazione di un Mediatore. Durante la fase di progettazione e realizzazione del sistema, è stato necessario introdurre una ipotesi che restringesse il campo applicativo del sistema stesso (e di conseguenza che restringesse il campo dei problemi a cui dare risposta). Tale ipotesi consiste nel fatto che il mediatore, per il momento, si occupi esclusivamente di sorgenti di dati di testo strutturati e semistrutturati, quali possono essere basi di dati relazionali, ad oggetti, file di testo, pagine HTML ed XML. L'approccio architetturale scelto è quello classico, che si sviluppa principalmente su 3 livelli:

- utente: attraverso un'interfaccia grafica l'utente pone delle query su uno schema globale e riceve un'unica risposta, come se stesse interrogando un'unica sorgente di informazioni;

- mediatore: il mediatore gestisce l'interrogazione dell'utente, combinando, integrando ed eventualmente arricchendo i dati ricevuti dai wrapper, ma usando un modello (e quindi un linguaggio di interrogazione) comune a tutte le fonti;

- wrapper: ogni wrapper gestisce una singola sorgente, ed ha una duplice funzione: da un lato converte le richieste del mediatore in una forma comprensibile dalla sorgente, dall'altro traduce informazioni estratte dalla sorgente nel modello usato dal mediatore.

Facendo riferimento ai servizi descritti nelle sezioni precedenti, l'architettura del nuovo mediatore è riportata in Figura 1.2. Il passo successivo, ossia l'integrazione di dati all'interno del global schema, viene effettuata mediante l'impiego di appositi wrappers. Un wrapper svolge la funzione di interfaccia tra il nucleo del mediatore. Tale wrapper restituisce la descrizione ODL_I [3] della sorgente alla quale è connesso e permette l'esecuzione a livello locale delle queries generate dal query manager. L'impostazione architetturale presentata dimostra l'intenzione di progettare un mediatore che si distacchi dall'approccio strutturale, cioè sintattico, tuttora dominante tra i sistemi presenti sul mercato. L'approccio strutturale, adottato da sistemi quali TSIMMIS [CHA 94], è caratterizzato dal fatto di usare un self-describing model per rappresentare gli oggetti da integrare, limitando l'uso delle informazioni semantiche alle regole predefinite dall'operatore. In pratica, il sistema non conosce a priori la semantica di un oggetto che va a recuperare da una sorgente (e dunque di questa non possiede alcuno schema descrittivo) bensì è l'oggetto stesso che, attraverso delle etichette, si autodescrive, specificando tutte le volte, per ogni suo singolo campo, il significato che ad esso è associato. Questo approccio porta quindi ad un insieme di vantaggi, tra i quali possiamo identificare:

- la possibilità di integrare in modo completamente trasparente al mediatore, basi di dati fortemente eterogenee e magari mutevoli nel tempo: il mediatore non si basa infatti su una descrizione predefinita degli schemi delle sorgenti, bensì sulla descrizione che ogni singolo oggetto fa di sè. Oggetti simili provenienti dalla stessa sorgente possono quin-

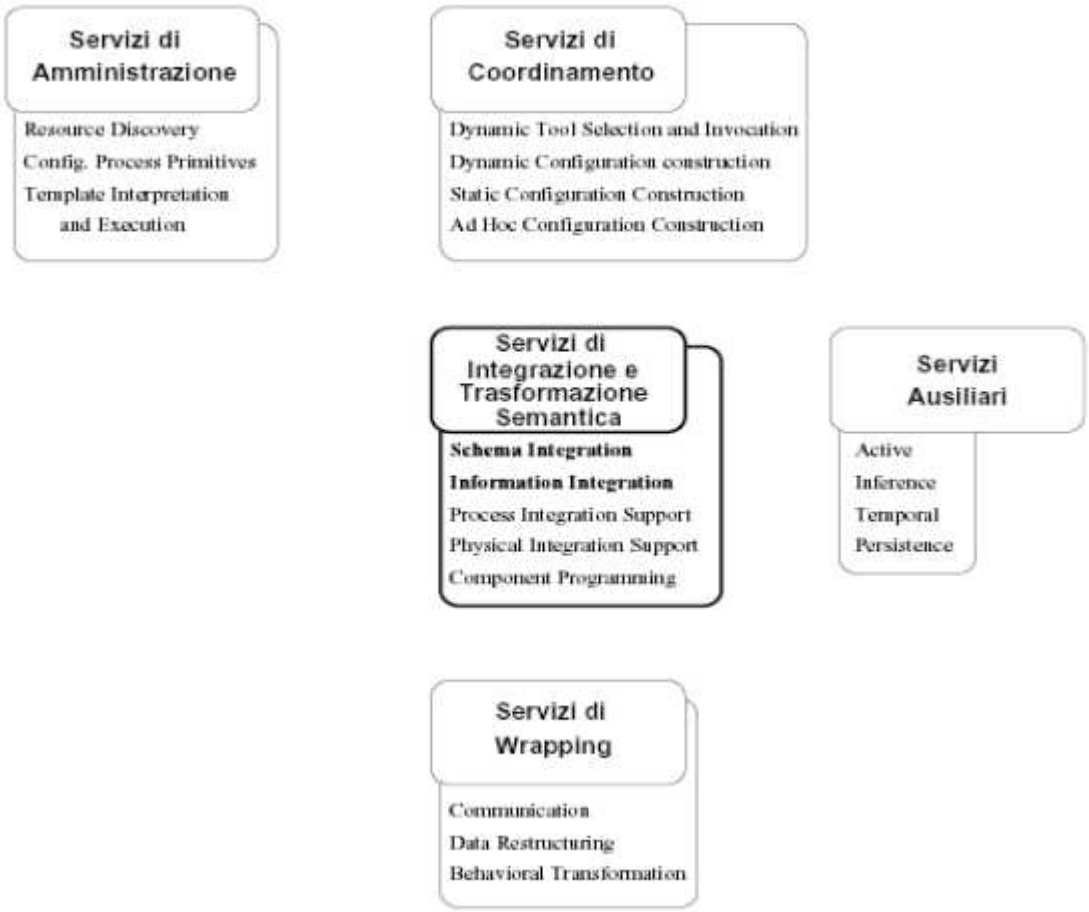

Figura 1.2: Servizi I^3 presenti nel mediatore.

di avere strutture differenti, cosa invece non ammessa in un ambiente tradizionale object-oriented;

- per trattare in modo omogeneo dati che descrivono lo stesso concetto, o che hanno concetti in comune, ci si basa sulla definizione manuale di rule, che permettono di identificare i termini (e dunque i concetti) che devono essere condivisi da più oggetti.

Altri progetti, e tra questi quello proposto, seguono invece un approccio definito semantico, il quale è caratterizzato dai seguenti punti:

- il mediatore deve conoscere, per ogni sorgente, lo schema concettuale (metadati);
- informazioni semantiche sono codificate in questi schemi;
- deve essere disponibile un modello comune per descrivere le informazioni da condividere (e dunque per descrivere anche i metadati);
- deve essere possibile una integrazione (parziale o totale) delle sorgenti di dati.

In questo modo, sfruttando opportunamente le informazioni semantiche che necessariamente ogni schema sottintende, il mediatore può individuare concetti comuni a più sorgenti e relazioni che li legano.

1.2.1 L'approccio semantico all'integrazione

Il cuore dell'architettura I^3 è rappresentato dai Servizi di Trasformazione ed Integrazione Semantica. Essi si occupano dell'integrazione vera e propria delle informazioni cercando di risolvere le differenze tra gli schemi, i modelli ed i tipi di dati, mentre le differenze dalla prospettiva delle piattaforme (Hw, DBMS, API) sono risolte dai protocolli di rete e dagli standard: SQL, OLE (Object Linking Embedded), per legare oggetti in applicazioni contenitori, ODBC (Object Database Connectivity): per la connessione fra basi di dati eterogenee, ODMG, che fornisce un modello dei dati ed i linguaggi di descrizione, manipolazione ed interrogazione per basi di dati orientate agli oggetti,

CORBA, per lo scambio degli oggetti fra sorgenti eterogenee. Un secondo problema cui si dedica la Semantica riguarda il potere espressivo delle interrogazioni: molti sistemi, specialmenti quelli accessibili via Web, supportano solo un ristretto numero di query predefinite, aggiungendo conoscenza semantica è possibile ricondurre le query più complesse in quelle predefinite o in un loro sovrainsieme.

L'integrazione degli schemi porta alla definizione di viste omogenee: queste rappresentano un superschema, virtuale nella maggior parte dei casi, degli schemi delle sorgenti integrate. Le viste possono essere convenzionali se rappresentano in maniera omogenea gli schemi delle sorgenti o parti di esse, altrimenti sono dette integrate: in questo caso gli schemi sono fusi e combinati e non è più facilmente riconoscibile il contributo portato dalle diverse sorgenti. La fusione degli schemi può essere completata dalla materializzazione dei dati, come ad esempio si verifica nei data warehouse: in cui i dati sono duplicati e fusi per essere conservati presso l'integratore. Questa scelta presenta vantaggi in termini di velocità di risposta, specialmente quando l'esecuzione dei dati richiede l'esecuzione di join su molti attributi, però diventa pesante il mantenimento della consistenza tra le sorgenti e la vista. In ragione di queste considerazioni si sta diffondendo sempre più l'approccio virtuale: esso consiste nel non replicare i dati, ma nell'eseguire a run-time la decomposizione della query globale, l'interrogazione delle sorgenti e la fusione delle informazioni.

1.2.2 Trattamento dei MetaDati e delle Ontologie

Il concetto su cui si basa l'integrazione semantica di sorgenti, autonome nel fine ed eterogenee nella struttura, riguarda la sovrapposizione dei rispettivi domini, che si traduce in corrispondenze fra gli schemi delle sorgenti. Per introdurre i problemi legati al processo di individuazione delle similitudini fra questi schemi introduciamo i concetti di Metadato ed Ontologia.

Metadati: rappresentano informazioni relative agli schemi: significato delle classi (o relazioni) e delle loro proprietà (attributi), relazioni sintattiche e/o semantiche esistenti tra i modelli nelle sorgenti, caratteristiche delle istanze

nelle sorgenti: tipi degli attributi, valori null. Utilizzare i metadati può essere di supporto nell'affrontare problematiche relative ad aspetti semantici. Vediamo di approfondire l'argomento. Pur ipotizzando che anche sorgenti diverse condividano una visione similare del problema da modellare, e quindi un insieme di concetti comuni, nulla assicura che i diversi sistemi usino esattamente gli stessi vocaboli per rappresentare questi concetti, ne tantomeno le stesse strutture dati. Al contrario, dal momento che le diverse sorgenti sono state progettate e modellate da persone differenti, è molto improbabile che queste persone condividano la stessa concettualizzazione del mondo esterno: in altre parole non esiste nella realtà una semantica univoca a cui chiunque possa riferirsi.

La causa principale delle differenze semantiche (anche se non l'unica) è identificabile nelle diverse concettualizzazioni del mondo esterno che persone distinte possono avere. Una medesima realtà può essere rappresentata su DBMS differenti che utilizzano modelli diversi: partendo così dalla stessa concettualizzazione, determinate relazioni tra concetti avranno strutture diverse a seconda che siano realizzate, ad esempio, attraverso un modello relazionale o ad oggetti. L'obiettivo dell'integratore di fornire un accesso integrato ad un insieme di sorgenti, si traduce quindi nel difficile compito di identificare i concetti comuni all'interno delle sorgenti e risolvere le eventuali differenze semantiche. Si può classificare queste incoerenze semantiche in tre gruppi principali:

1. *eterogeneità tra le classi di oggetti:* benchè due classi in due differenti sorgenti rappresentino lo stesso concetto nello stesso contesto, possono usare nomi diversi per gli stessi attributi o per i metodi, oppure avere gli stessi attributi con domini di valori differenti o ancora avere regole diverse sui valori;

2. *eterogeneità tra le strutture delle classi:* comprendono le differenze nei criteri di specializzazione, nelle strutture per realizzare un'aggregazione, ed anche le discrepanze schematiche, quando cioè valori di attributi sono invece parte dei metadati in un altro schema (ad esempio, l'attributo SESSO presente in uno schema può essere assente in un al-

tro schema che rappresenta le persone attraverso le classi MASCHI e FEMMINE);

3. *eterogeneità nelle istanze delle classi:* ad esempio, uso di diverse unità di misura per i domini di un attributo, o la presenza/assenza di valori nulli.

Ontologie: per ontologia si intende, in questo ambito, l'insieme dei termini e delle relazioni esistenti in un dominio per denotare concetti ed oggetti. Un'ontologia relativa ad un insieme di sorgenti è costituita da tutti quei termini che identificano in maniera non ambigua lo stesso concetto o la stessa porzione di conoscenza. Un'ontologia può essere più o meno generale a seconda del livello cui ci si riferisce: se ci si riferisce ad una precisa applicazione l'ontologia sarà limitata dipendendo dal dominio e dall'obiettivo della stessa. Le ontologie di livello superiore sono più vaste riferendosi solo al dominio ma essendo indipendenti dall'obiettivo, quelle di livello ancora superiore sono indipendenti anche dal dominio e definiscono concetti molto generali. Più precisamente, mutuando la classificazione fatta da Guarino [GUA 97] è possibile individuare i seguenti livelli di ontologia:

1. **top-level ontology**: descrive concetti molto generali come spazio, tempo, evento, azione. . . , che sono quindi indipendenti da un particolare problema o dominio: si considera ragionevole, almeno in teoria, che anche comunità separate di utenti condividano la stessa top-level ontology;

2. **domain e task ontology**: descrive, rispettivamente, il vocabolario relativo a un generico dominio (come può essere un dominio medico, o automobilistico) o a un generico obiettivo (come la diagnostica, o le vendite), dando una specializzazione dei termini introdotti nelle top-level ontology;

3. **application ontology**: descrive concetti che dipendono sia da un particolare dominio che da un particolare obiettivo.

E' lecito supporre che, integrando sorgenti seppur autonome, il livello dell'ontologia sia vincolato all'interno di un certo dominio; questa restrizione è

congruente con l'idea di integrare i domini comuni di sorgenti che descrivono campi almeno parzialmente sovrapposti. Non disponendo delle informazioni sui Metadati e sulle Ontologie risulta impossibile realizzare una buona integrazione. Questa affermazione risulta facilmente comprensibile osservando che, già trattando una sola sorgente, la rappresentazione del dominio per la costruzione ad esempio di un DB sarà fatta in una infinità di modi diversi a seconda del progettista incaricato, delle informazioni disponibili e della specifica applicazione richiesta; questo fatto risulta ancora più evidente dovendo integrare domini che solo parzialmente sono sovrapposti, che sono stati realizzati da persone diverse, in momenti diversi e con fini del tutto autonomi. Se dunque si desidera integrare è indispensabile poter disporre di strumenti in grado di dedurre, direttamente dagli schemi, informazioni sui metadati e sulle ontologie. Le ontologie di supporto a questo processo sono basate su tecniche di manipolazione della conoscenza, studiate nell'area dell'intelligenza artificiale (AI): queste tecniche sono in grado di simulare intelligenza trasformando problemi di apprendimento in un equivalente problema di ricerca. Una volta ricavata questa conoscenza si possono stabilire dei mapping intersorgente e costruire lo schema integrato.

Capitolo 2

Il sistema MOMIS

MOMIS - acronimo di **M**ediator envir**O**nment for **M**ultiple **I**nformation **S**ources - è un sistema intelligente per l'integrazione di informazioni da sorgenti di dati strutturati e semi-strutturati. Il contributo innovativo di questo progetto, rispetto ad altri similari, risiede nella fase di analisi ed integrazione degli schemi sorgenti, realizzata in modo semiautomatico [BER 98a, BER 98b, BER 99]. MOMIS nasce all'interno del progetto MURST 40% INTERDATA dalla collaborazione tra i gruppi operativi dell'Università di Modena e Reggio Emilia e di quella di Milano.

2.1 L'approccio adottato

MOMIS adotta un approccio di integrazione delle sorgenti semantico e virtuale. L'approccio semantico è descritto in 1.2.1. Con virtuale si intende invece che la vista integrata delle sorgenti, rappresentata dallo schema globale, non viene materializzata, ma il sistema si basa sulla decomposizione delle query e sull'individuazione delle sorgenti da interrogare per generare delle subquery eseguibili localmente; lo schema globale dovrà inoltre disporre di tutte le informazioni atte alla fusione dei risultati ottenuti localmente per poter ottenere una risposta significativa. Le motivazioni che hanno portato all'adozione di un approccio come quello descritto sono varie:

- la presenza di uno schema globale permette all'utente di formulare qualsiasi interrogazione che sia con esso consistente;

- le informazioni semantiche che comprende possono contribuire ad una eventuale ottimizzazione delle interrogazioni;

- l'adozione di una semantica type as a set per gli schemi permette di controllarne la consistenza facendo riferimento alle loro descrizioni;

- la vista virtuale rende il sistema estremamente flessibile, in grado cioè di sopportare frequenti cambiamenti sia nel numero che nel tipo delle sorgenti, ed anche nei loro contenuti (non occorre prevedere onerose politiche di allineamento).

Inoltre si è deciso di adottare un unico modello dei dati basato sul paradigma ad oggetti, sia per la rappresentazione degli schemi sia per la formulazione delle interrogazioni. Il modello utilizzato è ODM_{I^3}. Si tratta di un modello di alto livello che facilita la comunicazione fra il mediatore e il wrapper. Per definire questo modello si è cercato di seguire le raccomandazioni relative alla proposta di standardizzazione per i linguaggi di mediazione: un mediatore deve poter essere in grado di gestire sorgenti dotate di formalismi complessi (ad es. quello ad oggetti) ed altre decisamente più semplici (come i file di strutture), è quindi preferibile l'adozione di un formalismo il più completo possibile.

Per la descrizione degli schemi si è arrivati a definire il linguaggio ODL_{I^3} che si presenta come estensione del linguaggio standard ODL proposto dal gruppo di standardizzazione ODMG-93.

Per quanto riguarda il linguaggio di interrogazione si è adottato OQL_{I^3} che adotta la sintassi OQL senza discostarsi dallo standard. Questo linguaggio richiede un maggiore sforzo dal punto di vista dello sviluppo di moduli per l'interpretazione e la gestione delle interrogazioni (implementando le funzionalità tipiche di un ODBMS), ma risulta altresì estremamente versatile ed espressivo fornendo la possibiltà di sfruttare le informazioni rappresentate nello schema globale. Le maggiori difficoltà implementative sono quindi ampiamente giustificate da una maggiore versatilità e da una migliore facilità d'uso per l'utente finale.

Riassumendo: in MOMIS i wrapper si incaricano di tradurre in ODL_{I^3} le descrizioni delle sorgenti, i moduli del mediatore di costruire lo schema

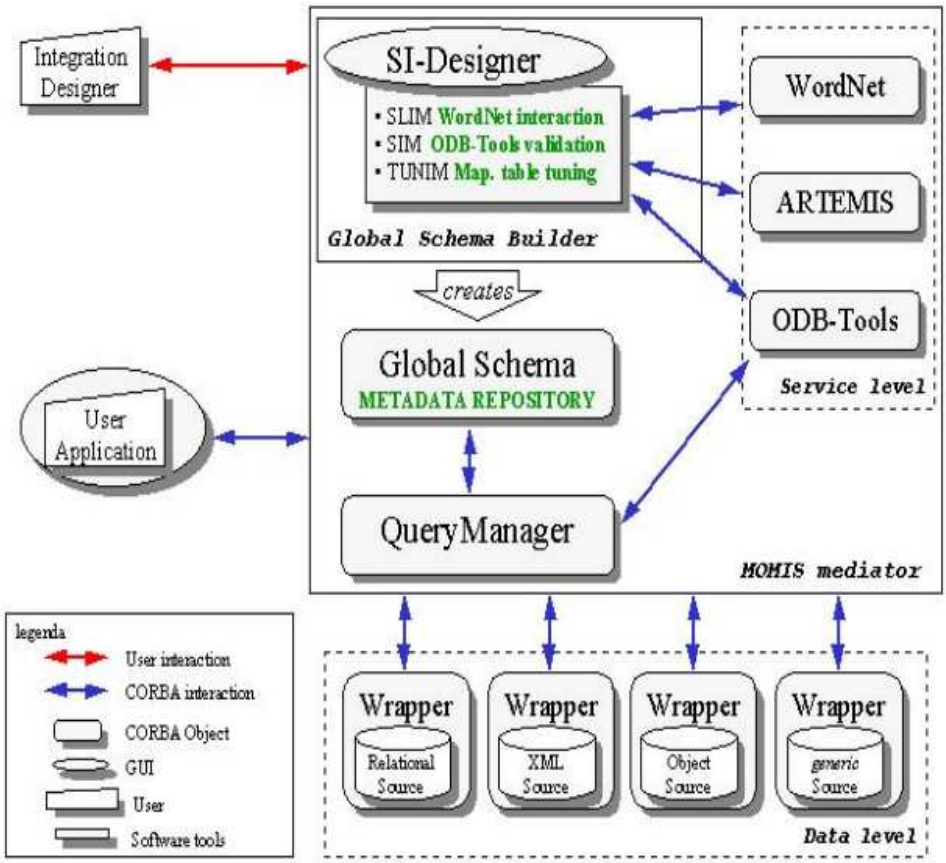

Figura 2.1: L'architettura di MOMIS.

globale, mentre tutte le interrogazioni sono poste dall'utente su questo schema utilizzando il linguaggio OQL_I [3] disinteressandosi dell'effettiva natura ed organizzazione delle sorgenti; sarà il sistema stesso che si incaricherà di tradurre le interrogazioni in un linguaggio comprensibile dalle singole sorgenti e di riorganizzare le risposte ottenute in una unica corretta e completa da fornire all'utilizzatore.

2.2 L'architettura generale di MOMIS

In Figura 2.1 è illustrata dettagliatamente l'architettura generale di MOMIS. Lo schema evidenzia l'organizzazione a tre livelli utilizzata.

Livello Mediatore. Il nucleo centrale del sistema è costituito dal Mediatore (o Mediator) che presiede all'esecuzione di diverse operazioni. Per meglio comprendere i suoi compiti, è opportuno a questo punto illustrare le due fasi ben distinte in cui si articola la sua attività. La prima funzionalià del Mediatore è quella di generazione dello Schema Globale. In questa fase il modulo del Mediatore denominato Global Schema Builder riceve in input le descrizioni degli schemi locali delle sorgenti espressi in ODL_{I^3} forniti ognuno dal relativo wrapper. A questo punto (utilizzando strumenti di ausilio quali ODB-Tools Engine, WordNet, ARTEMIS) il Global Schema Builder è in grado di costruire la vista virtuale integrata (Global Schema) utilizzando tecniche di clustering e di Intelligenza Artificiale. In questa fase è prevista anche l'interazione con il progettista il quale, oltre ad inserire le regole di mapping, interviene nei processi che non possono essere svolti automaticamente dal sistema (come ad es. l'assegnamento dei nomi alle classi globali, la modifica di relazioni lessicali, . . .). Oltre allo Schema Globale, altri prodotti di questa fase sono le Mapping Table, tabelle che descrivono il modo in cui gli attributi globali presenti nello schema generato hanno corrispondenza (mappano) nei vari attributi locali presenti negli schemi delle sorgenti. Un secondo importante modulo che compone la struttura del mediatore è il Query Manager che presiede alla fase di query processing. In questa fase la singola query posta in OQL_{I^3} dall'utente sullo Schema Globale (che chiameremo Global Query) sarà rielaborata in più Local Query (anche'esse espresse in OQL_{I^3} da inviare alle varie sorgenti, o meglio ai wrapper predisposti alla loro traduzione. Questa traduzione avviene in maniera automatica da parte del Query Manager utilizzando tecniche di logica descrittiva.

L'ultimo modulo del Mediatore è rappresentato dall'Extensional Hierarchy Builder il quale si occupa della generazione della Conoscenza Estensionale (Gerarchie Estensionali e Base Extension) necessaria per ottimizzare le interrogazioni.

Livello wrapper. I Wrapper costituiscono l'interfaccia tra il mediatore e le sorgenti; ad ogni sorgente corrisponde un determinato wrapper ed ogni wrapper deve essere disegnato esclusivamente per la sorgente (o la tipologia di sorgenti) che sovrintenderà. Ogni wrapper ha due compiti ben precisi:

- in fase di integrazione deve fornire al Global Schema Builder la descrizione della sorgente da integrare in formato ODL_{I^3};
- in fase di query processing deve tradurre la local query (rivolta alla sua sorgente) che gli è stata indirizzata dal Query Manager (e che è espressa in OQL_{I^3}) nel linguaggio di interrogazione specifico della sorgente per cui è progettato.

Collegate ai wrapper sono quindi le Sorgenti, per questo a volte si parla anche di quattro livelli. Esse sono le fonti da integrare, possono essere DataBase (ad oggetti o relazionali) o parti di essi, file system ed anche sorgenti semistrutturate.

Livello utente: L'utilizzatore del sistema dovrà potere interrogare lo schema globale. L'accesso ai dati si propone del tutto simile a come avvengono le usuali interrogazioni di un DataBase tradizionale: le sorgenti ed il modo in cui i dati vengono recuperati risultano all'utente del tutto trasparenti, in quanto è il sistema ad occuparsi di tutte le operazioni necessarie per reperire le informazioni e combinare le risposte in un'unica risposta corretta, completa e non ridondante. Tra i tools di ausilio del mediatore si possono riconoscere:

ODB-Tools è uno strumento software sviluppato presso il dipartimento di Ingegneria dell'Università di Modena e Reggio Emilia [BEN 97a, BEN 97b]. Esso si occupa della validazione di schemi e dell'ottimizzazione semantica di interrogazioni rivolte a Basi di Dati orientate agli Oggetti (OODB).

WORDNET [MIL 90a] è un DataBase lessicale on-line in lingua inglese. Esso è capace di individuare relazioni semantiche fra termini; cioè, dato un insieme di termini, WordNet è in grado di identificare l'insieme di relazioni lessicali che li legano. Si tratta di un componente fondamentale sia per MOMIS sia per l'agente mobile progettato e realizzato in questa tesi. Dato il suo ruolo cruciale, maggiori dettagli verranno forniti in 2.3.

ARTEMIS [CAS 97] riceve in ingresso il thesaurus, cioè l'insieme delle relazioni terminologiche (lessicali e strutturali) precedentemente generate,

e sulla base di queste assegna ad ogni classe coinvolta nelle relazioni un coefficiente numerico indicante il suo grado di affinità. Questi coefficienti verranno utilizzati per raggruppare le classi locali in modo tale che ogni gruppo (cluster) comprenda solo classi con coefficienti di affinità simili.

2.3 WordNet, un componente fondamentale di MOMIS

WordNet [MIL 90a] è sviluppato dal Cognitive science Laboratory sotto la direzione del professore George A. Miller presso l'università di Princeton [1] ed è diventato una delle più importanti risorse per lo sviluppo di tecniche di linguistica computazionale [HIR 98] [ALK 97] e di altre aree associate [GRE 99]. Sebbene la versione attualmente disponibile on line sia la 1.7.1, MOMIS utilizza ancora la vecchia versione 1.6. Ad essa si farà riferimento anche per quanto riguarda il progetto realizzato in questa tesi. WordNet è un sistema di gestione di un dizionario lessicale basato sulle teorie psicolinguistiche della memoria lessicale umana. WordNet riconosce quattro categorie sintattiche: nomi, verbi, aggettivi ed avverbi, ognuna delle quali è organizzata in insiemi di sinonimi (*synonym sets o synsets*). Ogni insieme di sinonimi si riferisce ad un particolare concetto ed è posto in relazione con altri synsets tramite relazioni lessicali. Inoltre, WordNet riconosce che il termine "parola" è inerentemente ambiguo, perchè non permette di distinguere fra quello che è il modo con cui una parola viene scritta o pronunciata e il significato che essa assume. Pertanto, WordNet definisce **lemma** la forma scritta o il suono di una parola e indica con **significato** il concetto ad essa associato. In questo modo si possono spiegare i fenomeni di:

Sinonimia: proprietà di un concetto di avere due o più parole in grado di esprimerlo.

Polisemia: proprietà di una parola di avere due o più significati.

[1] Il sito web ufficiale si trova all'indirizzo http://www.cogsci.princeton.edu/~wn/.

In WordNet esistono due tipi di relazioni: le relazioni semantiche e quelle lessicali. Mentre le prime sussistono fra significati, le seconde sussistono fra parole. WordNet è quindi una rete di relazioni semantiche e lessicali, ognuna delle quali è rappresentata da un puntatore. La regola generale che i puntatori devono seguire prevede che non possano esistere relazioni fra due diverse categorie sintattiche, a meno di casi eccezionali.

Le relazioni lessicali sono classificate in:

Synonymy se una espressione può essere sostituita ad un'altra in un certo contesto senza che il significato subisca rilevanti alterazioni.

Antonimy se una espressione è il contrario di un'altra. Ad esempio i termini "ricco" e "povero" sono l'uno il contrario dell'altro. La relazione di antonimy rappresenta l'organizzazione fondamentale degli aggettivi e degli avverbi.

See also se è previsto un puntatore che da un synset di aggettivi o verbi permetta di recuperare altri synsets ad esso correlati. Ad esempio dal synset di AWAKE/ASLEEP esiste un puntatore "see also" verso il synset di ALERT/UNALERT.

Pertaynym in cui una coppia formata da un nome ed un aggettivo è in relazione con un altro nome. Ad esempio la locuzione "fraternal twin" è in relazione con "brother", così come "dental hygiene" è in relazione con "tooth".

Derived From è una relazione che sussiste fra quegli aggettivi o verbi che, pur esprimendo lo stesso concetto, hanno una diversa forma perchè derivanti da lingue diverse. Ad esempio "rhinal" e "nasal" hanno un significato simile ma derivano rispettivamente dal greco e dall'idioma anglosassone.

Participle of verb. La lingua inglese ha vari metodi per alterare il significato di un nome creando espressioni composte da aggettivo e nome o da una coppia di nomi. L'obiettivo è quello di specializzare un concetto altrimenti poco selettivo.

Le relazioni semantiche possono essere invece classificate in:

Hyponymy/Hypernymy Si dice che un synset A ed un synset B sono in relazione di hyponymy se un madrelingua inglese riconosce che A *is a kind of* B. Si tratta di una relazione transitiva ed asimmetrica che determina l'organizzazione base del file dei nomi.

Meronymy/Holonymy Si dice che un synset A ed un synset B sono in relazione di meronymy se un madrelingua inglese riconosce che A *has a part of* B. Anche la relazione meronymy è transitiva ed asimmetrica.

Entailment Si tratta di una relazione di implicazione fra verbi. Ad esempio la frase "He is snoring" implica la frase "He is sleeping". La relazione di Entailment è simile alla relazione di meronymy.

Cause To La maggior parte delle lingue riesce ad esprimere le relazioni di causa-effetto tramite un lessico appropriato. Spesso non si tratta di un singolo termine ma di una coppia di termini, ad esempio "show-see".

Verb group Al contrario dei nomi, l'organizzazione dei verbi è più complessa dal punto di vista semantico. Esistono infatti dei gruppi di verbi, detti *cluster of verbs* che possono essere organizzati in maniera tassonomica, ad esempio i verbi di creazione e comunicazione. Lo studio dell'organizzazione dei verbi è sicuramente interessante, ma riveste un'importanza marginale in MOMIS.

Similar To I synset di aggettivi sono organizzati in cluster in cui tutti gli elementi esprimono un concetto semanticamente simile. Poichè gli aggettivi sono organizzati soprattutto sulla base della relazione lessicale di antonymy, dire che un aggettivo è simile ad un altro significa determinare anche la similarità fra gli aggettivi contrari.

Attribute Gli attributi di un oggetto sono espressi generalmente da aggettivi. Spesso l'utilizzo degli aggettivi permette di discriminare, fra un insieme di oggetti, quello di interesse. In particolare, la maggior parte degli attributi sono espressi tramite gli *descriptive adjectives*, i quali sono collegati con i nomi tramite opportuni puntatori.

2.4 Generazione del Common Thesaurus

Si prenda come esempio di riferimento quello proposto in [BER 98a], in cui si vogliono integrare tre sorgenti eterogenee:

1. `UNIVERSITY`, una sorgente relazionale che contiene informazioni sugli studenti e lo staff di ricerca di un'università;
2. `COMPUTER_SCIENCE`, una sorgente semistrutturata che contiene informazioni sulla facoltà di informatica della università di cui al punto 1.
3. `TAX_POSITION` rappresentata da un file in cui è descritta la posizione fiscale di ogni studente.

La descrizione delle sorgenti in ODL_{I^3} è la seguente:

```
UNIVERSITY source:
interface Research_Staff
  ( source relational University
    extent Research_Staff
    key name
    foreign_key dept_code, section_code )
  { attribute string name;
    attribute string relation;
    attribute string e_mail;
    attribute integer dept_code;
    attribute integer section_code; };
interface Department
  ( source relational University
    extent Department
    key code )
  { attribute string dept_name;
    attribute integer dept_code;
    atribute integer budget; };
interface Room
  ( source relational University
```

```
    extent Room
    key room_code )
  { attribute integer room_code;
    attribute integer seats_number;
    attribute string notes; };
interface School_Member
  ( source relational University
    extent School_Member
    key name )
  { attribute string name;
    attribute string faculty;
    attribute integer year; };
interface Section
  ( source relational University
    extent Section
    key section_name
    foreign_key room_code )
  { attribute string section_name;
    attribute integer section_number;
    attribute integer length;
    attribute integer room_code; };



COMPUTER_SCIENCE source:
interface CS_Person
 ( source object Computer_Science
  extent CS_Persons
  keys first_name, last_name )
 { attribute string first_name;
  attribute string last_name; };
interface Student: CS_Person
 ( source object Computer_Science
   exyends Students )
```

```
 { attribute integer year;
   attribute set<Course> takes;
   attribute string rank; };
interface Location
 ( source object Computer_Science
   extent Locations
   keys city, street, county, number )
 { attribute string city;
   attribute string street;
   attribute string county;
   attribute integer number; };
interface Professon: CS_Person
 ( source object Computer_Science
   extent Professors )
 { attribute string title;
   attribute Office belongs_to;
   attribute string rank; };
interface Office
 ( source object Computer_Science
   extent Offices
   key description
 { attribute string description;
   attribute Location address; };
interface Course
 ( source object Computer_Science
   extent Courses
   key course_name )
 { attribute string course_name;
   attribute Professor taught_by; };


Tax_Position source;
interface Univesity_Student
 ( source file Tax_Position
```

```
   extent University_Student
   key student_code )
{    attribute string name;
     attribute integer student_code;
     attribute string faculty_name;
     attribute integer tax_fee; };
```

L'integrazione delle sorgenti realizzata dal *Global Schema Builder* si basa sull'individuazione di un'ontologia comune alle diverse fonti. In MOMIS un'ontologia di questo tipo è definita *Common Thesaurus*. Il Common Thesaurus esprime le relazioni inter-schema ed intra-schema degli schemi delle sorgenti da integrare. Si riconoscono relazioni di

Sinonimia SYN definita da due termini che possono essere scambiati nelle sorgenti senza modificare il concetto da rappresentare nello schema comune.

Specializzazione BT definita da due termini in cui il primo ha un significato più generale del secondo.

Aggregazione RT definita tramite due termini fra i quali esiste una relazione generica.

La costruzione del Common Thesaurus è un complesso processo incrementale in cui vengono man mano aggiunte relazioni semantiche intra-schema, relazioni lessicali, relazioni aggiunte dal progettista e relazioni intensionali inferite.

Per quanto riguarda le relazioni intra-schema delle sorgenti ad oggetti, possono essere agevolmente estratte le relazioni BT/NT dalla gerarchia di generalizzazione e le relazioni RT dalla gerarchia di aggregazione. Per le sorgenti relazionali è invece possibile ricavare relazioni RT e BT dall'analisi delle *foreign key*.

L'estrazione di relazioni lessicali (nel senso di relazioni fra termini indipendentemente dalla classificazione fatta nel Paragrafo 2.3) ha la sua ragion d'essere nel fatto che il progettista di una sorgente ha codificato implicitamente nello schema delle sorgenti parte della loro semantica. Ad esempio,

se una particolare classe farà riferimento agli studenti, allora ci si aspetta che essa venga denominata come STUDENT o con un termine dal significato analogo. L'ipotesi alla base di questo tipo di approccio è che i nomi utilizzati per descrivere entità, etc. . . siano significativi e descrittivi. Grazie a WordNet è possibile estrarre varie relazioni, tra le quali quelle di sinonimia, ipernimia, iponimia, olonimia e meronimia sono le più significative.

Il limite di WordNet risiede nel fatto che è in grado di catturare solo relazioni che sussistono fra due termini a livello linguistico. In alcuni casi potrebbe essere necessario l'intervento di un operatore che, sulla base della propria esperienza, possa aggiungere altre relazioni importanti. In questa fase il modulo ARTEMIS può fornire importanti indicazioni circa l'affinità strutturale delle varie classi. ARTEMIS calcola l'affinità strutturale tramite un coefficiente detto *Global Affinity*, a sua volta combinazione lineare di altri due coefficienti, il primo valuta l'affinità strutturale (*Structural Affinity*) delle classi, il secondo valuta l'affinità dei nomi delle classi e degli attributi (*Name affinity*). Questi tre passi devono subire un processo di validazione, per garantire che vi sia compatibilità fra i tipi di dati dei termini messi in relazione.

Infine, la quarta ed ultima fase vengono inferite nuove relazioni semantiche intensionali. L'obiettivo è quello di produrre uno schema virtuale che contiene descrizioni strutturali delle sorgenti da integrare riorganizzate secondo quanto ricostruito nel Common Thesaurus. Si tratta quindi di uniformare la descrizione delle classi riconosciute simili, così come si deve fare in modo che per due classi tra le quali esista una relazione di specializzazione vi sia anche una relazione di specializzazione fra le due descrizioni e così via.

2.5 Generazione delle classi globali

Tramite il SI-Designer vengono generate le classi globali. In primo luogo vengono calcolate tramite ARTEMIS le affinità e vengono generati dei cluster, in cui le classi sono rappresentate in una struttura ad albero. Le foglie dell'albero rappresentano le classi locali. Si dice che due foglie, quindi due classi, contigue hanno un elevato grado di affinità. Nell'organizzazione gerarchica i

nodi rappresentano un livello di clusterizzazione. Ogni nodo ha associato un coefficiente che combina i coefficienti di affinità dei sottoalberi che unisce.

Per ogni cluster, il SI-Designer crea un insieme di attributi globali e per ognuno di essi determina la corrispondenza con gli attributi locali. Inoltre, l'insieme degli attributi globali da associare ad un cluster viene determinato in due fasi. La prima prevede l'unione di tutti gli attributi delle classi affini rappresentate nel cluster di interesse; la seconda elimina le ridondanze fondendo attributi simili. In questa seconda fase è fondamentale la conoscenza espressa nel Common Thesaurus.

L'attività di SI-Designer si conclude con la generazione di una mapping-table per ogni classe globale. SI-Designer mette a disposizione del progettista un'interfaccia che permette di avere una visione completa di tutte le classi (nomi ed attributi).

2.6 Il sistema MIKS

Come si è visto, il sistema MOMIS è in grado di realizzare l'integrazione di sorgenti di dati eterogenee, ma quando si tratta di uno scenario complesso in cui le sorgenti sono distribuite, allora l'architettura client-server potrebbe non essere più una soluzione ottimale.

Per questo motivo è stato integrato in MOMIS l'utilizzo di agenti intelligenti. Il sistema proposto è rappresentato in Figura 2.2 ed è denominato **MIKS** - **M**ediator Agent for **I**ntegration of **K**nowledge **S**ource. MIKS è basato sull'utilizzo di sistemi multi-agente (MASs).

Il modello multi-agente si propone di progettare sistemi in cui gli agenti si comportano in modo autonomo, cooperando ed interagendo fra di loro per svolgere mansioni complesse.

L'utilizzo di agenti all'interno del sistema MIKS, oltre a rendere più efficiente i processi di integrazione e la fase di interrogazione può essere sfruttato per altre funzionalità fra cui la ricerca e l'acquisizione di nuove sorgenti dati. Spesso si può presentare la necessità di integrare nuove sorgenti di informazione per migliorare la quantità e la qualità dei dati a disposizione. In questo caso il sistema dovrà occuparsi di ricercare le nuove potenziali sorgenti dati

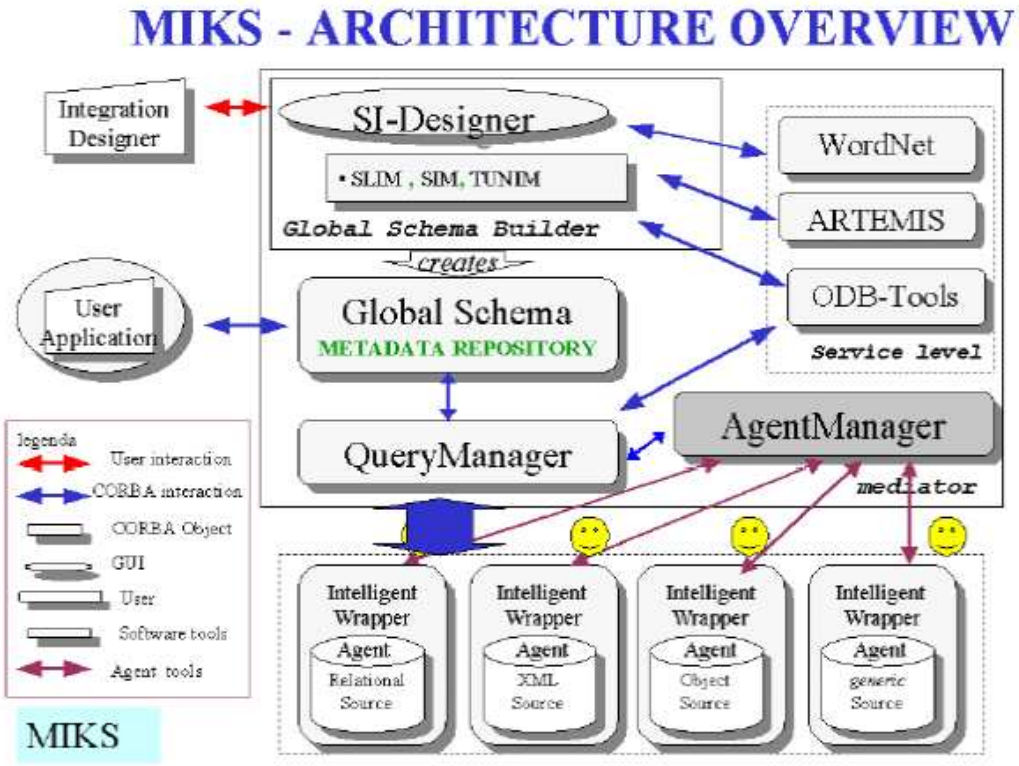

Figura 2.2: Il sistema MIKS.

in un determinato dominio (ad esempio il Web) e gestire la fase di analisi ed acquisizione dei dati.

Per far fronte a queste problematiche può essere utile lo sfruttamento delle caratteristiche degli agenti mobili. Il sistema, infatti, potrebbe delegare ad un certo numero di agenti il compito di ricercare nuove fonti di informazione specificando alcuni parametri che permettano di individuare dati effettivamente utili alle necessità del sistema. Altri agenti potrebbero occuparsi dell'acquisizione e dell'archiviazione dei dati ritenuti interessanti.

Grazie alle loro caratteristiche, tutti gli agenti coinvolti anche in mansioni diverse potrebbero comunicare e cooperare. In questo modo un agente potrebbe decidere di modificare il proprio comportamento, interrompendo o variando l'attività svolta, in modo da aumentare l'efficienza della ricerca.

Nella tesi di Enrico Natalini [NAT 02] è stata studiata la piattaforma ad agenti JADE realizzata dall'Università di Parma [BEL 02a, BEL 02b]. La piattaforma JADE è una struttura di sviluppo software mirata alla gestione di applicazioni e sistemi multi-agente conformi agli standard FIPA. Si faccia riferimento alla tesi di Natalini per ogni altro dettaglio su JADE.

In questa tesi viene presentato un nuovo agente hunter in grado di adattarsi alle condizioni della rete e di utilizzare le informazioni raccolte durante la fase di esplorazione per determinare una strategia sub-ottima per recuperare sorgenti di informazioni interessanti prima di altre meno rilevanti. Inoltre, l'agente presentato è innovativo anche perchè adotta una nuova tecnica per valutare il grado di interesse di una risorsa rispetto al Common Thesaurus del MOMIS. Il nuovo agente, la cui realizzazione è descritta nel Capitolo 8, è stato denominato **TUCUXI** - In**T**elligent H**U**nter Agent for **C**oncept **U**nderstanding and Le**X**ical Cha**I**ning.

Parte II

La scoperta delle informazioni

Capitolo 3

Topologia del Web: Il *Web Graph*

Il World Wide Web è stato paragonato ad un enorme grafo, in cui i nodi sono rappresentati dalle pagine, mentre i *links* costituiscono gli archi. Pertanto, dal punto di vista topologico, il Web viene chiamato anche *Web Graph*.

Vari studi hanno cercato estrapolare la dimensione del Web [LAW 98a, LAW 99a, BHA 99, BAR 00] e, pur fornendo dati riferiti a periodi di osservazione diversi, tutti concordano sul fatto che il numero di documenti pubblicati ha ormai superato i due miliardi. Per quanto riguarda il tasso di crescita del Web, Lawrence e Giles osservano che la dimensione del Web raddoppia ogni 15 mesi [LAW 99a].

Studi più approfonditi e recenti hanno cercato di individuare proprietà del *Web Graph* non solo quantitative, con particolare riferimento a caratteristiche di connettività. L'interesse per la connettività è legato essenzialmente alla possibilità di utilizzare strumenti *software*[1] per esplorare automaticamente grandi porzioni di Web. Proprio i risultati di una sessione di crawling[2] dello spider di *Altavista* [8] sono alla base dell'analisi di Broder e altri [BRO 00], dalla quale si può trarre una conclusione apparentemente inaspettata: per la maggior parte delle coppie di pagine u e v non esiste un percorso che permetta di raggiungere v a partire da u. Quindi, alla luce di questo dato,

[1] Spiders, crawlers, robots, gatherers...

[2] Esperimenti effettuati tra maggio 1999 e ottobre 1999 su 200 milioni di pagine e 1,5 miliardi di links, con strategia di crawl di tipo breadth-first a partire da seed sites random.

l'utilizzo degli strumenti *software* precedentemente citati potrebbe sembrare poco fruttuoso, ma la sessione di crawling rivela ulteriori informazioni. Ad esempio, le pagine visitate possono essere classificate in insiemi (*components*) *weakly-connected* o *soft-connected*.

Definizione 1 *Un* **Weakly-Connected Component** *è un set di pagine, ognuna delle quali è raggiungibile dalle altre se gli iperlinks possono essere seguiti o nella direzione* **forward** *o nella direzione* **backward**.

Definizione 2 *Un componente* **Strongly-connected** *è un insieme tale che per ogni coppia di pagine (u, v) esiste un percorso diretto fra di esse. Ciò significa che un utente può raggiungere* v *seguendo l'iperlink contenuto in* u.

Quantitativamente, circa il 90% del crawl (186 milioni di pagine) è stato classificato come componente *weakly-connected*, mentre al componente *strongly-connected* di maggiore dimensione è risultato appartenere solo il 28% dei documenti recuperati.

Il *Web Graph* può essere rappresentato in una mappa simile a quella di Figura 3.

Il Web è paragonato ad un *bowtie*, in cui il componente centrale è proprio il set di tipo *strongly-connected* (SCC o core) citato in precedenza. Alla sinistra di SCC è rappresentato un gruppo di 44 milioni di pagine denominato IN, le quali consentono un flusso di navigazione unidirezionale verso SCC. Ciò significa che a partire da IN si raggiungono i documenti appartenenti a SCC, mentre non è possibile procedere in direzione inversa.

Analogamente, le pagine contenute in OUT (44 milioni circa) sono recuperabili da SCC ma non viceversa. Nella figura si indica con TUBES l'insieme degli iperlinks che collegano direttamente alcune pagine di IN ad altre di OUT senza attraversare il componente centrale

Esiste, infine, una quarta regione denominata TENDRILS in cui sono classificabili quelle pagine che non possono nè raggiungere SCC, nè essere raggiunte dall' esterno. La porzione di Web rappresentata da Tendrils è significativa, in quanto di dimensione paragonabile a quella di IN e OUT.

Quindi, grazie a questi dati numerici, è possibile giustificare statisticamente l'affermazione fatta in precedenza per cui per la maggior parte delle

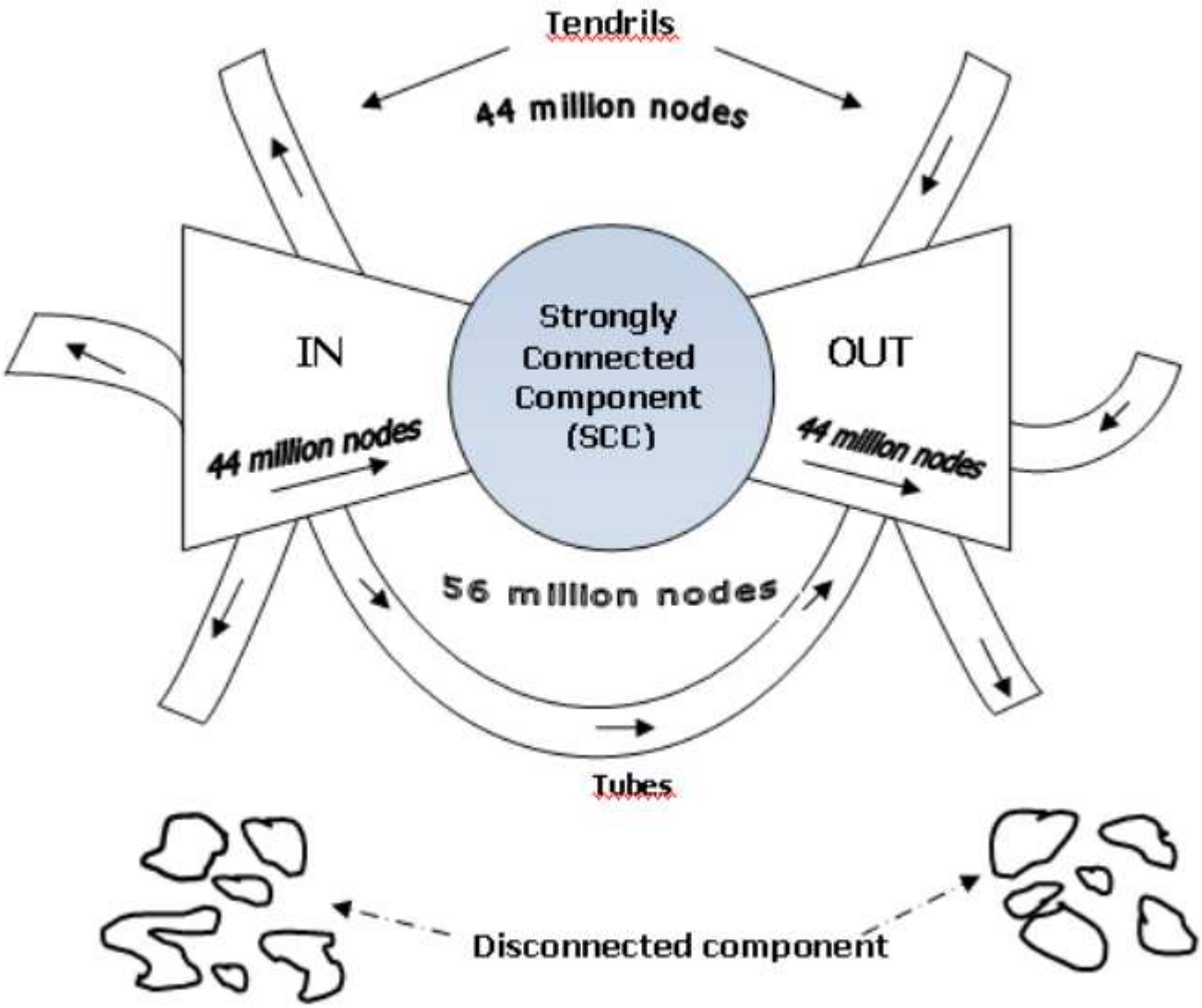

Figura 3.1: Il bowtie secondo Broder.

coppie di pagine u e v non esiste un percorso che le colleghi. Per ipotesi, si consideri che la pagina u appartenga all'insieme $IN \cup SCC$ e che la pagina v appartenga invece all'insieme $SCC \cup OUT$. La probabilità che una pagina appartenga ad uno qualsiasi dei due insiemi è stimabile nel 50% circa. La probabilità che la pagina sorgente u e la pagina destinazione v appartengano entrambe ad uno dei due insiemi è del 25%. Di conseguenza, la probabilità che fra due pagine non esista un percorso è del 75%.

Assumendo di analizzare solo le pagine u e v tra le quali esiste un percorso, si calcola che il numero medio di out-link da seguire per raggiungere v sia un numero finito pari a 19 [ALB 99]. Questo dato che, seppur imprecisamente rappresenta il diametro del Web (*Web Diameter*), permette di concludere che il Web è veramente immenso, ma un crawler potrebbe comunque muoversi da un sito all'altro senza impiegare eccessive risorse.

Il diametro del Web è impreciso innanzitutto perchè viene calcolato esclusivamente sulla base di quelle pagine tra le quali esiste un percorso diretto, e perchè non considera il fatto che il Web non è statico. Infatti, non solo la sua dimensione raddoppia ogni 15 mesi, ma il contenuto delle pagine cambia, si evolve. Cho e Garcia-Molina [CHO 00a] hanno effettuato esperimenti volti a quantificare il tempo di vita medio delle pagine pubblicate sul Web. Il Standfor WebBase crawler ha infatti monitorato giornalmente 270 siti per un periodo di circa 3 mesi (un totale di 720000 pagine). Sulla base dei dati raccolti, risulta che oltre il 20% di tutte le pagine visitate (ved. Figura 3.2) varia il proprio contenuto almeno una volta al giorno. Ciò significa che il contenuto viene costantemente aggiornato, soprattutto per quanto riguarda i domini di carattere commerciale. Al contrario i domini `.org` e `.edu` risultano essere molto più statici ed infatti oltre il 50% delle pagine di questi domini non varia il contenuto per un periodo superiore ai 4 mesi (Figura 3.3).

La grande volatilità delle pagine Web è confermata anche dalle statistiche recentemente fornite da *Google*[11], dalle quali si evince che oltre il 70% delle pagine ha un'età inferiore ad un anno. Questo dato comprende sia le nuove pagine pubblicate sia le pagine già presenti il cui contenuto subisce un'evoluzione nel tempo.

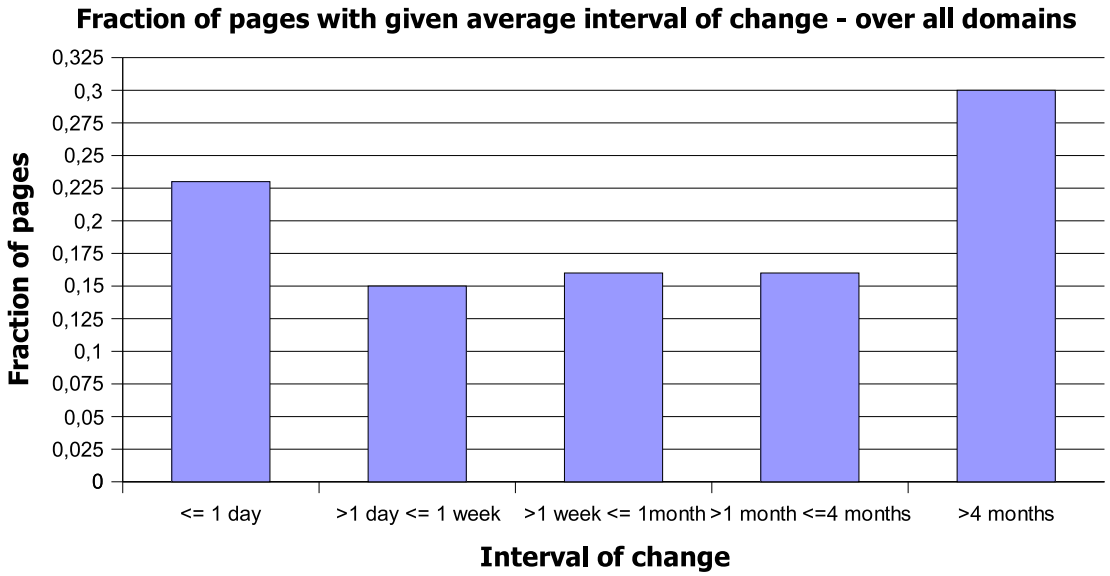

Figura 3.2: Average change interval over all web page. Fonte: Cho e Garcia-Molina: The Evolution of the Web and Implications for an Incremental Crawler, 26th VLDB Conference, 2000.

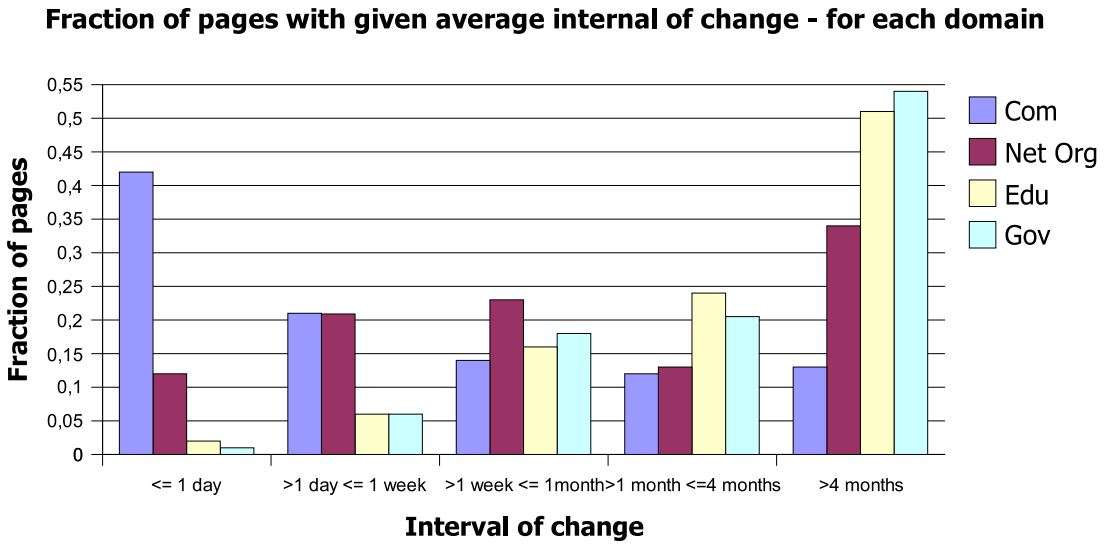

Figura 3.3: Average change interval for each domain. Fonte: Cho e Garcia-Molina: The Evolution of the Web and Implications for an Incremental Crawler, 26th VLDB Conference, 2000.

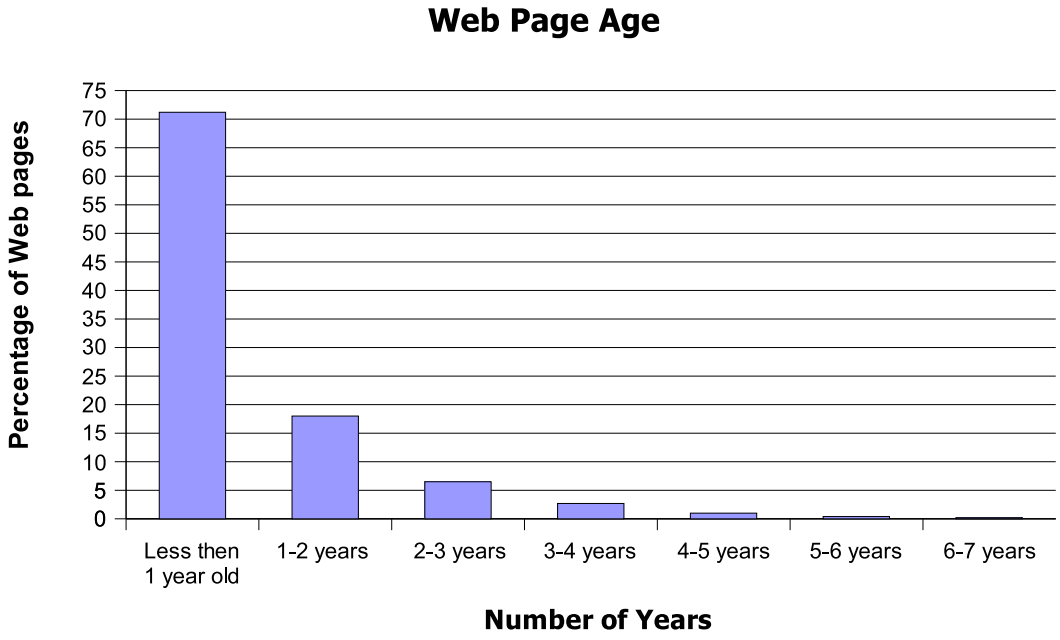

Figura 3.4: Età media delle pagine indicizzate da Google, così come indicato dai Web Servers. Fonte: www.Google.com

Capitolo 4

Strumenti di ricerca generali

Nel capitolo precedente si sono illustrate le conclusioni ottenute da importanti studi sul Web. In sintesi, si può affermare che i documenti *on-line* sono ormai miliardi e il loro contenuto è costantemente in evoluzione. Di conseguenza chi cerca informazioni in rete deve necessariamente utilizzare qualche strumento che gli permetta di districarsi nella massa dei dati potenzialmente interessanti (*Information Overload*).

Attualmente è possibile utilizzare tre diverse tipologie di strumenti: le *Web directories,* i *search engines* e i *metasearch engines.* Nel linguaggio comune questi *tools* sono designati genericamente con il nome di motori di ricerca, ma dal punto di vista concettuale ed architetturale si tratta di strumenti completamente diversi.

Nei paragrafi seguenti verranno illustrati alcuni esempi significativi.

4.1 Web Directories

Le *Web directories* forniscono un servizio di riorganizzazione tematica del Web, classificando le pagine pubblicate in una gerarchia. Ogni nodo della gerarchia rappresenta un argomento, il quale, a sua volta, può essere raffinato in un certo numero di concetti più specifici. Ogni nodo genera quindi un numero di figli pari al numero di sotto-concetti individuati, in modo che esplorando la gerarchia si visitino dapprima i *broad topics*, ovvero i nodi di primo livello, successivamente i *fine topics*, o nodi di livello inferiore, fino

alle foglie. Lo scopo della categorizzazione è quello di popolare la gerarchia ottenuta.

Sebbene il problema della classificazione sia stato ampiamente studiato nell'ambito dell'*Information Retrieval* (IR) e dell'*Artificial Intelligence* (AI), si tratta comunque di un'operazione complessa e delicata, tanto che spesso viene ancora affidata a *team* di esperti (*surfers*). L'uso di una gerarchia per accedere ai contenuti del Web è uno strumento semplice da utilizzare e permette una facile navigazione anche agli utenti meno esperti. Purtroppo, una pagina tratta normalmente più di un argomento, pertanto potrebbe essere classificata in più categorie. La multiclassificazione tende ad evolvere la struttura prettamente gerarchica precedentemente descritta verso una struttura ricca di riferimenti incrociati. La creazione di collegamenti *interbranch* dovrebbe essere limitata ai soli casi strettamente necessari,per garantire la coerenza della navigazione per argomento. Considerando il fatto che le tassonomie create dalle attuali Web directories contengono centinaia di concetti, appare evidente come l'attività degli esperti diventi indispensabile per mantenere la compattezza della struttura.

Lo svantaggio delle Web directories è insito nel loro vantaggio. L'attività di esperti, infatti, è ottima per quanto riguarda la capacità di multiclassificazione, ma è altrettanto vero che la classificazione ottima è impossibile da ottenere, perchè persone diverse, con un background culturale diverso, tendono ad organizzare la conoscenza in maniera diversa. A sostegno delle Web directories si può comunque dire che le pagine classificate sono di buona qualità perchè, dato un argomento, è proprio l'intervento di esperti che consente di discriminare pagine di maggior interesse da documenti di interesse marginale, fino ad ottenere un set di pagine ben selezionate. Questa organizzazione, però, non offre sufficienti prospettive di scalabilità rispetto alla crescita impressionante del Web.

Le raccolte di documenti delle Web directories sono sicuramente vaste, ma poco aggiornate rispetto al numero di nuove pagine che ogni giorno appaiono *on-line*. Inoltre, per effetto della volatilità dei contenuti alcune pagine, correttamente classificate in un dato istante, potrebbero non corrispondere più alla categoria assegnata.

D'altro canto anche gli stessi *fine topics* non dovrebbero essere statici:

gli argomenti trattati sul Web sono in continua evoluzione, così come sono in evoluzione gli interessi degli utenti che utilizzano le *Web directories*. Per rispondere a questo elenco di problematiche, alcune Web directories affiancano al proprio *team* di esperti alcuni classificatori di testo semiautomatici, i quali sono in grado di elaborare un numero di documenti sicuramente superiore al numero di documenti analizzati dagli operatori umani. Molti di questi classificatori automatici sfruttano un approccio al problema di tipo *Machine Learning* da set di esempi. Con l'aumentare delle classi della tassonomia, però, è dimostrato che anche questi strumenti non offrono una buona scalabilità.

Altri tools sono in grado di generare automaticamente anche la gerarchia, vale a dire sono in grado di individuare nuove possibili categorie non appena compaiono sul Web. Questi strumenti sfruttano alcune tecniche di clustering adattate al contesto del Web. Interessante è il loro uso anche per quanto riguarda l'information filtering. Una nuova tendenza dei motori di ricerca in generale e non solo delle Web directories è quella di presentare i risultati ad una query pre-classificandoli *on-the-fly*.

4.1.1 *Yahoo!*

Yahoo! [7] è tuttora una delle risorse on-line più utilizzate per le ricerche in rete ed è ancora uno dei siti web più frequentati in assoluto. L'idea di una web directories nacque da due studenti dell'università di Stanford, Jerry Yand e David Filo, che verso la fine del 1993 decisero di impegnarsi nel catalogare efficacemente la loro lista di *bookmarks*, ossia i riferimenti dei siti Internet più interessanti. Il *core set* di questi siti comprendeva circa 200 siti che vennero organizzati in una lista di *links* suddivisa per categorie. Ogni sito (e quindi non solo la singola pagina) veniva descritto sinteticamente ma esaustivamente. Quando una categoria diventava troppo grande, si creavano delle sottocategorie. La tentazione di utilizzare dei programmi che catalogassero in modo automatico i siti fu molto forte, ma continuarono a svolgere questo lavoro manualmente; questa scelta che, come accennato, tuttora prevale è stata probabilmente determinante per il successo di *Yahoo!* ed ha permesso di sviluppare un archivio completo e ben strutturato. In seguito *Yahoo!* è

diventato di proprietà di una compagnia commerciale, evento che ha dato origine a non poche polemiche. Infatti, c'è un lato molto discusso di Yahoo! e cioè il criterio con il quale inserisce i siti nel suo archivio. Rispetto agli altri principali motori di ricerca, è praticamente l'unico che non garantisce l'inserimento, anzi la maggior parte delle volte non lo effettua affatto (circa il 77% delle richieste di inserimento non viene accolto). Questa discriminazione non sembra essere legata a dei criteri qualitativi nell'analisi dei siti, ma sembra dipendere dal fatto che le richieste che pervengono a Yahoo! sono gestite da persone (surfer) e non da sistemi automatici come per gli altri motori di ricerca; se un numero eccessivo di richieste perviene al sistema, molte di queste, inevitabilmente, vengono cestinate. *Yahoo!* è comunque una vetrina molto utilizzata e ambita. Alla fine degli anni '90 Yahoo ha subito un'evoluzione che lo ha portato a creare versioni in italiano, tedesco, francese, spagnolo, giapponese e coreano e dal punto di vista commerciale ha introdotto il submit del sito a pagamento e la portalizzazione.

Yahoo! presenta molti dei limiti delle Web directories, soprattutto la scarsa copertura. Per offrire un servizio migliore *Yahoo!* ha scelto *Google* come partner e come motore di *back end*. In questo modo i risultati di *Google* si fondono con le categorie di *Yahoo!*: il database frutto del crawling di *Google* è integrato nella directory. L'effetto della collaborazione fra questi due colossi ha reso i listing di *Yahoo!* più simili a quelli di *Google*.

4.1.2 *Open Directory Project*

Verso la metà degli anni '90 *Yahoo!* classificava una porzione di Web di dimensione sicuramente inferiore rispetto ai diretti concorrenti quali *Excite, Lycos* o *AltaVista*, ma la sua tassonomia costituiva per milioni di utenti una corsia preferenziale per accedere alle informazioni pubblicate sul Web. Alla fine degli anni '90 *Yahoo!* era ormai diventato un colosso dell'*Information Technology* e come tale aveva stretto importanti *partnership* commerciali. *Yahoo!* stava diventando pertanto un autorevole strumento di marketing per siti di *e-business*. Apparire in una categoria di *Yahoo!* era, ed è tuttora, un risultato ambito, poichè garantisce la visibilità ad una vasta potenziale clientela. L'utilizzo di *Yahoo*! come strumento di marketing ne sbilanciò

però la tassonomia, tanto che certi *topics*, tra i quali *Shopping*, divennero sovra-popolati, a scapito di altri che furono completamente o parzialmente trascurati. Sotto questo punto di vista*, Yahoo!* non sembrava offrire un servizio di buona qualità come quello che forniva agli albori della sua storia e che, fra tutti gli strumenti per la ricerca di informazioni in rete, lo aveva reso noto.

Come [SHE 00] ricorda, un gruppo di ingegneri della *Sun Microsystems*, tra i quali Rich Skrenta e Bob Truel, ispirati dal successo di progetti *open-source* quali GNU e Apache, decisero di affidare a volontari (*editors*) la creazione di una nuova Web directory che venne dapprima chiamata GNUHoo e poi NewHoo. L'appello lanciato da Skrenta e Truel fu sottoscritto da migliaia di volontari, il cui lavoro contribuì a fondare l'*Open Directory Project*, detta anche *ODP* o *DMOZ* [12] (1998).

Per rendere la nuova Web directory una buona alternativa rispetto alle altre già presenti *on-line*, essa doveva soddisfare essenzialmente tre requisiti. Innanzitutto, la tassonomia doveva essere ben progettata, così come lo era la gerarchia di Yahoo! ai suoi albori. Pertanto, non doveva trascurare categorie per favorirne altre, così come ogni topic doveva essere popolato da un set esaustivo ma non ridondante di pagine. Un altro problema riscontrato con Yahoo! era quello dei *dead-link*, ovvero i riferimenti a pagine non più esistenti. Il terzo requisito, quindi, riguardava la manutenzione della Web directory, che doveva essere puntuale ed efficiente. Periodicamente un crawler (*batch mode* crawler*) visita* i siti catalogati per verificare che essi siano ancora attivi, in caso contrario vengono generati opportuni *reports* da sottoporre all'attenzione degli *editors*. Questa soluzione al problema dei *dead-link* è estremamente efficace. Infatti, la percentuale di collegamenti non più esistenti fra tutti gli iperlink nelle categorie è inferiore all' 1%.

Per quanto riguarda le caratteristiche tecniche, le categorie di primo livello sono attualmente 16 [1], ognuna delle quali è suddivisa in una *list* di categorie dettagliate. Per ogni categoria è possibile consultare FAQ e articoli per comprendere le linee guida con le quali i contenuti sono stati organizza-

[1] *Arts, Business, Computer Games, Home, Health, Kids and Teens, News, Recreation, Reference, Regional, Shopping, Society, Science, Sport* e *World.* La categoria *Shopping*, ad esempio, viene suddivisa in *Autos, Books ...*

ti. Il problema della indicizzazione multilingua è risolto riportanto per ogni categoria links alle categorie corrispondenti nelle varie lingue.

Un progetto *open-source* come *DMOZ* ha subito varie critiche, ma il lavoro svolto dai volontari è riconosciuto essere di buona qualità anche dai più importanti motori di ricerca, a titolo di esempio è possibile citare *Google*, *HotBot* e *DirectHit*. In particolare, Google include DMOZ nei propri risultati nella sezione ***directory*** e li riorganizza applicando l'algoritmo PageRank. Il vantaggio di Google è infatti quello di presentare i risultati in ordine di rilevanza, piuttosto che in ordine alfabetico come DMOZ attualmente fa. Come Sergey Brin[2] ha dichiarato "*The ODP has very useful information, but it is tedious to browse. So we put our technology on top to make it far easier to get pinpoint results*".

4.1.3 Tools automatici o semiautomatici

La catalogazione dei siti è sicuramente un'attività *time-consuming* se affidata ad operatori umani. L' introduzione di tools automatici o semiautomatici dovrebbe risolvere in parte il problema della scarsa scalabilità dell'attività umana rispetto al numero crescente di pagine e di contenuti pubblicati sul Web, favorendo inoltre il controllo di consistenza e la manutenzione del catalogo.

Categorisation by content. In questo paragrafo verrà descritta una tecnica a mio avviso particolarmente interessante: la *categorisation by context*, complementare alla forse più nota *categorisation by content*. Tramite quest'ultima tecnica, si cercano di ottenere le informazioni necessarie per classificare un documento direttamente dal documento stesso. In letteratura sono state proposte varie implementazioni della *categorisation by content* [SCH 95], alcune delle quali compiono un'analisi linguistica per determinare le porzioni di testo più significative dalle quali estrarre le *features*. Una volta estratte le feature, viene applicato un modello statistico che, pesando le singole features, è in grado di assegnare correttamente un documento ad una o più categorie.

[2]President of *Google*.

La tecnica denominata *categorisation by context* sfrutta alcuni principi utilizzati anche nell'ambito della bibliometrica e delle reti sociali (*social network*). Ad esempio, in bibliometrica è possibile valutare il prestigio di un documento scientifico determinando l'autorevolezza degli articoli citati. Più autorevoli sono, infatti, gli articoli citati, maggiore è l'autorevolezza del documento in esame. D'altro canto, è altrettanto vero che un documento può veder aumentare il proprio prestigio se viene incluso nella bibliografia di articoli a loro volta autorevoli. Un altro principio utilizzato in bibliometrica che è più attinente alla classificazione di documenti è il seguente: se due documenti citano gli stessi articoli allora è ragionevole pensare che gli argomenti trattati siano simili o comunque attinenti.

In sintesi, la *categorisation by context* applicata in ambito Web si pone come obiettivo la classificazione delle pagine senza analizzarne direttamente il contenuto, ma sfruttando i suggerimenti che la natura ipertestuale dei documenti HTML offre. Il tool realizzato da Attardi [ATT 98] è a mio avviso interessante perchè combina un modulo di *spidering* con un algoritmo di classificazione che si interfaccia con un *part of speech tagger* e con WordNet per realizzare un'analisi linguistica.

Il tool ottiene una lista di URL, recupera ogni documento e ne estrae gli iperlinks contenuti. Ad ogni indirizzo individuato viene associata una context phrase, ossia un insieme di stringhe che possono aiutare a determinare l'argomento trattato dal riferimento. Le stringhe considerate significative includono, tra l'altro, il titolo della pagina in cui l'URL compare, nonchè l'eventuale testo racchiuso fra i due tag ancora <A> e </A> (*anchor text*). Se l'indirizzo compare all'interno di una lista o di una tabella, allora anche il titolo di questi elementi viene incluso nella context phrase.

Dopo aver estratto gli URL e costruito le context phrases, il modulo di classificazione va a popolare un *category tree*. Il *category tree* è una struttura del tutto analoga alle gerarchie tassonomiche precedentemente descritte (più precisamente si tratta di un grafo diretto aciclico), dove ad ogni nodo è associato un titolo, che può essere costituito da una singola parola o da una frase. Lo scopo del classificatore è associare un numero di pesi pari al numero di categorie contenute nell'albero ad ogni URL estratto. L'assegnazione di un URL ad una categoria è possibile quando il peso relativo alla voce del

catalogo è maggiore di una certa soglia. E' pertanto prevista la possibilità di classificare un indirizzo sotto più di un nodo, ma mai in due nodi in cui uno è discendente dell'altro.

I pesi relativi ad ogni categoria sono calcolati considerando il matching fra le stringhe identificative dei nodi dell'albero e le context phrases. Dalle context phrases un modulo di preclassificazione estrae le feature che in questo caso sono i nomi e le cosiddette *noun phrases*. In questa fase è un part of speech tagger che riconosce i nomi dai verbi, avverbi e aggettivi. Il confronto fra le noun phrases e i titoli delle categorie viene realizzato grazie al database lessicale WordNet. Questo tipo di tool è interessante perchè implementa implicitamente una tecnica predittiva, nel senso che cerca di classificare il documento prima di averne analizzato effettivamente il contenuto. Un principio simile potrebbe essere applicato anche all'ambito del crawling: fissato un argomento L'architettura del tool è illustrata in Figura 4.1.

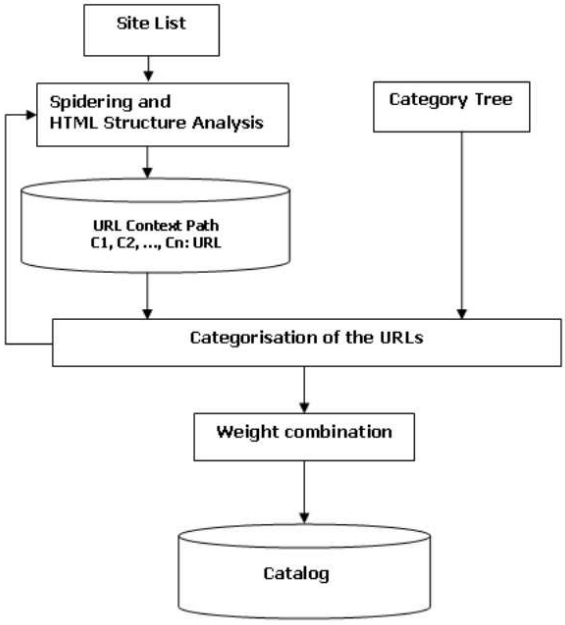

Figura 4.1: Architettura del tool *"Categorisation by Context"*

Automatic resource compiler. Questo tool è stato realizzato da Chakrabarti e altri nell'ambito del *CLEVER Project on Information Retrieval*

dell'IBM Almaden Research Center [CHA 98]. Lo scopo dell'Automatic Resource Compiler (ARC) è costruire automaticamente una lista di risorse interessanti in relazione ad una serie di *topics* abbastanza ampi e ben rappresentati nel Web. Anche in questo caso, come il precedente, la tecnica realizzata si basa su un principio mutuato dalla bibliometrica, più precisamente il principio della co-citazione, nella formulazione matematica sviluppata da Kleinberg [KLE 98].

L'algoritmo di ARC prevede tre fasi principali:

Search-and-growth: dato un *topic* il sistema raccoglie un insieme di pagine in cui compaiono un certo numero di termini. Questa fase di raccolta viene realizzata in collaborazione con un motore di ricerca vero e proprio, al quale viene sottosposto l'elenco di termini come keywords. Ovviamente il motore di ricerca presenta il risultato della query ordinandolo in base ai propri criteri di scoring. Tra i risultati vengono selezionati un numero fissato di documenti che andranno a formare il *root set*. Il root set viene espanso aggiungendo ad esso qualsiasi documento che sia raggiungibile a partire dagli iperlink contenuti nelle pagine appartenenti al root set stesso. In modo analogo, vengono aggiunte anche quelle pagine che contengono link ai documenti del root set. In questo modo sono state recuperate quelle risorse che si trovano *one-link-away* dagli elementi del root set. Fino a questo punto l'algoritmo è del tutto analogo all'algoritmo HITS di Kleinberg [KLE 98], ad eccezione del fatto che la fase di espansione sopra descritta viene reiterata per due volte, fino ad ottenere l'*augmented set* che sarà oggetto delle fasi successive.

Weighting phase: anche questa fase ricalca l'algoritmo di Kleinberg, perchè considera che le pagine possano essere divise in due categorie: *authority* ed *hubs*. Un authority è un documento che contiene informazioni utili circa un topic, mentre un hub è una pagina che contiene un buon numero di link ad altre pagine importanti circa l'argomento di interesse. Analogamente al principio di bibliometrica precedentemente descritto, un "buon" hub contiene riferimenti a "buone" authorities e viceversa. Ad ogni pagina dell'augmented set vengono assegnati due punteggi,

uno di hub ed uno di authority. In questa fase iterativa sono previsti due step, il primo dei quali ricalcola il punteggio di authority di ogni pagina come somma dei punteggi di hub dei documenti che ne contengono i riferimenti. Il secondo passo realizza l'operazione complementare calcolando il punteggio di hub di ogni documento come somma dei punteggi di authority dei documenti puntati. Nell'algoritmo di Kleinberg il peso associato ai punteggi è unitario, mentre ARC prevede che ai punteggi possa essere associato un peso compreso fra 0 e 1. ARC utilizza lo stesso principio della *categorisation by context*, per il quale il testo che circonda la coppia di tag <A> </A> è generalmente descrittivo del contenuto del documento puntato. I pesi vengono pertanto calcolati verificando il numero di match fra i termini descrittivi del topic (gli stessi utilizzati come keywords per il motore di ricerca) e i termini contenuti nella finestra (opportunamente dimensionata) centrata sui tag ancora.

Iteration-and-reporting: lo scopo è, al termine della fase iterativa, determinare i punteggi di hub e authority per ogni pagina dell'augmented set, punteggi che dal punto di vista matematico corrispondono agli autovalori del prodotto fra la matrice dei pesi e della sua trasposta. Ogni passo della fase iterativa consiste, come si è già spiegato, in due step distinti, ognuno dei quali contribuisce a rendere più stringente il legame fra hubs ed authorities. Dagli esperimenti effettuati risulta che generalmente il numero di iterazioni necessarie è un numero finito. Ciò è giustificato anche dalla teoria degli autovalori, in base alla quale in caso di matrici non negative il valore dei punteggi di hub ed authority converge.

4.2 Motori di ricerca veri e propri

Con il termine *Search Engines* si vogliono designare quei *tools* di ricerca che possono essere descritti così come in Figura 4.2.

L'architettura di un motore di ricerca vero e proprio è generalmente un segreto commerciale. Pertanto, la figura non vuole essere sicuramente esau-

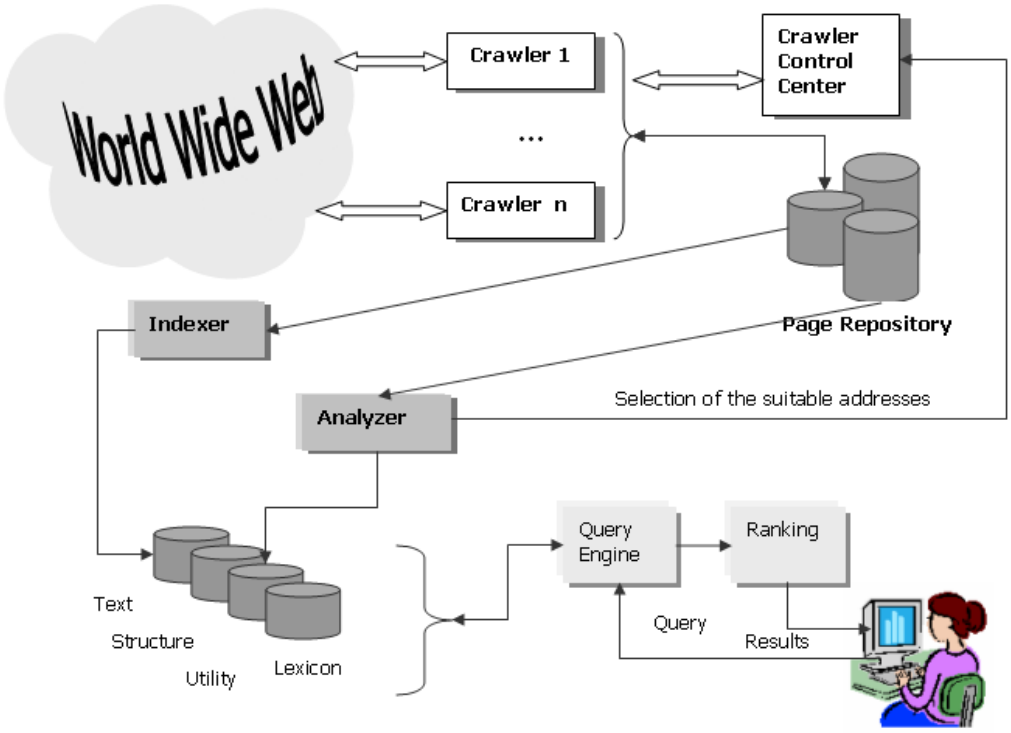

Figura 4.2: Architettura di un motore di ricerca vero e proprio

stiva ma vuole evidenziare ciò che accomuna la maggior parte dei motori di ricerca attualmente disponibili *on-line*.

Nell'architettura di ogni motore è riconoscibile un modulo spider[3] che ha il compito di raccogliere i dati navigando in rete, così come si comporterebbe un operatore umano seguendo gli iperlinks contenuti in ogni pagina. Come illustrato in Figura 4.2, il modulo spider è a sua volta formato da un *Crawler Control Center (CCC)* che si occupa di coordinare un certo numero di crawler veri e propri, a seconda del grado di parallelismo. La corretta gestione di questi crawlers è fondamentale in quanto gli attuali motori di ricerca utilizzano un elevato grado di parallelismo. Il CCC assegna ad ogni crawler una lista di indirizzi di documenti da recuperare (*seed set*). Ogni crawler recupera, se possibile, le pagine assegnate e provvede ad estrarne gli iperlink contenuti. Gli URL estratti vengono comunicati al control center, il quale si preoccuperà di assegnare il compito del loro reperimento al crawler o ai crawlers più idonei. Mercator [HEY 99] fornisce uno dei pochi esempi dell'architettura di un crawler disponibili in letteratura (Figura 4.3).

Per quanto riguarda, invece, il documento vero e proprio, i singoli crawler interagiscono un un *Page Repository*, nel quale i documenti verranno opportunamente memorizzati.

Lo scopo di un motore di ricerca di tipo general purpose è quello di rispondere ad ogni possibile query che gli viene soyyoposta. Per ottimizzare l'interazione con il modulo del *Query Engine,* ogni documento raccolto nel *Page Repository* viene indicizzato secondo alcuni criteri. Anche in questo caso si tratta di criteri di indicizzazione proprietari. In generale, l'attività del modulo *indexer* prevede una fase preliminare in cui da ogni documento vengono estratti i termini contenuti. Nella fase successiva l'indexer costruisce una struttura simile ad una look-up table, in cui ad ogni URL recuperato

[3]Uno spider è, secondo la definizione di Cheong [CHE 96]:

> ″a software program that traverse the World Wide Web information space by following hypertext links and retrieving Web documents by standard HTTP protocol″.

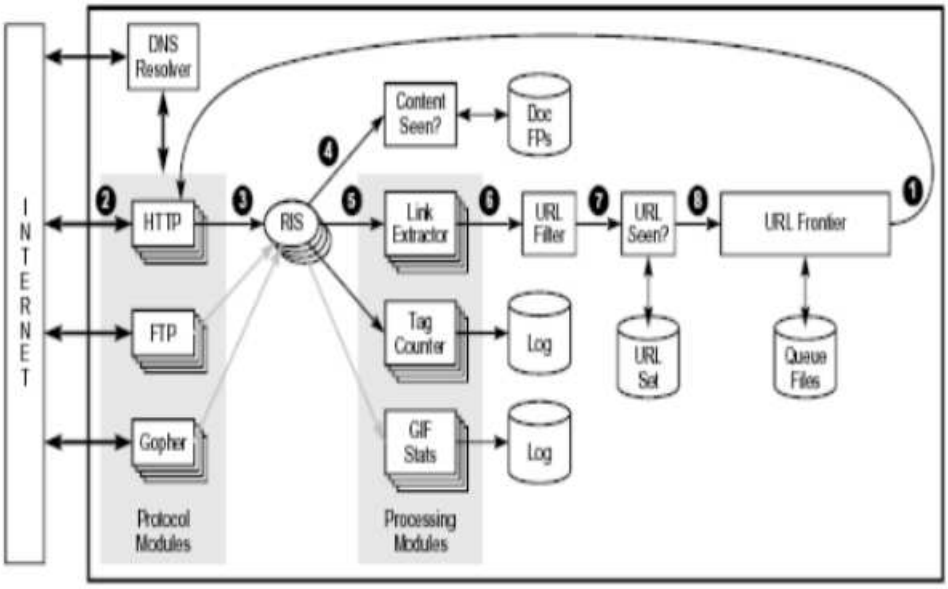

Figura 4.3: Architettura di Mercator

viene associata la lista di termini contenuti. L'indice basato sul contenuto testuale non è generalmente la sola struttura costruita, infatti molti motori di ricerca affiancano a quello che viene chiamato *text indexer* un altro modulo detto *collection analysis module*, il quale si preoccupa di fornire un rapido accesso a certe categorie di documenti, quali pagine di una certa lunghezza, di una certa importanza, con un certo numero di immagini contenute all'interno.

Come già accennato, il *Query Engine* ha la funzione di ricevere le richieste degli utente e di rispondere restituendo gli indirizzi delle pagine che soddisfano i criteri di ricerca specificati. Generalmente l'elenco degli indirizzi delle pagine viene arricchito con informazioni che possono essere utili all'utente. Ad esempio Google illustra per ogni URL restituito un indicazione circa il formato (HTML, PDF, PPT, etc...) e un insieme di frasi estratto dal contenuto della pagina in cui le keywords inserite nella query vengono evidenziate in grassetto. Anche il modo con il quale i risultati vengono ordinati costituisce un aspetto molto importante. Infatti, anche il *ranking module* è spesso un segreto commerciale. Le tecniche di scoring più tradizionali utilizzano una misura di similarità tra il testo della keywords ed il testo dei documenti raccolti nella collezione del *Page Repository*, mentre quelle più avanzate

adattano altre tecniche di IR al contesto del Web, realizzando implicitamente anche l'*information filtering* ed, in certi casi anche un *topic distillation*. Nei paragrafi seguenti verranno analizzati i singoli moduli in dettaglio.

CRAWLER MODULE. Come Cho e Garcia-Molina illustrano [ARA 01], il modulo di crawling recupera le pagine dal Web[4] e le predispone per una successiva analisi dell'*indexer* o del *collection analysis module*. Per quanto riguarda le strutture dati, il crawler gestisce tramite una coda sia gli URL ricevuti dal CCC, sia quelli estratti dai documenti recuperati. Le difficoltà di progettazione di un modulo di crawling sono relative a tre importanti aspetti, uniti alla necessità di impiegare il minor numero di risorse possibili [5].

Poichè una sessione di crawling in grado di raccogliere tutti i documenti pubblicati *on-line* è impossibile da ottenere, sia perchè il tasso di crescita del Web e la volatilità dei suoi contenuti rendono ben presto la collezione di pagine recuperate non aggiornata, sia perché una parte di Web non é esplorabile (deep Web o hidden Web). Inoltre, per risparmiare risorse di rete, sarebbe opportuno che un crawler raggiunga le pagine ritenute importanti prima delle altre. Per questi motivi, oramai molti crawler adottano strategie di esplorazione del grafo più complesse rispetto alla tradizionale *breadth-first*. Alcuni autori, tra i quali Garcia-Molina [ARA 01], propongono due metodi possibili per definire una misura di qualità del crawler:

Crawl & Stop: il crawler inizia l'esplorazione del Web a partire da un elenco di indirizzi P_o e termina dopo aver visitato un numero fissato K di pagine, corrispondenti al numero massimo di pagine che è possibile recuperare in una singola sessione. Un crawler ideale, che implementi una strategia di esplorazione del grafo guidata dalla qualità delle pagine, dovrebbe visitare in ordine le pagine R_1,R_2,...,R_k, dove R_1 è la pagina con il punteggio di rilevanza maggiore e così via (Le pagine R_1,R_2,...,R_k sono dette *hot pages*) . Fra le K pagine raccolte da un

[4] I crawler sono utilizzati anche nell'ambito del Web caching.

[5] L'attività di crawling tende infatti a generare un traffico anomalo rispetto alle normali condizioni di funzionamento della rete, pertanto il comportamento di uno spider dovrebbe attenersi il più possibile alle linee guida dell'etica dei robots [5].

crawler non ideale, invece, vi saranno solo $M \leq K$ pagine con un punteggio maggiore od uguale a quello di R_k. La performance del crawler viene calcolata come:

$$PF = (M \cdot 100)/K. \tag{4.1}$$

Ovviamente la performance di un crawler ideale è pari al 100%. Per quanto riguarda, invece, un crawler che visiti le pagine in maniera random, è possibile che un certo documento venga recuperato più volte. In questo caso la performance viene definita come:

$$PF = (K \cdot 100)/T \tag{4.2}$$

dove T rappresenta il numero totale di pagine nel Web. Ogni pagina visitata ha la probabilità K/T di essere una hot page. Al termine di una sessione di crawling di tipo crawl and stop il numero di hot pages recuperate è, dal punto di vista statistico, pari a K^2/T.

Crawl & Stop with Threshold: Anche in questo caso il crawler visita K pagine, mentre viene definito un parametro I di importanza. Le pagine con un punteggio maggiore di I sono dette *hot pages*, per ipotesi in numero di H. La performance del crawler è data dalla percentuale delle H hot pages che sono state visitate durante la sessione. Se K è minore di H, allora un crawler ideale avrà una performance di

$$PF = (100 \cdot K)/H \tag{4.3}$$

. Se K è maggiore di H la performance di un crawler ideale sarà pari al 100%. Invece, per un crawler che visiti le pagine random, allora il numero di pagine hot recuperate durante la sessione è pari a $(H/T) \cdot K$, con una performance calcolabile come $(K \cdot 100)/T$, che sarà pari al 100% solo se le T pagine vengono tutte visitate.

Un altro parametro che influesce sulla qualità del servizio offerto dal motore di ricerca è la freschezza (*freshness)* della collezione dei documenti. Infatti, una volta recuperate le pagine durante una sessione di crawling, queste devono essere mantenute il più possibile aggiornate. La definizione di freshness comunemente adottata è la seguente:

Definizione 3 *Sia S=$\{e_1,...,e_N\}$ una collezione di N pagine. La freshness di una pagina della collezione e_i ad un certo tempo t è*

$$F(e_i;t) \doteq \begin{cases} 1 & se\,la\,pagina\,e_i\,risulta\,aggionata\,al\,tempo\,t \\ 0 & \text{altrimenti} \end{cases} \tag{4.4}$$

Definizione 4 *Si definisce freshness di una collezione S al tempo t il valor medio della freshness dei documenti contenuti in S.*

$$F(S;t) = \frac{1}{N}\sum F(e_i;t) \tag{4.5}$$

La misura complementare a quella di freshness è quella di *age.*

Definizione 5 *L'età (age) di una pagina e_i al tempo t è*

$$A(e_i;t) \doteq \begin{cases} 0 & se\,la\,pagina\,e_i\,risulta\,aggionata\,al\,tempo\,t \\ \text{t} & \text{altrimenti} \end{cases} \tag{4.6}$$

Analogamente, l'età di una collezione S è

$$A(S;t) = \frac{1}{N}\sum A(e_i;t) \tag{4.7}$$

Considerando un periodo di tempo sufficientemente ampio,

Definizione 6 *Il tempo medio di freshness di una pagina e_i e il tempo medio di freshness di una collezione S si definiscono rispettivamente come:*

$$\bar{F}(e_i) = \lim_{t\to\infty} \frac{1}{t}\int_0^t \bar{F}(e_i;t)dt \tag{4.8}$$

$$\bar{F}(S) = \lim \frac{1}{t}\int_0^t \bar{F}(S;t)dt \tag{4.9}$$

Le politiche di refresh sono essenzialmente due:

Uniform refresh policy: il crawler rivisita le pagine della collezione con una frequenza fissata *f*. La frequenza con cui rivisitare le pagine dovrebbe essere commisurata alla frequenza con la quale le pagine cambiano.

Proportional refresh policy: più spesso una pagina cambia, più spesso il crawler la rivisita. Supponendo che λ_i sia la frequenza con cui una pagina e_i cambia e che f_i sia la frequenza con la quale il crawler rivisita la pagina e_i. Il rapporto fra λ_ie f_i dovrebbe essere lo stesso per ogni i, ciò significa che se una pagina cambia 10 volte più di un'altra pagina, allora la prima dovrebbe essere visitata 10 volte più spesso rispetto alla seconda. Per implementare questa politica, però, il crawler deve stimare la frequenza con cui ogni pagina cambia. Tale stima può essere ricavata analizzando la storia della pagina sulla base di sessioni di crawling successive [CHO 00b, CHO 00a]. Ad esempio, se un crawler visita una pagina $p_1$ogni giorno per un mese e rileva 10 cambiamenti, si può stimare una frequenza λ_1 pari a 1/3.

La frequenza degli update del *page repository* permette di distinguere i crawler progettati per funzionare in modalità *batch* da quelli di tipo *steady*. Un crawler di tipo batch esplora la rete con una cadenza predefinita, ad esempio una volta al mese. Una sessione di crawling termina quando non sono raggiungibili nuove pagine oppure quando il numero fissato di pagine viene recuperato. Al contrario, un crawler di tipo steady è perennemente in esecuzione. La progettazione e la gestione di un crawler di quest'ultimo tipo è molto più complicata e implica un'efficiente organizzazione del page repository.

STORAGE SYSTEM. L'insieme delle pagine Web raccolte durante una sessione di crawling viene memorizzato nel *Page Repository*. Le funzioni svolte da un page repository sono molto simili a quelle di un database system, ma data la natura particolare degli oggetti da memorizzare e il contesto al quale deve essere applicato, un repository dovrebbe soddisfare i seguenti requisiti:

Scalabilità. Il repository dovrebbe essere in grado di rispondere alla crescita del Web. Spesso il page repository è uno storage system distribuito fra cluster di workstation e dischi. Inoltre, data la necessità di mantenere aggiornata la collezione dei documenti, lo spazio occupato dalle versioni obsolete delle pagine deve essere riorganizzato e reso disponibile per le nuove versioni, anche se alcuni storage system mantengono contemporaneamente più versioni di una stessa pagina.

Modalità di accesso differenziate. Vi sono tre moduli che interagiscono con il page repository: il crawler, l'indexer e il query engine. Mentre il crawler e l'indexer (incluso anche il collection analysis module) necessitano di un accesso di tipo *stream*, per il query engine è più efficiente un metodo di accesso *random*, tramite il quale una pagina può essere recuperata velocemente tramite il suo identificatore unico.

Politiche di distribuzione delle pagine. Come diretta conseguenza della scalabilità, i moderni page repository sono in realtà sistemi distribuiti, formati da un certo numero di nodi. Se ogni nodo ha la stessa importanza e viene pertanto considerato al pari di tutti gli altri, si dice che la politica di distribuzione delle pagine è di tipo uniforme (uniform distribution policy). Ciò significa che, data una pagina, essa potrebbe essere assegnata per la memorizzazione ad uno qualsiasi dei nodi. Al contrario, quando una pagina viene assegnata ad un nodo in base ad un criterio come, ad esempio, l'identificatore della pagina, non si tratta di una distribuzione uniforme. In questo caso, infatti, si parla di distibuzione *hash.*

Organizzazione del nodo. All'interno di un singolo nodo, le operazioni di inserimento ed eliminazione delle pagine sono sicuramente le più caratterizzanti, assieme ai metodi di accesso random e stream. Il metodo con il quale le pagine sono organizzate influisce sicuramente sull'efficienza delle citate operazioni. Ad esempio, in una organizzazione hash il dispositivo di memorizzazione viene suddiviso in un certo numero di buckets di dimensione limitata. Le pagine vengono assegnate ai vari buckets in base al valore di una chiave, ad esempio l'identificatore unico

della pagina. Questo tipo di organizzazione è però svantaggiosa sia per quanto riguarda il metodo di accesso stream e le operazioni di inserimento di nuove pagine. Al contrario, in una organizzazione del disco in cui le nuove pagine vengono aggiunte in coda alle altre (*log-structured organization*) le operazioni di inserimento sono sicuramente favorite. Per quanto riguarda l'accesso random, accanto alla coda delle pagine viene mantenuto un indice B-tree. Altri sistemi, invece, basano la propria organizzazione su un sistema ibrido, in parte hash ed in parte log. Il supporto di memorizzazione viene suddiviso logicamente in *extents* di dimensione maggiore rispetto ai buckets hash. Le pagine vengono assegnate ai vari extents tramite una funzione di hash. La gestione di ogni extent è realizzata in modalità log.

Strategie di update. Se il page repository deve interagire con un crawler di tipo batch, allora riceverà nuovi dati per un periodo limitato di tempo. Al contrario, se il crawler è di tipo steady il repository dovrà essere in grado di ricevere dati in modo pressochè continuo. E' dimostrato ([CHO 00a]) che le politiche di update della collezione di documenti non possono essere implementate senza considerare le modalità di funzionamento dei crawler. Si parla di *in-place update* se le pagine ottenute dal crawler sono inserite direttamente, rimpiazzando eventualmente le vecchie versioni con quelle nuove. Con la tecnica di *shadowing*, invece, le nuove versioni sono memorizzate separatamente rispetto alle vecchie e solo in uno step successivo l'intera collezione verrà completamente aggiornata. Generalmente, i page repository che adottano una strategia di shadowing update distinguono i nodi in *read nodes* e in *update nodes*. I read nodes sono utilizzati per l'accesso random e stream, mentre l'inserimento viene effettuato negli update nodes. In questo modo si realizza una perfetta separazione fra le operazioni di accesso e quelle di inserimento, con lo svantaggio, però, di diminuire il grado di freshness dell'intera collezione.

INDEXING MODULE. I dati contenuti nel page repository vengono analizzati da un modulo indexer e, nei motori di ricerca più avanzati, anche

da un collection analysis module. L'indexer costruisce tradizionalmente due strutture: il *text index* (o *content index*) e il *link index*. Entrambi gli indici sono fondamentali per fornire un adeguato supporto al query engine, in particolare il text index rappresenta l'applicazione al contesto del Web dei tradizionali strumenti di *Information Retrieval* e costituisce un parametro fondamentale per determinare quali documenti siano rilevanti per una query. Dal punto di vista delle strutture dati, il text index viene generalmente realizzato come un *inverted index*. Un inverted index è un insieme di *inverted list*, una per ogni termine appartenente ad un lessico selezionato (*lexicon*). Tramite l'inverted list è ricostruibile l'insieme delle pagine nelle quali il termine compare, perchè per ogni termine viene ricavato un elenco di *locations*. Una location (o location entry) indica il documento in cui un termine compare e la posizione di tale termine all'interno del documento. Alcuni algoritmi di ranking, tra i quali anche quello di Google [BRI 98], calcolano la rilevanza di un documento rispetto ad una query data pesando differentemente le location. Ad esempio, se un termine compare in un titolo o con un font grassetto o corsivo è presumibile che tale termine possa assumere, all'interno di un documento, una maggior importanza.

Anche il text index risente dello storage system distribuito. A questo riguardo, Ribeiro-Nieto e Barbosa [RIB 98] illustrano due diverse strategie di gestione dell'indice. La prima soluzione è detta *local inverted file*, in cui ogni nodo ha la gestione di una frazione di pagine dell'intera collezione. Ciò determina la necessità da parte del query engine di raccogliere le inverted list da tutti i nodi. Al contrario, la politica di gestione del *global inverted file* prevede che la suddivisione avvenga sulla base dei termini del lessico e non in base ai documenti.

Un altro indice costruito dal modulo indexer è il *link index,* che rappresenta sinteticamente la topologia del Web Graph. Anche questo indice è particolarmente importante perchè alcuni algoritmi di ranking, tra cui PageRank [PAG 98, KAM 03], HITS [KLE 98]e CLEVER[CHA 98] determinano la rilevanza dei documenti in relazione ad una query anche sulla base di un'analisi di connettività. Accanto al link index, alcuni motori prevedono la costruzione di utility index da parte del collection analysis module. In letteratura, però, sono stati descritti pochi esempi di utility index, perchè queste

strutture sono parte integrante dei moduli di ranking, i quali sono oggetto di segreti commerciali. Un possibile impiego di un utility index è quello descritto in [ARA 01], dove un motore di ricerca offre un servizio di ricerca limitato all'interno di un dominio. In questo caso l'utility index contiene la mappa di ogni sito da cui sono state prelevate pagine per la collezione, in modo che, escludendo i domini non di interesse, la ricerca sia effettuata in un tempo inferiore.

RANKING SYSTEM. Le query poste ad una search interface contengono generalmente un numero variabile di keywords e il motore di ricerca risponde estraendo dalle proprie strutture dati un elenco di documenti che sembrano essere di maggiore rilevanza (*ranking*).

I motori di ricerca attuali hanno sviluppato tecniche di ranking che integrano tecniche tradizionali di Information Retrieval con altre di *Data Mining on Link Structure* e *Prestige Analysis*. Infatti, le tradizionali tecniche di Information Retrieval sono insufficienti per realizzare una funzione di ranking efficace, in primo luogo perchè la ricerca per keywords permette di discriminare documenti che le contengono da documenti che non le contengono, non documenti di buona qualità da documenti di bassa qualità. La scarsa qualità dei documenti può anche dipendere sia da tentativi di spamming (per cui in una pagina vengono aggiunti volutamente termini in modo da farne aumentare artificialmente il ranking), sia dal fatto che le pagine Web non sono sempre sufficientemente autodescrittive. Ad esempio, la pagina della search interface di Google non contiene i termini "search engine" e nonostante ciò si tratta del più noto e prestigioso motore di ricerca. Quest'ultima osservazione è alla base di una serie di studi circa la possibilità di individuare risorse prestigiose non più sulla base del contenuto. Il primo di questi studi è sicuramente quello di Kleinberg, il quale ha proposto l'algoritmo HITS ([KLE 98]).

Kleinberg osserva che l'insieme dei criteri di rilevanza (*relevance*) utilizzati dai motori di ricerca spesso non coincidono con le esigenze degli operatori umani, per cui un algoritmo per filtrare i risultati di un'interrogazione individuando le risorse "authority" e le risorse "hub", potrebbe migliorare l'efficacia del sistema di ranking. Come premessa fondamentale, Kleinberg suddivide le query in tre categorie fondamentali:

Specific ad esempio, "Quale è la più recente versione di Jade?", query per la quale l'ostacolo maggiore è costituito dal fatto che un numero irrisorio di documenti rispetto al numero totale di pagine pubblicate contengono l'informazione desiderata (*Scarcity problem*).

Broad-Topic ad esempio, "Trovare informazioni su Java", query per la quale esistono centinaia di migliaia di pagine più o meno interessanti, un numero di pagine che rende proibitiva l'analisi da parte di un operatore umano (*Abundance problem* e *Information overload*).

Similar-Page ad esempio, "Trovare pagine simili a java.sun.com", in cui la difficoltà consiste nel definire cosa si intende per similarità fra pagine.

Il problema di determinare le pagine prestigiose (authorities) sussiste nelle broad-topic queries e viene risolto da Kleinberg ipotizzando che nella struttura degli iperlinks contenga in forma latente le informazioni necessarie per determinare un ranking sulla base del prestigio. Infatti, se il creatore di una pagina p inserisce in essa un collegamento verso la pagina q, conferisce implicitamente a quest'ultima un ruolo di authority, quindi, a partire da un insieme di pagine l'analisi del numero di collegamenti verso una pagina (inlinking structure o *in-degree*) è un forte indizio di prestigio.

HITS si articola nelle seguenti fasi:

1. Data una query di tipo broad-topic rappresentata da una stringa σ, viene ricostruito il sottografo del Web nel quale sono inseriti i documenti che soddisfano σ. Al pari del Web Graph, i documenti costituiscono i nodi e gli iperlink i vertici. I nodi del sottografo potrebbero anche essere centinaia di migliaia, con evidente aumento dei tempi di calcolo e comunque il set di documenti in esame potrebbe non contenere affatto pagine prestigiose. Per questi motivi, l'insieme di documenti dovrebbe avere dimensioni relativamente limitate ed essere ricco di pagine rilevanti. HITS raccoglie le pagine che soddisfano i criteri sopracitati interrogando un motore di ricerca tramite la query σ e selezionando i primi t risultati. In questa fase, quindi, HITS si "fida" del ranking proposto dal motore di ricerca.

2. I t documenti costituiscono il *root Set*, che sicuramente contiene i termini indicati nella query ed è di dimensioni limitate, ma non sempre contiene un numero sufficiente di pagine prestigiose. Per questo motivo, questa seconda fase prevede un'estensione del sottografo per includere in esso tutte le pagine che hanno un link verso i documenti del root set e, parallelamente, anche le pagine raggiungibili dal root set.

3. Questa fase consiste nel vero e proprio ranking dei documenti, il quale, nella sua forma più semplice, consiste nel calcolo dell' *in-degree* di ogni pagina del sottografo. Kleinberg, però, raffina questo calcolo osservando che accanto alle pagine authorities è possibile riconoscere anche delle pagine dette hubs, le quali sono caratterizzate da un elevato out-degree, ovvero un alto numero di collegamenti verso altri documenti. Quindi, l'importanza di una pagina come authority può e deve aumentare se viene riferita da hubs che, a loro volta, sono hub prestigiosi, allo stesso modo l'importanza di una pagina come hub deve crescere se riferisce buone authorities (*mutually reinforcing relationship*). HITS è un algoritmo del punto fisso, in cui in un passo preliminare vengono assegnati, ad ogni nodo del sottografo, una coppia di punteggi, uno di authority x ed uno di hub y. La mutually reinforcing relationship viene espressa tramite due diverse operazioni che modificano iterativamente i punteggi di authority e di hub. L'algoritmo può essere sintetizzato nel seguente modo:

Algoritmo 1 *HITS*

Iterate (G, K)
G is a collection of n linked pages
k is a natural number
Let z denote the vector (1,1,1...,1)$\in R^n$
set $x_0 = z$
set $y_0 = z$
Fori=1,2...,k
Calculate new x_i *as the sum of* y_{i-1} *(only suitable pages)*

Calculate new y_i *as the sum of* x_{i-1} *(only suitable pages)*
Normalize both x_i *and* y_i
End for
Filter (G,k,c)
c is a natural number
report as authorities the c pages with largest coordinates x_i
report as hubs the c pages with largest coordinates y_i

Bharat e Henzinger [BHA 98], parallelamente a Chakrabarti [CHA 98], osservano che l'algoritmo HITS spesso non dà buoni risultati per due ragioni fondamentali:

1. secondo Kleinberg, un operatore che inserisce in un documento un link ad una pagina vuole conferire ad essa un ruolo di authority. Spesso, però, le pagine Web sono generate da tools automatici, pertanto alcuni degli embedded URLs sono stati inseriti per pubblicità, piuttosto che per conferire prestigio;

2. spesso nelle broad topic queries i documenti raccolti nel set non sono così rilevanti e se i documenti sono ben interconnessi, le pagine con i migliori punteggi di hubs e di authorities tendono a non rispettare il topic originale.

Il secondo problema è quello che influenza negativamente il risultato di HITS in maniera più sensibile. Se dal set di documenti fosse possibile eliminare quelli che non appartengono al topic di interesse e pesare l'influenza dei nodi sulla base della rilevanza rispetto al topic, allora l'algoritmo HITS potrebbe diventare un efficace strumento di *Topic Distillation*.

4.2.1 *Google*

Il prototipo di Google è stato sviluppato a Stanford da Sergey Brin e Lawrence Page [BRI 98] ed in breve è diventato il più noto e prestigioso motore di ricerca disponibile on-line. Attualmente Google dichiara di indicizzare oltre 3 miliardi di documenti, cosa che gli permette di ottenere la più elevata copertura fra tutti i motori di ricerca (Figura 4.4).

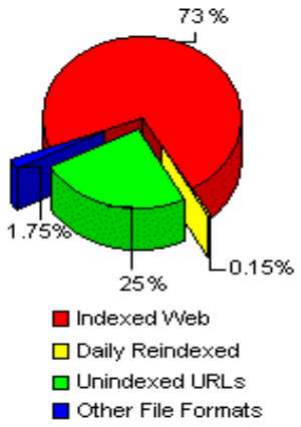

Figura 4.4: Copertura di Google

Analogamente a quanto descritto nel precedente paragrafo, Google ha un'architettura complessa nella quale si possono riconoscere tre moduli fondamentali: crawling, indexing e searching (Figura 4.5).

Il modulo di crawling di Google è un sistema distribuito in cui un URL-Server assolve alle funzioni del Crawler Control Center (CCC), assegnando ad ogni singolo crawler un elenco di URL da recuperare. Ad ogni URL corrisponde una risorsa (una pagina .html, un file .pdf, un'immagine, un video ...), la quale viene memorizzata in un page repository e, per facilitare le successive fasi di analisi, lo store server assegna ad ogni documento un identificatore unico (docID).

Mentre i crawlers assolvono i propri compiti in maniera relativamente semplice e rapida, le funzioni svolte dall'indexer sono sicuramente più complesse. L'interposizione di un page repository permette di rendere indipendente la fase di costruzione degli indici dalla fase di recupero delle risorse.

L'indexer analizza quindi ogni documento contenuto nel page repository dapprima decoprimendolo, poi estraendo una serie di informazioni utili al sistema di ranking. Come la maggior parte dei motori di ricerca, anche Google trasforma i documenti da una forma human-oriented ad una forma più consona per l'applicazione di tecniche di Information Retrieval. In pratica, l'indexer sintetizza ogni documento in un set di record detti word occurrences o hits. In ogni record viene memorizzata una parola e la sua posizione

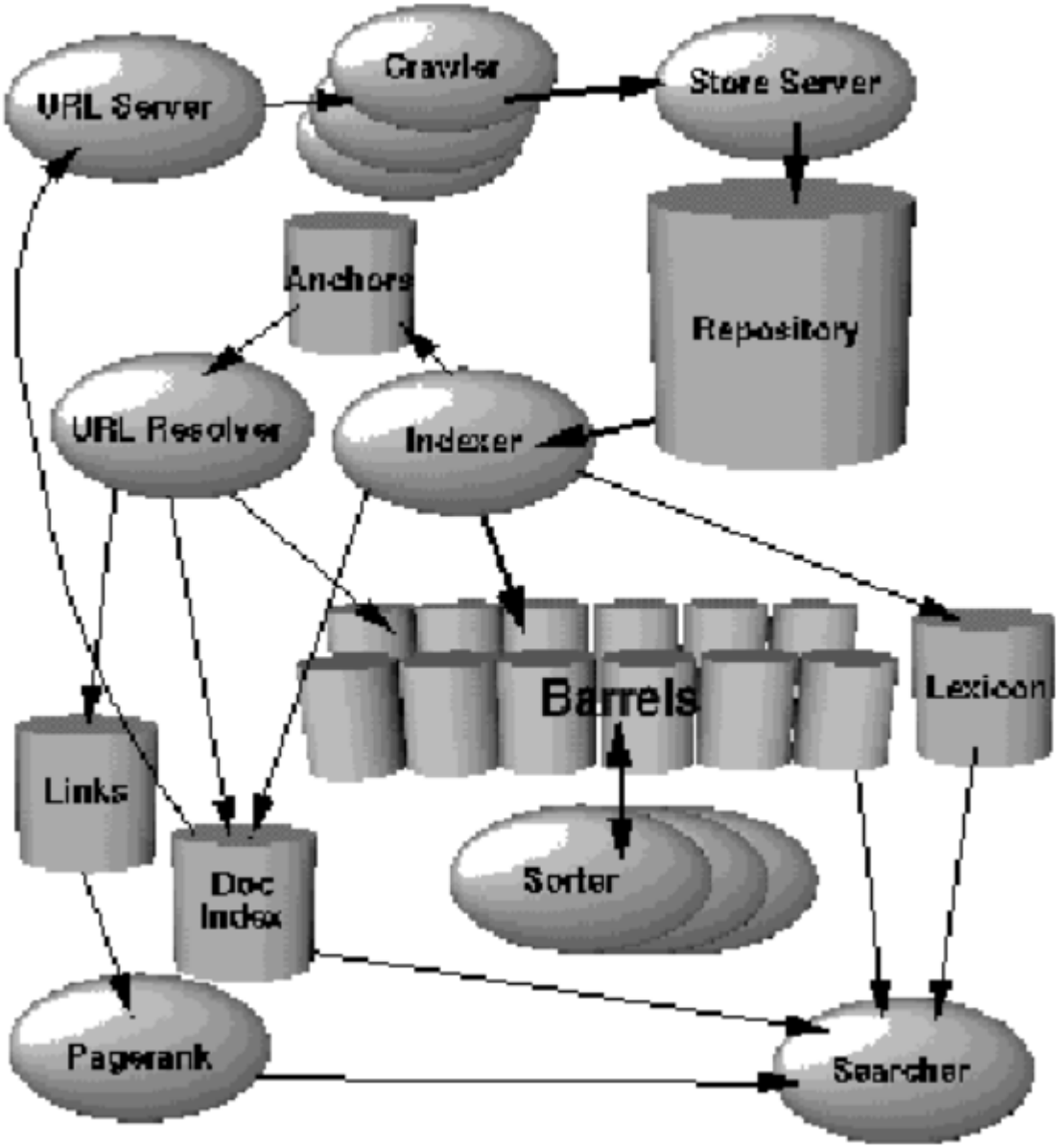

Figura 4.5: Architettura di Google, così come illustrata in [BRI 98].

all'interno del documento, nonchè una misura approssimativa del font con il quale un browser (o un altro applicativo idoneo) dovrebbe mostrarla. Quindi, oltre alla presenza di una parola in un documento, Google valuta anche informazioni circa il layout con la quale essa dovrebbe essere rappresentata da un browser (o un applicativo idoneo). Ciò perchè, mentre i documenti oggetto dell'Information Retrieval tradizionale sono linguisticamente e stilisticamente ben strutturati (si pensi ad esempio a collezioni di articoli di giornali, etc. . .), le risorse pubblicate sul Web perdono parte di queste caratteristiche, a favore di un tipo di comunicazione legato più all'aspetto grafico che ai contenuti.

Ma l'aspetto innovativo di Google nell'ambito dei motori di ricerca risiede nella costruzione di un anchor file. Durante la costruzione dei record delle word occurrences, infatti, l'indexer effettua il parsing dei documenti alla ricerca dei riferimenti verso risorse esterne. Ogni collegamento individuato viene memorizzato in un file, l'anchor file, appunto, assieme al testo che circonda i tag <A> </A> (anchor text). In questo modo Google tiene conto del fatto che la maggior parte delle pagine Web è in realtà poco autodescrittiva, mentre il testo di contorno ad un iperlink descrive sinteticamente il contenuto del documento riferito. Il principio è lo stesso utilizzato da Attardi e da Chakrabarti, inoltre Brin e Page osservano che generalmente l'anchor text esiste anche per quei documenti quali immagini, programmi, file audio che l'indexer non può elaborare direttamente.

Google estrae dai documenti un numero di informazioni maggiore rispetto a tutti gli altri motori di ricerca. Il sistema di ranking integra queste informazioni con i risultati dell'algoritmo PageRank per determinare, fra tutti i documenti recuperati non solo i documenti che soddisfano la query (ossia presenza all'interno del testo dei termini cercati), ma anche i documenti di qualità. Ad esempio, quando Google riceve una query, un parser estrae i termini e recupera i relativi WordIDs, ossia gli identificatori un ici delle parole. L'indexer, infatti, estrae le word occurrences dai documenti e le distribuisce in un certo numero di buffer detti barrels. I Barrels sono mantenuti parzialmente ordinati sulla base dei DocIDs, finchè un modulo sorter non provvede alla costruzione di un inverted index, riordinando il contenuto dei barrels rispetto ai WordIDs.

Quando viene richiesta l'esecuzione di una query, il sistema accede ai barrels tramite gli identificatori unici delle parole; verifica l'esistenza di documenti che contengano tutti i termini indicati e restituisce un elenco di URLs, ai quali corrisponde il valore più elevato di ranking. La funzione di ranking integra due diversi punteggi, il primo derivante da analisi testuali, il secondo calcolate tramite l'algoritmo PageRank.

L'algoritmo PageRank [PAG 98] è simile a HITS perchè si basa sull'intuizione per cui una pagina ha un elevato *rank* se la somma dei ranks dei suoi documenti *backlink* è elevata. PageRank può essere definito nel seguente modo: Sia u una pagina Web e sia F_u il set di pagine a cui u si riferisce. Il set di pagine B_u rappresenta invece le pagine che contengono collegamenti a u, vale a dire le sue pagine *backlink*. Sia N_u=$|F_u|$ il numero di link contenuti in u e sia $c < 1$ un fattore di normalizzazione per mantenere costante il punteggio globale di tutte le pagine.

Un semplice rank può essere definito come l'equazione:

$$R_u = c \cdot \sum_{v \in B_u} \frac{R(v)}{N_v} \tag{4.10}$$

E' possibile costruire una matrice quadrata A in cui sia le righe che le colonne rappresentano le pagine Web. A è una matrice di adiacenza, dove gli elementi indicano se fra la pagina rappresentata in una riga e la pagina rappresentata in una colonna esiste un iperlink di collegamento. Il valore che un elemento assume se esiste l'iperlink è $1/N_u$, 0 in caso contrario. Considerando quindi tutte le pagine Web della collezione, R è un vettore in cui ogni elemento rappresenta il rank della relativa pagina e l'equazione 4.10 può essere riscritta come

$$R = cAR, \tag{4.11}$$

del tutto equivalente ad affermare che R è un'autovettore di A.

Ovviamente questa è una versione semplificata di PageRank, nella quale si può comunque osservare che si tratta di un algoritmo computazionalmente molto oneroso. Per questo motivo l'esecuzione di PageRank avviene normalmente off-line.

4.3 Meta Motori di ricerca

Gli attuali strumenti per la ricerca di informazioni in rete, siano essi web directories o motori di ricerca veri e propri, sono sicuramente strumenti complessi e avanzati che hanno risposto alla crescita del Web aumentando la dimensione delle loro collezioni di documenti, così come mostrato in Figura 4.6 (dati aggiornati al maggio 2003).

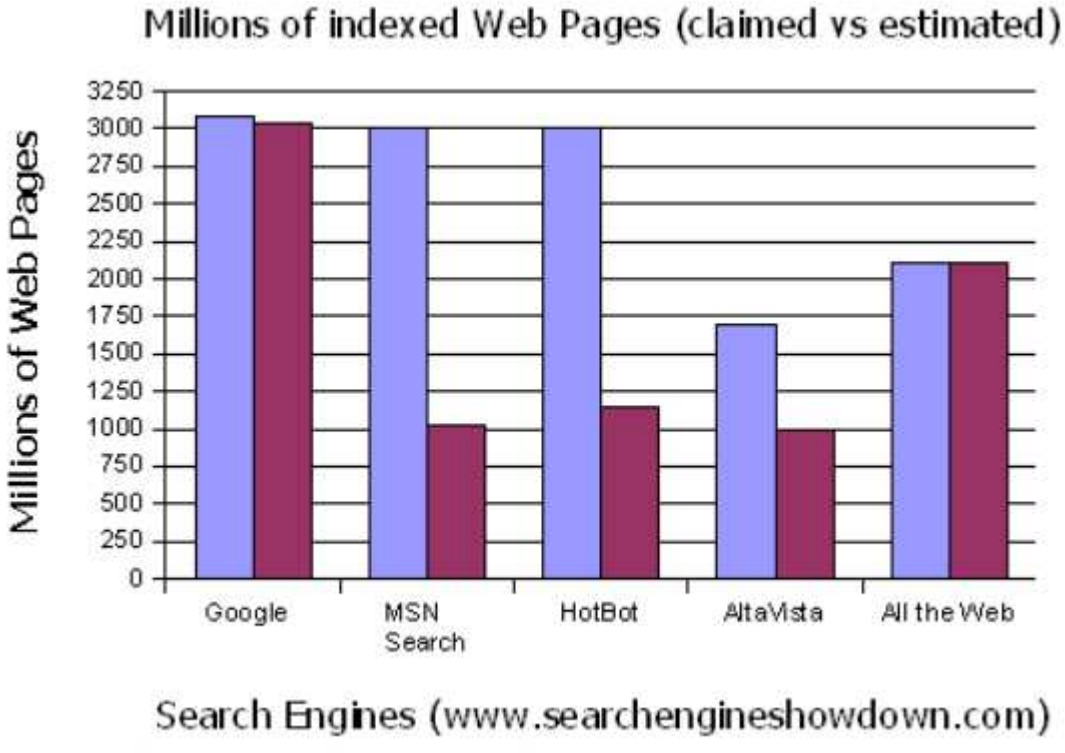

Figura 4.6: Il numero di pagine indicizzate dai più noti motori di ricerca

Definizione 7 *Si definisce copertura (coverage) la percentuale di Web accessibile tramite la search interface dello strumento di ricerca, sia esso un search engine o una web directory.*

Confrontando l'evoluzione della dimensione del Web indicizzato [6] con l'evoluzione del numero di documenti pubblicati, si osserva che anche gli strumenti di ricerca più noti non sono riusciti ad aumentare successivamente la

[6] i.e. accessibile dalla search interface dello strumento di ricerca.

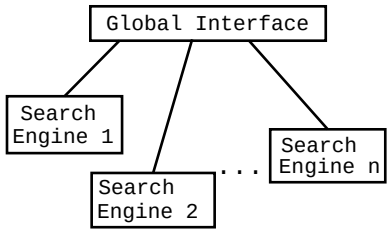

Figura 4.7: Architettura di un semplice metasearch engine.

copertura. Già nel 1998, uno studio di Lawrence e Giles [LAW 99a] dimostrava che la copertura dei principali motori di ricerca, invece di aumentare, diminuiva costantemente, confermando la scarsa scalabilità di questo tipo di strumenti, sia per quanto riguarda le risorse software (crawler, indexer...), sia per quanto riguarda le risorse hardware (dispositivi di memorizzazione ed infrastrutture di comunicazione).

Se invece uno strumento di ricerca potesse accedere ad un insieme di search engines e web directories, allora potrebbe aumentare la dimensione di documenti recuperabili tramite la sua search interface, con un conseguente aumento della copertura del Web. Sistemi in grado di ricevere richieste da parte degli utenti e trasparentemente redirezionare le query verso le search interface di un certo numero di search engines e web directories sono detti *metasearch engines*. Oltre ai possibili vantaggi di copertura del Web, generalmente un metasearch engine non dispone di storage system e di indici per ottimizzarne l'accesso, pertanto non subisce tutti gli svantaggi legati alla scarsa scalabilità. Per quanto riguarda la progettazione di metasearch engines, in letteratura [MEN 02] si parla di un'architettura a due livelli come architettura di base (Figura 4.7).

L'utente pone le proprie query ad una *global interface*, come se si trattasse di una tradizionale search interface. Compito del metasearch engine è interrogare i cosiddetti *component search engines*, i quali, con ogni probabilità accetteranno query in formati diversi e restituiranno i risultati con ordine e modalità differenti. Compito del meta-motore di ricerca è ricevere i documenti selezionati dai vari componenti secondo i propri algoritmi di ranking

ed analizzarli per presentare all'utente quelli che risultano essere i *potentially useful documents*.

Secondo la definizione proposta da [MEN 02],

Definizione 8 *Supponendo di aver definito una funzione che permetta di calcolare il grado di similarità fra un documento ed una query posta dall'utente, si dice che un documento d è potenzialmente interessante (potentially useful) rispetto alla query se soddisfa una delle seguenti condizioni:*

1. *Fissato in m (intero positivo) il numero di documenti da fornire come risultato, un documento d fa parte del risultato se il suo grado di similarità è compreso negli m migliori gradi di similarità.*

2. *Se il grado di similarità accettabile nei risultati è maggiore di un certo valore di soglia prefissato, allora il documento d con un grado di similarità maggiore della soglia può essere incluso nei risultati.*

Nella figura seguente è illustrata l'architettura di riferimento di un semplice meta-motore di ricerca.

Se il numero di motori di ricerca inclusi nell'architettura è sufficientemente basso, allora non è troppo oneroso spedire ogni query ai singoli componenti. Quando però si tratta di centinaia di componenti, allora alcuni di questi possono non fornire documenti interessanti, contribuendo ad aumentare il traffico di rete e ad occupare risorse inutilmente. Pertanto, i metamotori di ricerca cercano di determinare a priori quali componenti interrogare in modo da minimizzare il numero di risorse impiegate e da ottenere il numero prefissato di documenti interessanti. Il problema descritto, noto come *database selection problem,* può essere ad esempio risolto calcolando tramite formule generalmente molto complesse la qualità di ogni repository rispetto ad una query; oppure applicando tecniche di machine learning, sulla base delle quali il meta-motore impara a determinare il set ottimo di database da interrogare, analizzando il risultato di alcune query di esempio.

Dopo aver selezionato a quali componenti sottoporre la query, un modulo detto *query dispatcher* provvede a stabilire le connessioni ed eventualmente a tradurre le richieste ricevute in un formato comprensibile per le varie search interface. Ciò non solo perchè motori diversi potrebbero accettare

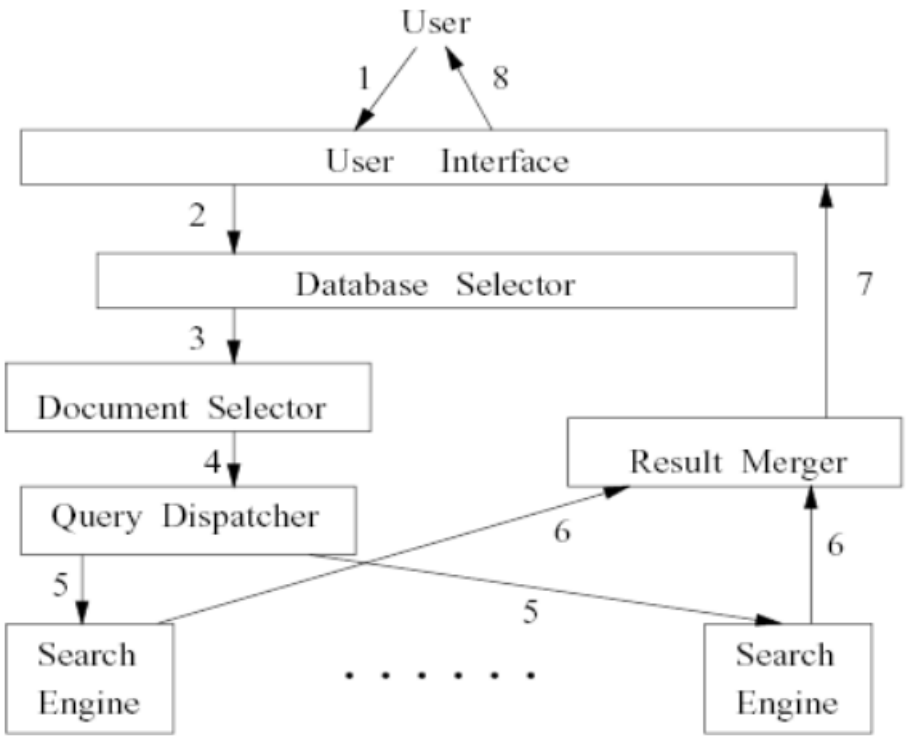

Figura 4.8: Architettura di riferimento di un semplice meta motore di ricerca

input in formato diverso, ma anche perchè dopo aver selezionato i database components, generalmente un modulo detto *document selector* interviene specificando una serie di parametri in relazione ai quali il sistema deve discriminare fra i documenti potentially useful e gli altri. Attenendosi alla definizione, infatti, potrebbe definire un valore di soglia per il grado di similarità o definire il numero massimo di pagine da includere nei risultati. Infine, il *Result merger* è lo strumento che analizza tutti i risultati ottenuti e li filtra in base alle richieste, eliminando dall'elenco i documenti duplicati [BRO 98].

Nei paragrafi seguenti saranno descritti alcuni aspetti dei più noti ed interessanti metasearch engines.

4.3.1 *MetaCrawler*

Selberg ed Etzioni presentano in [SEL 07] l'architettura di un tool di ricerca chiamato *MetaCrawler [10],* la cui architettura è illustrata in Figura 4.9. MetaCrawler è un robot software che presenta all'utente un unica search

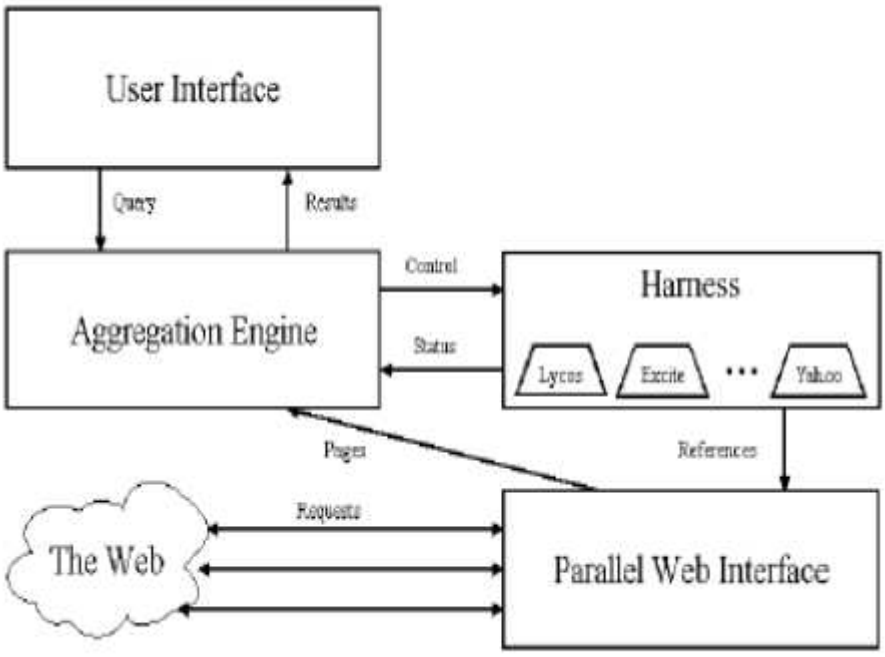

Figura 4.9: Architettura di MetaCrawler

interface tramite la quale interrogare altri motori fra i quali AltaVista e Lycos e si compone di un *Aggregation Engine*, di una *Parallel Web Interface* e di un *Harness*. Le query ricevute tramite l'interfaccia utente vengono analizzate dall'*Aggregation Engine* e tradotte dall'harness nei formati più opportuni. La Parallel Web Interface raccoglie tutti i documenti che soddisfano le queries e li trasmette all'Aggregation Engine, che in questa fase svolge le funzioni del Result merger. Infatti, i risultati ottenuti vengono di nuovo analizzati al fine di eliminare i documenti duplicati e di ordinarli in base ai criteri scelti. Il flusso di esecuzione del sistema è rappresentato in Figura 4.10. MetaCrawler è stato progettato per soddisfare una serie di requisiti, tra i quali l'adattabilità, la portabilità e la scalabilità. Così come possono essere volatili i contenuti delle pagine, sono volatili anche i servizi di ricerca delle informazioni in rete. Infatti, vi sono frequenti variazioni di elementi della search interface, dalle modalità ed opzioni di ricerca fino all'indirizzo dove reperire l'interfaccia stessa. Nei casi estremi, alcuni motori componenti potrebbero terminare la loro attività, mentre potrebbe essere necessario includere nuovi servizi ed è evidente come il modulo harness sia cruciale da questo punto di vista.

4.3.2 *SavvySearch*

SavvySearch ha iniziato la propria attività nel 1995 e si pone come obiettivo quello di fornire un servizio che minimizzando il numero di risorse impiegati, massimizzi il grado di rilevanza dei documenti recuperati. Il trade-off proposto da SavvySearch è determinare, sulla base della query ricevuta e sulla base delle performance dei componenti, il set ottimo di motori di ricerca da interrogare [HOW 97]. Il flusso di controllo di SavvySearch può essere sintetizzato nelle seguenti fasi:

1. l'utente sottopone la query specificando uno o più termini ed eventualmente altri parametri di ricerca, tra i quali il numero di documenti da restituire;

2. SavvySearch decide quali motori interrogare e in che ordine. Determinare il set ottimo di componenti è un'operazione complessa che prevede un'analisi delle risorse disponibili all'istante in cui la query viene posta

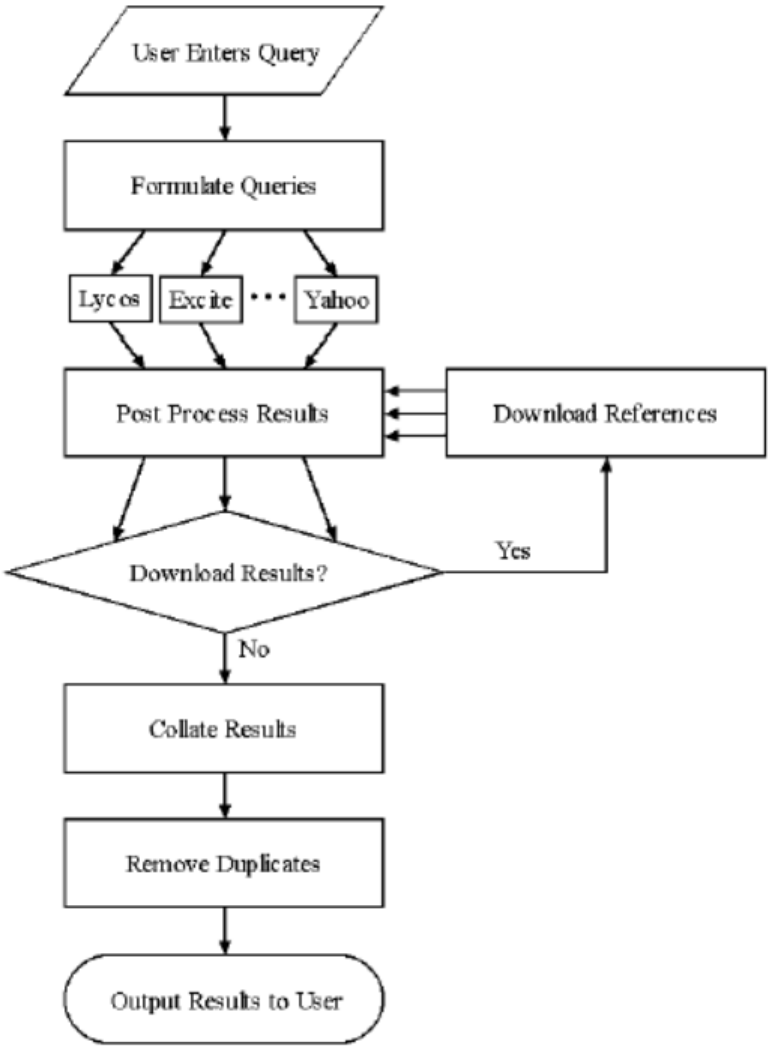

Figura 4.10: Flusso di controllo di MetaCrawler

(traffico di rete e carico di CPU), mentre definire l'ordine in base al quale interrogare i componenti implica utilizzare un criterio di ranking, che per SavvySearch è basato su un *meta-index;*

3. un modulo dispatcher interagisce con i componenti selezionati adattando la query posta dall'utente ai requisiti richiesti dalle varie search interface;

4. SavvySearch mostra i risultati in base ai parametri selezionati nella fase 1.

La peculiarità di SavvySearch è il meta-index, il quale per ogni termine mantiene una lista di *effectiveness values*, uno per ogni motore componente. Gli effectiveness values derivano da un insieme di osservazioni circa la performance dei vari componenti in relazione alle query utente. Per la precisione, gli effectiveness values sono calcolati sulla base del numero di link restituiti in risposta e sulla base di quelli visitati dall'utente. Se un motore componente non restituisce indirizzi per un certo insieme di termini, allora il suo repository non è completo, mentre maggiore è il numero di link restituiti e visitati dall'utente, maggiore è la significatività dei risultati [DRE 97]. Il criterio di ranking dei componenti utilizza le informazioni contenute nel meta-index penalizzando l'aspetto negativo della bassa qualità dei risultati e privilegiando la disponibilità delle risorse (tempo di risposta, carico di lavoro, traffico di rete).

4.3.3 *ProFusion*

ProFusion [4]è un metasearch engine il cui prototipo è stato sviluppato alla Kansas University. La peculiarità di ProFusion è l'approccio di tipo machine learning [GAU 96], in cui sono definite 13 *pre-set categories*[7]. Per ogni categoria viene selezionato un insieme di termini rappresentativi e un set di query tipiche, che in una fase periodica di start-up verranno sottoposte ai vari search engine componenti al fine di testarne le capacità di risposta. La

[7]Science and Engineering, Computer Science, Travel, Medical and Biotechnology, Business and Finace, Social and Religion, Society Law and Government, Animals and Enviroment, History, Recreation and Entertainment, Art, Music, Food.

capacità di risposta cumulativa di un componente (*confidence factor*) viene espressa mediante una formula che considera sia il ranking proprio di ogni motore che la sua *precision.*

Dopo questa fase iniziale, ProFusion può accettare query dall'utente. Per ogni query ricevuta, il sistema provvede a realizzare un mapping tra la singola query e le categorie predefinite. Ad una query utente possono pertanto corrispondere una o più categorie, e verrà "girata" a quei componenti che sulla base dei confidence factors sembrano fornire la migliore capacità di risposta. Anche ProFusion adotta un proprio criterio di re-ranking dei risultati ottenuti che calcola i nuovi scores pesando opportunamente i punteggi assegnati dai singoli componenti con i relativi fattori di confidenza cumulativi.

La versione corrente di ProFusion è un sistema multiagente, così come illustrato in Figura 4.11.

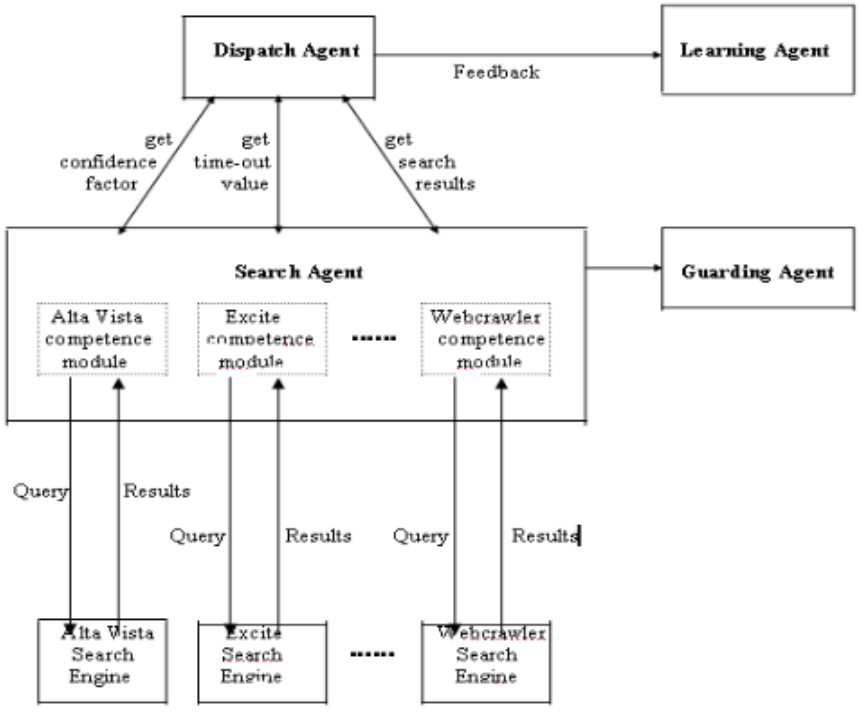

Figura 4.11: Architettura di ProFusion

Il *dispatch agent* ha il compito di interagire con l'utente, con il *search agent* e con il *learning agent.* Il search agent provvede invece ad interagi-

re con i singoli motori componenti, valutandone i fattori di confidenza. In questa versione di ProFusion [GAU 99], viene introdotto come un nuovo parametro di selezione dei componenti il tempo di risposta offerto, che non può superare un time-out fissato. Analogamente a SavvySearch [DRE 97], i tempi di risposta vengono periodicamente monitorati. Nel caso di ProFusion, tale compito è assegnato al *guardian agent*, il quale è anche in grado di individuare i componenti che, per vari motivi, interrompono il servizio. Per evitare di spedire query ad elementi inattivi e pertanto impossibilitati a rispondere, il guardian agent segnala i componenti *off* al dispatch agent.

4.4 Conclusioni

Gli strumenti presentati si propongono come obiettivo quello di rispondere a tutte le possibili query formulate da un utente, siano esse query basate su keywords o query implicite poste durante la navigazione in cataloghi. Uno dei limiti fondamentali di questi strumenti risiede proprio nella modalità con cui le query vengono poste: le keywords sono sicuramente insufficienti per esprimere le caratteristiche dei documenti desiderati, mentre i cataloghi non contengono tutte le possibili informazioni che un utente vorrebbe avere a disposizione.

Nonostante la crescita del numero di pagine indicizzate, la copertura offerta è ancora insufficiente, soprattutto per quanto riguarda le pagine il cui contenuto cambia frequentemente. In questo senso potrebbero essere utili i meta motori di ricerca, proprio in virtù di poter interrogare contemporaneamente più motori di ricerca.

Capitolo 5

Strumenti di ricerca specializzati

Gli strumenti di ricerca illustrati nel capitolo precedente hanno come obiettivo quello di rispondere a tutte le possibili query formulate da un utente, siano esse query basate su keywords o query implicite poste durante la navigazione all'interno di cataloghi. Accanto a queste soluzioni, coesistono una serie di strumenti che adottano un approccio special-purpose al problema della ricerca di informazioni nel Web.

Ad esempio, alcuni di questi strumenti si propongono di fornire un servizio efficiente ed esaustivo in relazione ad un determinato settore, mentre altri sono creati per guidare l'utente durante lo svolgimento di attività complesse o critiche. Mentre i primi sono detti motori di ricerca specializzati [Par.5.1], i secondi si configurano come *personal assistants* [Par.5.2].

5.1 Motori di ricerca specializzati

5.1.1 *CiteSeer*

CiteSeer [15] è un sistema avanzato per la ricerca di articoli scientifici disponibili on-line. Al pari dei tradizionali *citation indexing systems,* CiteSeer consente, dato un articolo, di recuperare sia i documenti in esso citati (navigazione backward), sia quelli nei quali compare come fonte (navigazione

forward). Il vantaggio di CiteSeer risiede nel fatto che gli articoli accessibili non provengono da una singola pubblicazione settoriale, ma dall'intero Web; pertanto la collezione di documenti è più ampia ed aggiornata ed il ranking dei vari articoli è sicuramente più accurato.

Per quanto riguarda la ricerca dei documenti, CiteSeer sfrutta una serie di strategie complementari che comprendono richieste di informazioni ai motori di ricerca general purpose e l'adozione di euristiche per localizzare pagine in cui potrebbero essere contenuti riferimenti a risorse in formato elettronico (.ps, .pdf, .ps.gz, ...)[1].

Dopo il recupero dei documenti e la conversione degli stessi in formato testo, il sistema effettua un'analisi per verificare che si tratti di articoli scientifici. Per convenzione questo tipo di documenti deve presentare una serie di caratteristiche quali titolo, autori, eventuali abstract o paragrafi introdutti e soprattutto una sezione dedicata alla bibliografia (references). Quindi, di ogni articolo che soddisfa i requisiti vengono estratti, tramite parsing, gli elementi sopra citati e viene indicizzato l'intero testo. In questa fase l'attività critica è il parsing delle citazioni, in primo luogo perchè si tratta di parsing del linguaggio naturale, in secondo luogo perchè possono essere espresse tramite stili completamente diversi l'uno dall'altro.

La validità di CiteSeer come strumento automatico di citation indexing risiede proprio nella capacità di individuare i riferimenti ad uno stesso articolo, anche se le citazioni sono espresse in forme e termini diversi. Questo problema viene risolto dagli autori [LEE 98a, BOL 98] "normalizzando" le citazioni, convertendole cioè in una base comune. Data una citazione,ad esempio, essa viene convertita in lettere minuscole, vengono espanse le abbreviazioni come proc. (proceedings), conf. (conference), trans. (transactions), etc.., mentre vengono rimosse quelle relative alle pagine, volumi, numeri, isbn, ...

Per quanto riguarda i servizi offerti agli utenti, CiteSeer permette di recuperare i documenti correlati ad un dato articolo combinando tre diverse

[1]Nella versione originaria CiteSeer utilizzava un crawler simile a quello dei motori di ricerca che cercava pagine contenenti parole chiave come "publications", "papers", "documents".

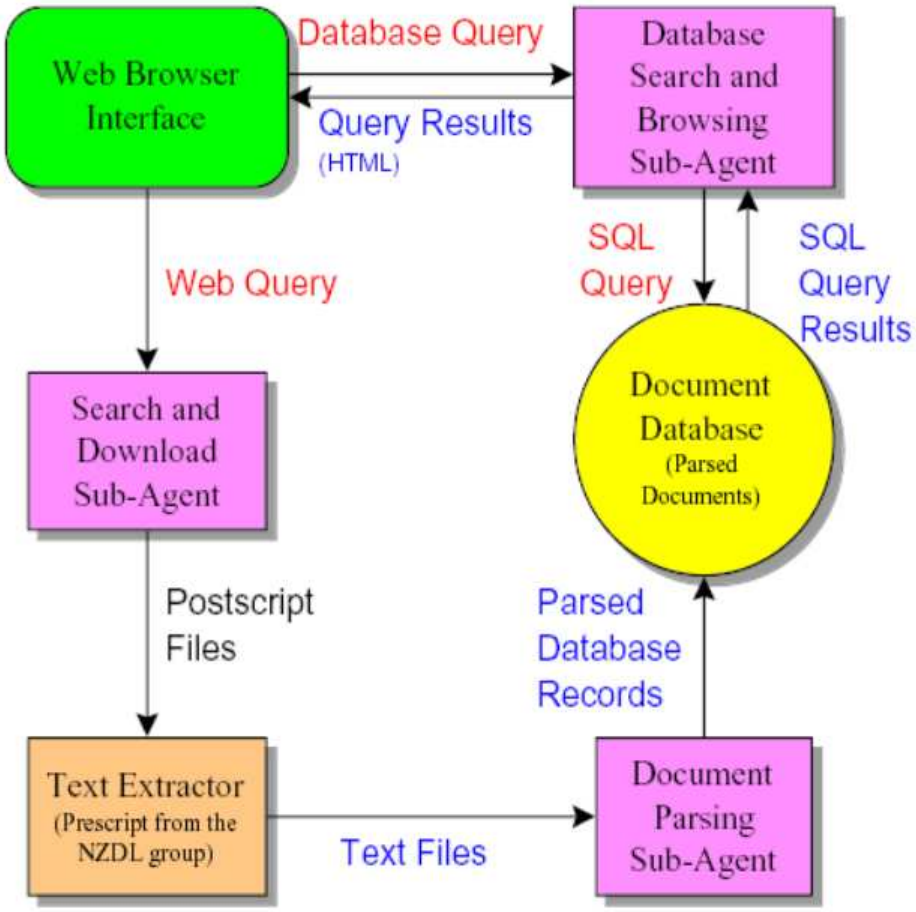

Figura 5.1: CiteSeer Architecture

misure di similarità studiate in ambito di Information Retrieval. La prima misura utilizzata è la tradizionale *Word Vector*, la quale prevede che la similarità fra due documenti venga calcolata a partire da una rappresentazione sintetica dei documenti stessi. La rappresentazione sintetica estratta è un vettore di termini, ognuno dei quali pesato in base allo schema TFIDF (term frequency x inverse document frequency) . La seconda misura è la String Distance applicata agli header (titoli ed intestazioni) di due documenti. La terza è chiamata Citation e si basa sull'osservazione che le singole parole non hanno la potenza espressiva necessaria per rappresentare adeguatamente l'argomento trattato in un articolo, mentre se due articoli condividono alcune citazioni, allora è presumibile che si tratti di articoli inerenti agli stessi argomenti[2]. In modo analogo alla Word Vector, anche questa tecnica cerca di estrarre una rappresentazione sintetica dei documenti basata sulla misura CCIDF (Common Citation x Inverse Document Frequency), analoga alla TFIDF.

5.2 Strumenti personali per la ricerca di informazioni nel Web

La ricerca di informazioni in rete rappresenta un'attività cruciale e complessa che spesso richiede l'attenzione di un operatore umano. Sebbene l'operatore umano sia efficace dal punto di vista della distinzione fra documenti rilevanti e quelli non rilevanti, non è però efficiente in termini di prestazioni. Al contrario, i tool automatici sono efficienti perchè in tempi limitati possono analizzare milioni di documenti, ma non sempre sono efficaci. Inoltre questi strumenti sono limitati dalle modalità attraverso le quali l'utente esprime i propri interessi. Verso la metà degli anni '90, la soluzione al problema del recupero dei documenti interessanti pubblicati on-line sembrava essere quella dei *personal assistants* dotati di comportamenti intelligenti per comprendere e soddisfare le necessità dell'utente. Questi strumenti sono quindi di tipo special-purpose perchè sono modellati sulle esigenze di chi li utilizza.

[2]In pratica si tratta di uno dei basilari principi adottati nell'ambito della bibliometrica

Tra questi personal assistants, i più noti sono certamente Letizia e WebWatcher.

Letizia Letizia[LIE 95] è un agente software sviluppato da Henry Lieberman al Media Laboratory del Massachusetts Institute of Technology. L'agente collabora con l'utente impegnato in una sessione di esplorazione del Web, fornendogli dei suggerimenti su quali iperlink seguire.

Generalmente l'utente esplora il Web analizzando ogni documento tramite un browser, decidendo quali link seguire e quali ignorare. Al contrario degli strumenti general-purpose e dei motori di ricerca specializzati, l'utente non deve esprimere i propri interessi esplicitamente tramite keywords o altro, perchè il sistema è in grado di interpretare ogni comportamento associandovi un significato. Ad esempio, se l'utente salva un documento come bookmark, adotta un comportamento indice di un particolare interesse verso il contenuto del documento. Analogamente, se l'utente decide di seguire un link significa che l'argomento trattato è anch'esso di particolare interesse.

Letizia osserva il comportamento adottato dall'utente cercando di capire, sulla base dei documenti visitati, quelli che sembrano essere interessanti, per poi sfruttare le informazioni raccolte fornendo suggerimenti al momento opportuno. Infatti, mentre l'utente è impegnato nella lettura di un documento, l'agente esplora la porzione di Web raggiungibile per determinare quali pagine potrebbero essere interessanti, per cui da visitare prima delle altre.

WebWatcher WebWatcher[ARM 95] è un sistema sviluppato alla Carnegie Mellon University da Robert Armstrong,Tom Mitchell, Dianne Freitag e Thorsen Joachims.

Come Letizia, WebWatcher è un personal assistant che guida l'utente durante l'esplorazione del Web, con la differenza che gli argomenti di interesse devono essere espressi esplicitamente tramite keywords. I suggerimenti proposti da WebWatcher sono riconoscibili perchè durante la navigazione attraverso un browser gli iperlink verso documenti ritenuti interessanti vengono evidenziati. Ciò significa che ogni collegamento deve essere valutato al fine di determinarne un punteggio detto *Link Utility*, dipendente da una serie di parametri tra i quali il numero di parole chiave scelte per specificare l'inte-

resse, la pagina in esame e i gli stessi link contenuti nella pagina corrente. Il punteggio rappresenta la probabilità che seguendo un dato link all'interno della pagina corrente si raggiunga un documento interessante sulla base delle specifiche date.

Capitolo 6

Un nuovo approccio alla scoperta di informazioni

I crawlers dei motori di ricerca general-purpose sono sistemi complessi caratterizzati da un elevato grado di parallelismo e progettati per recuperare il numero maggiore possibile di documenti nel minor tempo possibile, indipendentemente dal contenuto o tantomeno dalla qualità. Infatti, lo scopo di un motore di ricerca come Google o Altavista è quello di rispondere a tutte le possibili query che un utente può formulare. Per alcune di queste query la risposta comprende centinaia di migliaia di risultati, mentre per altre l'insieme delle pagine rilevanti ha una dimensione molto limitata. Quindi, sia che si tratti di una query relativa ad un argomento ampiamente trattato dai documenti pubblicati sul Web, sia che si tratti di un topic ben più circoscritto, i motori di ricerca devono comunque effettuare un crawl esaustivo della rete, con periodici aggiornamenti per garantire la freschezza dei dati. Ciò significa che, parallelamente a sessioni di crawling di refresh, i motori di ricerca devono mantenere la consistenza degli indici, operazioni particolarmente onerosa e complessa.

A partire da queste osservazioni e considerando il fatto che le query basate su keywords sono da sole poco idonee per esprimere le vere esigenze di un utente ed insufficienti per localizzare risorse rilevanti, Chakrabarti, Van Den Berg e Dom [CHA 99] propongono il Focused Crawling come nuovo paradigma per la ricerca di informazioni in rete. A differenza dei crawlers

tradizionali, un focused crawler si prefigge come scopo quello di recuperare il maggior numero di pagine rilevanti visitando il numero minore possibile di pagine non rilevanti [DIL 00]. Per questo aspetto, i focused crawlers sono la diretta evoluzione di un serie di tecniche di esplorazione del grafo denominate best-first, che verranno introdotte nel prossimo paragrafo.

6.1 Strategie di esplorazione del *Web graph*

Se dal punto di vista topologico il Web è assimilabile ad un grafo orientato, allora il problema dell'Information Discovery può essere in parte risolto adattando alcune strategie di ricerca sviluppate nell'ambito dell'Artificial Intelligence(AI).

La maggior parte di queste strategie ipotizza che l'esplorazione di un grafo sia un'operazione onerosa ed implichi il consumo delle risorse disponibili, le quali sono per natura limitate. Anche un crawler, infatti, utilizza un certo numero di risorse sottraendole a quella che viene definita la normale attività all'interno della rete, in cui sono gli utenti e non software automatici a recuperare documenti. Un crawler dovrebbe adottare un comportamento etico così come indicato nelle linee guida dei Robot Writers[1], limitando il consumo di risorse allo stretto necessario. Pertanto, se la strategia di esplorazione del grafo fosse guidata da un'euristica in grado di raggiungere le pagine rilevanti prima di tutte le altre, allora il consumo indesiderato di banda di comunicazione e di capacità di risposta dei server potrebbe essere compensato da una maggiore efficienza.

Definizione 9 *Per efficienza (efficiency o harvest rate) si intende il rapporto fra il numero di pagine rilevanti e il numero totale di pagine visitate.*

E' ora evidente come il crawl esaustivo della rete non comporti necessariamente l'adozione di una strategia best-first. Una sessione di crawling esaustiva può essere realizzata applicando una tecnica semplice ed efficiente come quella breadth-first, poichè lo scopo è quello di visitare completamen-

[1]Web Robots GuideLines: www.robotstxt.com.

te il grafo[2], piuttosto che determinare a priori l'ordine ottimo o subottimo con cui recuperare i vari documenti. Diverso è invece il caso di un crawler impegnato in una sessione di refresh dei documenti, in cui l'ordine di recupero potrebbe essere rilevante. Ad esempio, il crawler potrebbe recuperare i documenti più richiesti dagli utente prima di tutti gli altri, con l'obiettivo, quindi, di garantire una maggior freschezza dei dati ritenuti di migliore qualità. Quest'ultimo approccio è tuttora piuttosto controverso, soprattutto alla luce dei risultati presentati da Najork e Wiener [NAJ 01], dai quali si evince che, anche applicando una semplice tecnica breadth-first, le pagine di migliore qualità sono recuperate durante le prime fasi della sessione di crawling.

Algoritmo 2 Breadth-First Crawl (seeds, maxDownloads)

```
/** seeds is a list of URLs
 * maxDownloads represents the number of pages to fetch
 */

URLQueue ← ∅
downloadedURL ← ∅
foreach page ∈ seeds
        push page into URLQueue
while URLQueue ≠ and size(downloadedURL)<maxDownloads
        URL ← pop URLQueue
        if (URL ∉ downloadedURL)
                page ←download URL
                push URL into downloadedURL
                linkList ← extractLinks page
        foreach link ∈ linkList
                push link into URLQueue
```

[2]In realtà, si tratta comunque di sessioni di crawling limitate ad una frazione del Web [KUM 00].

Algoritmo 3 Best-First Crawl (seeds, maxDownloads, Resolver)

```
/** seeds is a list of URLs
 * maxDownloads represents the number of pages to fetch;
 * the pages should be visited in the order calculated by the Resolver
 * /

SortedURLQueue ← ∅
downloadedURL ← ∅
foreach page ∈ seeds
        push page into SortedURLQueue with score 1
while SortedURLQueue ≠ and size(downloadedURL)<maxDownloads
        bestURL ← pop SortedURLQueue
        if (bestURL ∉ downloadedURL)
                page ←download bestURL
                push bestURL into downloadedURL
                linkList ← extractLinks page
        foreach link ∈ linkList
                linkScore ← score link using Resolver
                push link into SortedURLQueue with linkScore
```

Le strategie di ricerca best-first si adattano, quindi, ad un tipo di ricerca client-side, in cui l'utente specifica i propri interessi. Uno dei primi tool ad aver raggiunto una certa notorietà nel settore è quello proposto da DeBra e Post [DEB 94]. Questo tool implementa una strategia denominata *Fish-Search*, in cui il processo di ricerca di informazioni viene metaforicamente paragonato alla ricerca di cibo da parte di un branco di pesci. La sopravvivenza e la riproduzione di un branco di pesci dipende dalla quantità di cibo a disposizione. Generalmente i pesci si muovono nella direzione in cui il cibo abbonda, ma non è infrequente che un certo numero di individui si distacchi dal branco. Questi ultimi sopravviveranno e daranno origine ad un nuovo branco se troveranno sufficiente sostentamento.

Nell'algoritmo Fish-Search, ogni URL corrisponde ad un pesce. Quando un documento viene recuperato, il pesce genera un numero di figli che di-

pende dal numero di iperlink contenuti nel documento stesso e dal grado di rilevanza del documento. Per DeBra e Post si intende rilevante un documento se contiene una o più parametri indicati dall'utente (keywords, espressioni regolari, filtri). Se un documento viene considerato rilevante, allora il pesce ed i suoi figli possono nutrirsi e sopravvivere, mentre in caso contrario il pesce diventa più debole e si riproduce meno frequentemente. Dopo aver seguito un certo numero di link senza trovare documenti rilevanti, il pesce muore e la ricerca nella corrispondente direzione termina. Inoltre, l'algoritmo ipotizza che i problemi sorti durante il recupero dei documenti, ad esempio servers irraggiungibili o banda di comunicazione satura, possano essere paragonati all'acqua inquinata. Pertanto, se un pesce raggiunge queste acque muore in breve tempo, perchè non riesce a nutrirsi.

In sintesi, l'algoritmo Fish-Search riceve in ingresso un URL (seed URL) a partire dal quale esplorare il Web e una query utente in base alla quale stabilire la rilevanza dei documenti recuperati. Il documento associato all'URL ricevuto in ingresso è il primo ad essere analizzato: viene determinato un punteggio che esprime il grado di attinenza con la query data e vengono estratti gli URLs eventualmente contenuti nel testo (embedded URLs). Ogni nuovo indirizzo così individuato, contribuisce alla costruzione dinamica di una priority list, la quale determina l'ordine con cui recuperare i documenti. Si tratta quindi di una strategia di tipo depth-first guidata da un'euristica che favorisce il recupero dei documenti associati ad URLs individuati in pagine rilevanti. Poichè l'algoritmo simula il comportamento di un branco di pesci alla ricerca del cibo, l'esplorazione non continua all'infinito ma si esaurisce quando vengono visitati un limite massimo di indirizzi senza trovare nuovi documenti interessanti o quando scade un time-out (o time limit).

Algoritmo 4 Fish Search Algorithm

*/** Input parameters: seedURL, width W(number of generated children),*
** depth D (number of links to follow in the specified direction),*
** user-defined query Q,*
** size S of the desired graph to be explored, the time limit*

```
*/
Set depth of the initial node as depth=D and insert it into an empty list
While the list is not empty, and the number of processed nodes is less
than Sand the time limit is not reached
        Pop the 1st node from the list and make it the currentNode
        Compute the relevance of the current node:
        if depth >0
          if the currentNode is not relevant then
            foreach child of the 1st width children of currentNode
                    Set potentialScore(childNode)=0.5
            foreach child of the rest of the children of the
                    currentNode
                    Set potentialScore(childNode)=0
          else
              foreach child of the 1st (α*width) children
              of currentNode where α is set to 1.5
                      Set potentialScore(childNode)=1
              foreach child of the rest (α*width) children of currentNode
                        Set potentialScore(childNode)=0
        foreach child of the currentNode
                if childNode alread exists in the priority list then
                  Compute max(list.score(childNode), potentialScore(childNode))
                  Replace list.score(childNode) with max
                  Reorder the priority list
                else
                    Insert childNode in the list
        foreach child of the currentNode
                Compute the depth depth(childNode) as follows:
                        if currentNode is relevant then
                                Set depth(childNode) = D
                        else depth(childNode) = depth(currentNode)-1
                        if childNode is already in the priority list then
                                Compute max between the existing depth
                                in the list to the newly computed depth
```

Replace *the list depth by max.*

EndWhile

Il limite più evidente dell'algoritmo Fish-Search risiede nel fatto che i relevance scores vengono assegnati in modo discreto (1 se il documento soddisfa i parametri indicati, 0.5 o 0 altrimenti), pertanto vi è una scarsa differenziazione fra i punteggi di priorità degli indirizzi contenuti nella lista. Soprattutto quando il limite di tempo entro il quale recuperare i documenti è breve, il crawler tende a visitare solamente i primi elementi della lista.

L'algoritmo Shark-Search[HER 98] è una variante dell'algoritmo Fish-Search, in cui l'obiettivo è trovare il numero maggiore di documenti rilevanti a parità di time limit. Quindi, invece di valutare i documenti in modo pressochè binario, questa versione determina il grado di rilevanza di un documento calcolando la similarità dello stesso con la query data. Il grado di rilevanza è un valore compreso fra 0 e 1, in cui 0 significa nessuna attinenza, mentre 1 rappresenta il matching perfetto. Ad ogni nuovo URL individuato viene associato un nuovo punteggio di priorità (inherited score), il quale dipende dal grado di rilevanza del documento padre corretto da un fattore fuzzy δ. Cercando di privilegiare il recupero di documenti raggiungibili dalle pagine più interessanti, il nuovo modo di calcolare la rilevanza determina una maggiore differenziazione fra gli elementi della priority list. Inoltre, l'algoritmo Shark-Search è stato tra i primi ad utilizzare le informazioni contenute nell'anchor text per predire il grado di interesse del documento. L'osservazione fatta dagli autori è la seguente: se un operatore umano di fronte ad una pagina sceglie quale link seguire in base al testo descrittivo del collegamento, allora lo stesso principio può essere sfruttato per migliorare l'efficienza del crawler.

Algoritmo 5 Shark Search Algorithm

*/** Input parameters: seedURL, width W(number of generated children),*
** depth D (number of links to follow in the specified direction),*

** user-defined query Q,*
** size S of the desired graph to be explored, the time limit*
**/*

Set *depth of the initial node as depth=D and* **insert** *it into an empty list*
While *the list is not empty, and the number of processed nodes is less than Sand the time limit is not reached*
Pop *the 1st node from the list and make it the currentNode*
Compute *the relevance of currentNode:*
if *depth >0*
Compute *inheritedScore(childNode) as follows:*
if *relevance(currentNode)> 0* **then**
*inheritedScore(childNode)= δ*sim(Q, currentNode)*
else *inheritedScore(childNode)= δ*inheritedScore(currentNode)*
Let*anchorText be the textual contents of the anchor pointing to childNode, and anchorTextContext the textual context of the anchor (up to given predefined boundaries)*
Compute *the relevance score of the anchor text as*
anchorScore = sim(Q, anchorText)
Compute *the relevanceScore of the anchor textual context as follows:*
if *anchorScore > 0* **then**
anchorContextScore =1
else *anchorContextScore = sim(Q, anchorTextContext)*
Compute *the score of the anchor, neighbourhoodScore as follows*
*neighbourhoodScore =β*anchorScore +*
*+ (1- β)*anchorContextScore*
where β is a pre-defined constant
Compute *the potential score of the child as:*
*potentialScore(childNode) =γ*inheritedScore(childNode)+*
*+ (1 -γ)*neighbourhoddScore(childNode)*
where γ is a pre-defined constant.
foreach *child of the currentNode*
if *childNode alread exists in the priority list* **then**
Compute *max(list.score(childNode), potentialScore(childNode))*

Replace *list.score(childNode) with max*
Reorder *the priority list*
else
Insert *childNode in the list*
foreach *child of the currentNode*
Compute *the depth depth(childNode) as follows:*
if *currentNode is relevant* **then**
Set *depth(childNode) = D*
else *depth(childNode) = depth(currentNode)-1*
if *childNode is already in the priority list* **then**
Compute *max between the existing depth in the list to the newly computed depth*
Replace *the list depth by max.*

EndWhile

Altri autori, tra i quali Chen, Chung, Ramsey e Yang propongono una strategia di ricerca guidata da algoritmi genetici [CHE 98]. Un algoritmo genetico (GA) è un processo stocastico basato sui concetti di evoluzione ed ereditarietà, in cui una funzione di fitness viene utilizzata come metrica per determinare, all'interno di un ambiente di ricerca, la/e soluzione/i ottima/e. Generalmente gli algoritmi genetici prevedono una fase preliminare in cui viene determinato lo spazio iniziale delle soluzioni, chiamato anche la prima generazione della popolazione degli individui. Nelle fasi successive, la popolazione si evolve e si possono osservare nascite di nuovi individui che ereditano parte del patrimonio genetico dei genitori (genetic crossover), mentre altri elementi scompaiono o subiscono mutazioni casuali. Il processo termina quando due generazioni consecutive non producono una sufficiente variazione del valore della funzione di fitness.

Algoritmo 6 Genetic Algorithm based Crawl

*/**
* *SeedURLs are the pages provided by the user*

** CG is the current generation*
**/*

1. Initialize the search space: *saves all SeedURLs in asetcalled Current Generation, CG=(cg_1, cg_1,...,cg_n)*

2. Iteration until convergence:

New pages connected to the starting ones are retrieved

2.1 Crossover:

Documents that have been connected to multiple starting pages are considered Crossover Pages and saved in a new set C=(c_1, c_2, ...)

2.2 Mutation:

the system retrieves pages from a Yahoo's user specified category to from a Mutation Set M=(m_1, m_2, ...)

2.3 Stochastic selection:

each page is evaluated by a fitness function (i.e. Jaccard's function) and it survive if the score is greater than a predefined threshold. The survivors forms a new population for the next generation.

3. Output: *the final generation is the set of the relevant pages.*

In sintesi, tutti gli studi descritti dimostrano che generalmente l'adozione di una strategia di esplorazione best-first permette di recuperare documenti rilevanti prima di altri di minore importanza, aspetto particolarmente interessante se il crawler è impegnato in una sessione di crawling dedicata alla costruzione di un database specializzato su un argomento specifico, oppure quando un'esplorazione esaustiva non è realizzabile per scarsità di risorse o tempo a disposizione.

Quindi, l'adozione di strategie best-first implica necessariamente la definizione di metriche per poter valutare a priori l'importanza dei documenti non ancora recuperati. Come osservano Cho, Garcia-Molina e Page [CHO 98], è possibile definire metriche di caratterere generale, quali:

1. ***Similarity to a Driving Query Q.*** Il crawler esegue una sessione di esplorazione alla ricerca di documenti simili alla query Q data. Per determinare il grado di similarità di due documenti P e Q, gli

autori utilizzano la rappresentazione degli stessi sotto forma di vettori n-dimensionali $<w_1,w_2,\ldots,w_n>$, in cui il termine w_i rappresenta la keyword i-esima del vocabolario. Se il termine w_i non compare nel documento allora assumerà un valore pari a 0, mentre in caso contrario dovrebbe rappresentare l'importanza relativa della parola all'interno del documento e all'interno della collezione di documenti, così come indicato da Salton [SAL 89]. Durante la fase di esplorazione, però, un crawler non ha a disposizione l'intera collezione di documenti, cosa che gli consentirebbe di applicare correttamente la tecnica TDIDF, per cui si limita a stimare la *inverse document frequency* sulla base delle pagine precedentemente raccolte. La similarità fra i due documenti viene calcolata tramite la misura coseno.

2. ***Backlink Count.*** Si tratta sostanzialmente del numero di riferimenti verso un documento P, per cui si ipotizza che una pagina altamente referenziata sia molto più importante di una pagina citata raramente. Anche in questo caso, come nel precedente, il crawler ha una visione parziale del Web, per cui può valutare il numero di riferimenti solo sulla base delle pagine già recuperate.

3. ***PageRank.*** La metrica *Backlink Count* considera tutti i riferimenti di pari importanza, mentre il Web è una rete sociale in cui può essere definita anche una misura di prestigio dei documenti. L'algoritmo PageRank è descritto con maggiori particolari nel paragrafo dedicato a Google

4. ***Forward Link Count.*** Analogamente alla metrica *Backlink Count*, si può determinare quanti iperlink sono contenuti in un documento P. Se il numero di collegamenti verso l'esterno è estremamente elevato, allora la pagina potrebbe essere quella di una Web Directory.

5. ***Location Metric.*** L'interesse verso una pagina P può essere valutato analizzando semplicemente il suo indirizzo. Per alcune queries, ad esempio, visitare i domini .com potrebbe essere più fruttuoso rispetto al recupero di documenti da domini .edu, oppure si potrebbero privilegiare quelle pagine il cui indirizzo contiene uno dei termini indicati

dall'utente. Inoltre, potrebbero essere visitati gli indirizzi con pochi slash rispetto a quelli con molti slash. In questo modo si ipotizza che i siti abbiano un'organizzazione gerarchica in cui i documenti più importanti si trovano nei livelli più vicini alla radice.

Gli esperimenti realizzati con WebBase[3] evidenziano come PageRank sia un ottimo strumento per determinare il prestigio di un documento e come una strategia di esplorazione guidata dalle suddette metriche o da una combinazione delle stesse permetta di recuperare un numero elevato di documenti interessanti prima di altri non rilevanti.

6.2 Focused Crawlers

Il Focused Crawling è stato introdotto da Chakrabarti, Van Der Berg e Dom come nuovo paradigma di ricerca di informazioni in rete [CHA 99] . Un Focused crawler tenta di identificare i link piiù promettenti e di ignorare quelli che non corrispondono agli interessi specificati dall'utente.

Un crawler standard segue ogni link, tipicamente tramite una strategia breadth-first o depth-first. Pertanto, se un crawler inizia l'esplorazione da un seed URL che dista i step da un documento target, tutte le pagine fino al livello *i-1* devono essere visitate (Figura 6.1). Al contrario, a parità di seed url, un focused crawler visita solo un subset di documenti (dal livello 0 al livello *i-1*) (Figura 6.2).

L'adozione di una strategia di ricerca differenzia un crawler standard da un crawler focalizzato anche per quanto riguarda l'architettura. Mentre un crawler standard si occupa solo del recupero dei documenti (in casi particolari anche dell'estrazione degli URLs) , un crawler focalizzato deve disporre degli strumenti necessari per valutare il grado di rilevanza dei documenti. L'architettura di un crawler focalizzato è illustrata in Figura 6.3 e come si può notare il crawler è dotato di un parser delle pagine .html, di un modulo per il calcolo della priorità degli indirizzi individuati e di una lista che rappresenta l'ordine con cui devono essere visitati i documenti.

[3]Il crawler dell'università di Stanford, prototipo dello spider di Google

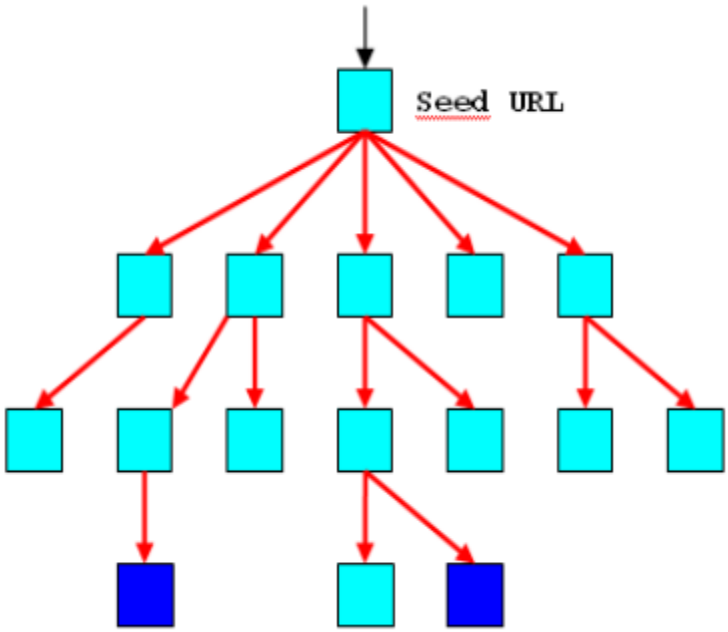

Figura 6.1: Esplorazione del Web realizzata da un crawler standard (in azzurro le pagine visitate, in blu i targets, in rosso i link seguiti).

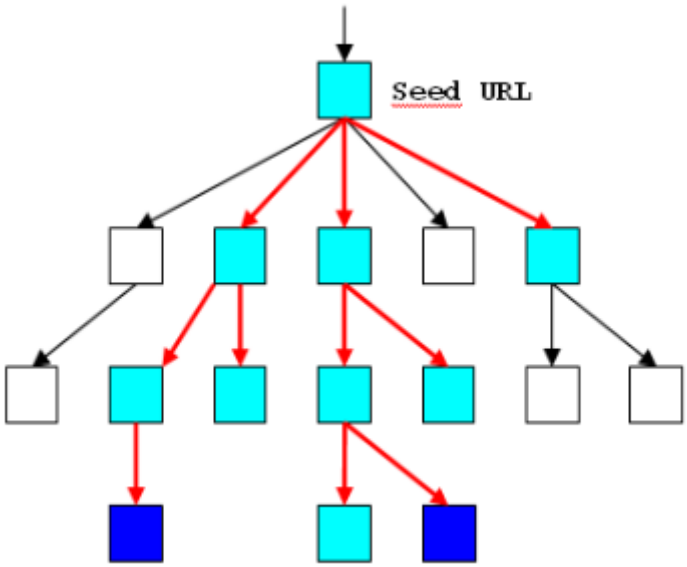

Figura 6.2: Esplorazione del Web realizzata da un focused crawler (in azzurro le pagine visitate, in bianco quelle non visitate, in blu i targets, in rosso i link seguiti, in nero i link scartati .

Nel Focused Crawling di Chakrabarti gli interessi dell'utente sono specificati per mezzo di una tassonomia e di un set di documenti che costituiscono esempi di documenti rilevanti. Il crawler sfrutta tecniche di Machine Learning per discriminare i documenti interessanti da quelli non interessanti ed inoltre, un *Hypertext Classifier* analizza i documenti recuperati e li classifica all'interno della tassonomia. Sotto questo punto di vista il crawler di Chakrabarti è un'evoluzione degli studi di Labrou e Finin [LAB 99], in cui viene descritto un sistema per la classificazione di pagine Web all'interno delle categorie di Yahoo!.

Come tutti i sistemi che utilizzano tecniche di Machine Learning da set di esempi, anche questo crawler deve essere sottoposto ad una fase di training all'interno della quale possono essere riconosciuti 5 step principali:

1. **Canonical taxonomy creation.** Il classificatore viene sottoposto a training con la tassonomia di Yahoo! o ODP ed un set di pagine di esempio. Il risultato di questo step preliminare è un albero decisionale detto coarse-grained tree.

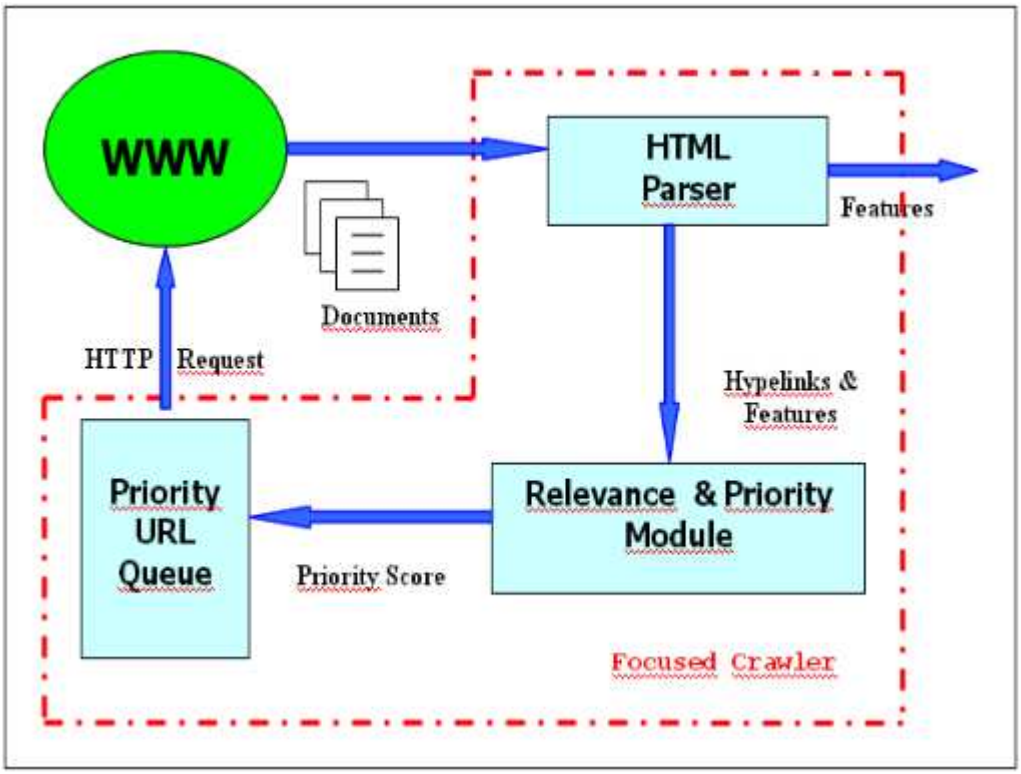

Figura 6.3: Architettura di un Focused Crawler.

2. **Example collection.** L'utente propone altri esempi di documenti considerati di particolare interesse, ad esempio importando i propri bookmark files.

3. **Taxonomy selection and refinement.** Le pagine indicate dall'utente vengono classificate all'interno della tassonomia e il risultato viene proposto all'utente per la validazione (classificazione corretta o meno). In questo step l'utente può raffinare la tassonomia inserendo nuove categorie più specifiche oppure spostare i documenti da una classe in un'altra.

4. **Interactive exploration.** Il sistema esegue un crawl preliminare in cui cerca di classificare i documenti direttamente raggiungibili dal set di esempi. Anche questa è comunque una preclassificazione che l'utente deve valutare, ad esempio integrando alcune pagine nella tassonomia e scartandone altre.

5. **Training.** Il sistema integra le informazioni ricevute durante l'interazione con l'utente con il coarse-grained tree, con lo scopo di estrapolare il modello statistico che guida il processo di classificazione.

La fase successiva riguarda il recupero di nuovi documenti e la relativa classificazione. Periodicamente o concorrentemente il sistema esegue un algoritmo di topic distillation, il quale ha l'obiettivo di identificare quelle pagine che contengono un elevato numero di link. Per Chakrabarti, un numero elevato di link è sinonimo di una pagina hub, ossia di una pagina che contiene riferimenti a nuove risorse. Se questa pagina è stata inclusa all'interno di una categoria della tassonomia, allora il punteggio di priorità degli indirizzi contenuti dovrebbe aumentare. L'attività del sistema viene supervisionata periodicamente dall'utente, il quale conferma o meno le scelte del classificatore. Tutte le informazioni raccolte durante l'interazione con l'utente sono utilizzate come feedback, pertanto sia le categorie della tassonomia che il modello statistico del classificatore sono costantemente raffinati.

Diligenti e altri [DIL 00] osservano che gli iperlinks sono per natura unidirezionali e che la maggior parte dei siti sono organizzati in maniera gerarchica.

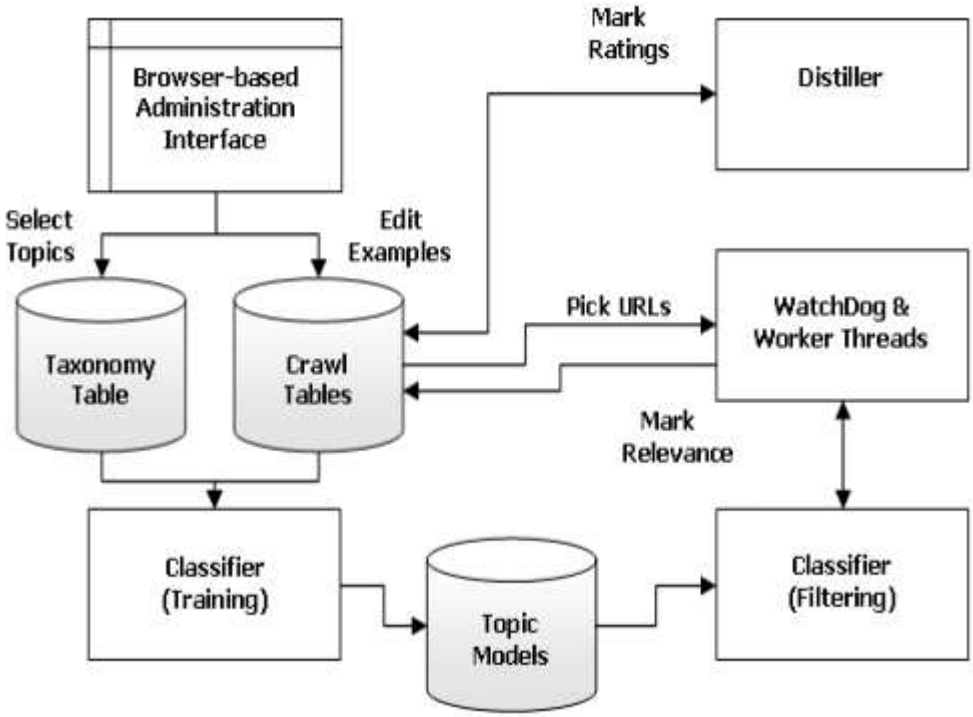

Figura 6.4: Architettura del Focused Crawler di Chakrabarti, Van Der Berg e Dom [CHA 99].

Tutti i crawlers, compreso quello focused di Chakrabarti, hanno una visione forward del grafo del Web. Ciò significa, che dato un documento P, il crawler riesce a recuperare i riferimenti contenuti in esso, ma non gli è possibile risalire alle pagine che puntano a P (backward crawl). Pertanto, se il crawler raggiunge un sito ancora da esplorare tramite un nodo terminale, la probabilità di trovare nuove risorse interessanti diminuisce.

Durante una sessione di esplorazione del Web, un crawler ha una visione parziale della struttura degli iperlink, al contrario un search engine mantiene una visione pressochè globale e, se interrogati, sono in grado di elencare le pagine che contengono riferimenti ad un documento dato. La capacità di backcrawling è limitata, ma è sufficiente per permettere la costruzione di quello che Diligenti chiama il Context Graph. Anche questo tool adotta un approccio di tipo Machine Learning, per cui spetta all'utente fornire alcuni esempi di documenti intessanti. Questo crawler, però, ricostruisce il contesto prettamente gerarchico nel quale i documenti di esempio sono inseriti. Ipotizzando che il training set sia composto da un solo documento, il sistema ne ricostruisce il context graph sfruttando la capacità di backcrawling di Google o Altavista. Gli eventuali documenti recuperati costituiscono il primo livello di una gerarchia incrementale, in cui si suppone che i documenti meno distanti dal core (target) siano di maggiore rilevanza. Il processo di backcrawling viene iterato finchè non è stato raggiunto il numero specificato di livelli (depth). Se, come è auspicabile, il numero di documenti del set di esempio è maggiore di 1, si procede ad una fusione dei context graph associati ad ogni seme (merged context graph). Il merged context graph costituisce il training set per il classificatore [4], il quale deve considerare che ogni livello del grafo rappresenta una categoria. Un'altra differenza rispetto ai precedenti lavori sul Focused Crawling risiede nel fatto che il crawler non utilizza una sola coda di priorità, ma il classificatore organizza gli URL inesplorati in N code, dove N è la profondità massima del merged context graph. Nella fase di inizializzazione, il crawler recupera tutti i documenti riferiti dal seed URL, estrae le feature di interesse e tutti gli iperlinks. Il classificatore analizza ogni documento recuperato e, determinando la categoria in cui può essere inserito,

[4] Il classificatore è ben più complesso di quello utilizzato in [CHA 99], soprattutto perchè si tratta di un set di classificatori.

lo assegna alla relativa coda di priorità. La strategia che guida l'esplorazione della rete prevede che vengano visitate quelle pagine con priorità più alta (livello 1) e via via tutte le altre. Gli esperimenti effettuati dimostrano che un crawler di questo tipo è più efficiente di un focused crawler standard. A mio avviso, il limite maggiore dei crawler di Diligenti e di Chakrabarti risiede proprio nel tipo di approccio adottato. Le tecniche di Machine Learning possono guidare la strategia di ricerca efficacemente, ma il modello statistico ricavato dal classificatore è un modello ad hoc per un particolare argomento.

Gli esperimenti effettuati dimostrano che un crawler di questo tipo ha una maggiore efficienza rispetto ai focused crawler standard, ma, a mio avviso, il limite maggiore di questi strumenti risiede proprio nel tipo di approccio adottato. Le tecniche di Machine Learning forniscono sicuramente buoni risultati (a patto di aver scelto un training set significativo), ma il modello statistico estrapolato dal classificatore è un modello ad hoc per un particolare argomento e difficilmente le informazioni estrapolate in un contesto possono essere riutilizzate in altri ambiti.

6.3 Intelligent Focused Crawlers

Aggarwal, Al-Garawi e Yu osservano in [YU 01] che le strategie di ricerca adottate da Chakrabarti e Diligenti sono interessanti perchè basate su due proprietà della struttura degli iperlinks.

Linkage Locality , in base alla quale pagine che trattano di un certo argomento contengono riferimenti ad altre pagine inerenti al medesimo argomento.

Sibling Locality , in base alla quale se una pagina contiene riferimenti ad alcuni documenti che trattano di un particolare topic, allora anche gli altri link riguardano lo stesso topic. E' questo il caso delle pagine hubs o delle Web directories.

La strategia di ricerca del crawler di Chakrabarti o di Diligenti utilizza un modello statico della struttura degli iperlinks, proprio perchè viene guidata da informazioni estrapolate tramite tecniche di Machine Learning.

Un crawler intelligente, invece, dovrebbe essere in grado di imparare come e quanto influisce la struttura degli iperlinks sulla scoperta di nuove informazioni, indipendentemente dal topic di interesse. Inoltre, un crawler intelligente dovrebbe valutare anche quali altri aspetti([CHO 98] oltre la struttura degli iperlinks possono efficacemente guidare la strategia di ricerca.

Il crawler proposto dagli autori riceve in input uno o più seeds URLs e progressivamente si autofocalizza sulle pagine rilevanti (in questo caso pagina rilevante è una pagina che contiene le keywords specificate in una query). Ogni nuovo documento recuperato dal crawler viene analizzato per estrarre il testo e gli URLs contenuti. Ogni URL individuato costituisce il riferimento ad una pagina candidata, per la quale viene calcolato un punteggio di priorità sulla base di alcuni parametri forniti da un modello statistico della struttura dei riferimenti e degli altri aspetti che possono efficacemente guidare la strategia di ricerca. Il modello statistico proposto prende in considerazione 4 features per determinare la priorità di un URL candidato:

1. il contenuto delle pagine che contengono riferimenti all'URL candidato;
2. una location metric (vedi pag. 96), perchè gli URL stessi non sono scelti a caso, ma rispecchiano una organizzazione semantica delle pagine;
3. la natura delle *inlinking pages*, o *Linkage Locality*(ved. pag. 104), per cui se le pagine che contengono collegamenti ad un URL candidato sono rilevanti, allora anche la pagina riferita può soddisfare il criterio di rilevanza;
4. la natura delle *siblings pages*, o *Sibling Locality* (ved. pag. 104), per cui se le pagine che contengono collegamenti ad un URL candidato contengono riferimenti anche a pagine che sono già state recuperate e che soddisfano il criterio di rilevanza, allora il documento associato all'URL potrebbe essere anch'esso rilevante.

Gli autori denotano con C l'evento in cui una pagina recuperata soddisfa il predicato definito dall'utente (contiene cioè le keywords della query). La probabilità *P(C)* di una pagina candidata di soddisfare il criterio indicato può essere calcolato come il semplice rapporto fra il numero di documenti rilevanti e il numero totale di documenti recuperati. Tale probabilità rappresenta

anche la priorità con la quale il crawler dovrebbe recuperare la pagina ed è identica per tutti gli URL candidati. La differenziazione delle priorità avviene proprio grazie alle informazioni raccolte durante l'esplorazione. Infatti, conoscendo a priori alcune caratteristiche degli URL candidati, incrementa il valore della priorità. Sia E un'informazione nota circa un URL candidato, ad esempio una delle 4 features sopracitate. La conoscenza di questa informazione può aumentare il valore della priorità della pagina associata. Il nuovo valore della priorità viene calcolato sotto forma di probabilità condizionata *P(C|E)* secondo la seguente formula:

$$P(C|E) = P(C \cap E)/P(E), \tag{6.1}$$

dove *P(E)* è la frequenza con la quale l' evento E si verifica.

Analogamente, si può calcolare un fattore di interesse (*Interest Ratio* per l'evento C, noto un evento E:

$$I(C, E) = P(C|E)/P(C). \tag{6.2}$$

Se l'evento è favorevole, ossia se *P(C|E)>P(C)*, allora l'interest ratio è maggiore di 1 e la pagina dovrebbe essere recuperata prima delle altre con un interest ratio minore. Inoltre, se la strategia è guidata da più di una feature, si può esprimere l'interest ratio per l'evento composto ε come

$$I(C, E) = \pi_{i=1}^{k} I(C, E_i) \tag{6.3}$$

Il crawler compie costantemente un processo di self-learning raccogliendo le informazioni necessarie per calcolare gli interest ratio relative alle sopracitate features. Per ogni feature il crawler esegue quindi una particolare fase di learning:

Content Based Learning. Dato un lessico composto da W_1, W_2, W_n termini, solo una piccola parte di esso compare nel testo delle inlinking pages degli URL candidati. La variabile logica Q_i assume valore vero quando il termine i-esimo compare in una delle inlinking pages. Pertanto, l'insieme $M = \{$ *i: Q_i is true* $\}$ contiene gli indici dei termini che compaiono nelle inlinking pages. Il relativo interest ratio è

$$I(C, Q_i) = P(C \cap Q_i)/(P(C) \cdot P(Q_i)) \tag{6.4}$$

Non sempre la presenza delle keywords nelle inlinking page porta alla scoperta di nuove pagine rilevanti. Il fattore di interesse complessivo viene filtrato introducento un *significance factor* calcolato empiricamente [5]

$$I_c = \pi_{i\,:\,i \in M,\ S(C,Q_i) \geq threshold}\, I(C, Q_i) \tag{6.6}$$

URL Token Based Learning. Il processo di learning sulla base degli indirizzi delle pagine candidate, consiste nella verifica del matching fra le keywords date e i token estratti dagli URL. Il processo è del tutto analogo al content based learning. I_u è il corrispondente interest ratio.

Link Based Learning. Se il crawler recupera 100 pagine di cui solo 10 sono rilevanti, la probabilità *P(C)* è pari al 10%. Se la struttura dei link fosse completamente random, allora la probabilità di recuperare una nuova pagina rilevante è esattamente pari a *P(C)*. La probabilità che da una pagina rilevante si raggiunga un'altra pagina rilevate è $P(C) \cdot P(C)$. Quest'ultimo è un valore atteso, ma se sulla base dei documenti recuperati risulta che la frazione di documenti in cui sia il nodo sorgente che il nodo destinazione sono rilevanti è maggiore del valore atteso, allora significa che per l'argomento d'interesse esiste un'organizzazione strutturale dei contenuti. Formalizzando questo aspetto e considerando una pagina che viene riferita da altre k pagine, m delle quali soddisfano il predicato, per ognuna delle m pagine che soddisfano il predicato il corrispondente interest ratio è dato da

$$p = f_1/(P(C) \cdot P(C)), \tag{6.7}$$

dove f_1 rappresenta la frazione di pagine rilevanti per le quali anche le inlinking page sono rilevanti. Analogamente, l'interest ratio per ognuna delle k-m pagine che non soddisfano il predicato può essere calcolato come

$$q = f_3/(P(C) \cdot (1 - P(C))), \tag{6.8}$$

[5] Gli autori propongono come formula di calcolo del significance factor:

$$S(C, Q_i) = (P(C|Q_i) - P(C))/(\sqrt[2]{P(C) \cdot (1 - P(C))/(N \cdot P(C))}) \tag{6.5}$$

con N il numero totale di pagine recuperate

dove f_3 è la frazione di pagine che non soddisfano il predicato, nonostante le inlinking page fossero rilevanti.

L'interest ratio complessivo è dato da:

$$I_l = p^m \cdot q^{k-m}. \tag{6.9}$$

Sibling Based Learning. Una pagina candidata ha una maggiore probabilità di soddisfare il predicato se i documenti recuperati dalle stesse inlinking page sono a loro volta rilevanti. Supponendo che una pagina contenga n embedded URLs (siblings), di cui r già recuperati, allora il numero atteso di pagine che soddisfano il predicato è $r \cdot P(C)$. Se durante il crawling il numero di siblings che soddisfano il predicato è maggiore di $r \cdot P(C)$, allora la priorità dei siblings non ancora recuperati deve aumentare. I_s è l'interest ratio corrispondente e viene calcolato come:

$$I_s = s/e, \tag{6.10}$$

dove s è il numero di siblings recuperati che soddisfano il predicato ed e è il numero atteso di siblings rilevanti.

La priorità assegnata ad una pagina candidata è la combinazione dei fattori sopracitati:

$$PriorityValue = w_c \cdot \log(I_c(C)) + w_u \cdot \log(I_u(C)) + w_l \cdot \log(I_l(C)) + w_s \cdot \log(I_s(C)) \tag{6.11}$$

Parte III

Estrazione delle informazioni: un approccio linguistico

Capitolo 7

Catene Lessicali - Stato dell'arte

Leggeva lentamente, mettendo insieme le sillabe, mormorandole a mezza voce come se le assaporasse, [...] e così si impadroniva dei sentimenti e delle idee plasmati sulle pagine. Quando un passaggio gli piaceva particolarmente lo ripeteva piú volte, tutte quelle che considerava necessarie per scoprire quanto poteva essere bello anche il linguaggio umano. [1]

7.1 Generazione automatica di sommari

La crescita del Web, misurata in termini di numero di pagine pubblicate, ha provocato il fenomeno dell'*information overload*. Il desiderio e la necessità di fornire servizi *on-line* efficaci ed efficienti, soprattutto per quanto riguarda la ricerca di informazioni, ha contribuito al raffinamento di tecniche già esistenti di Information Retrieval (IR) e allo sviluppo di nuovi strumenti di NLP (Natural Language Processing).

Le pagine HTML sono pensate per essere comprese da operatori umani. Si pensi, innanzitutto, che l'HTML é nato come linguaggio di markup, in cui i tag definiscono e danno indicazioni circa il layout con il quale i documenti

[1] *Luis Sepúlveda.* Il vecchio che leggeva romanzi d'amore*, Ugo Guanda Editore, Parma, 1993.*

devono essere visualizzati (allineamento dei paragrafi, dimensione dei font, colore di sfondo, etc. . .).

Nei documenti HTML, il contenuto informativo e la relativa rappresentazione sono inscindibili l'uno dall' altra. Per questo motivo le pagine Web sono dette *human-readable*. L'ideale sarebbe che fossero anche *machine-readable*.

Il termine *Information Extraction* (IE) é spesso utilizzato con molteplici significati. L'IE cosí come definito dalla Message Understanding Conferences (MUC) si riferisce al problema di estrarre automaticamente i dettagli essenziali che un testo contiene. Si tratta di estrarre la semantica dell'informazione testuale!

Ad esempio, data una pagina web circa un annuncio di un seminario, un sistema di IE estrarrá le informazioni salienti circa l'argomento, lo speaker, la data, l'ora, il luogo e cosí via. L'estrazione avverrá seguendo una serie di regole.

Una volta estratte le informazioni da questi testi *free-form*, vi sono sistemi particolarmente evoluti che sono in grado di aggiornare database o produrre reports in modo completamente automatico.

L'IE puó anche essere visto come l'insieme di tecniche che permettono di identificare i frammenti di testo che rappresentano la risposta a domande standard (*query template*). Generalmente un *template* é formato da piú elementi detti campi o *slot*, che devono essere completati cercando, nella collezione dei documenti, quei segmenti di pagine che rispondono alla query (*slot filling*). I *template* cosí compilati sono detti *answer template*. Tra tutti gli *answer template* possono poi essere scelti quelli migliori, perché vi sono probabilmente frammenti di testo che meglio rispondono alla query rispetto ad altri.

Determinare quali siano i frammenti migliori coinvolge alcuni aspetti di classificazione, problema che é stato ampiamente studiato sotto vari punti di vista. In pratica, un problema di IE puó essere considerato come un'estensione del problema di *text classification*.

Vi sono sistemi di IE che utilizzano un approccio di *Wrapper Induction* ([ASH 97], [KUS 97]). Come dimostrato in [LUG 03], questi sistemi estraggono informazioni in modo efficace quando vengono applicati a pagine *human-readable* con struttura molto regolare , ad esempio pagine ottenute in risposta

ad interrogazioni a database. Purtroppo l'applicabilità dei sistemi di cui sopra é evidentemente limitata nell'ambito delle pagine altamente strutturate, che non rappresentano l'intero Web.

Come notato in [FRE 98], peró, esistono pagine che sono strutturate in senso relazionale. Si tratta di pagine che descrivono entità del mondo reale, con lo scopo di raccoglierne aspetti salienti (attributi e loro valori). In queste pagine vi sono links ad altri documenti rappresentanti altre entità e così via. I links corrispondono quindi alle associazioni del modello relazionale. Interrogando queste pagine come un normale database relazionale, ci si aspetta di riuscire ad ottenere risposte a un set di domande standard.

Ad esempio, in ambito universitario, un'entità é la pagina di un professore, nella quale genalmente si potrà leggere il suo nome, cognome, indirizzo e-mail, campi di ricerca, affiliazioni...
Se un professore é titolare di corsi, allora ci si aspetta anche di trovare link alle pagine dei corsi. Un corso é un altro tipo di entità e in una home page tipica di un corso potrà essere indicato il nome del corso, il codice, l'elenco delle lezioni, i testi d'esame...

Le informazioni di cui sopra possono essere presentate con svariati termini e forme (elenchi puntati, tabelle, immagini). Molti studi, rinunciano all'applicazione delle tecniche di *Natural Language Processing*, poiché giudicano di grande importanza il contenuto informativo di tipo grafico/visuale (liste, tabelle).

A sostegno di questa tesi, essi affermano che il testo presente nelle pagine HTML perde parte della struttura sintattica dei documenti tradizionali ([SOD 97]). Infatti, spesso le informazioni sono date in modo sintetico e non in frasi complete. Dove non vi é una frase completa, la semantica dell'informazione é comunicata in modo visivo. Quindi, le tecniche di NLP, pensate per documenti tipici dell'Information Retrieval tradizionale (es.: articoli scientifici) i quali per natura dispongono di una struttura sintattica, non sono cosí facilmente adattabili al Web.

Le soluzioni proposte si basano su aspetti di *machine learning* e, nel caso di struttura relazionale, su algoritmi di copertura di *relational learning* o *inductive logic programming* ([FRE 98]). Spiegare la differenza fra *machine learning* e *relational learning* esula dallo scopo di questa tesi.

Si puó dire peró che, mentre entrambi condividono l'obiettivo di estrarre regole da esempi a cui vengono poste *labels* (set di esempi preclassificati) per poter imparare a classificare istanze a cui non é stata assegnata alcuna etichetta, le istanze degli algoritmi di machine learning sono rappresentati da vettori di coppie (attributo, valori) di lunghezza fissa.
Nel relational learning, le istanze di un universo relazionale sono contenute in un dominio, ossia gli attributi non sono isolati, ma sono associati l'un l'altro logicamente. Inoltre, i learners relazionali possono derivare logicamente nuovi attributi da quelli giá esistenti.

Un interessante sistema é **WebKB**[MIT 98] in cui le suddette tecniche sono utilizzate da un crawler che esplora con strategia breadth-first il Web. Ogni documento recuperato viene esaminato e se si tratta di una pagina classificabile all'interno di un'ontologia data WebKB, é in grado di estrarre vari tipi di informazione in via automatica.

WebKB riceve due input: un'ontologia di classi e relazioni fra classi e un set di esempi che rappresentano istanze delle classi e delle relazioni. Un esempio di istanza di classe e/o relazione puó essere costituito da una sola pagina, da piú pagine o da una stringa di testo. Il classificatore utilizzato é di tipo bayesiano. Ogni documento viene rappresentato come *bag of words* con particolare attenzione, però, ai titoli e alle stringhe che compaiono nell'anchor tag. Le ragioni per cui questi elementi sono stati preferiti sono analoghe a quelle descritte in [ATT 98].

Il crawler popola un database di classi e relazioni mentre esplora il Web. Ciò significa che ogni volta che il valore della funzione di matching fra un documento recuperato e l'ontologia indica che si tratta di un' istanza di una certa classe, la pagina viene aggiunta alla base di conoscenza. Riconoscere una nuova istanza permette di modificare la strategia breadth-first in modo da mantenere una coda di prioritá delle pagine da esplorare.

Il sistema é stato testato nell'ambito del dominio universitario, in particolare sono state prelevate le pagine delle università statunitensi. WebKB sembra funzionare soddisfacentemente, ma é lampante come il metodo non abbia una validità generale. Infatti, il dominio é ristretto ad un ambito specifico e non compie una ricerca esaustiva di tutti i siti delle università nel mondo, siti che potrebbero avere una struttura ed un'organizzazione totalmente

differente da quella degli esempi forniti nel training set.

Un approccio basato su tecniche di NLP è adottato dai moderni sistemi di generazione automatica di sommari summarization systems. Se un motori di ricerca, oltre a fornire gli URLs in risposta ad una query, fosse in grado di riportare una semplice sintesi degli argomenti esposti nei documenti individuati, allora un utente potrebbe distinguere più facilmente le pagine rilevanti da quelle non rilevanti. In questo senso, potrebbe essere interessante analizzare quante e quali tecniche di NLP sono sviluppate da questa tipologia di strumenti.

In [SPA 93] la generazione di un sommario viene descritta come due step sequenziali:

1. **Sintesi automatica del testo** Purtroppo non si tratta di un'operazione semplice. Infatti, riassumere un testo comporta la comprensione del significato espresso, al fine di rappresentarlo in una forma equivalente ma concisa.

2. **Costruzione del sommario** Estrazione, a partire dalla forma equivalente, dei concetti fondamentali.

I primi summarization systems sfruttavano metodi statistici per determinare i concetti fondamentali espressi in un testo. Ad esempio [LUH 58], considerava la *frequency word table* come forma concisa di rappresentazione di un documento, considerando come le parole piú frequenti siano un buon indizio per individuare concetti importanti. Questa soluzione si adatta bene alla maggior parte dei testi, indipendentemente dallo stile utilizzato.

Sistemi piú avanzati, tra i quali [EDM 69] e [HOV 97] sfruttano, oltre ai metodi statistici, anche alcuni marcatori linguistici (detti anche rethorical markers) come *cue phrases*. Ad esempio, in un documento scientifico espressioni del tipo "the paper describes..." oppure "in conclusion..." sono estremamente indicative per identificare le frasi salienti. Altre frasi importanti possono essere individuate considerando la formattazione delle parole (grassetto, corsivo...), nonché la loro posizione all'interno del testo. Infatti, generalmente le frasi iniziali e finali di ogni paragrafo sono piú significative di altre (*location cue*). Paice dimostra [PAY 90] che queste euristiche non sono estremamente complesse da implementare e producono migliori risultati

rispetto ai metodi puramente statistici. Purtroppo l'accuratezza di questi sistemi varia enormemente a seconda del tipo di documento analizzato. Infatti, come riportato da [ONO 94] i location e clue phrases methods sono adatti a sintetizzare documenti scientifici o tecnici, ma falliscono spesso quando elaborano articoli tratti da quotidiani, perché i rhetorical markers variano enormemente a seconda dello stile utilizzato.

Proprio in considerazione del fatto che anche lo stile linguistico ha la sua importanza, sono stati sviluppati sistemi che affiancano alle euristiche suddette tecniche di machine learning da set di esempi con abstracts generati manualmente (ad esempio [KUP 95]).

Il limite maggiore delle soluzioni citate consiste proprio nella rappresentazione di un documento come semplice unione di termini, troppo povera per generare buoni sommari. Allo stesso modo, il risultato prodotto dall'uso di euristiche tipo cue phrases e location cues é troppo dipendente dal tipo di testo, per avere una validitá generale.

Ció che questi metodi non prendono in considerazione é il fatto che fra le parole usate in un testo esistono delle relazioni. Se queste relazioni potessero essere incluse nella forma concisa, allora si otterrebbe realmente una rappresentazione semantica del documento.

Mentra alcune tecniche di NLP prevedono la completa compresione dei documenti *full semantic understanding* per generare buoni sommari, altre si limitano ad un *partial semantic understanding*. Tra queste ultime é particolarmente interessante, a mio avviso, il metodo delle catene lessicali che, dal punto di vista linguistico, ha la propria teorizzazione nello studio di Michael Halliday e Ruqaiya Hasan [HHC 76].

Il capitolo é organizzato come segue:

- la sezione 7.2 é dedicata alle definizioni di coerenza, coesione e presenta un'ampia panoramica degli studi linguistici di Hasan e Halliday;

- la sezione 7.3 descrive gli algoritmi proposti per l'estrazione delle catene lessicali;

- la sezione 7.4 illustra alcune estensioni al metodo delle catene lessicali proposte in letteratura.

7.2 Coerenza e Coesione: definizioni

Hasan e Halliday descrivono in [HHC 76] e in [HHG 85] alcuni aspetti della lingua inglese, osservando che c'é un'importante differenza fra un insieme di frasi (*random set of sentences*) ed un testo (*text*).

Infatti, un *set of sentences* puó essere definito testo quando un lettore, considerando le frasi nella loro successione, non solo ne comprende il significato, ma si rende anche conto di come esse siano piú o meno dipendenti l'una dalle altre. Quest'ultima proprietá é detta anche *texture* [HHG 85].

Pertanto, se ció che rende un documento testo é la dipendenza interfrasale, allora é importante individuare con quali strumenti tale dipendenza viene realizzata. Hasan e Halliday definiscono la **coesione** come *"the set of possibilities that exists in the language for making text hang together "*, piú precisamente come l'insieme di caratteristiche grammaticali e lessicali grazie alle quali una frase all'interno di un testo viene collegata alle precedenti e alle successive.

Per fare un esempio, consideriamo il seguente brano:

Antonio José Bolívar sapeva leggere, ma non scrivere. Al massimo riusciva a scarabocchiare il suo nome quando doveva firmare qualche documento...[2]

e analizziamone il significato. La prima frase utilizza un'ellissi, infatti viene omessa la parola scrivere che una corretta costruzione grammaticale richiederebbe, mentre nella seconda frase il soggetto (omesso) é Antonio José Bolívar. Appare evidente come, grazie alla coesione, possiamo interpretare correttamente alcuni elementi del discorso a partire da altri espressi in precedenza.

Peró, esiste anche un legame fra segmenti diversi di uno stesso testo. Si tratta di una forma di coesione di piú alto livello e viene detta coerenza o *coherence* . La coerenza si ottiene tramite relazioni di:

[2]Luis Sepúlveda. *Il vecchio che leggeva romanzi d'amore*, Ugo Guanda Editore, Parma, 1993.

Elaboration: descrizione di alcuni concetti in dettaglio, specialmente a sostegno di una tesi.

Cause/Effect: relazioni di causa/effetto fra due concetti;

Explanation: spiegazione di alcuni concetti in modo che possano essere compresi piú facilmente.

Riconoscere la coerenza significa determinare la struttura del discorso espressa in un testo. Purtroppo, non si tratta di un'operazione banale, perché fortemente dipendente dal punto di vista e dalle conoscenze del lettore. Infatti, distinguere relazioni di elaboration e di explanation puó essere, a volte, molto difficile.

Fortunatamente anche le relazioni di coerenza possono essere espresse tramite relazioni di coesione. Queste ultime sono oggettive e classificabili in due categorie principali: le relazioni di coesione grammaticale e le relazioni di coesione lessicale.

Le relazioni di coesione grammaticale possono essere classificate in 4 categorie principali:

Reference: collegamento a qualche elemento giá apparso nel testo. Spesso, questo tipo di collegamento permette di interpretare correttamente il significato dell'elemento in esame.

Ellipsis: eliminazione di uno o piú termini da una frase, in modo che il significato possa essere compreso ugualmente.

Substitution: utilizzo di un termine al posto di un altro, in modo perfettamente equivalente ai della comprensione del significato.

Conjunction: utilizzo di termini che uniscono fra loro altre parole o frasi, per ottenere un legame logico e semantico. Si tratta di una relazione esplicita, ad esempio espressa tramite le congiunzioni e, o, etc...

Invece, la coesione lessicale viene espressa tramite la scelta dei termini che devono apparire in un testo scritto. A questo riguardo sono state individuate due categorie principali:

Reiterazione: (*reiteration*) di un concetto, ottenuta tramite la semplice ripetizione di un termine, l'uso di sinonimi, sostituzione di un termine specifico con uno più generico e/o viceversa (*hyponymy* e *hypernymy*). Più precisamente sono stati riconosciuti tre diversi tipi di reiterazione.

1. *Reiterazione con identità nel riferimento*, ad esempio:
   Yesterday Jane lost her *cat*. Fortunately, the *cat* was in the neighbour's garden.
2. *Reiterazione senza identità nel riferimento*, ad esempio:
   Yesterday Paul bought a *dog*. He likes *dogs* very much.
3. *Reiterazione attraverso ipernimi etc...*
   Jane has a *cat* and Paul has a *dog*. They like *animals* very much.

Collocation: uso di termini che tendono ad apparire assieme in una frase, in un discorso comune. Collocation puó essere ottenuta per relazioni semantiche di tipo sistematico come *meronymy* e *antonymy*, oppure attraverso membri di un set ordinato (*one, two, three*) membri di un set non ordinato (*while, black, red*), oppure per relazioni non sistematiche. L'individuazione di queste ultime é piú difficile perché dipendenti dal contesto. Infatti le relazioni non sistematiche si riferiscono all'uso di termini che rappresentano la descrizione di un certo argomento.

1. *Systematic semantic relation*, ad esempio
   Jane likes *white* cats. She does not like *black* ones.
2. *Non-systematic semantic relation*
   Jane is in the *garden*. She is *digging* flowers.

L'importanza del lavoro di Halliday e Hasan é dato dalla dimostrazione di come la coesione lessicale determini la proprietá di texture. Infatti, analizzando vari tipi di testo, hanno calcolato la frequenza d'uso delle singole tecniche di coesione e hanno osservato che, da sola, la coesione ottenuta attraverso aspetti lessicali, viene utilizzata nel 40 per cento dei casi ed oltre. A questo riguardo un risultato analogo é stato ottenuto anche da Hoey [HOE 91], il quale afferma:

"The study of the greater part of cohesion is the study of lexis ".

Pertanto, se esistono relazioni di coesione lessicale fra un termine ed un altro, allora esistono relazioni di coesione lessicale anche fra un gruppo di termini ed un altro. Unendo questi gruppi (sequenze) di termini, si ottengono quelle che sono definite come **catene lessicali** , o *lexical chains*.

Secondo Hasan ([HAC 84], le catene lessicali possono essere classificate e descritte nel seguente modo [3]:

Identity chains (IC), che contengono termini che si riferiscono allo stesso oggetto. Sono caratterizzate dall'uso di pronomi, ripetizione di termini o di parole equivalenti.

Similarity chains (SC), in cui i termini sono collegati fra loro da relazioni semantiche che vanno al di lá del contenuto di un documento scritto. Si tratta di relazioni sovra-testuali, che hanno la loro ragione d'essere nel linguaggio stesso.

Le catene lessicali forniscono un importante indizio per verificare la coerenza all'interno di un testo e contribuiscono a delineare la struttura del discorso. Queste due caratteristiche - coerenza e struttura del discorso - permettono di comprendere il significato più ampio di un testo.

7.3 Algoritmi per l'estrazione di catene lessicali

7.3.1 Algoritmo di Morris e Hirst

Jane Morris e Graeme Hirst [MOR 91] utilizzano le proprietà coesive di un testo per determinare i significati in esso contenuti. Se le catene lessicali rappresentano le unità concettuali che contribuiscono ad esprimere il significato

[3]Le catene lessicali di Hasan e Halliday sono, cosí come sono state riassunte in [HIR 98] liste lineari di termini e relazioni fra termini. Vi sono sette tipi di relazioni *word-to-word*: *sinonimy, meronymy, repetition, taxonomy, atonomy, co-taxonomy, co-meronymy.*
Mentre i tipi di relazioni *chain-to-chain* sono sei: *epithet-thing, medium-process, process-phenomenon, actor-process, process-goal, process-location of process.*

di un testo, allora esse possono risolvere eventuali ambiguitá che i metodi statistici classici dell'Information Retrieval non individuano. Infatti, il significato di un termine non sempre é univoco, ma deve essere interpretato proprio in base al contesto in cui appare. Inoltre, oltre a questo aspetto locale del testo, le catene lessicali contribuiscono a determinare la coerenza e la struttura del discorso. Una catena lessicale non é confinata all'interno di una sola frase, ma si dipana all'interno dell'intero testo.

Il contributo fondamentale di Morris e Hirst allo sviluppo della teoria delle catene lessicali consiste nell'introduzione di un *thesaurus* come base di conoscenza per estrarre le relazioni fra termini. Il thesaurus scelto (*Roget's International Thesaurus*[4]) è organizzato in una struttura gerarchica in cui il primo livello è formato da 8 classi[5] Ogni classe è suddivisa in *sub-classes* e ogni sub-class in *sub-subclasses*. All'interno delle sub-subclasses sono sviluppate 1042 categorie base, organizzate per coppie antitetiche e composte da paragrafi che raggruppano termini correlati all'interno di una stessa categoria sintattica . Le relazioni inter-paragrafo e/o inter-categoria sono mantenute tramite puntatori. In Figura 7.1 è illustrata l'organizzazione del Roget's Thesaurus.

L'algoritmo proposto si compone di due fasi, la prima delle quali è dedicata all'individuazione delle **parole candidate** per essere incluse nelle catene lessicali. Ad esempio, pronomi, preposizioni e ausiliari verbali possono essere ignorati, così come termini di uso comune (stop words).

Nella seconda fase vengono determinate le relazioni tra le parole candidate al fine di costruire le catene lessicali. Le relazioni fra termini sono di 5 tipi e sono basate sull'organizzazione del Thesaurus. Ad esempio, due parole sono in relazione se hanno una categoria in comune oppure se la categoria di una parola può essere raggiunta grazie ad un puntatore contenuto nella categoria dell'altra sia in maniera diretta che indiretta. L'organizzazione di questo thesaurus è sicuramente ottima per recuperare le relazioni fra termini appartenenti a categorie sintattiche diverse, ma può determinare incongruenze se si permette di seguire tutti i puntatori indiscriminatamente. Se un puntatore collega un termine a ad un termine b ed un puntatore collega b con c, allora

[4] 4th Edition, 1997.

[5] *Abstract Relations, Space, Physics, Matter, Sensation, Intellect, Volition, Affections.*

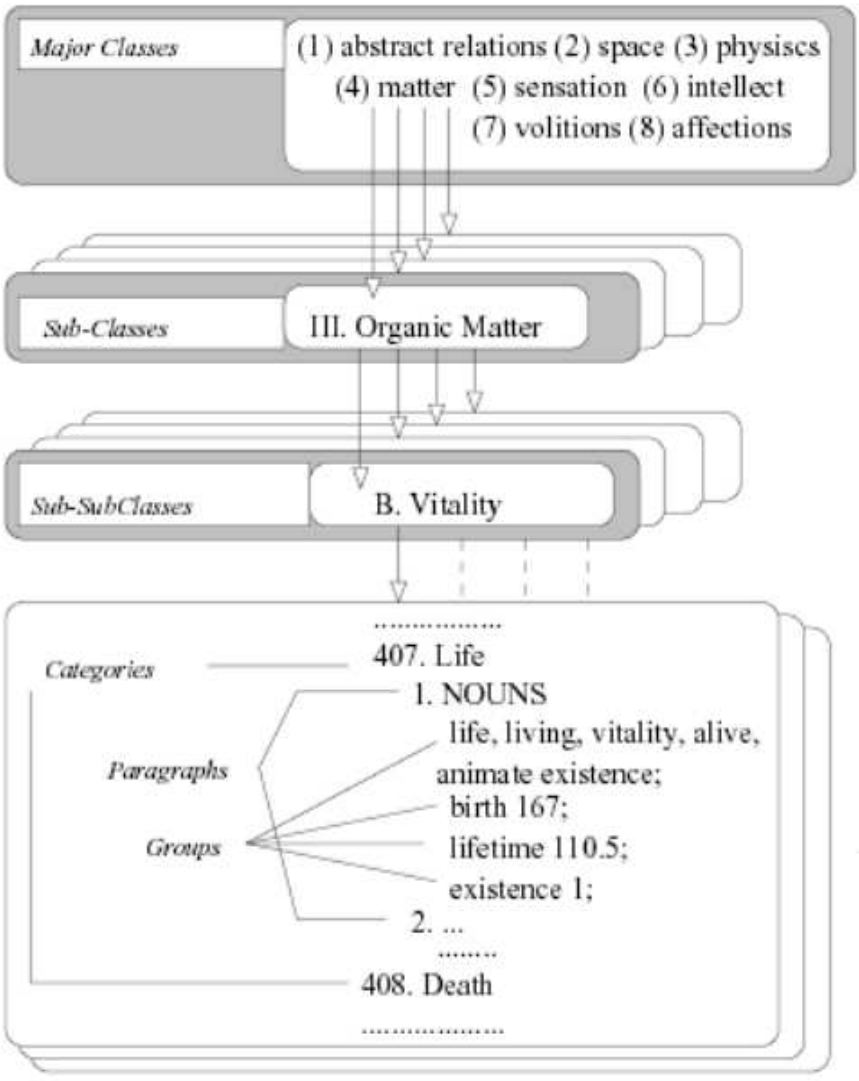

Figura 7.1: Organizzazione del Roget's Thesaurus.

per effetto della proprietà transitiva si può dire che a è in relazione con c. Però, a e b sono semanticamente più vicini rispetto ad a e c e ciò dovrebbe essere preso in considerazione assegnando alle catene un punteggio. L'obiettivo è quello di evidenziare alcune catene *strong* che meglio concentrano i concetti espressi in un testo. Secondo Morris e Hirst il punteggio dovrebbe essere assegnato pesando tre diversi fattori:

1. Reiteration - un termine ripetuto più volte indica una maggiore rilevanza del concetto associato, pertanto il punteggio della catena in cui compare dovrebbe aumentare.

2. Length - maggiore è il numero di termini in una catena, maggiore è la coesione semantica.

3. Density - è calcolabile come il rapporto fra il numero di termini della catena e il numero delle parole nell'intero testo. Un'elevata densità indica che la struttura del discorso è ampiamente catturata dalla catena.

Algoritmo 7 - Morris & Hirst' s Algorithm for Lexical Chaining

Repeat
READ next word
If *word is suitable for lexical analysis* **then**
CHECK for chains within a suitable span
CHECK thesaurus for relationships
CHECK other knowledge sources if available
If *chain relationship is found* **then**
INCLUDE word in chain
CALCULATE chain so far
End if
If *there are words that have not formed a chain for a suitable number of sentences* **then**
ELIMINATE words from the span
End if

CHECK new word for relevance to existing chains
that are suitable for checking
ELIMINATE chains that are not suitable for checking.
End if
End Repeat

L'algoritmo proposto funziona nel 90% dei casi analizzati in [MOR 91], ma vi sono delle limitazioni dovute al *Roget's Thesaurus*, perché ad esempio non contiene (ovviamente) le voci relativi a toponimi (nomi di cittá, strade, etc...) o ai nomi propri di persone. Inoltre, la tecnica fallisce quando é richiesta ulteriore conoscenza riguardo un argomento, rispetto a quella contenuta nel *thesaurus*. Morris e Hirst consigliano di sopperire alla mancanza di conoscenza integrando il thesaurus con altri strumenti, peraltro non chiaramente specificati.

In realtà l'algoritmo di Morris e Hirst non è mai stato implementato, perchè nel 1991 non era disponibile una versione completa del Roget's Thesaurus in formato elettronico.

7.3.2 Algoritmo di Okamura e Honda

In [OKA 94] Okamura e Honda applicato il metodo delle catene lessicali alla lingua giapponese e dimostrano come questa tecnica può sfruttare la conoscenza contenuta in thesaurus diversi dal Roget's mantenendo la capacità di catturare la struttura del discorso. Inoltre, Okamura e Honda osservano che un effetto secondario nella costruzione delle catene lessicali è la disambiguazione dei termini di un testo. Si consideri ad esempio il seguente brano:

> In the universe that continues expanding, a number of stars have appeared and disappared again and again. And about ten billion years after the birth of the universe, in the same way as the other stars, a primitive galaxy was formed with the primitive sun as the center.

e si supponga di aver costruito la catena C formata da {*universe, star, universe, star, galaxy*}. Se la parola *earth* comparisse nelle successive frasi del testo e venisse aggiunta alla catena C, allora il termine assumerebbe il significato di *planet* piuttosto che quello di *ground*.

La generazione delle catene lessicali secondo Okamura e Honda deve privilegiare in primo luogo l'aspetto della località dei riferimenti dei significati. Ciò significa che da un testo possono essere estratte più catene, ma non tutte sono contemporaneamente attive e per una disambiguazione corretta occorre ordinare le catene in ordine di salienza (*salience*), parametro che indica sia il grado di attività della catena che il grado di plausibilità. L'estrazione delle catene lessicali prevede una fase preliminare in cui il testo (giapponese) viene automaticamente segmentato in una sequenza di parole candidate.

Quando possibile, il sistema effettua una disambiguazione delle parole candidate focalizzata sulle frasi in cui compaiono. Alcuni termini sono correttamente disambiguati quando, ad esempio, hanno un solo significato oppure compaiono assieme ad altre parole per cui l'interpretazione risultante è univoca. Nella maggior parte dei casi, però, la disambiguazione avviene determinando le relazioni interfrasali, pertanto ogni parola candidata viene inserita nella migliore catena verso la quale verifica una coesione lessicale. Per migliore catena si intende la catena con il miglior punteggio di salienza. Aggiungere un termine ad una catena permette di aumentare il punteggio di salienza. Inoltre, un'inserimento in una catena permette di disambiguare progressivamente quei termini che nell'analisi intra-frasali sono rimasti irrisolti.

7.3.3 Algoritmo di Hirst e St-Onge

Graeme Hirst e David St-Onge [HIR 98] utilizzano le catene lessicali per risolvere il problema del riconoscimento e la correzione di quei particolari errori di ortografia detti *malapropisms*. Un malapropism si verifica quando al posto di una certa parola compare un termine che, pur esistendo nel linguaggio in esame, ha un significato completamente diverso (*Malapropism detection problem*).

Hirst e St-Onge utilizzano la classificazione degli errori data da Kukich (1992):

sintattici: The students are doing **there** homework;

semantici: o *malapropisms*, He spent his summer travelling around the **word**;

strutturali: I need **three** ingredients: red wine, sugar, cinnamor and cloves;

pragmatici: He studies at the University of Toronto in **England**.

Mentre i tradizionali *spelling checker systems* si limitano ad identificare gli errori di ortografia che danno origine a sequenze di lettere che non sono parole di senso compiuto, i *malapropism detection systems* devono, necessariamente, comprendere il significato del testo.

Il *thesaurus* utilizzato da Hirst e St-Onge é *WordNet*. *WordNet* é però strutturalmente e concettualmente diverso dal *Roget's Thesaurus*. Infatti, *WordNet* utilizza i *synonymy sets* o *synsets*, come unitá logiche fondamentali. Ogni *synset* é composto da un certo numero di termini che risultano avere lo stesso significato. Una parola puó corrispondere a piú synset, poiché ad una parola puó essere associato piú di un significato (ved. cap.1 e Fig. 7.2).

L'utilizzo di un thesaurus diverso implica una ridefinizione e un'estensione delle possibili relazioni lessicali, le quali vengono classificate in ordine di importanza:

Extra-Strong Relation: quando la parola viene ripetuta piú volte, evento che é indice, al pari dei sistemi di tipo statistico, dell'importanza del termine all'interno del testo;

Strong Relation: essenzialmente sono dovute a tre fenomeni diversi. Si ha una relazione forte del primo tipo (fig. 7.3) quando due parole condividono un *synset*, cioé quando hanno, fra tutti i possibili *synsets*, un *synset* in comune. Una relazione del secondo tipo (fig. 7.4)é individuabile quando esiste un link orizzontale fra un *synset* di una parola e un *synset* di un altro termine. Tale link puó essere ad esempio di *antonymy*. Infine, il terzo tipo (fig. 7.5) riguarda l'eventualitá in cui non

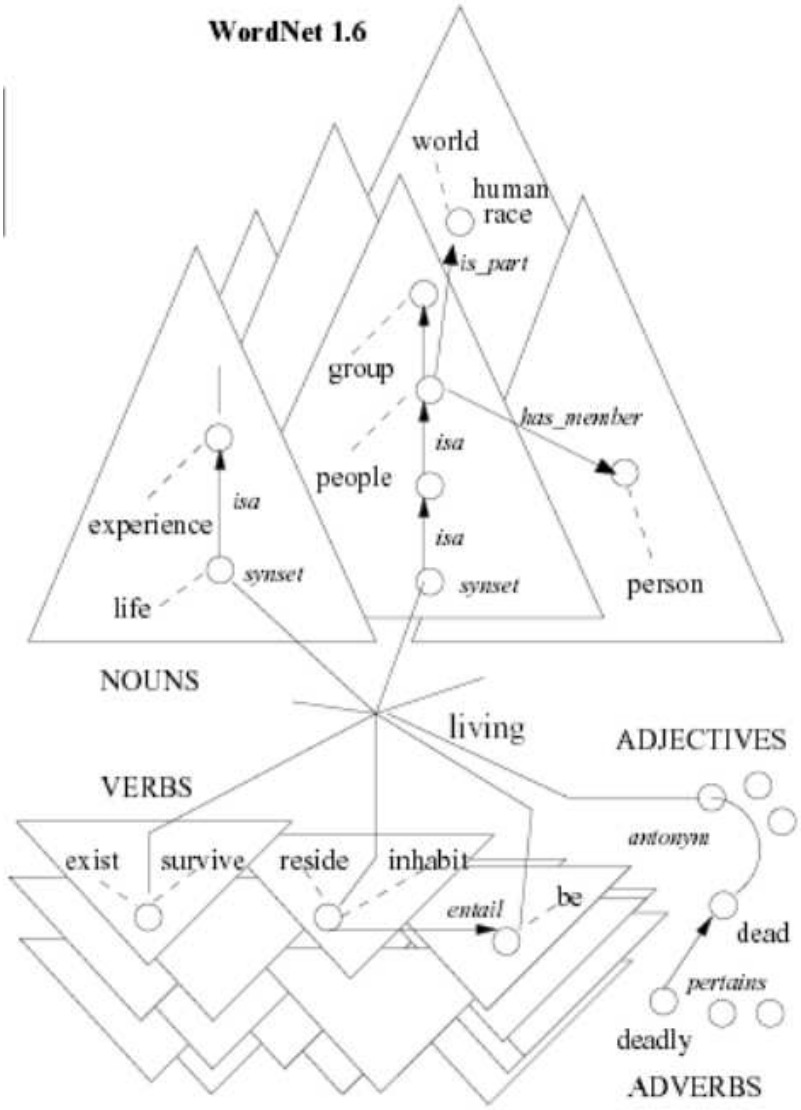

Figura 7.2: Struttura di WordNet.

esiste un collegamento di alcun genere fra due *synsets* se una parola é una combinazione dell'altro termine. E' il caso abbastanza frequente nella lingua inglese, in cui una parola viene scritta come si trattasse di un solo termine, ma in realtá é formata da piú elementi separati da un trattino (es.: *brown-eyed girl...*)

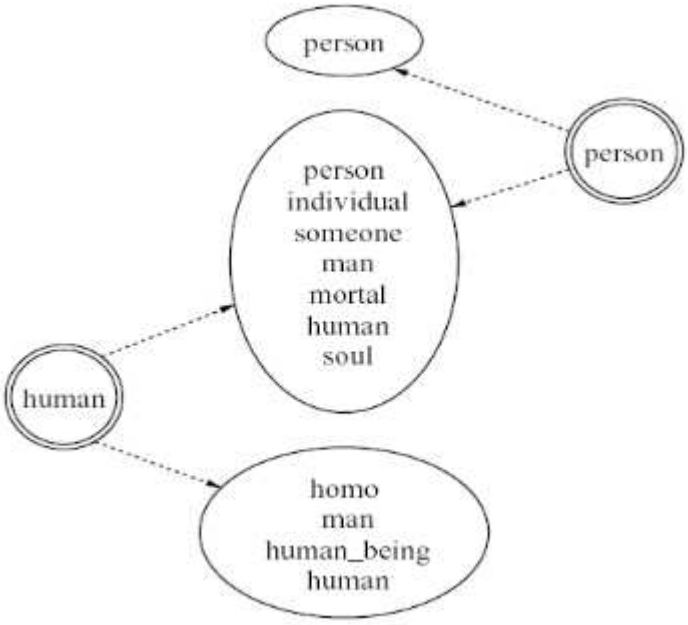

Figura 7.3: Strong Relation: synset in comune

Medium-Strong Relation: quando esiste un percorso che collega due *synsets*. Un percorso (*path*) é una sequenza di un minimo di due fino ad un massimo di cinque link fra i *synsets*. Un percorso é ammissibile se corrisponde ad una delle forme mostrate in Figura 7.6),

ma non in una di quelle di Figura 7.7.

A differenza degli altri tipi di relazioni, il peso associato varia in funzione della lunghezza del percorso e del numero "cambi di direzione", secondo la formula

$$weight = C - PathLenght - K * NumChangeOfDirection$$

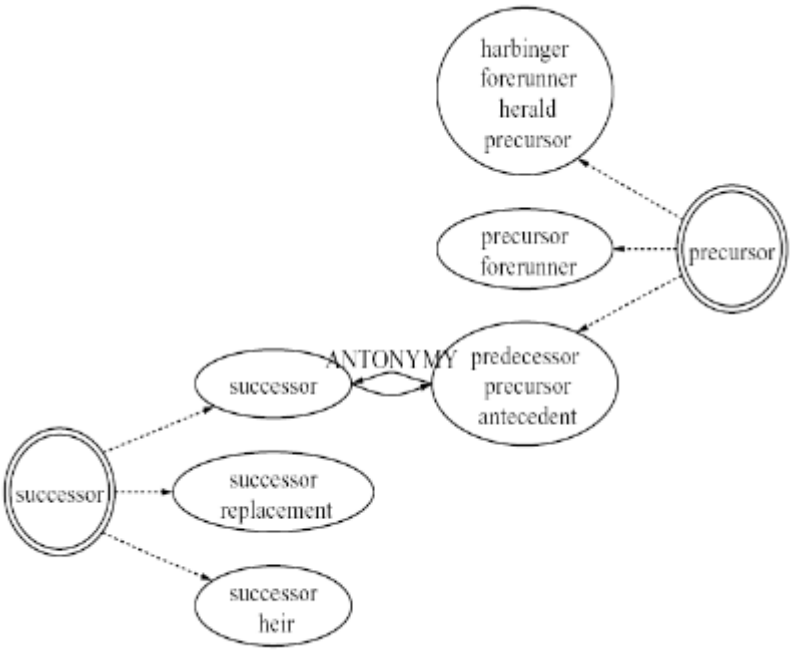

Figura 7.4: Strong Relation: un link orizzontale

dove C e K sono due parametri costanti.
Ció significa che piú lungo é il percorso e maggiore é il numero di cambiamenti di direzione, allora minore é il peso. Il significato indicato da Hirst e St-Onge é il seguente: se un percorso *multilink* esiste fra due *synsets* é indicativo della prossimitá o vicinanza semantica, per cui la semantica di ogni relazione lessicale deve essere presa in considerazione. In particolare, una direzione dal basso verso l'alto (*upward direction*) corrisponde ad una generalizzazione.

Ad esempio in Figura 7.8, il link *upward* fra *apple* e *fruit* significa che *fruit* é un synset semanticamente piú generale di *apple*. Al contrario, un link di senso inverso corrisponde ad una specializzazione. I link orizzontali sono meno frequenti e raramente un synset ne ha piú di uno. Questi link si riferiscono infatti a campi molto specifici. Ad esempio nella Figura 7.4 il link orizzontale fra *successor* e *predecessor, precursor, antecedent* é una specificazione molto accurata del significato della parola *successor*. Per assicurare che il percorso corrisponda ad una rela-

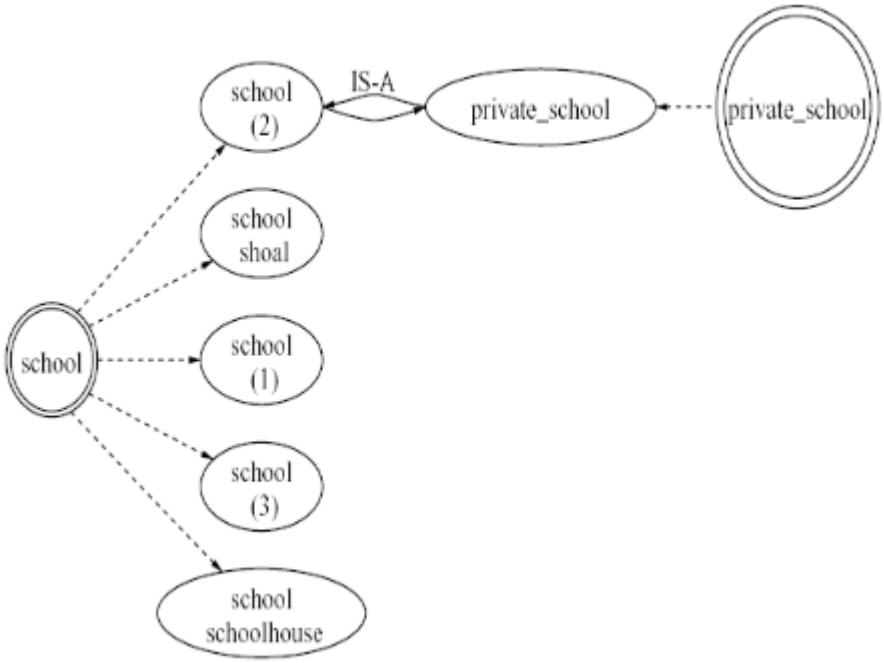

Figura 7.5: Strong Relation: compound word

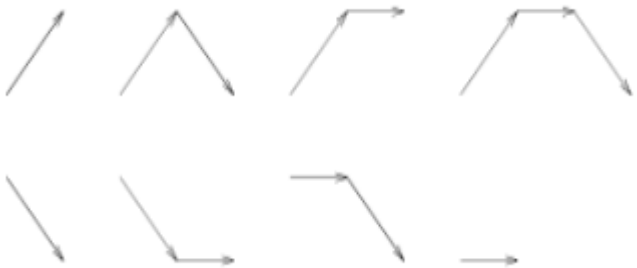

Figura 7.6: Medium-Strong Relation: path ammissibili

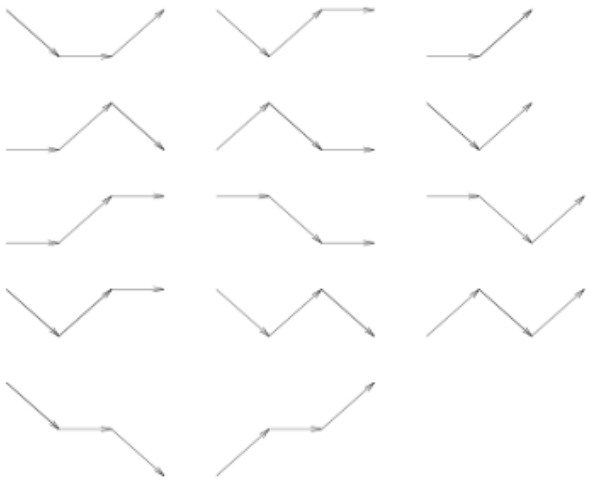

Figura 7.7: Medium-Strong Relation: path non ammissibili

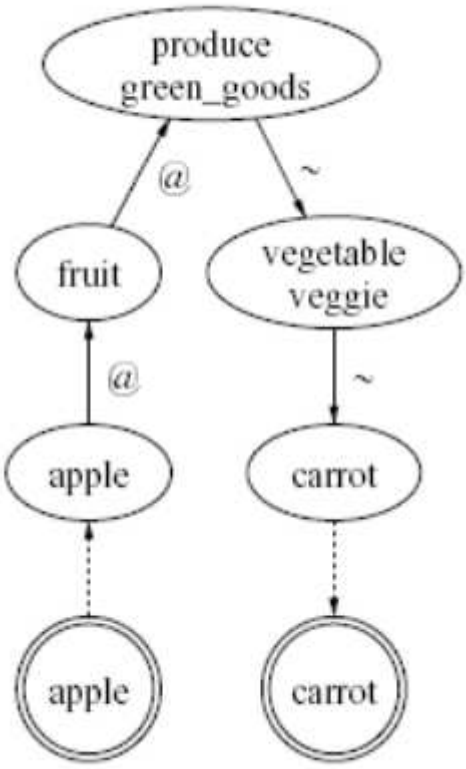

Figura 7.8: Esempio di relazione fra due parole

zione possibile e ragionevole fra la sorgente e la parola target, due regole sono state definite per determinare quali siano i percorsi ammissibili:

- nessun'altra direzione puó precedere un upward link. Ció significa che se esiste giá un link orizzontale e/o di specializzazione), allora non é permettere estendere la catena con un link upward;
- al piú di un cambio di direzione viene ammesso. Ogni cambio di direzione costituisce un grande passo semantico (aumento la distanza) e ció deve essere limitato, ma vi é un'eccezione. Infatti, é permesso utilizzare un link orizzontale per ottenere una transizione fra un upward ed un download. I link orizzontali corrispondono ad una piccola distanza semantica, per cui unire due superordinate non implica un grande gap semantico.

Una catena lessicale può sembrare un semplice elenco di parole, ma ha una struttura interna più complessa. Consideriamo la costruzione di una catena lessicale (Figura 7.9) che contenga i termini *economy, sectors, economic_system*. Per ogni elemento da inserire in una catena viene creato un record che viene aggiunto ad una lista costruita dinamicamente. Il primo termine (*economy*) è il primo elemento della lista. Il secondo termine viene aggiunto in testa e il record creato per memorizzare il lemma contiene anche un puntatore verso l'elemento con cui è posto in relazione. Infine, il termine *economic_system* viene riferito ad *economy*. L'ordine con cui i termini compaiono all'interno della lista non è significativo delle relazioni lessicali: esse possono essere ricostruite grazie ai puntatori.

Una lemma, però, può essere incluso in più synsets, per cui la gestione dei puntatori deve essere accurata. Infatti, quando una parola inizia una nuova catena, si considerano tutti i synsets in cui compare (Figura 7.10).

L'inserimento di un secondo termine grazie ad una relazione extra-strong comporta il collegamento fra tutti i synset attivi (Figura 7.11).

Invece, quando la relazione è forte si connettono tutte le coppie dei relativi synsets.

Quando la relazione è medio-forte, la coppia o le coppie di synsets il cui peso è il maggiore (o di pari peso) fra tutte le coppie vengono connessi.

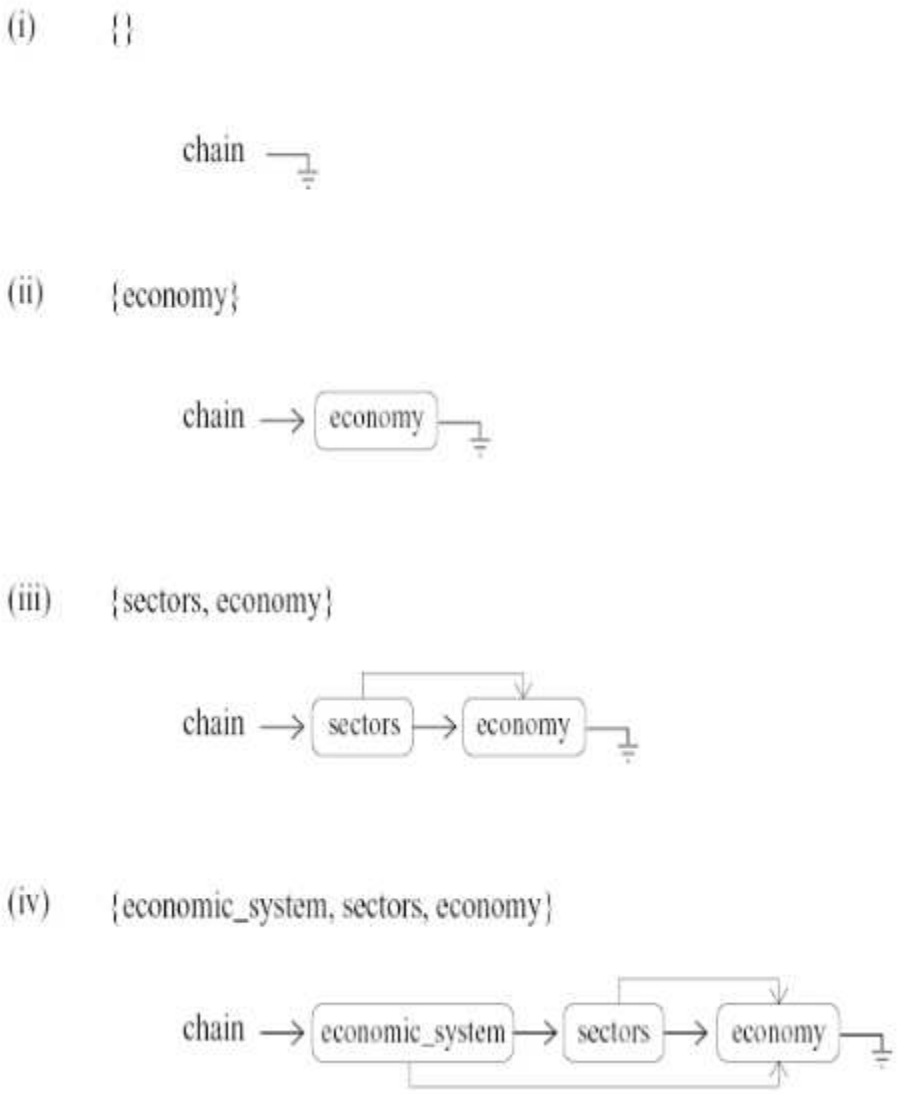

Figura 7.9: Esempio di costruzione dinamica di una catena lessicale.

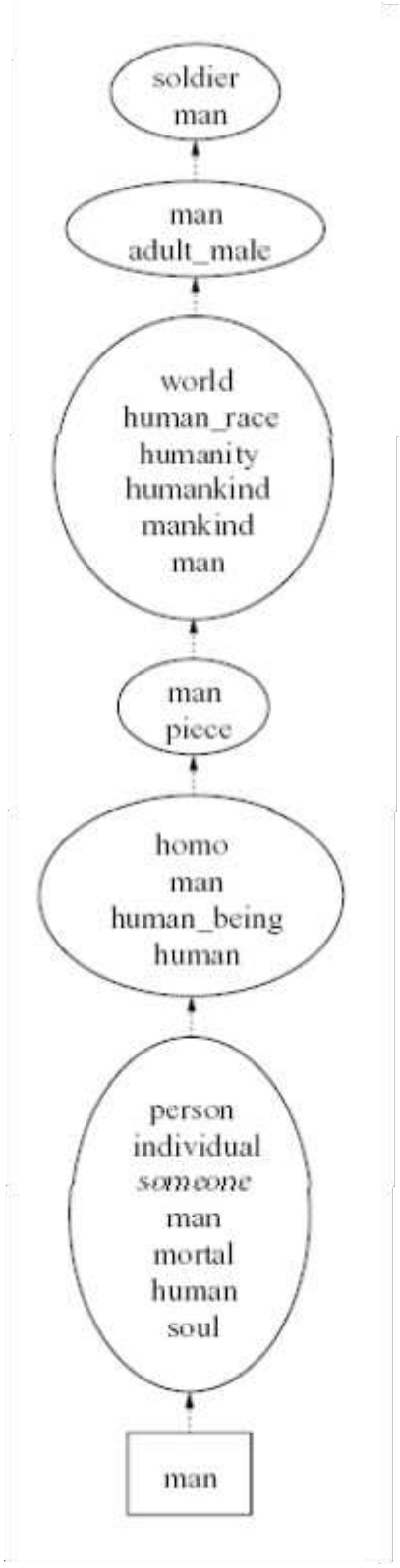

Figura 7.10: Creazione di una nuova catena.

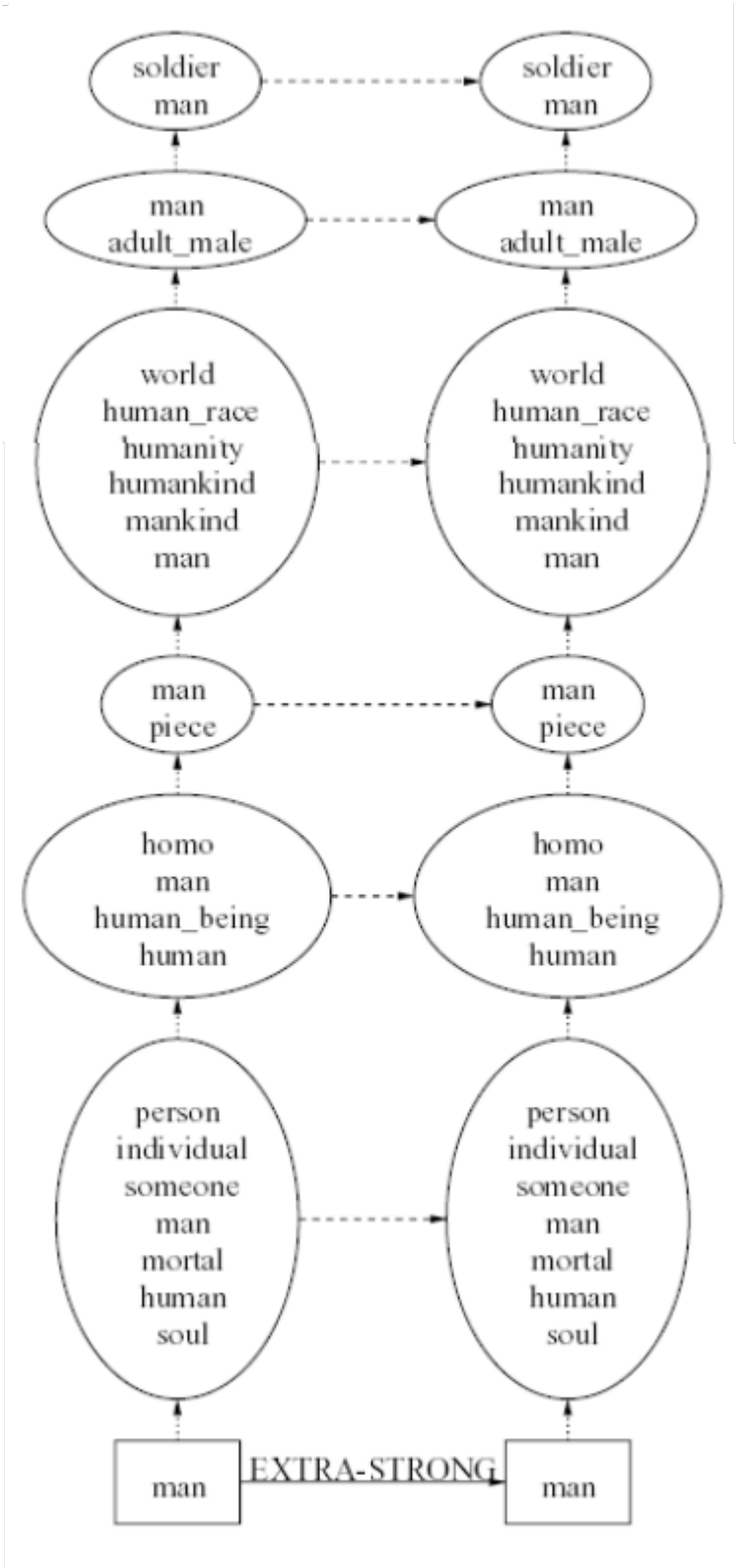

Figura 7.11: Inserimento di una relazione extra-strong

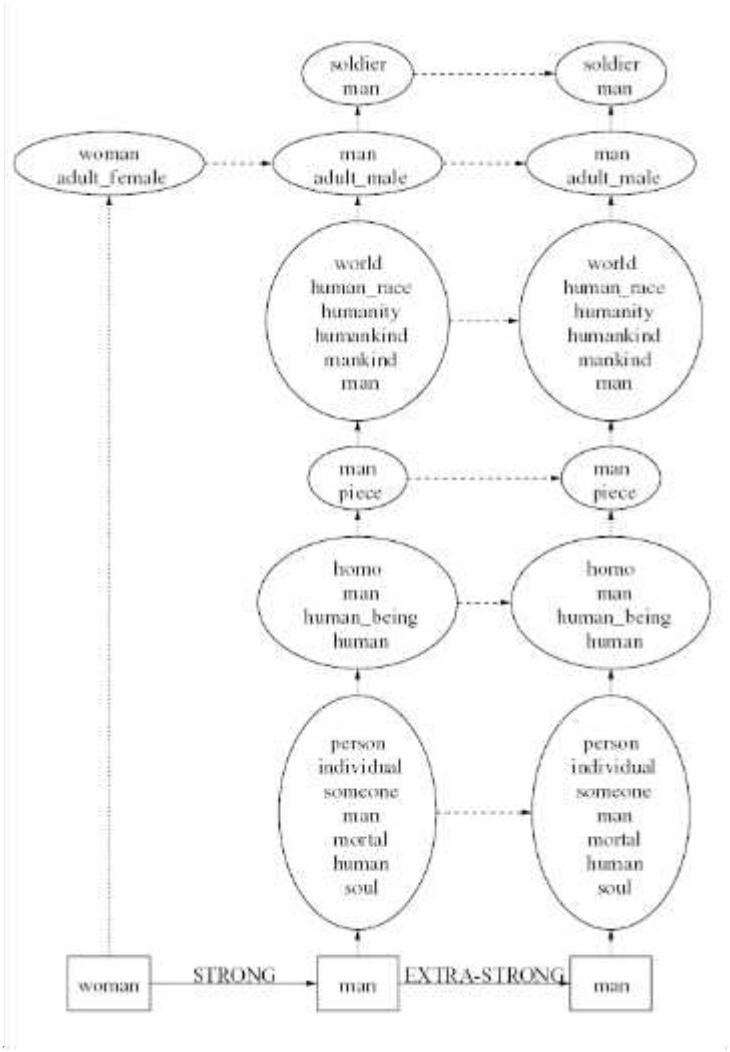

Figura 7.12: Inserimento di una relazione strong

Dopo aver connesso le parole, ogni synsets non collegato della nuova parola è cancellato e la catena è scandita per rimuovere, dove possibile, ogni altro synset. La rimozione dei synsets permette di determinare progressivamente il s

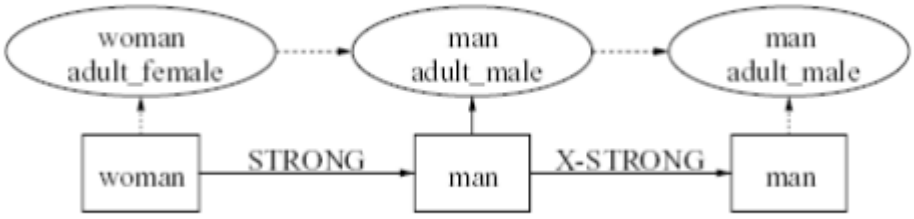

Figura 7.13: Risultato finale.

Anche l'algoritmo di Hirst e St-Onge privilegia l'inserimento di nuovi lemmi in catene con elevato grado di salienza. Il grado di salienza è visto, in questo caso, in stretta connessione con i tipi di relazione lessicale (extra-strong, strong, medium-strong). Un lemma è generalmente inseribile in più catene lessicali. Se esiste una relazione extra-strong verso una catena allora il termine viene inserito, altrimenti si procede alla ricerca di una relazione strong limitata alle catene formate o aggiornate nelle 7 frasi precedenti. Se non vi sono relazioni forti, si cercano relazioni medium-strong nelle 3 frasi precedenti.

Poichè il lavoro si basa sui sistemi di spell-checking, allora l'uso delle catene lessicali è importante perchè le parole isolate, per le quali sono state create catene atomiche, possono essere, con buona probabilità dei *malapropisms*.

7.3.4 Algoritmo di Barzilay e Elhadad

L'algoritmo proposto da Barzilay e Elhadad [BAR 97a, BAR 97b] differisce dai precedenti perchè tenta di applicare una strategia meno "greedy" per la costruzione delle catene lessicali. Infatti, l'algoritmo di Hirst e St-Onge applicato al seguente testo:

Mr. Kenny is the **person** that invented an anesthetic **machine** which uses **micro-computers** to control the rate at which

> an anesthetic is pumped into the blood. Such **machines** are nothing new. But his **device** uses two **micro-computers** to achieve much closer monitoring of the **pump** feeding the anesthetic into the patient.

prevederebbe la creazione di una nuova catena per la parola *Mr.*. *Mr.* appartiene ad un solo synset, per cui è perfettamente disambiguata. La parola *person* è in relazione con questa catena quando assume il significato di *a human being*. Il secondo lemma viene pertanto aggiunto ad una catena grazie ad una relazione medium-strong. La terza parola *machine* ha anche un significato di *efficient person*, che in olonimia con *person* viene aggiunta alla catena. La disambiguazione di *machine* non è avvenuta in modo corretto, perchè avrebbe dovuto assumere il significato di *device*. Pertanto, la disambiguazione non deve avvenire in modo greedy e la decisione su quali synset rimuovere deve essere presa in un momento successivo. L'algoritmo proposto da Barzilay e Elhadad elabora il testo in esame nel seguente modo:

1. viene creato un primo nodo contenente la parola *Mr.*;
2. la parola *person* ha due significati, la scelta sul significato viene rimandata e viene creata una nuova catena (*splitting*), così come in Figura

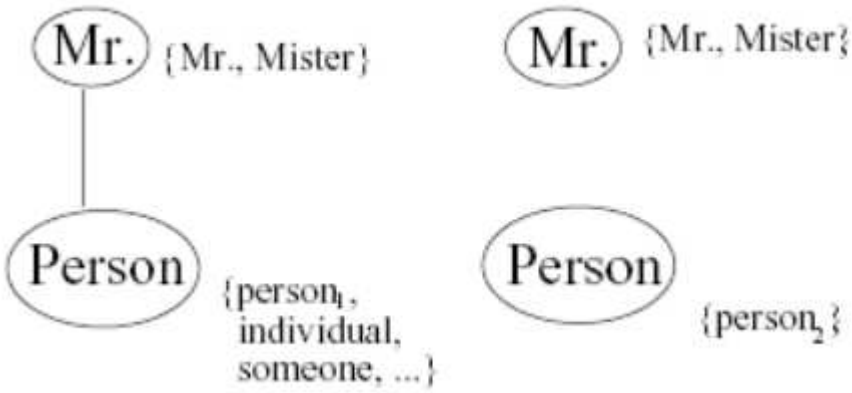

Figura 7.14: Split di una catena

Barzilay ed Elhadad definiscono *componente* una lista di possibili interpretazioni che sono mutualmente esclusive. Le parole contenute nei

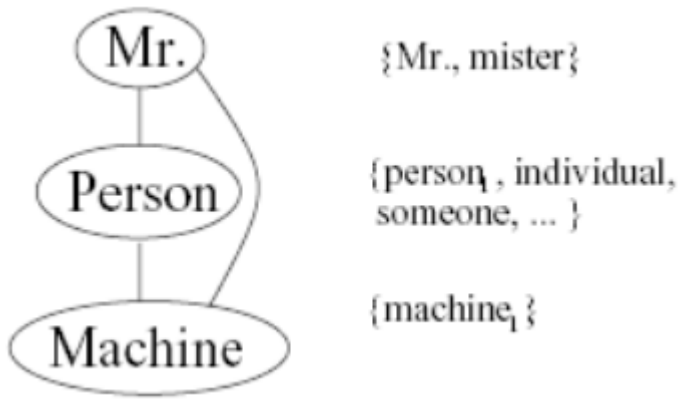

Figura 7.15: Inserimento di un nuovo termine in relazione con quelli esistenti

componenti si influenzano l'una con l'altra durante la determinazione dei significati.

3. la parola candidata *anesthetic* non è in relazione con alcuna parola del primo componente, così viene creata una nuova catena.

4. la parola *machine* appartiene a cinque synset. Quando assume il significato di *efficient person* è in relazione con la parola *person* della prima catena, pertanto viene inserita così come in Figura 7.15. Gli altri significati non vengono però eliminati subito, ma vengono memorizzati nella seconda catena (Figura 7.16)

5. L'inserimento dei termini *micro-computer*, *device* e *pump* provoca un incremento elevato del numero di possibili alternative. Sotto l'ipotesi che il testo sia coeso, Barzilay e Elhadad scelgono la migliore fra tutte le interpretazioni. Per interpretazione migliore si intende la catena che ha il maggior punteggio calcolato in termini di numero e tipo di relazioni che sono state ricostruite al suo interno. Le migliori catene sono quelle nelle figure 7.17 e 7.18

   Per la precisione, Barzilay ed Elhadad osservano che la lunghezza (*Length*) della catena è un'ottimo parametro per discriminare fra catene forti

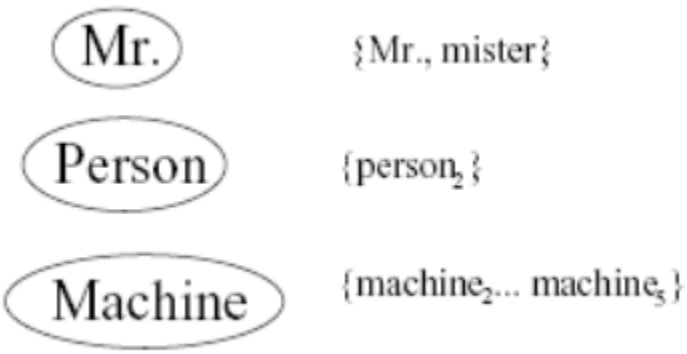

Figura 7.16: Inserimento di un termine con più significati

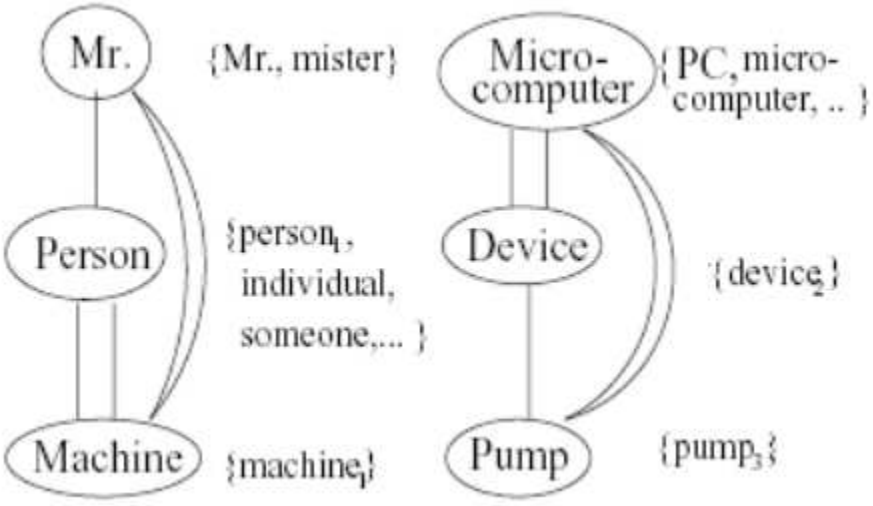

Figura 7.17: Strong Chain (a)

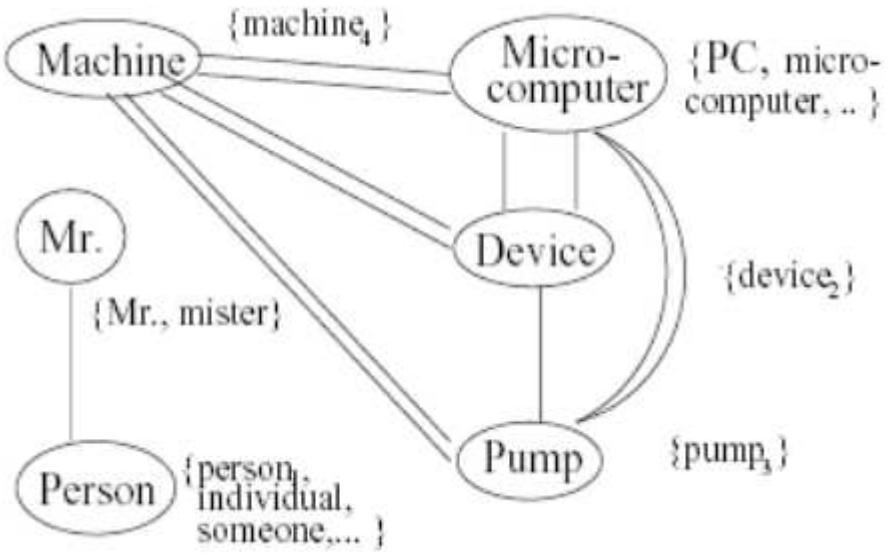

Figura 7.18: Strong Chain (b)

e meno forti. Inoltre, in una catena può essere importante determinare l'omogeneità. Se un termine compare in una catena più volte, allora il concetto associato viene rappresentato in maniera forte. Al contrario, se la catena è composta di termini tutti diversi, allora non è possibile individuare un concetto prevalente. L'omogeneità viene calcolata tramite la formula

$$HomogeneityIndex = 1 - N_{do}/N_{cw}, \tag{7.1}$$

dove N_{do} è il numero di parole distinte e N_{cw} è il numero totale di termini di una catena.

Il punteggio di una catena viene calcolato come:

$$Score(Chain) = Length * HomogeneityIndex \tag{7.2}$$

Le catene forti sono quelli per le quali vale la relazione:

$$Score(Chain) > Average(Scores) + 2 * StandardDeviation(Scores) \tag{7.3}$$

7.3.5 Algoritmo di Silber e McCoy

Gregory Silber e Kathleen McCoy ([SIL 02]) sviluppano ulteriormente l'algoritmo di Barzilay e Elhadad. L'algoritmo di Barzilay e Elhadad é offre un approccio sicuramente meno *greedy* rispetto ai precedenti, ma il suo limite è la complessità computazionale. Infatti, rimandare la scelta dei synset da rimuovere provoca un proliferare di possibili interpretazioni, la cui gestione rende esponenziale la complessità computazionale.

Lo studio di Silber e McCoy si pone come obiettivo quello di rendere le catene lessicali applicabili anche in contesto come quello delle pubblicazioni Web. In primo luogo, quindi, ristudiano il problema della costruzione delle catene cercando di ottenere una complessità computazionale lineare, pur mantenendo tutte le possili interpretazioni.

L'idea di Silber e McCoy è di memorizzare implicitamente ogni possibile interpretazione in un struttura idonea, senza la necessità di creare le catene esplicitamente. In una fase preliminare il metodo di accesso a WordNet viene modificato, così da trasformare il file dei nomi in formato binario. Il formato binario viene poi mappato in memoria, per permettere di recuperare un elemento come se si trattasse di accedere ad un elemento di un array.

A differenzia degli altri algoritmi, ogni documento da elaborare subisce una fase in cui uno strumento detto *part of speech tagger* ne evidenzia i nomi.

La seconda fase dell'algoritmo prevede la costruzione di un array di meta-catene (*meta-chains*). La dimensione di questo array è il numero dei synset dei nomi di Wordnet più il numero dei nomi contenuti nel documento.

L'algoritmo proposto può essere sintetizzato nel seguente modo:

1. Per ogni nome nel documento sorgente, si formano tutte le possibili catene lessicali recuperando le relazioni di sinonimia, hyponyms, hypernyms e sibling. Queste informazioni sono memorizzate in un vettore indicizzato in base alla posizione che il significato del lemma ha in WordNet.

2. Per ogni nome nel documento sorgente, si usa l'informazione ottenuta nel passo precedente per inserire la parola in ogni *meta-chain*. Una

meta-catena é chiamata così perchè rappresenta tutte le possibili catene in cui la prima parola ha un numero di significato dato . L'inserimento di una parola in una meta-catena permette di aumentarne il punteggio nella misura in cui il nuovo elemento è semanticamente correlato a quelli già presenti.

L'algoritmo continua poi per cercare di trovare la migliore interpretazione di tutti i termini individuati. Ogni meta-catena è, dal punto di vista della rappresentazione, un grafo chiuso transitivo in cui i vertici sono condivisi. La migliore interpretazione del testo sarà data dal set di grafi all'interno dei quali vi è il maggior grado di connessione. In sintesi, la disambiguazione dei termini segue le seguenti fasi:

1. Per ogni parola nel documento:
   (a) Per ogni catena a cui la parola appartiene
      i. Trovare la catena il cui punteggio varierá in modo maggiore rimuovendo la parola da essa;
      ii. Inizializzare il punteggio della parola corrente in ogni altra catena alla quale appartiene a 0 e aggiornare il punteggio di tutte le catene nelle quali la parola riflette quella rimossa;

In questo modo si trovano i set di catene che massimizzano il punteggio complessivo senza averle costruite tutte esplicitamente. Questa é sicuramente l'aspetto piú interessante. Quindi, estraendo l'interpretazione del testo (set indipendente di catene non intersecanti) con il punteggio piú alto senza avere costuito nessuna altra interpretazione é ció che permette all'algoritmo di elaborare in tempo lineare.

7.3.6 Algoritmo di Fuentes e Rodriguez

Il lexical chainer di Fuentes e Rodriguez estende quello di Barzilay e Elhadad all'ambito multilingua. Il *thesaurus* utilizzato é ***EuroWordNet***. Le parole candidate ad essere incluse in una catena sono i nomi comuni, i nomi propri, i pronomi, i *named entities* e le *noun phrases*. L'estensione é possibile perché il

sistema prevede l'uso di un *tagger* semantico in grado di mettere in relazione i nomi comuni con gli altri elementi sopracitati.

I tipi di relazioni sono ridefiniti sulla base di EuroWordNet e seguono la falsariga di [HIR 98]. Infatti, è possibile distinguere tre diversi tipi di relazioni:

Extra-strong relations: intercorre fra una parola, le sue ripetizioni e le parole contenute nello stesso synset.

Strong relations: intercorre fra due parole collegate da una relazione di EuroWordNet.

Medium strong relations: se il percorso che collega i synsets delle parole ha una lunghezza maggiore di uno.

Le catene vengono costruite rispettando i vincoli determinati dal tipo di relazione fra i vari membri, per cui la lunghezza delle catene é limitata perché é limitata la sua crescita. Come nel sistema di Barzilay ed Elhadad, il testo viene segmentato e ogni singola unitá viene elaborata. Quando si estraggono *identity chains* si puó concatenare il risultato alle catene giá esistenti. In caso contrario occorre un'analisi piú approfondita per determinare a quale catena l'unitá lessicale considerata appartiene.

Per ottimizzare l'algoritmo, la crescita del numero di catene viene limitata eliminando catene che sembrano essere meno promettenti. Si vuole ottenere una rappresentazione del documento fatta soprattutto di catene rilevanti, alle quali viene assegnato un punteggio in cui la lunghezza e l'omogeneitá sono i parametri fondamentali.

7.3.7 Conclusioni

Le catene lessicali sono utilizzate in ambiti disparati, ma in generale si puó dire che l'algoritmo generale é rappresentabile come tre passi fondamentali:

1. Selezionare un set di parole candidate
2. Per ogni parola candidata, trovare una catena appropriata in base ad un criterio di correlazione fra i membri delle catene. Generalmente questo/i criterio/i di relazione fra parole si basa in termini di distanza fra

le parole (occorrenze) e la forma del percorso che le collega (soprattutto per WordNet).

3. Se una catena viene trovata, si inserisce la parola nella catena e la si aggiorna. Le catene hanno un punteggio calcolato sulla base di euristiche: la loro lunghezza, il tipo di relazioni fra le parole, la posizione nel testo, etc...

7.4 Sviluppi delle catene lessicali

7.4.1 Costruzione di link ipertestuali

Sempre nell'ambito dei sistemi di IR, vi sono studi circa la possibilitá di costruire automaticamente links ipertestuali all'interno di ampie collezioni di documenti, con particolare attenzione alle pagine web.

Infatti, i siti web sono, per loro natura, ricchi di iperlinks. Maggiori sono le "dimensioni" di un sito, maggiore é il numero di iperlinks contenuti, quindi maggiori saranno tempi e costi per il mantenimento della loro consistenza. L'aspetto della consistenza é particolarmente sentito da parte di quelle organizzazioni che si occupano di e-commerce. Il cliente potenziale é conscio che troppi link "morti" o scorretti sono indice di scarso controllo sulle risorse, per cui tende ad abbandonare il sito. Quindi, la mancata consistenza non é solo un fatto di immagine ma si puó trasformare in perdite economiche. É peró innegabile che costruzione, verifica e manutenzione dei links siano estremamente onerose se affidate ad operatori umani. Se questi iperlinks potessero, in qualche modo, essere costuiti in via automatica, allora il risparmio potrebbe essere notevole. La costruzione automatica di iperlinks si occupa, in realtá, di due problematiche differenti. Vi sono infatti **links strutturali** e **links semantici**. I links strutturali sono quelli che risentono dell'organizzazione logica del documento. Ad esempio, se in una pagina vi é un sommario, ogni voce del sommario deve essere collegata con il relativo capitolo, paragrafo etc....

I links semantici, invece, collegano documenti o parti di documenti in base alla loro similaritá semantica, vale a dire in base al contenuto, senza che vi sia una esplicita relazione strutturale fra di essi.

É evidente che in un sistema di Information Retrieval puó essere importante avere a disposizione tecniche per calcolare la similaritá semantica fra piú documenti: se un documento é rilevante per una query, allora anche i documenti "semanticamente collegati" possono essere rilevanti per la stessa query. Ció permette di risolvere in parte il problema dell'ambiguitá nell'uso delle sole keywords. Ad esempio, se due documenti "condividono" un numero sufficiente di termini, allora entrambi trattano dello stesso argomento. Se, al contrario, il numero di termini presenti in entrambi i testi é inferiore ad una certa soglia, allora i due documenti sono scorrelati. Queste affermazioni non sono vere, o comunque non sono cosí scontate, se si considerano problemi di *synonymy* e *polisemy*. Nel primo caso (parole diverse con lo stesso significato) un documento che parla di **hound** non risulta essere simile ad un documento che parla di **dog**. Nel secondo caso, (una parola con piú significati) potrebbero essere restituiti documenti che trattano di argomenti correlati a un diverso senso della parola. Quindi, é piú corretto affermare che é probabile che due documenti che trattano lo stesso argomento condividano lo stesso lessico, vale a dire condividano sia termini che relazioni fra i termini.

Green [GRE 99] suggerisce che le catene lessicali, grazie alla loro capacità di catturare la struttura del discorso, potrebbero essere sfruttate per individuare quali paragrafi di un documento trattano degli stessi argomenti.

Green definisce tre tipi di relazioni:

Extra-Strong Relation, ottenuta per ripetizione di uno o più termini.

Strong, fra due termini che appartengono allo stesso synset oppure fra due termini connessi tramite un link orizzontale (coppie di lemmi antitetici) o tramite una relazione ISA-INCLUDE.

Regular, quando fra due termini esiste un cammino che non supera una certa lunghezza fissata e segue tre regole:

1. nessuna direzione può precedere un *upward link*;
2. un solo cambio di direzione è ammissibile;
3. un link orizzontale può essere utilizzato per collegare un *upward* e un *downward link*.

La creazione di iperlink intra-document è subordinata ad una fase in cui le catene lessicali sono utilizzate come strumento per calcolare la similarità fra paragrafi. Poichè le catene lessicali si dipanano all'interno dell'intero testo, è importante riconoscere quando sono, rispetto ad un paragrafo dato, particolarmente dense. Green definisce la densità di una catena c in un paragrafo p (*density chain*) come:

$$d_{c,p} = \frac{w_{c,p}}{w_p}, \tag{7.4}$$

dove w_p è il numero di parole candidate contenute nel paragrafo p e $w_{c,p}$ è il numero di parole candidate del paragrafo p contenute nella catena c.

Per ogni paragrafo, quindi, si costruisce un vettore di *density chain*, che costituisce una rappresentazione più concisa del testo. La similarità fra due paragrafi viene determinata calcolando il prodotto scalare fra le due rappresentazioni in termini di *density chain vectors*.

Lo scopo di un generatore automatico di iperlinks è individuare quali paragrafi di un documento possono essere collegati perchè inerenti allo stesso topic. Dato una pagina con k paragrafi, viene costruita una matrice di dimensioni $k*k$ e solo quelle coppie di paragrafi la cui similarità[6] supera una certa soglia sono candidate ad essere collegate.

Anche a livello *inter-document* le catene lessicali possono fornire importanti indizi per individuare quei documenti inerenti allo stesso argomento. Nell'Information Retrieval classico la similarità di due documenti viene determinata tramite il prodotto scalare dei *word vectors*. Green osserva che le catene lessicali sono una rappresentazione più fluida e flessibile rispetto ad un semplice elenco di termini, pertanto anche il metodo di calcolo della similarità deve essere modificato. Le catene lessicali hanno come effetto secondario quello di ottenere il synset di appartenenza dei termini contenuti in un testo. La rappresentazione che se ne può ricavare è più vicina ad una rappresentazione semantica e/o concettuale di un testo. Inoltre, Green propone di rappresentare un documento con due vettori piuttosto che con uno solo. Ogni vettore contiene un numero di elementi pari al numero di synset in WordNet. Il primo è detto *member weighted synset vector* ed ognuno di essi

[6] calcolata con il coefficiente di Dice.

rappresenta il peso che il synset corrispondente ha all'interno del documento. Il secondo vettore (*linked weighted synset vector*) contiene i pesi dei synset quando si trovano *one link away* da un synset rappresentato nelle catene lessicali. Il calcolo della similarità avviene sulla base di tre misure distinte:

Member-Member similarity: calcolata tra i due *member weighted synset vectors* e cattura le relazioni più importanti (extra-strong e strong);

First Member-Linked similarity: calcolata fra il *member vector* del primo documento e il *linked weighted synset vector* del secondo e cattura le relazioni strong fra synset tra i quali esiste un cammino di lunghezza unitaria;

Second Member-Linked similarity: speculare alla precedente.

Dal punto di vista implementativo, Green calcola la similarità fra i due *member vectors* e se, superiore ad una soglia fissata, viene predisposto un collegamento fra i due documenti. All'interno di una collezione un documento può essere associato a più documenti. Il grado di rilevanza di questi documenti viene determinato sulla base delle altre due misure di similarità.

Parte IV

Progetto e realizzazione di un Agente Hunter

Capitolo 8

TUCUXI: un approccio linguistico alla ricerca di informazioni

Si pensi ad uno scenario in cui la **Global Virtual View** di MOMIS debba essere estesa includendo sorgenti HTML disponibili *on-line*. Il primo passo da compiere è individuare dove le nuove sorgenti sono localizzate, secondariamente è necessario valutare quali sorgenti sono realmente interessanti ed idonee ad essere integrate in MOMIS.

Se la ricerca di informazioni fosse demandata ad un operatore umano, egli interrogherebbe un motore di ricerca come *Google* specificando le proprie necessità tramite una o più *keywords*. Il motore di ricerca risponderebbe alla query restituendo un elenco ordinato di pagine, in cui il primo documento è considerato il più rilevante [1].

Purtroppo, la ricerca tramite keywords soffre di notevoli limiti. In primo luogo, una parola può avere più significati (*polisemia*) e pertanto la sua corretta interpretazione è strettamente dipendente dal contesto in cui appare. Inoltre, un termine può avere più sinonimi (*sinonimia*), quindi indicare una keyword non permette di specificare automaticamente anche le parole che hanno un significato simile.

[1] I problemi ed i limiti dei motori di ricerca sono stati descritti in dettaglio nei capitoli 4 e 5.

Per effetto della polisemia, il motore di ricerca tende a indicare come rilevanti documenti che contengono la parola richiesta, ma spesso essa non ha il significato desiderato. Al contrario, per effetto della sinonimia viene restituita solo una piccola parte di tutti i documenti potenzialmente rilevanti.

Scegliere le keywords potrebbe essere un processo non banale, in quanto esse sono uno strumento insufficiente e poco efficace per esprimere le reali necessità di un utente. Inoltre, una sola query non esaurisce la ricerca di informazioni e l'utente, sulla base del tipo e del contenuto dei documenti individuati dal motore è in grado di raffinare le proprie richieste.

Nonostante ciò, per alcuni argomenti trovare le informazioni che realmente interessano è un processo lungo e tedioso. Consideriamo il seguente esempio in cui si vogliono cercare i corsi di computer science. I risultati proposta da Google sono oltre 1500000, in cui i primi 4 sono siti sponsorizzati! Le keywords da sole provocano ambiguità!

L'ideale sarebbe poter esprimere le richieste in un modo più efficace, nonchè disporre di uno strumento di ricerca in grado di estrarre e valutare non tanto la presenza di keywords ma la semantica di un documento.

TUCUXI - In**T**elligent H**U**nter Agent for **C**oncept **U**nderstanding and Le**X**ical Cha**I**ning - è l'agente intelligente progettato e realizzato in questa tesi.

TUCUXI è uno strumento versatile che permette ad un utente di specificare i propri interessi tramite un'ontologia di dominio come il Common Thesaurus di MOMIS. TUCUXI [2] è un'agente intelligente sotto due diversi aspetti:

- adotta una strategia intelligente per localizzare risorse HTML interessanti rispetto all'ontologia data;

- il grado di interesse di una risorsa viene calcolato grazie alla tecnica delle catene lessicali mutuata dalla Linguistica Computazionale. In questa

[2]TUCUXI (pronuncia "tookooshee") è il nome indio di un delfino di fiume comune nel Rio delle Amazzoni e nell'Orinoco. Il delfino è dotato di un particolare sonar per localizzare branchi di pesce anche in acque sporche e paludose. Inoltre, i delfini sono noti per le capacità di eseguire ordini impartiti da operatori umani, nonchè la spiccata attitudine alla socialità.

tesi l'algoritmo di Silber e McCoy è stato adattato all'ambiente Web, in modo da estrarre la semantica e i concetti espressi in un documento HTML.

Durante la progettazione e l'implementazione di TUCUXI non sono venuta a conoscenza di progetti volti a realizzare agenti mobili in grado di sfruttare tecniche di *Natural Language Processing*, oltre a quelle "semplici" di uno *stemmer* o di un *part of speech tagger*. Vi sono, però, tre interessanti progetti (KAON, GATE e KeyConcept) che propongono tre diversi modi di specificare le esigenze di un utente. I dettagli sono illustrati in appendice.

8.1 Tecnologia ad agenti per un crawling efficiente

Il primo passo per scoprire nuove sorgenti di informazioni è... cercarle! Nel capitolo 4 si è utilizzata la definizione data da Cheong, secondo la quale un crawler è

> "a software program that traverse the World Wide Web information space by following hypertext links and retrieving Web documents by standard HTTP protocol".

Fiedler e Hammer [FIE 98] propongono di sostituire i tradizionali crawlers con agenti mobili e di sfruttarne le caratteristiche per esplorare la rete in maniera più efficiente (strategie *best-first*). Gli agenti mobili sono entità autonome intelligenti caratterizzate[WOO 95] da:

Autonomia: controllo diretto sulle proprie azioni e sul proprio stato interno e capacità di comportamente reattivi (recependo le condizioni ambientali) e proattivi (decidendo autonomamente quali azioni intraprendere).

Mobilità: capacità di muoversi da un'ambiente all'altro verso dati e risorse remote.

Socialità: capacità di comunicare e interagire e cooperare utilizzando linguaggi comuni.

Ad esempio, si supponga che un motore di ricerca voglia costruire o mantenere aggiornato un insieme di pagine riguardanti alcuni argomenti quali *Salute*, *Sport*, ... per fornire agli utenti un servizio di qualità. La creazione e la manutenzione di un indice tramite i crawler tradizionali è altamente inefficiente, perchè prevede il recupero di tutte le pagine. Al contrario, gli agenti crawler, spostandosi verso i Web Server possono analizzare in loco le pagine e restituire al modulo *indexer* del motore di ricerca solo quelle interessanti, con un notevole risparmio di banda di comunicazione . In questo modo, inoltre, il carico di lavoro per l'*indexer* viene drasticamente ridotto, risolvendo in parte i problemi di scalabilità dei tradizionali motori di ricerca.

Purtroppo, l'applicabilità della tecnologia ad agenti mobili limitata da due fattori:

1. l'esecuzione di un processo su un server remoto deve essere autorizzata dal proprietario/amministratore del sistema. E' necessario un meccanismo che fornisca all'amministratore di un server le oppurtune garanzie sul corretto comportamento e sull'eticità del codice che dovrebbe essere eseguito;

2. sul server remoto dovrebbe essere installato il *run-time environment* per consentire l'esecuzione del codice dell'agente.

TUCUXI è un agente JADE che implementa una strategia di crawling intelligente e pertanto è in grado di decidere autonomamente verso quale risorsa migrare. Purtroppo, per effetto dei fattori sopra citati, la mobilità di TUCUXI è per il momento limitata. Per questo motivo in seguito ci si riferirà alla migrazione verso una risorsa con l'espressione più generica "visitare una risorsa". Per maggiori dettagli e informazioni sulla piattaforma JADE si faccia riferimento a [BEL 02a, BEL 02b] e alla tesi di Enrico Natalini [NAT 02].

8.1.1 Una strategia di crawling basata sulla semantica

Nel capitolo 6 sono state presentate alcune euristiche per esplorare la rete visitando il prima possibile le risorse rilevanti e trascurando quelle non rilevanti.

Purtroppo gran parte delle strategie trascurano la semantica privilegiano tecniche di Information Retrieval classiche. Come si è visto, le keywords non sono da sole sufficienti per esprimere le reali necessità di un utente. Da questo punto di vista il *focused crawler* di Chakrabarti [CHA 99] è sicuramente più avanzato, perchè la rilevanza dei documenti recuperati viene valutata sulla base di esempi forniti dall'utente. Scegliendo opportunamente gli esempi e controllando periodicamente l'attività svolta dal crawler l'utente ottiene generalmente documenti di buona qualità.

Tramite le tecniche di *Machine Learning* da set di esempi si possono catturare informazioni anche circa il layout di un documento[3], non ritengo che si tratti di un approccio applicabile ad un ambiente ad agenti. Infatti, il crawler di Chakrabarti è un sistema con *feedback* che necessita di un costante ed attento intervento da parte dell'utente. Si tratta anche di un sistema poco flessibile perchè la ricerca di informazioni su un nuovo argomento prevede la determinazione di un training set significativo, senza poter riutilizzare modelli ricavati in precedenza.

Yu et al. [YU 01] propongono in questo senso una strategia più generale, in grado di adattarsi a diversi argomenti. Il crawler di Yu è un crawler che si costruisce run-time un modello statistico della porzione di Web esplorata e può riutilizzare le informazioni raccolte durante le precedenti sessioni di crawling. Il crawler di Yu dispone di un lessico composto da n termini e determina la priorità con la quale visitare le pagine calcolando quattro diversi parametri: *Content Based Learning, URL Token Based Learning, Link Based Learning e Sibling Based Learning* [4].

TUCUXI può adottare due comportamenti diversi per esplorare il Web. Il primo è analogo a quello dei crawler tradizionali, in cui i documenti vengono visitati senza tener conto della loro localizzazione all'interno di un sito; il secondo è più adatto alle caratteristiche di un agente mobile e prevede che quando un sito viene raggiunto venga esplorato completamente. Questa seconda modalità è più idonea anche per valutare immediatamente la qualità

[3]Il layout di un documento è un'aspetto estremamente importante in ambiente Web, perchè l'informazione testuale è espressa in forma sintetica e i concetti importanti sono evidenziati ed enfatizzati utilizzando font o colori diversi.

[4]Per una descrizione dettagliata del significato dei parametri si veda 6.

e l'attinenza dell'intero sito rispetto all'ontologia data.

Anche TUCUXI si costruisce un modello statistico delle pagine visitate seguendo quello proposto da Yu. L'aspetto innovativo della strategia è costituito da un nuovo parametro di *learning*, denominato *Synset Based Learning*. L'idea è applicare la tecnica delle catene lessicali per estrarre i significati e quindi i synset dalle pagine visitate e verificare se si tratta di un buon parametro per guidare una strategia di crawling. Poichè sono stati descritti in letteratura [CHA 99, DIL 00] casi in cui, per effetto dell'eccessivo numero di regole da seguire, il crawler tende a stagnare, ritengo che per il momento il parametro *URL Token Based Learning* possa essere trascurato. Invece mantenere il *Content Based Learning,* il *Link Based Learning* ed il *Sibling Based Learning* permette di confrontare la "robustezza"del nuovo parametro *Synset Based Learning* rispetto ai precedenti.

Supponendo di aver definito una funzione che esprima la rilevanza di un documento rispetto ad un'ontologia data con un valore compreso fra [0,1), è possibile formalizzare il modello statistico di TUCUXI in modo analogo a quello descritto in [YU 01]. Denotiamo infatti con C l'evento in una pagina recuperata ottiene un punteggio di rilevanza superiore ad una soglia fissata dall'utente. La probabilità *P(C)* di una pagina di superare la soglia può essere calcolata come il rapporto fra il numero di documenti rilevanti ed il numero totale di documenti recuperati. Anche per TUCUXI le pagine avrebbero tutte la stessa probabilità di contenere dati rilevanti se non determinasse la priorità da assegnare ai documenti candidati grazie all'analisi di alcuni dati statistici raccolti durante la sessione di crawling. Anche in questo caso sia E un'informazione nota circa un URL candidato. La conoscenza di tale informazione può far aumentare il valore della priorità della pagina associata. Il nuovo valore di priorità viene calcolato tramite la probabilità condizionata *P(C/E)*, la quale può essere determinata tramite la formula

$$P(C|E) = P(C \cap E)/P(E), \tag{8.1}$$

dove *P(E)* è la frequenza con la quale l' evento E si verifica.

Analogamente, si può calcolare un fattore di interesse (*Interest Ratio* per

l'evento C, noto un evento E:

$$I(C, E) = P(C|E)/P(C). \tag{8.2}$$

Se l'evento è favorevole, ossia se *P(C|E)>P(C)*, allora l'Interest Ratio è maggiore di 1 e la pagina dovrebbe essere recuperata prima di quelle con un Interest Ratio minore. Inoltre, se la strategia è guidata da più di una features, si può esprimere l'Interest Ratio per l'evento composto ε come

$$I(C, E) = \pi_{i=1}^{k} I(C, E_i) \tag{8.3}$$

Sulla base delle diverse feature, TUCUXI calcola i vari Interest Ratio nel modo seguente:

Content Based Learning. Dato un lessico composto da W_1, W_2,..., W_n termini, solo una piccola parte di esso compare nel testo delle inlinking pages degli URL candidati. La variabile logica Q_i assume valore vero quando il termine i-esimo compare in una delle inlinking pages. Pertanto, l'insieme $M = \{$ *i: Q_i is true* $\}$ contiene gli indici dei termini che compaiono nelle inlinking pages. Il relativo interest ratio è

$$I(C, Q_i) = P(C \cap Q_i)/(P(C) \cdot P(Q_i)) \tag{8.4}$$

Synset Based Learning. Il processo di learning sulla base dei significati espressi nelle pagine candidate consiste nella verifica del matching fra i synsets annotati nel Common Thesaurus e i synsets estratti dai documenti. Il processo è del tutto analogo al content based learning. I_{Synset} è il corrispondente Interest Ratio. Formalmente, dato un Common Thesaurus è composto da S_1, S_2,..., S_k synsets e solo un numero limitato di essi sono contenuti in un documento. La variabile logica R_j assume valore vero quando il synset j-esimo compare in una delle inlinking pages. Pertanto, l'insieme $N = \{$ *i: R_i is true* $\}$ contiene gli indici dei termini che compaiono nelle inlinking pages. Il relativo Interest Ratio è

$$I(C, R_i) = P(C \cap R_i)/(P(C) \cdot P(R_i)) \tag{8.5}$$

Link Based Learning. Se il crawler recupera 100 pagine di cui solo 10 sono rilevanti, la probabilità *P(C)* è pari al 10%. Se la struttura dei link fosse completamente random, allora la probabilità di recuperare una nuova pagina rilevante è esattamente pari a *P(C)*. La probabilità che da una pagina rilevante si raggiunga un'altra pagina rilevate è $P(C) \cdot P(C)$. Quest'ultimo è un valore atteso, ma se sulla base dei documenti recuperati risulta che la frazione di documenti in cui sia il nodo sorgente che il nodo destinazione sono rilevanti è maggiore del valore atteso, allora significa che per l'argomento d'interesse esiste un'organizzazione strutturale dei contenuti. Formalizzando questo aspetto e considerando una pagina che viene riferita da altre k pagine, m delle quali soddisfano il predicato, per ognuna delle m pagine che soddisfano il predicato il corrispondente interest ratio è dato da

$$p = f_1/(P(C) \cdot P(C)), \tag{8.6}$$

dove f_1 rappresenta la frazione di pagine rilevanti per le quali anche le inlinking pages sono rilevanti. Analogamente, l'interest ratio per ognuna delle k-m pagine che non soddisfano il predicato può essere calcolato come

$$q = f_3/(P(C) \cdot (1 - P(C))), \tag{8.7}$$

dove f_3 è la frazione di pagine che non soddisfano il predicato, nonostante le inlinking page fossero rilevanti.

L'interest ratio complessivo è dato da:

$$I_l = p^m \cdot q^{k-m}. \tag{8.8}$$

Sibling Based Learning. Una pagina candidata ha una maggiore probabilità di soddisfare il predicato se i documenti recuperati dalle stesse inlinking page sono a loro volta rilevanti. Supponendo che una pagina contenga n embedded URLs (siblings), di cui r già recuperati, allora il numero atteso di pagine che soddisfano il predicato è $r \cdot P(C)$. Se durante il crawling il numero di siblings che soddisfano il predicato è maggiore di $r \cdot P(C)$, allora la priorità dei siblings non ancora recuperati

deve aumentare. I_s è l'interest ratio corrispondente e viene calcolato come:

$$I_s = s/e, \tag{8.9}$$

dove s è il numero di siblings recuperati che soddisfano il predicato ed e è il numero atteso di siblings rilevanti.

La priorità assegnata ad una pagina candidata è la combinazione dei fattori sopracitati:

$$PriorityValue = w_c \cdot \log(I_c(C)) + w_{synset} \cdot \log(I_{synset}(C)) + w_l \cdot \log(I_l(C)) + w_s \cdot \log(I_s(C)) \tag{8.10}$$

Il valore dei pesi è determinabile da parte dell'utente. Inoltre è stata implementata anche una versione avanzata della strategia intelligente in cui i vari pesi sono calcolati dinamicamente valutando come le quattro feature hanno influenzato l' individuazione di sorgenti rilevanti. La determinazione dei pesi avviene in due modalità, la prima delle quali prevede una verifica quando l'*harvest rate* (il numero di pagine rilevanti sul totale di pagine visitate) scende sotto una soglia predefinita[5]; con la seconda l'agente ricalcola i pesi dopo aver visitato un certo numero di pagine specificato dall'utente. La prima è più adatta ad un crawler tradizionale non vincolato ad esplorare esaustivamente un sito prima di muoversi verso altre risorse.

8.1.2 Etica per un crawler

L'attività di crawling da parte di software automatici causa un traffico atipico all'interno della rete e consuma risorse destinate ad un'utenza "tradizionale". Infatti, se il normale traffico Web è quello generato da operatori umani che navigano utilizzando prevalentemente un browser, i software automatici possono effettuare migliaia di richieste al secondo. Se poi tutte le richieste venissero concentrate indiscriminatamente verso un unico server, allora la sua capacità di risposta subirebbe un deterioramento a scapito degli utenti

[5]Nei miei esperimenti questo parametro non è di vitale importanza. La soglia è stata fissata a 0,002

normali. Nei casi più gravi il server potrebbe subire un collasso. E' quindi importante che un crawler adotti un comportamento etico che secondo quanto descritto da Eichmann [EIC 95] deve essere caratterizzato da:

1. possibilità di determinare l'identità del crawler e del proprietario del crawler;

2. moderazione nell'uso delle risorse disponibili;

3. rispetto dei vincoli imposti dai proprietari dei siti ;

4. risultati delle sessioni di crawling consultabili pubblicamente;

Per quanto riguarda 3 è importante che anche un agente come TUCUXI rispetti le direttive impartite tramite i *META-TAG Robot* [HTML 99] e il *Robots Exclusion standard* [5].

Il META-TAG Robot è un meta-tag che viene introdotto nelle pagine HTML per indicare ai crawler come trattare il documento in esame. Generalmente i tradizionali motori di ricerca indicizzano i documenti recuperati. Per evitare che ciò accada, chi pubblica pagine Web può inserire un META-TAG Robot specificando la direttiva NOINDEX. La sintassi completa del META-TAG Robot è la seguente:

```
<META name="ROBOTS" content="ALL | INDEX | NOINDEX|
FOLLOW | NOFOLLOW | NONE">
```

Con la direttiva NOFOLLOW si impedisce al crawler di seguire i links contenuti nel documento. Le direttive possono essere anche combinate, ad esempio NOINDEX | NOFOLLOW indica al crawler di non indicizzare la pagina e di non recuperare gli embedded URLs.

Tramite il *Robots Exclusion Standard* si vuole risolvere un problema diverso, in parte generato, però, al momento dell'introduzione del META-TAG Robot nelle specifiche HTML solo pochi crawler seguivano le direttive. Il Robots Exclusion Standard si avvale di un semplice file di testo denominato *robots.txt* per esprimere istruzioni che limitino selettivamente l'accesso ai documenti da parte degli spiders. Secondo lo standard, il file *robots.txt*

dovrebbe essere pubblicato on-line nella directory principale di un sito. Un crawler dovrà accedere al file robots prima di poter accedere a qualsiasi altro documento e dovrà controllare quali directory NON possono essere visitate. La visibilità di un sito può essere resa differente a seconda dei crawler. Infatti, il file robots non è altro che un insieme di record, ognuno dei quali comprende due campi, il primo detto `User-Agent` ed il secondo `Disallow`. Il campo Disallow può apparire in un record più volte.

Il campo User-Agent serve ad indicare a quale robot/spider le direttive successive sono rivolte. La sua sintassi è:

```
User-Agent : <spazio> <nome_dello_spider>
```

Il campo Disallow serve a indicare a quali file e/o directory non può accedere lo spider indicato nel campo User-Agent. La sintassi di questo campo è:

```
Disallow <duepunti> <spazio> <nome_del_file_o_directory>
```

Quindi, un file **robots.txt** che impedisce allo spider di Google (GoogleBot) di prelevare un file di nome *personale.html* della directory principale del sito avrá la seguente sintassi:

```
User-Agent: googlebot
Disallow: personale.html
```

Per indicare invece allo spider di AltaVista (Scooter) di non accedere e non prelevare file dalla directory *privato*, sottodirectories comprese, il file **robots.txt** sará:

```
User-Agent: scooter
Disallow: /privato/
# questo e' un commento che viene ignorato dagli spider
# per le directories ricordarsi di inserire i due "/"
```

Il campo User-Agent può contenere un asterisco *, sinonimo di qualunque spider. Per cui l'esempio seguente dice a tutti gli spider di non prelevare il file temp.html:

```
User-Agent: *
Disallow: temp.html
```

Il campo Disallow può contenere un carattere / ad indicare qualunque file e directory. L'esempio che segue impedisce a Scooter di prelevare qualunque cosa:

```
User-Agent: scooter
Disallow: /
```

Infine, il campo Disallow può essere lasciato vuoto, ad indicare che non ci sono file o directory di cui si vuole impedire il prelievo. L'esempio seguente mostra come dire a tutti i motori di ricerca di prelevare tutti i file del sito:

```
User-Agent: *
Disallow:
```

l file robots.txt si compone di uno o più record, ognuno dei quali prende in esame spider differenti. Ecco dunque un esempio completo di file **robots.txt**, che blocca del tutto Google, impedisce a AltaVista l'accesso ad alcuni file e directory e lascia libero accesso a tutti gli altri motori di ricerca.

```
User-Agent: googlebot
Disallow: /

User-Agent: scooter
Disallow: listino.html
Disallow: vendita.html
Disallow: /temporanei/
Disallow: /cgi-bin/

User-Agent: *
Disallow:
```

Il nome (User-Agent) degli spider dei motori di ricerca può essere individuato cercando nei log del server HTTP che gestisce il sito web oppure consultando le pagine dei motori di ricerca dedicate ai webmaster. Quasi sempre, vengono indicati i nomi degli spider e come bloccarli usando il file robots.txt.

Quando gli spider non trovano il file robots.txt, si comportano come se avessero ricevuto via libera ed accederanno dunque a tutte le pagine trovate nel sito (salvo diversamente specificato dai meta tag ROBOTS).

Anche TUCUXI è stato registrato nel *Web Robots Database*. Parte del modulo di iscrizione è consultabile in Figura 8.1.2.

robot-id: TUCUXI
robot-name: TUCUXI
robot-owner-name: Roberta Benassi - University of Modena and Reggio Emilia
robot-owner-url: http://sparc20.ing.unimo.it/
robot-status: development
robot-purpose: indexing
robot-type: standalone
robot-exclusion: yes
robot-exclusion-useragent: TUCUXI
robot-noindex: yes
robot-nofollow: yes
robot-host: http://sparc20.ing.unimo.it/
robot-language: java
robot-description: TUCUXI - inTelligent hUnter agent
for Concept Understanding and leXical chaIning
robot-history: Roberta Benassi's final dissertation
at the University of Modena and
Reggio Emilia under the direction of
Prof. Sonia Bergamaschi
robot-environment: research
modified-date: Fri, 30 May 2003 12:53:22 GMT
modified-by: Roberta Benassi

Come si evince, TUCUXI rispetta sia il META-TAG Robots che lo standard per l'esclusione dei Robots. Una sorgente che volesse indicare a TUCUXI di non visitare la directory /cgi-bin/ dovrebbe includere nel proprio file robots.txt il seguente record

```
User-Agent: TUCUXI
Disallow: /cgi-bin/
```

Mi sembra però importante fare una precisazione: TUCUXI non indicizza le pagine visitate come farebbe un motore di ricerca, ma si limita a codificarne l'indirizzo, i synsets dei concetti contenuti e il punteggio di rilevanza in un file XML. Rispettare il META-TAG NOINDEX significa per TUCUXI non inserire l'URL del documento nel suddetto file.

8.2 Estrazione della semantica

8.2.1 Perchè le catene lessicali?

Uno degli scopi di TUCUXI è ottenere pagine rilevanti sulla base del loro contenuto e sulla base di una metrica di qualità. L'idea che ha guidato questo lavoro si basa su una semplice considerazione circa alcune delle più note tecniche di Information Retrieval. Esse cercano di costruire una rappresentazione alternativa del testo che sia più facilmente manipolabile con metodi matematici e statistici. Purtroppo queste rappresentazioni trascurano l'aspetto semantico dei termini contenuti in un testo, cosa che provoca effetti indesiderati dovuti alla polisemia e sinonimia. La domanda è: esistono altre rappresentazioni in grado di catturare la semantica di un testo e fornire quindi una metodologia alternativa alle *bag-of-words*? Per prima cosa è necessario comprendere la semantica di un testo. Tale compito è risolto dalle tecniche di *Natural Language Processing*. Alcune di queste tecniche sono estremamente complesse perchè si propongono di realizzare una completa comprensione del testo (*fully understanding*). Altre sono di tipo *partial undestanding* e si limitano a determinare a grandi linee il significato di un documento. Ritengo che le prime siano troppo complesse per essere implementate efficacemente

in un'architettura ad agenti ed inoltre, trattandosi spesso di tecniche sviluppate per una lingua in particolare, sarebbero scarsamente applicabili in un contesto multilingua.

La tecnica delle catene lessicali è di tipo *partial understanding* ed è a mio avviso una buona alternativa sia perchè è dimostrata l'applicabilità ad un contesto multilingua [FUE 02], sia perchè l'algoritmo di Silber e McCoy ha una complessità computazionale lineare.

Purtroppo, però, le pagine Web perdono parte della struttura formale e sintattica dei tradizionali documenti [SOD 97] per privilegiare una modalità di comunicare le informazioni più orientata verso l'aspetto visivo. Ciò che mi ha guidato verso la scelta di implementare un *lexical chainer* è stata l'osservazione di come le persone straniere tentano di comunicare in italiano. Non si può certo dire che le frasi siano grammaticalmente corrette, ma ciò che importa è comunicare e la comunicazione orale avviene tramite il lessico, proprio il principio cardine della teoria della coesione dei testi [HAS 84, HHC 76, HHG 85, HAC 84]!

TUCUXI implementa una versione adattata all'ambiente Web dell'algoritmo di Silber e McCoy, in cui la fase preliminare di segmentazione viene sostituita dalla semplice individuazione di tre regioni di particolare interesse sia per l'aspetto grafico/visivo che per l'aspetto testuale: il titolo ed headings, le liste e il corpo del documento. Le parole candidate ad essere inserite nelle catene sono solamente i nomi, che vengono individuati tramite un cosiddetto *part-of-speech tagger*. In letteratura sono stati proposti diverse tipologie di part-of-speech taggers, ma quasi tutti hanno in comune una fase di machine learning. Il più noto fra tutti gli strumenti di questo tipo è sicuramente il tagger di Brill [BRI 92, BRI 94]. In questa tesi si è preferito adottare MXPOST[6], il tagger di Ratnaparkhi [RAT 96, SAN 90] consente di ottenere un'accuratezza del 96,6%. Inoltre, mentre il tagger di Brill è disponibile in C, MXPOST è scritto interamente in Java, cosa che permette una semplice integrazione con TUCUXI, anch'esso scritto in Java. Purtroppo, i dati in ingresso per MXPOST devono subire una trasformazione. Ad esempio, l'individuazione dei nomi, verbi ed aggettivi avviene correttamente se viene

[6]MXPOST è stato sviluppato nell'ambito del Penn TreeBank Project dell'Università della Pennsylvania.

analizzato il testo una frase alla volta e se le forme verbali contratte (*don't*) sono sostituite dalla loro forma completa (*do not*). Lo stemmer di Porter [POR 80] è utilizzato per trovare la forma base nei nomi plurali. In alcuni casi, inoltre, è più significativa una coppia di termini piuttosto che un termine presto singolarmente, basti pensare a "computer science". Un semplice *shallow parser* individua le coppie di nomi in posizioni consecutive. Purtroppo, l'applicabilità di questo metodo ottiene dei buoni risultati solo quando la forma è contenuta in WordNet.

8.2.2 Estrazione di mappe concettuali

L'algoritmo di Silber e McCoy considera solo una piccola parte delle relazioni lessicali in cui un lemma può essere coinvolto. L'algoritmo è progettato per costruire le catene in un tempo lineare e per questo motivo considera solo relazioni *forti*, quali sinonimia, ipernimia e iponimia. Le catene di Silber e McCoy sono quindi cluster gerarchico-tassonomici, mentre il Common Thesaurus del MOMIS considera anche altri tipi di relazioni. Per poter confrontare equamente la struttura estratta dal testo con l'ontologia di dominio è inevitabile estendere il tipo di relazioni considerate, includendo anche le relazioni di *meronimy* ed *holonymy*. Proprio queste sono fondamentali per quei testi come le pagine Web che sono poco strutturati sintatticamente. Ovviamente, si rinuncia ad una complessità computazionale lineare[7], aspetto particolarmente interessante per un agente mobile. Il vantaggio, però, consiste nella possibilità di sfruttare eventuali estensioni di WordNet dedicate ad esempio a domini più specifici. Per maggiori dettagli sulle modalità di estensione dell'ontologia lessicale di WordNet si veda la tesi di Veronica Guidetti [GUI 02].

L'algoritmo implementato è un meccanismo di voto:

1. Per ogni lemma selezonato si individuano in WordNet (o in una delle sue estensioni [GUI 02] se richiesto) tutti i synset di appartenenza.

[7] Ottenuta in parte modificando i metodi di accesso a WordNet.

2. Ogni lemma[8] esprime la propria preferenza per comparire nelle catene in cui l'origine è uno dei synset di appartenenza.

   (a) Per ogni synset viene ricostruito il contesto in cui è inserito all'interno di WordNet, recuperando i synset padri, figli, fratelli, zii e cugini e quelli legati dalle relazioni di meronimia e olonimia.

   (b) Per ogni synset il lemma esprime una preferenza nel comparire in una catena la cui origine è uno dei synsets recuperati nel passo precedente. Il voto viene calcolato sulla base del contenuto della catena. Ciò significa che più forti sono le relazioni fra il nuovo synset e quelli già inseriti, più forte è la preferenza espressa.

3. Per ogni lemma inserito nelle catene viene determinato il significato più idoneo, determinando per quale/i catena/e la preferenza espressa è più forte. Una forte preferenza è indice di una forte interconnessione con gli altri lemmi inseriti nella catena.

4. Ogni synset di appartenenza del lemma diverso dal significato individuato al passo precedente viene eliminato dalle catene. Le interconnessioni fra i synset rimasti sono ricostruite.

Il risultato prodotto dall'algoritmo è un insieme di catene che contengono *cluster* di synsets. Ogni catena ha associato un punteggio determinato dai voti di preferenza espressi dai synsets contenuti. Tra tutte le catene ve ne saranno presumibilmente alcune di scarsa rilevanza rispetto ad altre ritenute *strong*. Le catene di scarsa importanza vengono eliminate proprio sulla base del punteggio ad esse associato. Questa fase di pruning prevede che vengano mantenute solo quelle catene per le quali il punteggio soddisfa la seguente relazione:

$$Score(chain) > AverageScore + 2 \cdot StandardDeviation. \tag{8.11}$$

Quello che si vuole ottenere è però la mappa concettuale del testo, ossia i concetti fondamentali e le relazioni fra essi . Solo i concetti forti sono i

[8]Ogni lemma avrà un'importanza diversa se compare in un titolo (peso 2), in una lista (peso 1.5) e nel corpo del testo (peso 1).

concetti fondamentali. In questa tesi si considerano forti i significati contenuti nelle catene determinate tramite la formula precedente. Inoltre, l'utente può intervenire limitando la costruzione delle mappe concettuali ad un numero fisso di catene forti (es. 25) o ad una percentuale di esse (es. 30%).

8.3 Relevance Scoring

La costruzione della mappa concettuale di una pagina Web è analoga alla costruzione di un grafo, in cui i concetti rappresentano i nodi e le relazioni rappresentano glia archi. Anche il Common Thesaurus del MOMIS è un grafo, pertanto il punteggio di una pagina può essere determinato confrontando due strutture analoghe. Il confronto di due grafi dal punto di vista strutturale è affrontato in [MEL 01]. Personalmente ritengo che il semplice confronto sulla base della struttura sia limitativo in primo luogo perchè la mappa concettuale estratta mantiene il significato espresso in un testo. Per questo motivo è più interessante analizzare la proposta di Maedche e Staab [MAE 01], in cui il confronto di due ontologie è affrontato sotto due punti di vista: sintattico e semantico.

Confrontare due ontologie dal punto di vista sintattico (*syntactic comparison level*) significa per Maedche e Staab utilizzare la *edit distance* di Levenshtein [LEV 66] per quantificare la similarità fra due stringhe.

Un'ontologia è formata in parte da un lessico 8.1, ossia un insieme di stringhe. Per ogni coppia di stringhe, L_1 prima dell'ontologia $\mathcal{O}_1$ e L_2 dell'ontologia $\mathcal{O}_2$, viene calcolata una misura di similarità detta *String Matching* (SM):

$$SM(\mathcal{L}_1, \mathcal{L}_2) = max(0, \frac{min(|L_1|, |L_2|) - editDistance(L_1, L_2)}{min(|L_1|, |L_2|)}) \qquad (8.12)$$

SM restituisce un grado di similarità compreso fra 0 e 1. Dal punto di vista sintattico, la similarità fra due ontologie viene calcolata come la similarità media delle stringhe di $\mathcal{L}_1$ e $\mathcal{L}_2$:

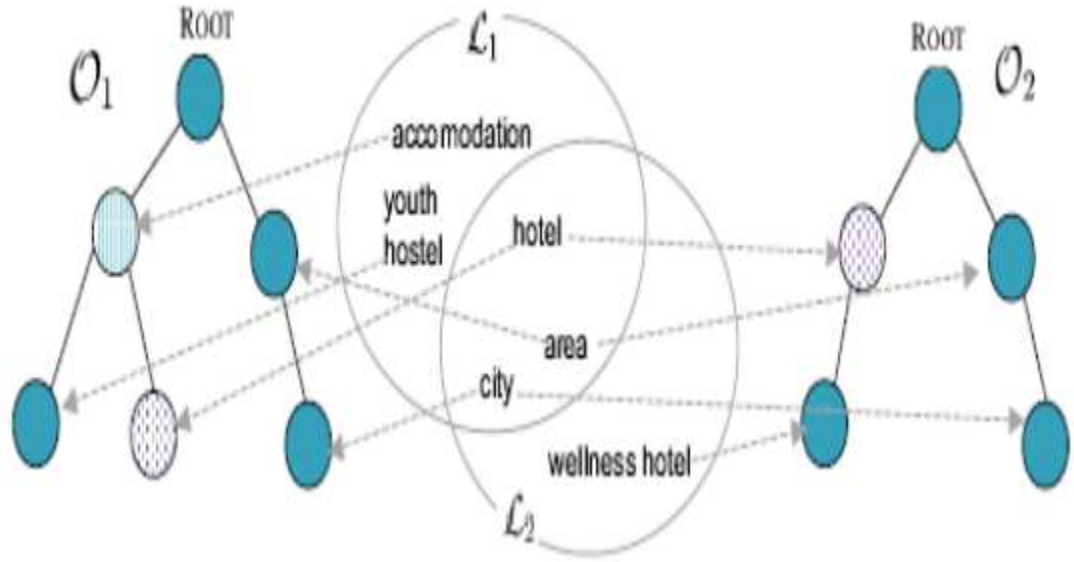

Figura 8.1: Due ontologie a confronto

$$SMAverage(\mathcal{L}_1, \mathcal{L}_2) = \frac{1}{|\mathcal{L}_1|} \cdot \sum_{L_i \in \mathcal{L}_1} max_{L_j \in \mathcal{L}_2} SM(L_i, L_j) \tag{8.13}$$

La misura *SMAverage* è una misura asimmetrica, tanto che $SMAverage(\mathcal{L}_1, \mathcal{L}_2)$ può essere diversa da $SMAverage(\mathcal{L}_1, \mathcal{L}_2)$. Lo scopo dell'uso della *edit distance* è mitigare le differenze di lessico dovuto all'uso di caratteri underscore, di plurali e singolari, ma può determinare un elavato grado di similarità anche per quei termini che pur avendo un diverso significato hanno quasi la stessa forma (es. "tower"e "power").

Per questo motivo viene data anche una misura di similarità rispetto in punto di vista semantico. Maedche e Staab ritengono che in un'ontologia sia forte la componente tassonomica. L'ipotesi è che due ontologie condividano concetti attraverso il lessico comune. Per ogni ontologia e per ogni concetto condiviso si determina un insieme formato dai relativi sub-concetti e super-concetti. Ad esempio, nella Figura 8.1 il lessico condiviso è l'insieme di termini {*hotel, area, city*}. Il termine *hotel* riferisce in $\mathcal{O}_1$ un nodo che ha come super-concetto *accomodation*. Nell'ontologia $\mathcal{O}_2$ *hotel* ha come sub-concetto *welness hotel*. L'unione dei due insieme è dato da {*hotel, welness hotel, accomodation*}. L'intersezione (*Taxonomic overlap*) invece è costitui-

ta dal solo termine *hotel*. La misura di similarità è data dal rapporto fra il numero di elementi nell'intersezione e il numero di elementi nell'unione. Considerando tutti i concetti condivisi, la similarità sarà pari al valore medio delle similarità concettuali.

La misura di similarità che TUCUXI adotta è una nuova misura che trova la sua ragione di essere proprio nella teoria linguistica che ha dato origine alla tecnica delle catene lessicali. TUCUXI suppone che ogni documento analizzato possa essere suddiviso in due parti, in cui la prima è reale ed è rappresentabile tramite la mappa, mentre la seconda è virtuale ed è costituita dall'ontologia di riferimento. Intuitivamente, TUCUXI calcola una misura di coesione fra la mappa concettuale estratta dal testo e l'ontologia di riferimento e costruisce una mappa globale (*G*lobal Map).

Una mappa concettuale ha un punteggio che viene calcolato sulla base del numero di interconnessioni fra i synsets contenuti. Aggiungendo i concetti dell'ontologia alla mappa, il punteggio di quest'ultima viene aggiornato. La misura di coesione (*Cohesion Measure* CM) è l'incremento percentuale del punteggio della mappa globale rispetto alla mappa concettuale del testo:

$$CM = \frac{Score(GlobalMap) - Score(ConceptualMap)}{Score(ConceptualMap} \tag{8.14}$$

Un'elevata misura di coesione indica sia un elevato numero di interconnessioni fra la mappa concettuale e l'ontologia sia un elevato peso associato alle relazioni.

La *Cohesion Measure* è a mio avviso insufficiente per determinare la similarità di una pagina rispetto ad un'ontologia. Ciò che spesso interessa determinare sono i synsets in un Common Thesaurus che sono contenuti in un documento. Per questo motivo, TUCUXI combina la misura di coesione con un punteggio assegnato sulla base del numero di synset che compaiono sia nella mappa concettuale che nell'ontologia.

Una buona misura di similarità dovrebbe aumentare all'aumentare del numero di synset comuni [9] in modo non lineare. Avendo a disposizione anche una misura di coesione si può dire che la misura di similarità ideale dovrebbe aumentare in modo non lineare anche all'aumentare della misura di coesione.

[9]Per ulteriori dettagli si veda [GUI 02].

TUCUXI calcola la rilevanza di una pagina rispetto a due misure di similarità diverse:

$$SimilarityScore(Page) = \begin{cases} (a + b\cdot \mathrm{CM}) + (c\cdot \mathrm{N}_{cs})/\mathrm{N}_{sCT} & \text{if } \mathrm{N}_{cs}{>}0 \\ (a + b \cdot (\mathrm{CM}/c) & \text{if } \mathrm{N}_{cs}{=}0 \end{cases} \tag{8.15}$$

dove N_{cs} rappresenta il numero di synsets in comune nella mappa concettuale e nell'ontologia e N_{sCT} è il numero di synset del Common Thesaurus. I parametri a, b sono determinati con le formule 8.16, mentre c è un valore costante. Negli esperimenti effettuati si è assunto un valore di c pari a 2.

$$\begin{aligned} a &= N_{sCT} + N_{sCM} \\ b &= (1 - a) \end{aligned} \tag{8.16}$$

La seconda misura di similarità è di tipo esponenziale analoga a quella utilizzata in [GUI 02]:

$$SimilarityScore(Page) = 1 - \exp\Big(- (\frac{N_{cs}^2}{N_{sCT}} + (a + b \cdot CM)\Big) \tag{8.17}$$

La misura 8.15 è una funzione ad handicap, in cui l'assenza di synsets in comune determina una diminuzione della misura di similarità. La seconda, invece, cresce più velocemente all'aumentare sia del numero di synsets in comune sia all'aumentare della misura di coesione.

Come dimostrato in [KLE 98, PAG 98, BRI 98], una misura di rilevanza basata solamente sul contenuto non esprime una misura della qualità delle pagine. Per questo motivo, TUCUXI implementa una versione dell'algoritmo HITS seguendo le indicazioni di [BHA 98] e pesando gli elementi proprio con il punteggio calcolato con le misure 8.15 e 8.17. Recentemente Diligenti e altri hanno proposto una versione di PageRank detta *Focused PageRank* [DIL 02].

Inoltre, TUCUXI combina il punteggio di similarità e quello di authority per ottenere un punteggio globale[10]:

$$GlobalScore(Page) = w_{sim}\cdot SimilarityScore + w_{aut}\cdot AuthorityScore \tag{8.18}$$

[10]Negli esperimenti i pesi sono entrambi posti a 0,5.

8.4 Esperimenti

Sono stati eseguiti esperimenti per la performance della strategia illustrata in 8.1.1. Purtroppo, non è possibile confrontare i risultati di TUCUXI con quelli di [YU 01], perchè gli argomenti trattati e il sistema di scoring sono completamente diversi. Come Common Thesaurus è stato scelto quello delle università di cui al Capitolo 2.4. Per poter trarre delle conclusioni circa l'efficacia della nuova strategia è stata implementata una tecnica[11] *Random* e una tecnica *Breadth-first*. Degli esperimenti effettuati verrano presentati i grafici dei risultati più significativi, nei quali viene considerata una *Score threshold* pari a 0.5.

8.4.1 Quattro strategie a confronto

Si vuole dimostrare che la strategia di esplorazione denominata *Intelligent* (e la sua estensione *Adaptive*) recupera, a parità di pagine visitate, un maggior numero di documenti rilevanti rispetto alle altre. Per quanto riguarda la modalità *Page mode* sono state eseguite tre sessioni di crawling di 5000 pagine per ognuna delle quattro strategie a partire dal seme *www.yahoo.com* (Figura 8.2). Altre tre sessioni di crawling di 5000 pagine ciascuna sono state eseguite a partire dal seme *www.berkeley.edu* (Figura 8.3). Nel primo caso si sono ottenuti degli harvest rate medi molto bassi, nel secondo caso i risultati sono migliori perchè l'agente viene subito indirizzato verso una sorgente potenzialmente rilevante (Tabella 8.1).

	Random	Breadth-First	Intelligent	Adaptive
Seme: www.yahoo.com	0.0333	0.0439	0.0594	0.0608
Seme: www.berkeley.edu	0.0452	0.0108	0.1227	0.1299

Tabella 8.1: Tabella comparativa degli Harvest Rate medi - risultati ottenuti da tre sessioni di crawling page mode per ognuna delle quattro strategie.

[11] In seguito ci si riferirà a queste due tecniche con il termine *strategia*, sebbene non si tratti di due strategie nel significato più stretto.

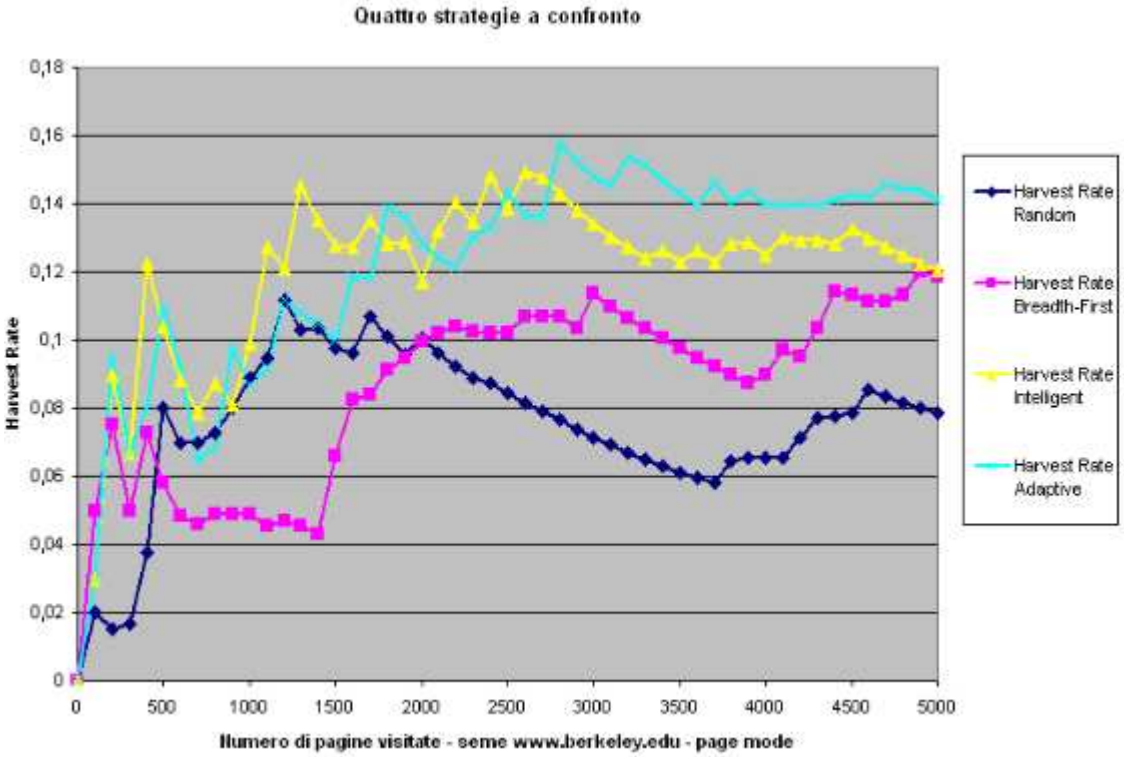

Figura 8.2: Esplorazioni page mode da www.yahoo.com

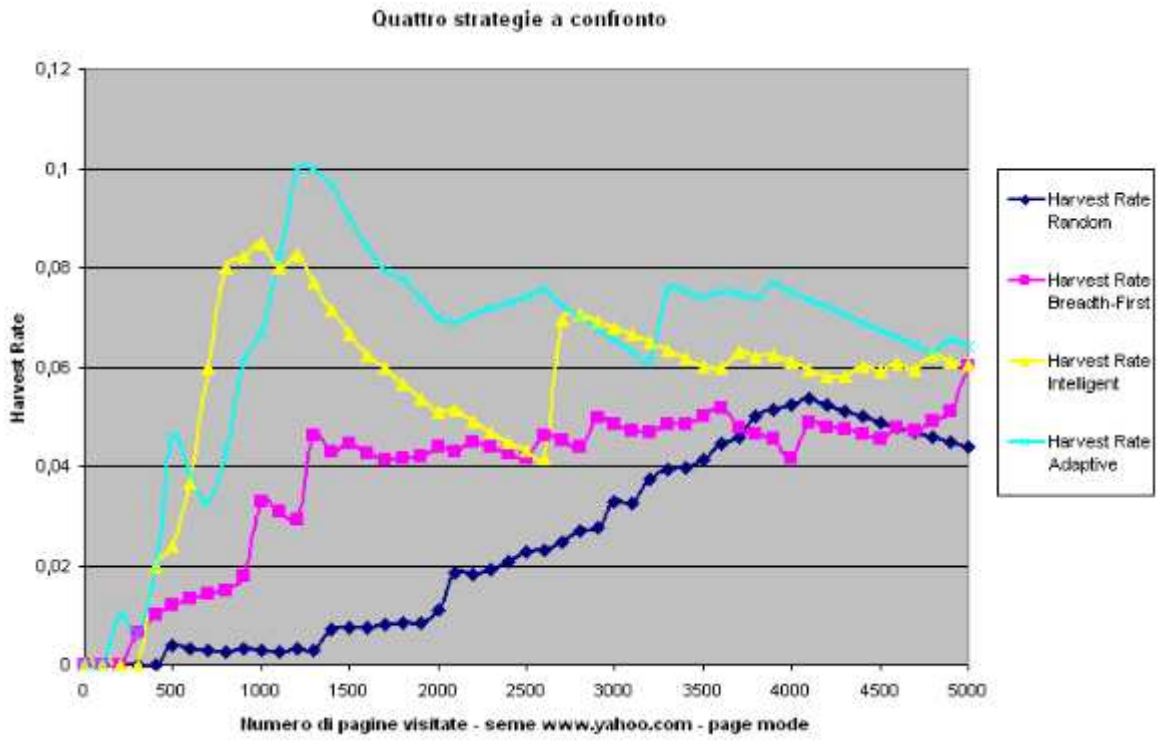

Figura 8.3: Esplorazioni page mode da www.berkeley.com

La Tabella rappresenta invece gli Harvest Rate medi ottenuti esplorando la rete in modalità *Site Mode*. Come si può notare gli harvest rate sono minori rispetto al caso precedente, ciò perchè l'agente è "costretto"a non abbandonare il sito in esame prima di aver scoperto tutte le pagine.

	Random	Breadth-First	Intelligent	Adaptive
Seme: www.yahoo.com	0.00209	0.00720	0.00893	0.01002
Seme: www.berkeley.edu	0.02193	0.0398	0.0581	0.06634

Tabella 8.2: Tabella comparativa degli Harvest Rate medi - risultati ottenuti da tre sessioni di crawling site mode per ognuna delle quattro strategie.

Anche in questo caso si riportano due grafici significativi, il primo in cui l'esplorazione inizia da *www.yahoo.com* (8.4), il secondo da *www.berkeley.edu* (8.5). Poichè *Yahoo!* è una Web directory molto estesa, si è impedito all'agente di esplorare esaustivamente il sito. Pertanto, uno degli indirizzi all'esterno del sito di *Yahoo!* è da considerarsi il vero *seed URL*. Se l'esplorazione di *Yahoo!* fosse stata permessa, l'agente si sarebbe "perso"in una delle cosiddette *Crawler Traps* [HEY 99].

Il problema da risolvere ora è scoprire se effettivamente il quarto parametro aggiunto alla strategia è un parametro sufficientemente robuto. A questo scopo sono state svolte 5 sessioni di crawling, di cui quattro guidate da uno solo dei parametri della strategia e una quinta cumulativa. I risultati sono illustrati in Figura 8.6. Come si può notare la strategia guidata dal solo synset match sembra essere migliore di tutte le altre. Si tratta di un risultato molto interessante, perchè significa che esiste all'interno del Web un'organizzazione implicita basata sui contenuti e una tecnica in grado di estrarre la semantica può sfruttare tale organizzazione per cercare efficientemente nuove risorse di informazione.

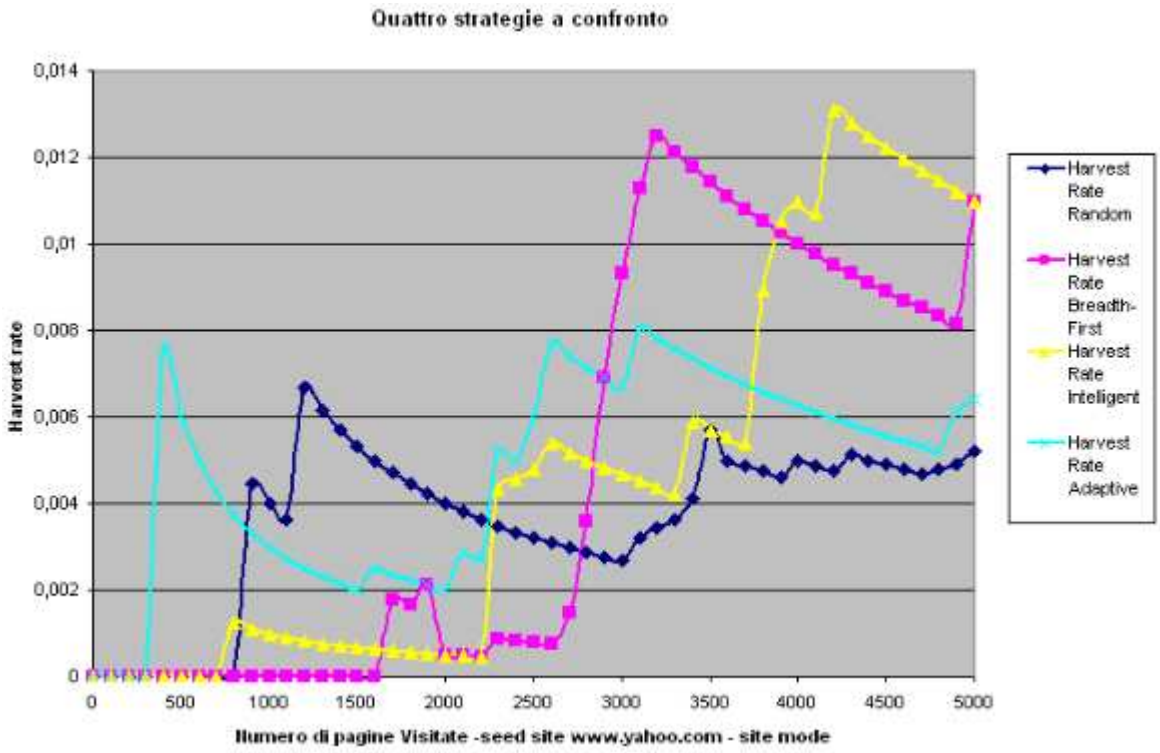

Figura 8.4: Esplorazioni site mode da “www.yahoo.com”

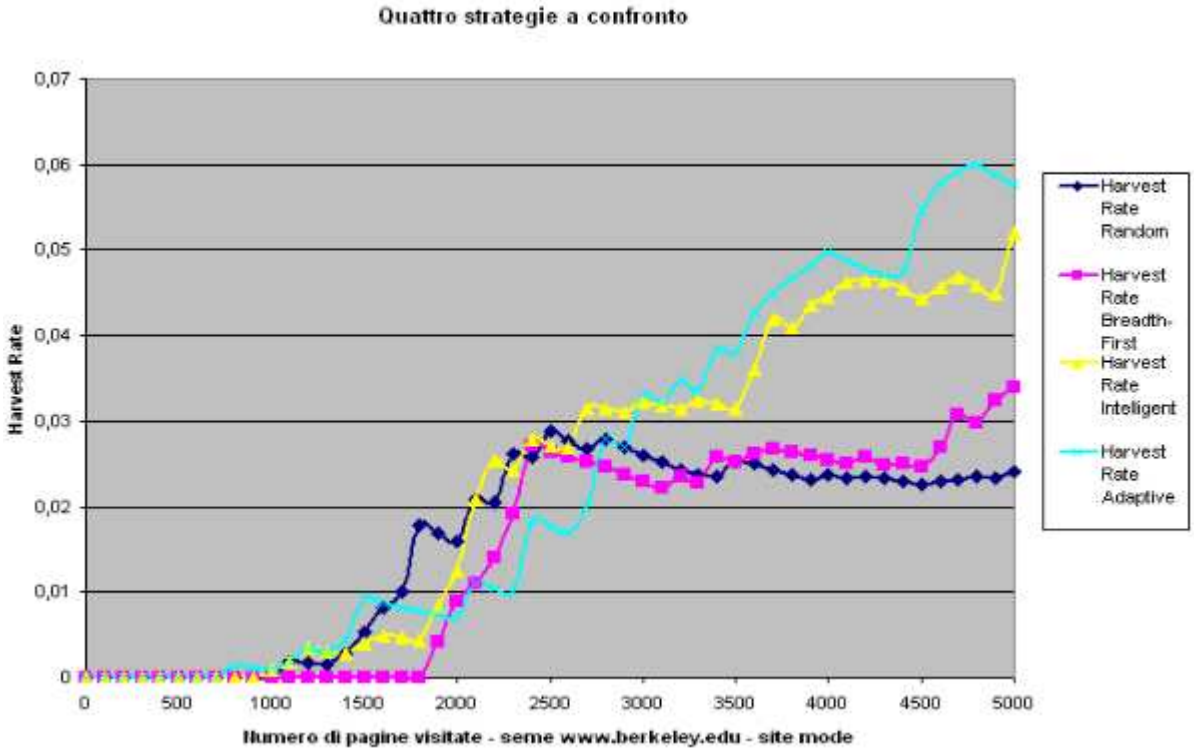

Figura 8.5: Esplorazioni site mode da www.berkeledy.edu

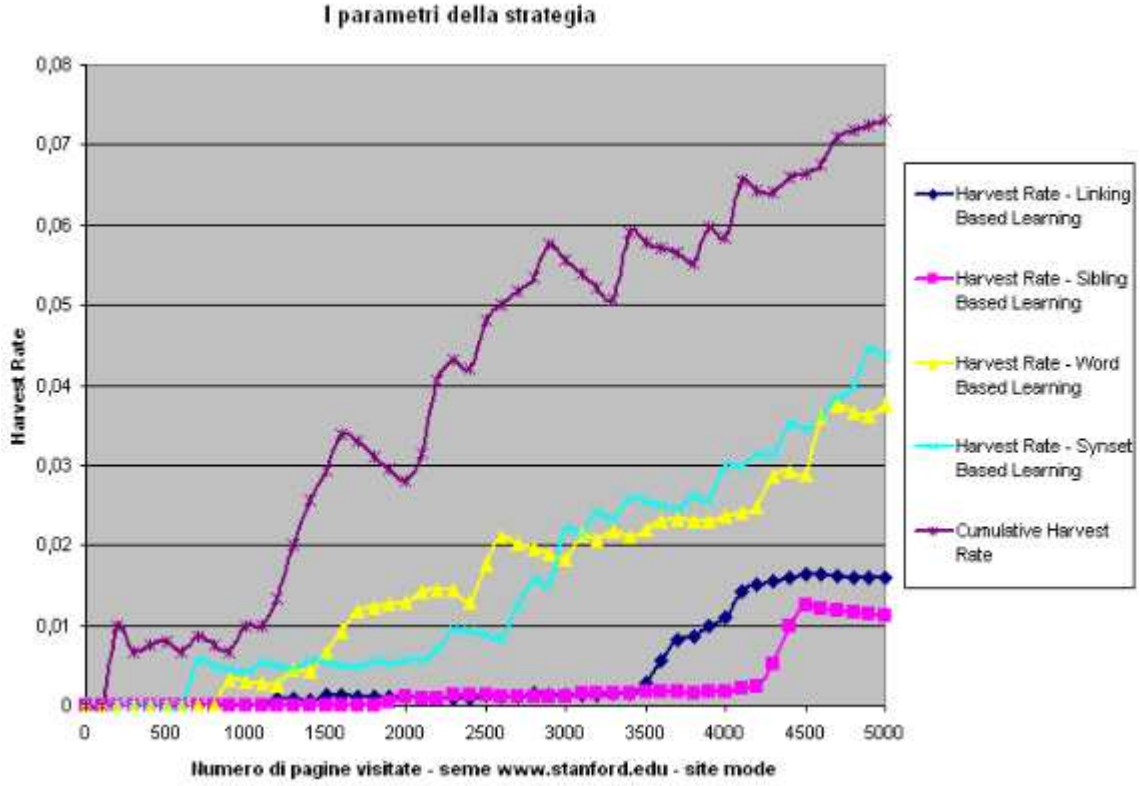

Figura 8.6: I parametri di learning a confronto

8.4.2 Confronto con Google

Nel precedente paragrafo si è dimostrato come la strategia di ricerca guidata dalla semantica sia in grado di recuperare pagine interessanti prima di altre non interessanti. In questo paragrafo si vuole dare una misura della bontà del metodo di ranking basato sulla coesione.

Si è pensato di realizzare un confronto con Google. Purtroppo, Google e TUCUXI sono due strumenti completamente differenti. Mentre il primo compie periodicamente esplorazioni esaustive, TUCUXI ha un approccio di tipo *focused*. Quindi, non è corretto fare confronti sulla copertura e sui fattori di *precision* e *recall*. Infatti, i concetti di *precision* e *recall* hanno senso se calcolati all'interno di una collezione controllata di documenti, come, ad esempio, l'indice di Google. TUCUXI è stato progettato per restituire pagine rilevanti non appena vengono scoperte e durante l'esplorazione non può mantenere che una visione parziale delle pagine pubblicate *on-line*.

Un altro fattore da considerare è l'algoritmo PageRank, il quale ha la propria ragione di essere sull'intera collezione di documenti. Il ranking di una pagina è sicuramente influenzato da migliaia di altre pagine che probabilmente TUCUXI non visiterà mai.

Inoltre, mentre per Google le query devono essere espresse tramite keywords, TUCUXI è più flessibile proprio perchè è portatore di un'ontologia.

Per realizzare un confronto equo, quindi, bisogna "normalizzare"le situazioni illustrate. In primo luogo entrambi devono poter accedere alla stessa collezione di documenti e devono avere le stesse possibilità di recuperare ed analizzare le varie pagine.

La soluzione proposta per realizzare un confronto equo prevede una fase preliminare in cui Google viene interrogato tramite keywords. Da qui in avanti si faccia riferimento alla query formata dai termini *faculty "computer science"course*. L'intenzione è quella di trovare i corsi avviati dalle facoltà di computer science. Inoltre, la ricerca di Google è stata confinata all'interno di siti ben specifici ed è stata filtrata in modo da ottenere solo le pagine .html, .htm, .shtml, .txt. Ciò perchè TUCUXI dovrebbe prettamente funzionare come crawler di siti e per il momento è in grado di analizzare solo alcuni tipi di documenti . Per il Common Thesaurus di esempio sono stati selezionati

quattro siti di prestigiose università statunitensi:

1. *www.umich.edu* - sito dell'Università del Michigan;
2. *www.stanford.edu* - sito della Stanford University;
3. *www.princeton.edu* - sito della Princeton University;
4. *www.berkeley.edu* - sito della Berkeley University.

Per ogni sito sono stati analizzati i primi 30 risultati, ipotizzando che 30 sia il numero massimo di risposte che l'utente visita prima di abbandonare una ricerca. Nelle tabelle seguenti, viene riportato il titolo sintetico di ogni pagina, il ranking di Google, nonchè due diversi punteggi attribuiti da TUCUXI. Il primo è riferito alla misura di coesione esponenziale, il secondo a quella con il meccanismo di *penalty*. Tra parentesi è indicato il ranking alla maniera di Google secondo quanto analizzato da TUCUXI. La colonna *Concetti* può assumere valore vero (Sì) o valore false (No). Essa assume valore vero quando nella pagina sono stati individuati i concetti espressi nella query (faculty, computer-science e course), mentre assume valore falso se non li contiene o si tratta di concetti *troppo deboli* rispetto al resto del discorso.

E' interessante notare che a volte Google riporta più volte un indirizzo, mentre TUCUXI si comporta come un crawler etico evitando di richiedere pagine già visitate. Inoltre, per il sito dell'Università del Michigan e per il sito di Berkeley vi sono alcune pagine che non è stato possibile recuperare (Errore 404), non per anomalie di TUCUXI ma per deficenze dei server. Si può quindi dire che Google ha dei dati poco aggiornati, nonostante i refresh periodici.

Per comprendere appieno le tabelle mi sembra doveroso dare una descrizione del *data set*. Si tratta generalmente di pagine a forte contenuto testuale piuttosto che visivo, soprattutto relative a news e/o articoli.

www.umich.edu (Titolo)	Google	TUCUXI 1	TUCUXI 2	Concetti
UM STS Undergraduate	1	-	-	NO
UROP Faculty Resource	2	0.73(7)	0.80(10)	SI
Amoco Undergraduate	3	0.83(4)	0.89(4)	SI
Seven Faculty Memb.	4	0.87(1)	0.91(3)	SI
Dell program fund	5	0.72(8)	0.82(6)	SI
Four Faculty Memb	6	-	-	SI
$5 Million Dollars	7	0.72(9)	0.81(9)	SI
UM College of Pharm	8	0.57(11)	0.65(11)	NO
7 Named A.F. Thurnau	9	0.87(2)	0.94(1)	SI
Undergraduate Courses	10	-	-	NO
13 receive Whitaker	11	0.53(12)	0.61(13)	SI
UM STS - Minor	12	-	-	NO
UMBS CIS: PhD	13	-	-	NO
Pharmacy SiteMap	14	Err. 404	Err. 404	?
UM unveils Ameritech	15	0.72(10)	0.84(5)	SI
Six faculty	16	0.74(5)	0.82(7)	SI
UM unveils Ameritech	17	Alr. Vis.	Alr. Vis.	SI
Business and education	18	0.52(13)	0.61(14)	NO
UROP Resource	19	0.86(3)	0.92(2)	SI
Six faculty	20	0.74(6)	0.82(8)	SI
Spring 2000	21	-	-	SI
Gilbert R. Whitaker	22	0.53(12)	0.62(12)	SI
Architecture FAQs	23	-	-	NO
Kang Geun Shin	24	-	-	SI
UM Kinesiology	25	-	-	NO
UROP Faculty	26	Err. 404	Err. 404	?
Health Science Scholars	26	Err. 404	Err. 404	?
Learning Collaboratory	27	Err. 404	Err. 404	?
20 faculty awarded	28	Err. 404	Err. 404	?
Graduate Education	29	Err. 404	Err. 404	?
17 faculty members	30	Err. 404	Err. 404	?

www.stanford.edu (Titolo)	Google	TUCUXI 1	TUCUXI 2	Concetti
Faculty and Courses	1	-	-	NO
Faculty 15 dept	2	0.93(1)	0,98(1)	SI
April 26 memorial	3	-	-	NO
Computer Science Dept	4	-	-	NO
CS 206 Technical Found.	5	-	-	NO
Hennessy gives...	6	-	-	NO
SCPD ProEd	7	-	-	NO
Untitled	8	-	-	NO
STS BA Thematic	9	0.71(6)	0.82(6)	SI
Computer scientists	10	-	-	NO
News	11	-	-	NO
Faculty Senate	12	-	-	NO
STS Adv. Commitee	13	-	-	NO
Stanford Biology	14	0.71(7)	0.84(7)	SI
Syllabus IHUM	15	0.85(3)	0.93(3)	SI
Orgish wins...	16	-	-	NO
M. E. Paté-Cornell	17	-	-	SI
Four faculty	18	0.72(5)	0.83(5)	SI
Stanford Computer	19	0.71(8)	0.83(8)	SI
About SiliconBase	20	-	-	NO
Stanford Science	21	-	-	SI
Charles Orgish	22	0.86(2)	0.95(2)	SI
Bio X awards	23	-	-	NO
Student projects	24	0.78(4)	0.83(4)	SI
Stanford launches...	25	-	-	NO
Majoring in East	26	-	-	NO
Anton Ushakov	27	-	-	NO
Concentrations	28	-	-	NO
CNS: INTERNET2 AT...	29	0.51(10)	0.62(10)	NO
2002 New Student	30	0.63(9)	0.75(9)	NO

www.princeton.edu (Titolo)	Google	TUCUXI 1	TUCUXI 2	Concetti
Grad. Student Announce	1	0.71(9)	0.81(9)	SI
Innovative courses	2	0.72(7)	0.79(10)	SI
Under. Student Announce	3	-	0.51(18)	SI
Dobkin named dean	4	0.53(11)	0.61(13)	SI
Computer Engineering	5	-	-	NO
Program of Mathematics	6	-	-	SI
E-Council honors	7	0.94(1)	0.99(1)	SI
From PowerPoint	8	0.94(2)	0.99(2)	SI
Engineering at Princeton	9	0.94(3)	0.99(3)	SI
Department of CS	10	-	-	SI
Bioengineering Track	11	Err.404	Err. 404	?
School of Engineerings	12	0.51(14)	0.62(12)	SI
Three faculty members	13	-	-	SI
Preceptorships	14	-	-	SI
Professor Debenedetti	15	0.53(12)	0.58(15)	SI
Professor Lipton	16	0.86(4)	0.92(6)	SI
Electronic Materials	17	0.51(15)	0.61(14)	SI
Princeton University	18	0.71(10)	0.82(8)	SI
Notebook: January	19	0.86(5)	0.93(4)	SI
Department of Mathematics	20	-	-	NO
Intro Departments	21	0.52(13)	0.64(11)	NO
Personal	22	-	-	NO
School of Engineering	23	-	-	NO
Optimization	24	-	-	SI
Undergraduate Engineering	25	0.85(6)	0.93(5)	SI
Lorene Lavora	26	0.72(8)	0.87(7)	NO
Notebook: January	27	Alr. vis.	Alr.vis	SI
Customized Concentration	28	-	-	SI
Forbes College	29	0.50(16)	0.58(16)	SI
Structural Engineering	30	0.50(17)	0.57(17)	SI

www.berkeley.edu (Titolo)	Google	TUCUXI 1	TUCUXI 2	Concetti
From Desks to Desktops	1	0.73(4)	0.81(5)	SI
On-line plagiarism	2	0.52(7)	0.61(11)	SI
Disability brainstorm	3	0.87(1)	0.91(2)	SI
From Desks to Desktops	4	Alr. vis.	Alr.vis	SI
Newest and oldest UC	5	-	-	NO
Awards and Honors	6	-	-	NO
Hewlett Courses	7	0.52(9)	0.60(13)	NO
Expanding Knowledge	8	0.86(2)	0.94(1)	SI
The Fine Art of Teaching	9	0.52(10)	0.67(6)	NO
Online	10	-	-	NO
UC Resource list	11	-	-	NO
Student programmers	12	0.72(5)	0.83(4)	NO
Four UC	13	-	0.56(14)	SI
UC News Jun 24	14	Err. 404	Err. 404	?
UC News Jul 17	15	Err. 404	Err. 404	?
UC News Sep 18	16	Err. 404	Err. 404	?
AZ-LIST	17	-	-	NO
In a Class	18	0.51(11)	0.63(9)	NO
News Aug 22	19	0.51(12)	0.64(7)	NO
Gazette 3/15/95	20	0.53(7)	0.62(10)	NO
General Catalog	21	-	-	NO
Bill Oldham	22	-	-	NO
Wong Named..	23	0.85(3)	0.88(3)	SI
James Named..	24	-	-	NO
University Relations	25	-	-	NO
Graduates to hear	26	0.54(6)	0.61(12)	NO
UC Berkeley students	27	-	-	NO
Academic Deparment	28	-	-	NO
Gazette, 3/8/95	29	0.51(13)	0.64(8)	NO
Governor	30	-	-	NO

L'aggiornamento di Google è comunque un fatto secondario. Qui interessa valutare la bontà della funzione di scoring, nonchè le capacità di TUCUXI di filtrare le informazioni rilevanti da quelle non rilevanti. Non è possibile riportare tutto il data set, sia per motivi di ordine pratico che per ragioni legali. Come si può osservare, TUCUXI considera rilevante solo una piccola parte dei risultati forniti da Google.

Inoltre, TUCUXI ha la capacità di individuare pagine rilevanti anche quando non sono contenuti i termini della query. In primo luogo perchè è in grado di individuare concetti che pur non essendo rappresentati esplicitamente sono comunque intuibili. Nella costruzione delle catene lessicali, infatti, un concetto potrebbe non essere espresso ma tramite esso è possibile aggregare e porre in relazione significati di termini contenuti nel testo. Secondariamente, TUCUXI può individuare un determinato concetto tramite i sinonimi. Ad esempio, molte delle pagine rilevanti contengono la locuzione *academic staff* che ha lo stesso significato di *faculty*, sebbene quest'ultimo termine non compaia nei documenti.

Vediamo ora le prime cinque pagine con il maggior punteggio di rilevanza all'interno dei quattro siti considerati:

www.stanford.edu (Titolo)	Google	TUCUXI 1	TUCUXI 2	Concetti
Stanford Courses	38	0.98(1)	0.99(1)	SI
Stanford University	44	0.97(2)	0.99(2)	SI
The Year un...	75	0.96(3)	0,98(3)	SI
Bachelor of Sc.	-	0.96(3)	0.98(4)	SI
Press release	-	0.95(5)	0.98(5)	SI

Dalle tabelle si evince che solo una piccola parte delle pagine rilevanti per Google sono rilevanti anche per TUCUXI.

Inoltre, TUCUXI è in grado trovare pagine rilevanti anche senza la presenza di keywords all'interno di un testo. Ad esempio la pagina dal titolo

Bachelor of Sc. del sito di Stanford non contiene la parola faculty nè la parola student. TUCUXI risale comunque ai concetti di facoltà e corso sulla base dei termini utilizzati, ad esempio nel testo ci si riferisce spesso ai metodi per un buon "undergraduate teaching".

www.berkeley.edu (Titolo)	Google	TUCUXI 1	TUCUXI 2	Concetti
Review the Computer-Science	43	0.98(1)	0.999(1)	SI
Review the Dept Chim	44	0.97(2)	0.96(2)	NO
News about courses	54	0.97(3)	0.96(3)	SI
Review the Dept Fis	46	0.97(4)	0.96(4)	NO
Review the Dept Math	67	0.97(5)	0.96(5)	NO

www.princeton.edu (Titolo)	Google	TUCUXI 1	TUCUXI 2	Concetti
Program in NeuroScience	65	0.97(1)	0.998(1)	SI
News Prof. Kean	-	0.96(2)	0.998(2)	SI
E-Council honors	7	0.94(3)	0.997(5)	SI
From PowerPoint	8	0.94(4)	0.997(4)	SI
Engineering at Princeton	9	0.94(5)	0.998(3)	SI

www.umich.edu (Titolo)	Google	TUCUXI 1	TUCUXI 2	Concetti
Program of tech	47	0.98(1)	0.99(1)	SI
Prof. Keen moves...	88	0.97(2)	0.98(2)	SI
News Regent July..	-	0.96(3)	0.98(3)	NO
News Regent May...	-	0.95(4)	0.97(4)	SI
Dell program	-	0.95(5)	0.97(5)	NO

8.4.3 Gli Errori di TUCUXI

Anche TUCUXI sbaglia. A mio avviso, la pagina che Google segnala al primo posto nel sito di Stanford è una pagina rilevante. In Figura 8.7 è rappresentato un estratto della pagina dal titolo *Faculty and Course*, così come è consultabile dall'interfaccia di TUCUXI.

Si tratta di un elenco di professori, con indicazione del dipartimento di afferenza e del corso presentato. Si nota che la parola *Department* viene ripetuta spesso, ma il concetto di dipartimento non viene rilevato. Non sembra si possa trattare di un concetto poco importante eliminato in fase di disambiguazione, perchè la sua ripetizione per decine di volte dovrebbe assicurarne la presenza. Quindi, l'errore non può che essere a monte ed è il part of speech tagger che non lo riconosce correttamente come nome.

Altre pagine in cui TUCUXI sembra sbagliare sono alcuni documenti in cui si parla di corsi, ma non ci si riferisce ad essi esplicitamente. Ad esempio, vi sono alcune pagine che elencano i corsi con un codice e descrivono sinteticamente gli argomenti trattati. In questi casi le informazioni a disposizione di TUCUXI sono insufficienti, e seppure il lemma *course* sia presente nel testo, il relativo concetto viene perso, perchè marginale rispetto alla componente testuale del programma dei corsi.

8.5 Il Manuale Utente

TUCUXI mette a disposizione dell'operatore un'interfaccia *user-friendly* in cui definire tutti i parametri necessari sia per la strategia di esplorazione che per il sistema di ranking.

Nel riquadro in alto a sinistra nella Figura 8.8 l'utente può selezionare il peso relativo di ognuno dei quattro parametri che guidano la strategia. Determinare il peso di uno dei quattro parametri significa determinare automaticamente anche i rimanenti. Se il peso di un parametro è posto a 0 significa che TUCUXI non assegnerà alle pagine da esplorare alcuna priorità dipendente da quel parametro.

Nel riquadro in basso a sinistra possono essere determinati altri parametri fondamentali quali il tipo di strategia (*Random*, *Breadth-first*, *Intelligent*

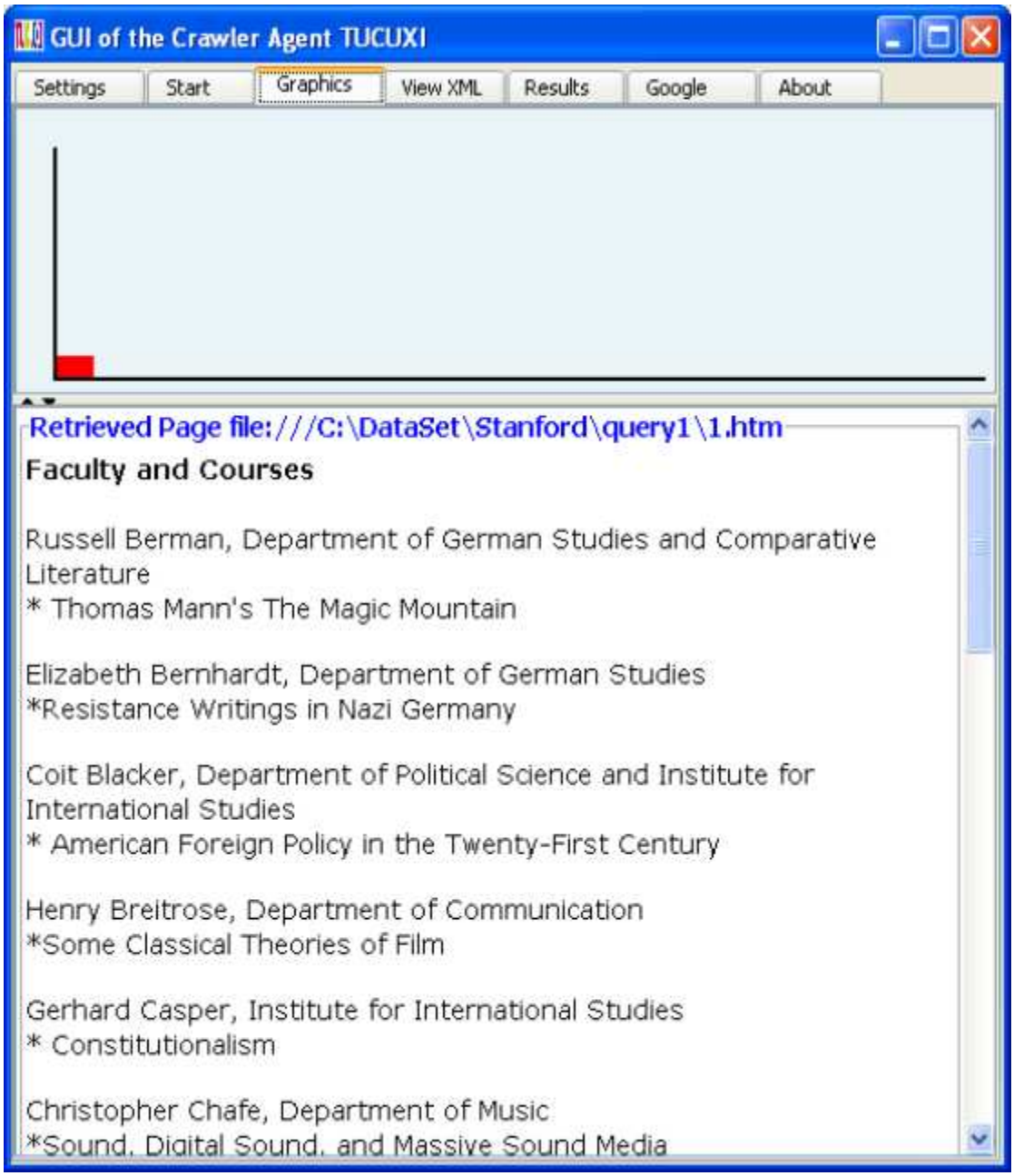

Figura 8.7: Un errore di TUCUXI??

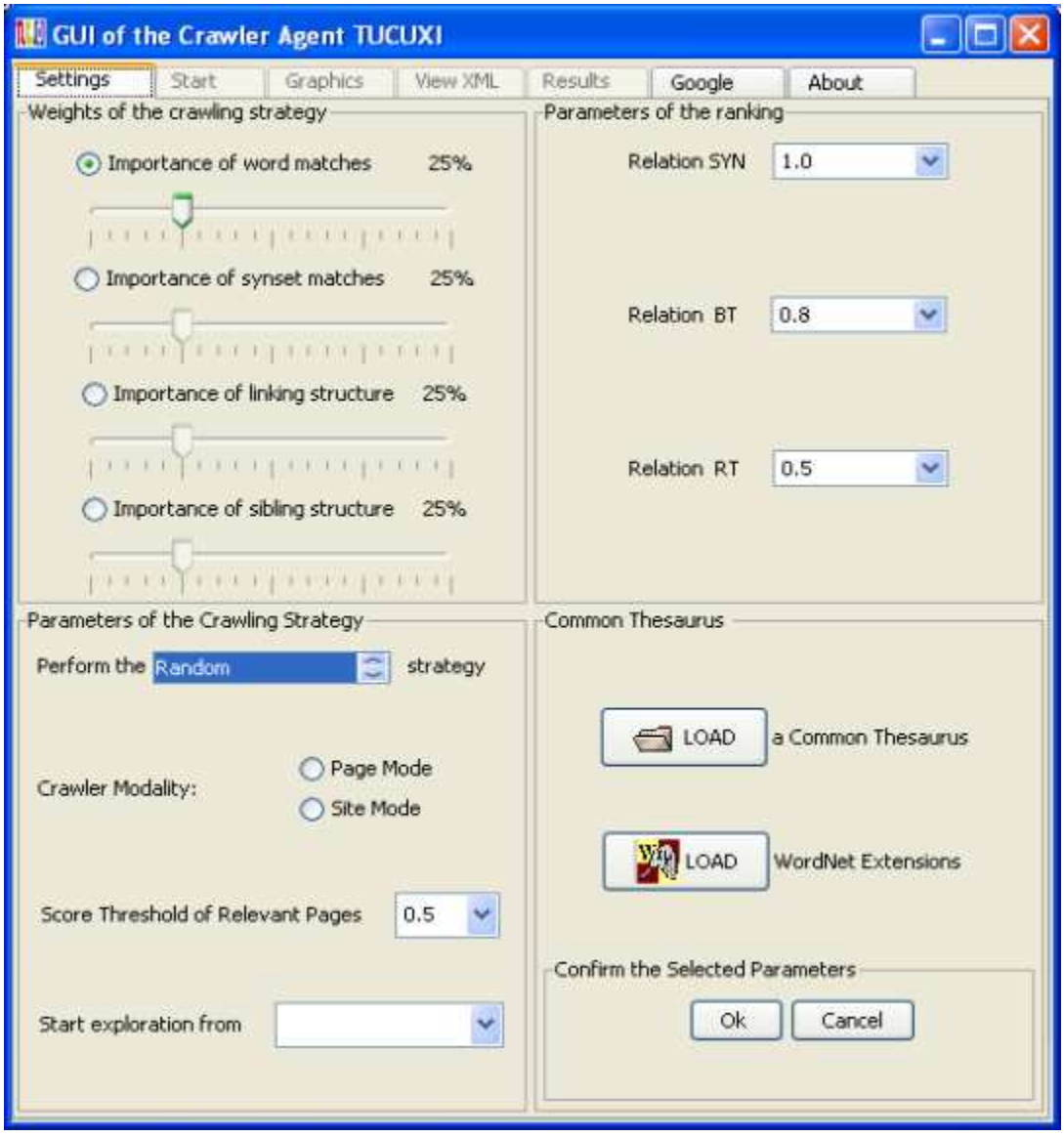

Figura 8.8: Setting Panels

ed *Adaptive*), la modalità con cui visitare i documenti e il sito o la pagina da cui iniziare l'esplorazione. La modalità *site mode* prevede che l'agente abbandoni il sito solamente quando ha compiuto un'esplorazione esaustiva dello stesso. La modalità *page mode* rende l'agente un comune crawler che visita le pagine indipendentemente dal sito di appartenenza. Un altro parametro estremamente importante è il valore di soglia (*Score threshold*) oltre il quale considerare una pagina rilevante. Oltre alle suddette modalità TUCUXI permette di analizzare anche una lista di indirizzi memorizzati in un file "url-list.xml". Gli stessi possono essere utilizzati anche come punto di partenza per l'esplorazione della rete. Un'altra caratteristica di TUCUXI è la capacità di filtrare i risultati di una query di Google. Dall'interfaccia è possibile interrogare questo motore di ricerca sia tramite la search interface, sia tramite protocollo **SOAP** - **S**imple **O**bject **A**ccess **P**rotocol [12]. Nel primo caso appare una maschera come quella di Figura 8.9, mentre nel secondo caso viene attivato il pannello *Google* (Figura 8.10). In questo pannello l'utente viene guidato nella costruzione delle query da porre a Google. Oltre ai termini contenuti nel Common Thesaurus è possibile scegliere anche di includere eventuali sinonimi (secondo WordNet) ed altri termini a scelta.

Nel riquadro in alto a destra l'utente può determinare i pesi delle singole relazioni lessicali. Determinare il peso delle relazioni significa influire su come vengono calcolati i punteggi delle catene lessicali e delle mappe concettuali derivate. Si consiglia di utilizzare i pesi indicati per uniformità con quelli standard utilizzati in MOMIS. Infine, nel pannello in basso a destra l'utente può scegliere il Common Thesaurus (Figura 8.11) da utilizzare ed eventuali estensioni di WordNet (8.12). Poichè WordNet è una base di conoscenza comune, le sue eventuali estensioni potrebbero rivelarsi fondamentali in ambiti altamente specialistici.

[12] È un protocollo basato su XML e nasce con lo scopo di migliorare il trasporto dei dati fra sistemi distribuiti eterogenei decentralizzati. SOAP rappresenta il mezzo per mettere in contatto piattaforme diverse: tralasciando problematiche legate alle comunicazioni fisiche, ai sistemi operativi, ai linguaggi di programmazione ed ai protocolli di comunicazione. Google mette a disposizione apposite API per interrogare la propria cache tramite protocollo SOAP

Figura 8.9: Finestra per la Google Search Interface

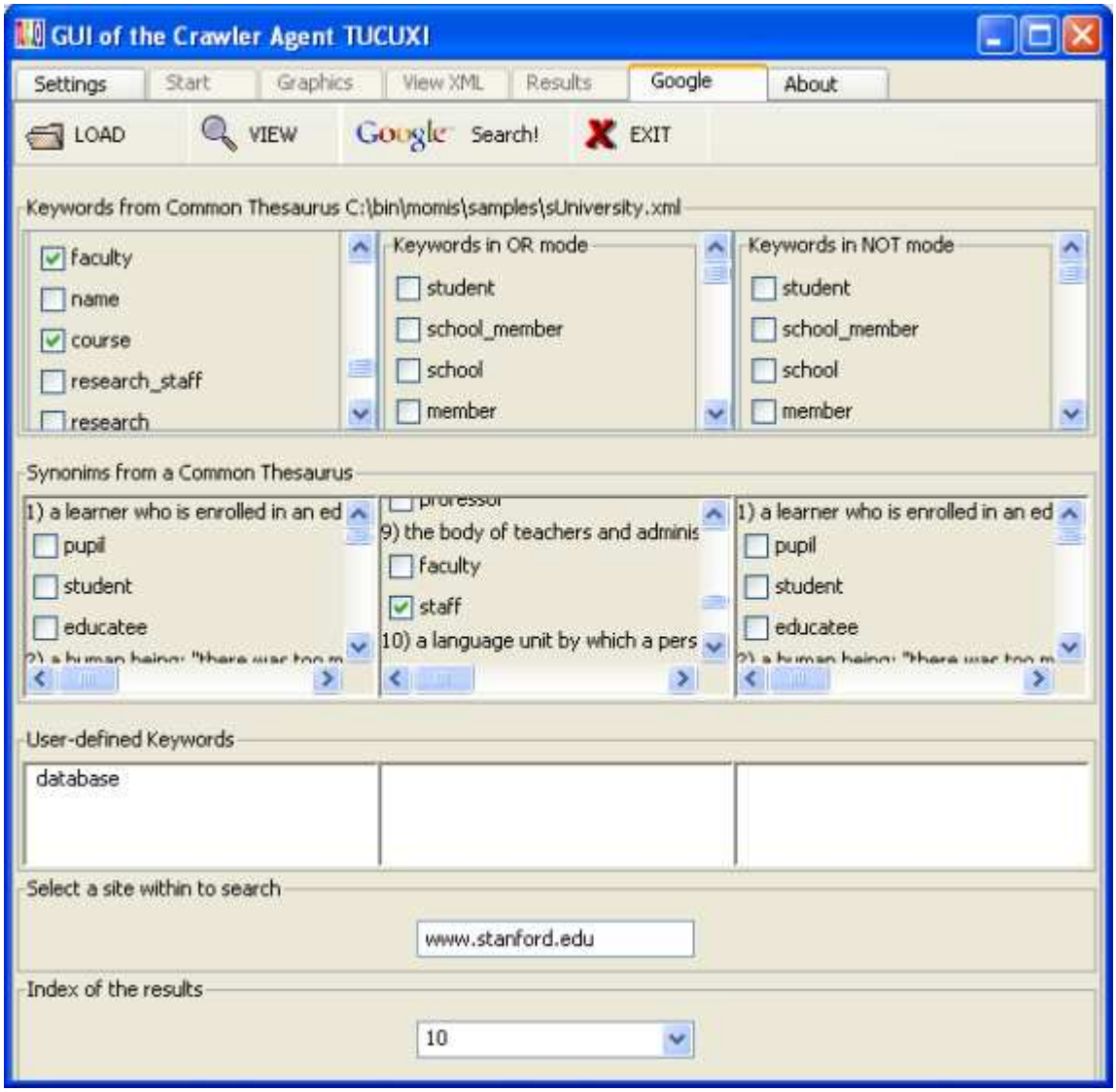

Figura 8.10: Finestra per l'interazione con Google tramite protocollo SOAP

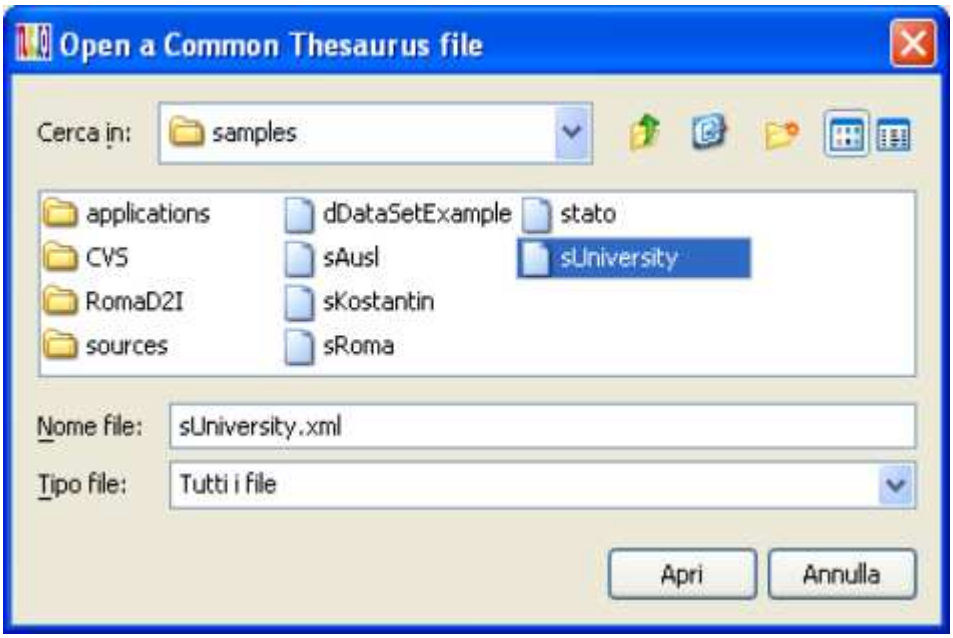

Figura 8.11: Scelta del Common Thesaurus

Figura 8.12: Scelta delle estensioni a WordNet.

Se tutti i parametri selezionati sono corretti, vengono attivati i pannelli *Start*(Figura 8.13), *Graphics*(Figure 8.14 e 8.15) e *View XML* (Figura 8.16).

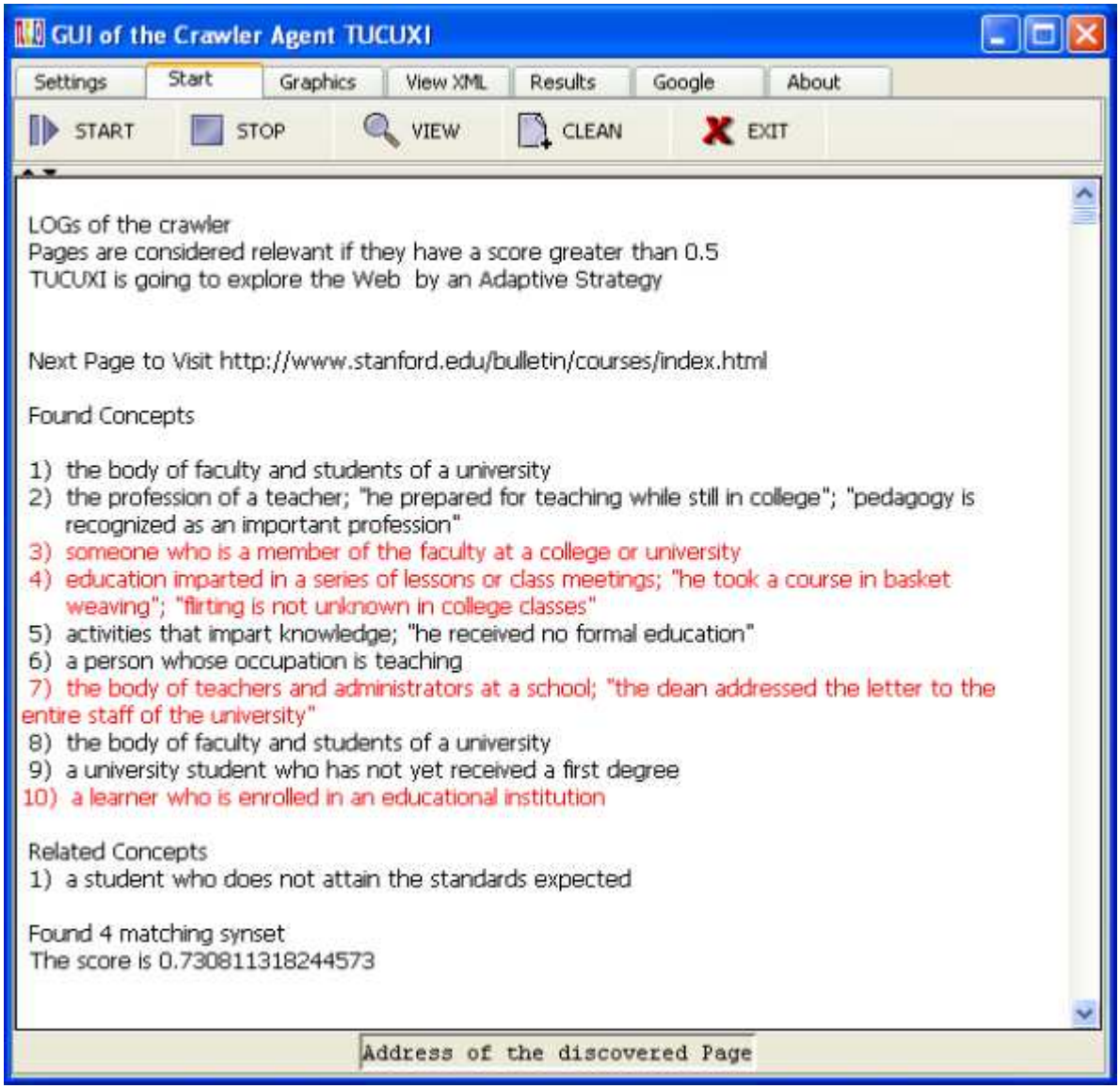

Figura 8.13: Il pannello Start.

Nel pannello *Start* è possibile monitorare l'attività dell'agente osservando la descrizione dei concetti contenuti nelle pagine visitate (*Found Con-*

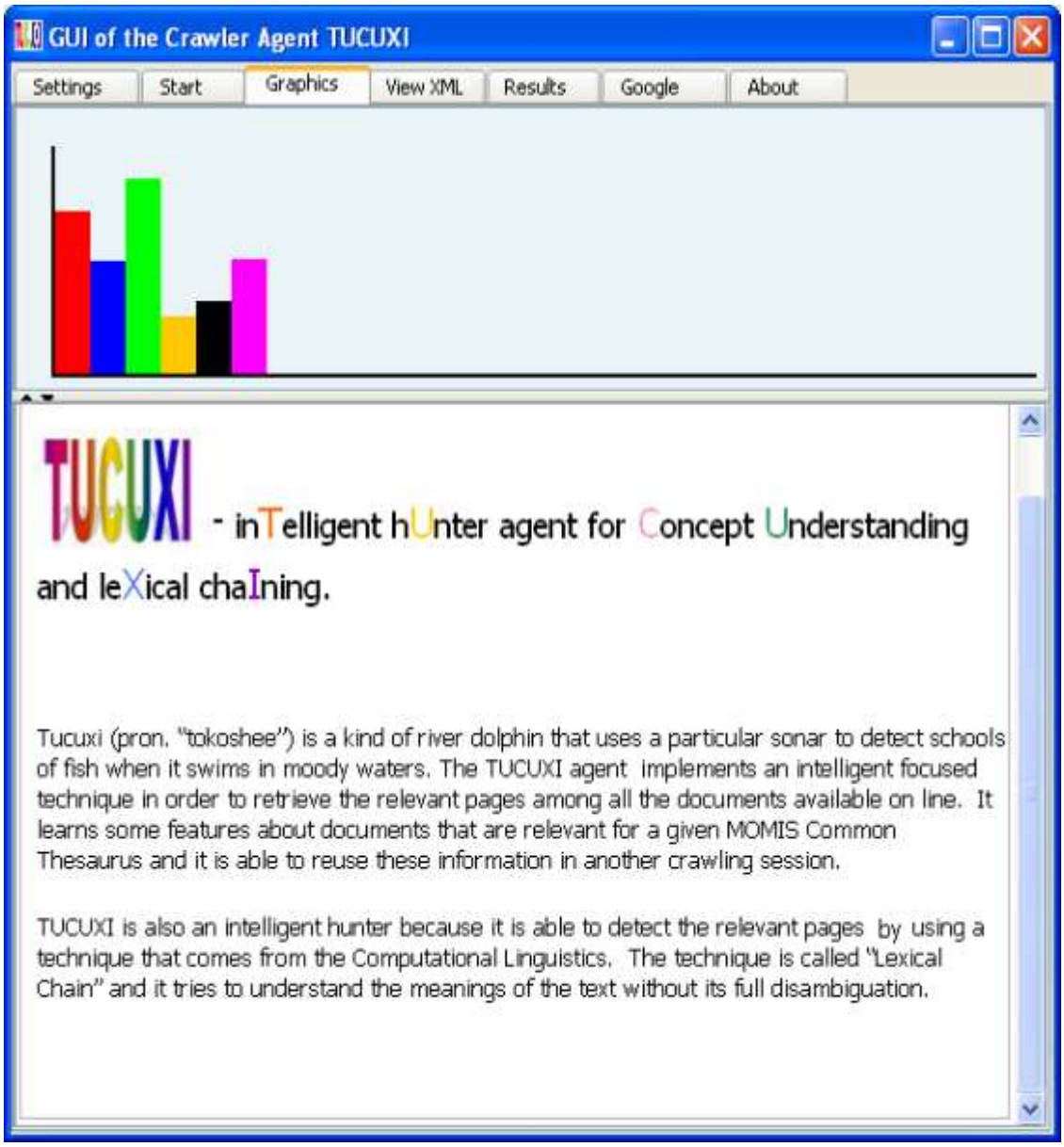

Figura 8.14: Il pannello Graphics - pagina di presentazione.

cept), nonchè le glosse dei concetti che, pur non essendo espressi nel testo esplicitamente tramite i rispettivi termini, sono comunque intuibili (*Related Concepts*).

Nel pannello *Graphics* è possibile monitorare l'attività dell'agente in modalità grafica. Nella parte superiore del pannello compare l'istogramma dei punteggi assegnati alle pagine visitate, più alta è la barra e più alta è la sua rilevanza nei confronti del Common Thesaurus. Nella parte inferiore del pannello, la pagina di presentazione di TUCUXI può essere sostituita con una qualsiasi delle pagine già visitate. Cliccando con il tasto destro del mouse su una delle barre dell'istogramma è possibile accedere alla relativa pagina e, ad esempio, esplorare la rete nell'ordine determinato dall'agente (Figura 8.15).

Nel pannello *View XML* è possibile visualizzare e salvare il file XML generato da TUCUXI durante l'esplorazione. Si ricorda che se nel tag META di HTML viene indicato di non indicizzare una pagina, essa non comparirà nel file XML. E' comunque possibile accedere alla pagina tramite l'istogramma dei punteggi.

Infine, al termine di una sessione di esplorazione, TUCUXI è in grado di riassegnare i punteggi alle pagine applicando una versione dell'algoritmo HITS. Nella parte superiore del pannello sono elencate tutte le pagine visitate con indicazione del punteggio ottenuto in base al confronto con il Common Thesaurus (*Content Score*) e il punteggio di authority (*Authority Score*). Nella parte inferiore del pannello, invece, viene assegnato un punteggio al sito, determinato grazie ai punteggi delle singole pagine. Questi punteggi vengono corretti con coefficienti che tengono conto della profondità in cui i vari documenti appaiono all'interno della gerarchia del sito.

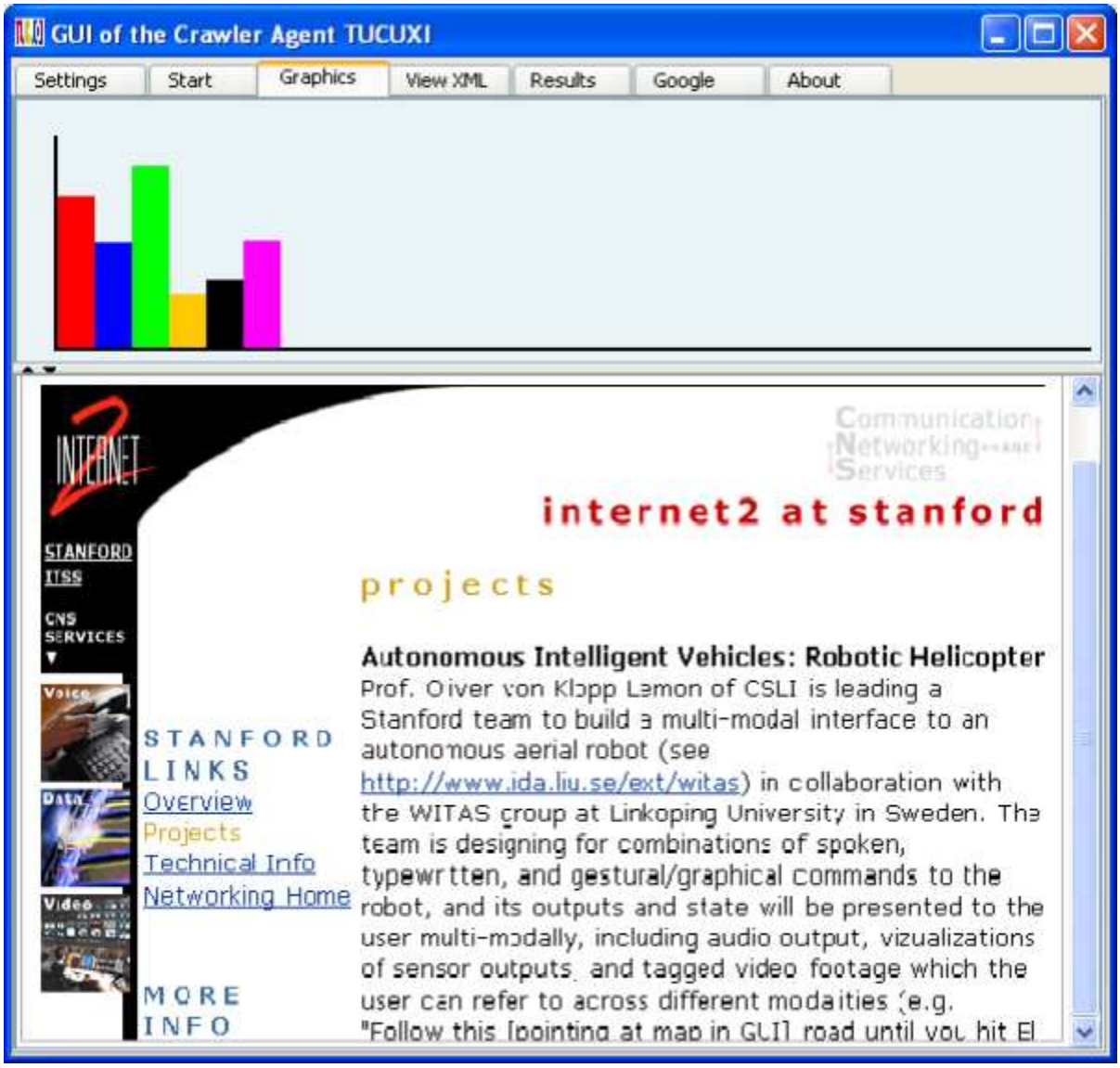

Figura 8.15: Il pannello Graphics - una pagina già visitata.

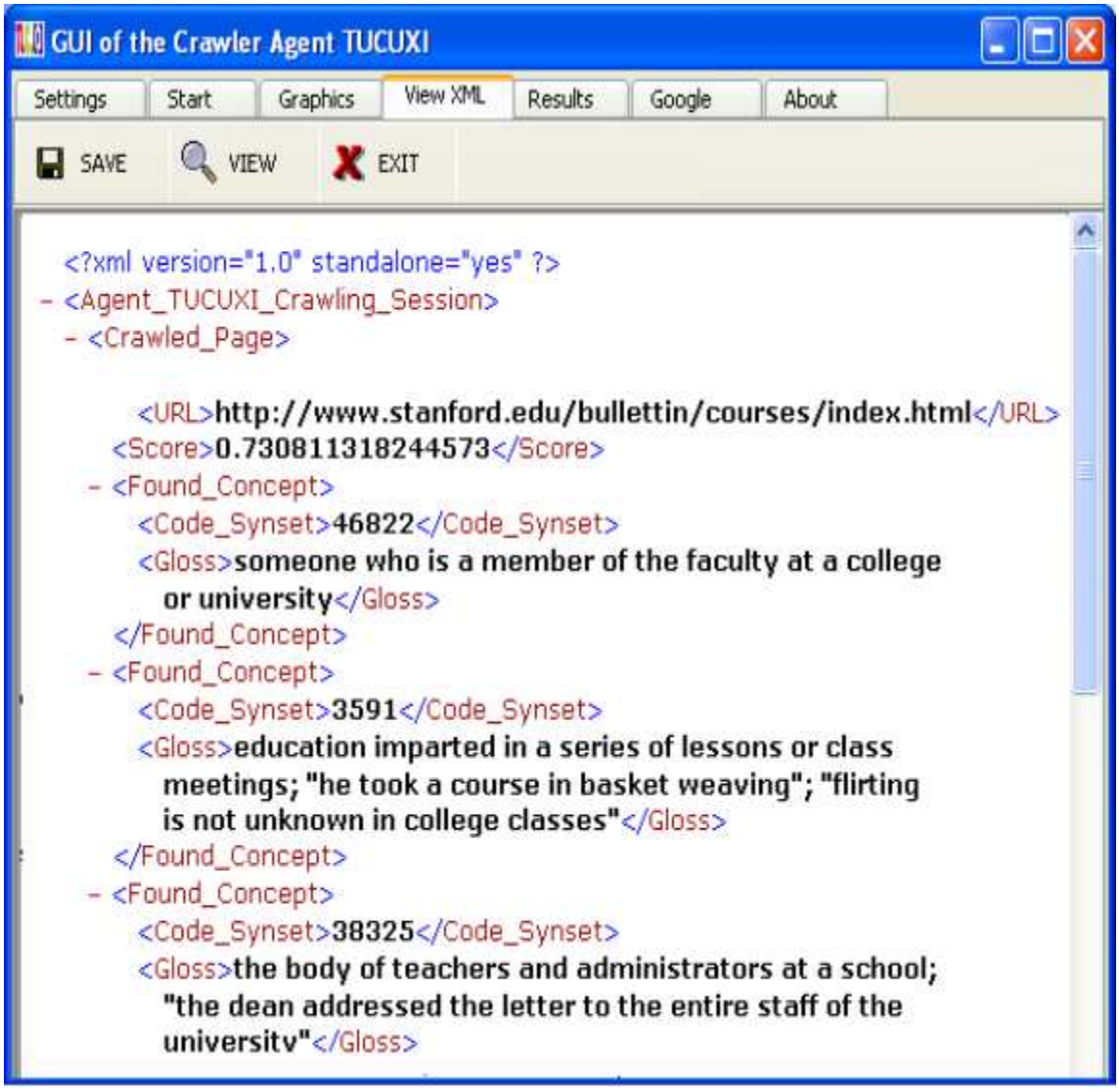

Figura 8.16: Il pannello View XML - il file XML generato.

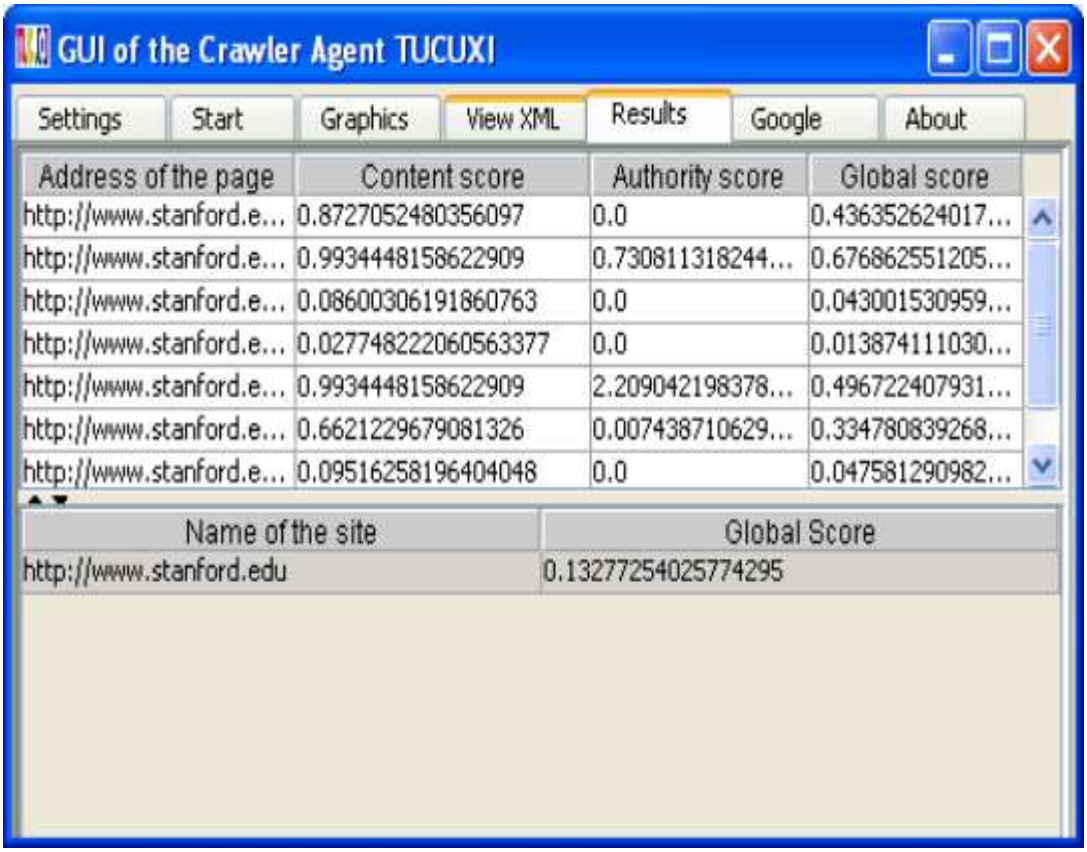

Figura 8.17: Il pannello Results - Punteggi globali.

Capitolo 9

Conclusioni e Sviluppi Futuri

In questa tesi è stato progettato e realizzato un agente hunter denominato TUCUXI (In**T**elligent H**U**nter Agent for **C**oncept **U**nderstanding and Le**X**ical Cha**I**ning).

TUCUXI è un agente JADE scritto interamente in Java che mette a disposizione dell'utente un'interfaccia *user-friendly* tramite la quale monitorare il processo di esplorazione del Web. Attualmente TUCUXI analizza documenti in formato .html, .htm, .shtml e .txt. Per quanto riguarda le pagine dinamiche, l'agente può recuperarle quando esse sono inserite come iperlink all'interno di un documento, ma non è ancora in grado di esplorare quello che viene definito *Hidden Web*.

TUCUXI è un agente innovativo sotto diversi punti di vista. In primo luogo esso adotta una strategia intelligente per visitare il maggior numero di pagine rilevanti attraversando il numero minore di documenti non interessanti. A differenza di altri approcci descritti in letteratura [DIL 00] [CHA 99], TUCUXI ha una validità generale ed è in grado di cercare informazioni rilevanti rispetto a diversi Common Thesaurus. Tale capacità gli deriva dalla costruzione di un modello statistico rappresentativo delle condizioni in cui opera. Inoltre, TUCUXI è in grado di riutilizzare parte delle informazioni contenute nel modello statistico anche in successive sessioni di crawling. La costruzione del modello statistico è simile a quella descritta in [YU 01], con l'introduzione di un nuovo parametro di ricerca basato sulla semantica del contenuto dei documenti.

A quanto mi risulta, TUCUXI è uno dei primi agenti, se non il primo in assoluto, ad utilizzare tecniche di *Natural Language Processing* per individuare sorgenti di dati interessanti. I risultati ottenuti sono estremamente incoraggianti: non solo la strategia di ricerca basata sulla semantica è una tecnica robusta, ma permette anche di realizzare un buon filtro per i motori di ricerca tradizionali come Google. Inoltre, è possibile utilizzare TUCUXI anche come crawler per costruire la collezione di documenti per motori di ricerca specializzati, proprio perchè è in grado di recuperare pagine di buona qualità e di elevato grado di rilevanza. Il grado di rilevanza di un documento è attualmente calcolato sulla base di tutto il testo contenuto. In alcuni casi si è notato che il punteggio può scendere drasticamente quando gli argomenti trattati in un documento sono molteplici. Segmentando manualmente il testo si è scoperto che il punteggio dei singoli paragrafi tende a privilegiare quelli più attinenti al Common Thesaurus. Sebbene in questo caso sussista il rischio che la disambiguazione dei termini sia incompleta, si potrebbe pensare ad un sistema automatico di segmentazione per poi individuare paragrafi altamente rilevanti e guidare eventualmente un estrattore di dati. La stessa tecnica delle catene lessicali si presta anche per realizzare un' indicizzazione concettuale, così come indicato in [AGG 2001].

Focalizzando l'attenzione proprio sulla tecnica di *Natural Language Processing* utilizzata, si può affermare che anch'essa lascia intravedere ulteriori possibilità di sviluppo. Innanzitutto, si tratta di una tecnica indipendente dalla lingua in cui un testo è scritto. La sua validità generale è dimostrata in [FUE 02] e deriva dal fatto che la coesione è una proprietà fondamentale dei testi in quanto tali.

Pertanto, un possibile sviluppo futuro è l'implementazione di un sistema multi-agente (MAS) basato non più su WordNet ma su EuroWordNet [GIL 96].

Appendice A

Appendice

A.1 KAON

KAON (**KA**rlsruhe **ON**tology and Semantic Web framework) è un'infrastruttura open-source per la creazione, l'evoluzione e la gestione di ontologie e fornisce il supporto per lo sviluppo di applicazioni ontology-based. KAON è tuttora concentrato nell'integrazione delle tradizionali tecnologie per l'ontology management con quelle utilizzate dalle business applications (es. database relazionali).

I componenti di KAON possono essere suddivisi in tre livelli (*layers*):

Applications and Services Layer: fornisce le necessarie interfacce per permettere l'interazione con i livelli sottostanti sia da parte di operatori umani che da software automatici. In particolare, l'applicazione OntoMat-SOEP guida l'utente durante il processo di modifica e validazione delle ontologie.

Middleware Layer: in cui KAON-API e RDF-API realizzano l'accesso alle ontologie indipendentemente dalle modalità con cui esse vengono memorizzate (database relazionali o RDF models).

Data and Remote Services Layer: fornisce servizi di data storage e garantisce l'atomicità delle transazioni di update delle ontologie.

All'interno del KAON framework è disponibile TEXT-TO-ONTO, un tool semiautomatico che implementa algoritmi di *text mining* per la creazione di

ontologie a partire da testi scritti. TEXT-TO-ONTO individua tramite un algoritmo di term extraction un set di termini che potrebbero essere inclusi in un'ontologia come nuovi concetti ed estrae le relazioni tassonomiche e non tassonomiche che intercorrono fra i nuovi concetti e quelli già esistenti. Tramite TEXT-TO-ONTO è possibile costruire una ontologia domain-specific a partire da una general ontology come WordNet.

Un'altro tool interessante è RDFCrawler, un crawler che individua frammenti RDF all'interno di pagine Web e costruisce una base di conoscenza a partire dai dati raccolti.

A.2 GATE

GATE [1] (**G**eneral **A**rchitecture for **T**ext **E**ngineering) è un'infrastruttura open source per lo sviluppo di sistemi di *Language Engineering* proposta dall'Università di Sheffield (UK). GATE si compone di 3 elementi principali:

GATE Document Manager - GDM, un database basato su di un modello object-oriented per il management delle informazioni generate durante l'elaborazione dei testi da parte di più moduli di un sistema di LE;

GATE Graphical Interface - CGI, un' interfaccia grafica ed un ambiente di sviluppo integrato;

Collection of REusable Objects for LE - CREOLE, un insieme di wrappers per realizzare l'interoperabilità fra sorgenti eterogenee, locali e/o remote (accessibili via HTTP);

Secondo GATE, le sorgenti del modulo CREOLE sono classificabili in tre categorie principali:

Language Resources: entità quali lexicons, corpora od ontologie;

Processing Resources: entità prevalentemente di tipo algoritmico: parsers, taggers, etc. . . ;

[1] *http://www.gate.ac.uk/index.html*

Visual Resources: entità per la rappresentazione e l'editing in un ambiente GUI.

Tutte le comunicazioni fra le risorse avvengono tramite il GDM, il quale fornisce un insieme di API per l'accesso e la manipolazione dei dati. Un importante obiettivo raggiunto da GATE è la separazione fra gli algoritmi dai dati che questi ultimi necessitano, proprio in considerazione del fatto che lo sviluppo di un'applicazione complessa come quella di un sistema di LE coinvolge due diverse figure professionali: lo sviluppatore di software ed il linguista.

Lo sviluppo di una nuova applicazione basata sull'architettura di GATE prevede una fase preliminare in cui l'utente sceglie quante e quali *processing resources* utilizzare, definendo per ognuna di esse le *language resources* coinvolte [2].

Il formato per la rappresentazione dei dati in GATE è un formato interno detto *annotation*, ma il sistema fornisce il supporto per l'elaborazione di testi in formato plain text, XML, HTML, RTF e SGML. Inoltre, GATE permette di realizzare un *Multilingual Text Processing*, favorendo la creazione di *Virtual Keyboards* quando la piattaforma non predispone il supporto all'internazionalizzazione.

Un' interessante applicazione basata su GATE è ANNIE, **A** **N**early-**N**ew **I**nformation **E**xtraction system[3]. ANNIE utilizza una serie di *language resouces* e di *processing resources* per riconoscere istanze di entità (Persone, Luoghi, Date, Indirizzi, ...)all'interno di un testo. In Figura A.1 è rappresentato il risultato della ricerca di istanze di Persona all'interno della pagina *http://www.dbgroup.unimo.it/Momis/partecipants.html*.

A.3 KeyConcept

KeyConcept [GAU 02] è un motore di ricerca in corso di sviluppo presso la Kansas University (USA). Il suo obiettivo è quello di fornire un servizio

[2]Le *language resources* possono essere memorizzate in database relazionali o in file (XML-based file format e Java serialisation-base file format).

[3]ANNIE Web Interface: *http://gate.ac.uk/annie/index.jsp*.

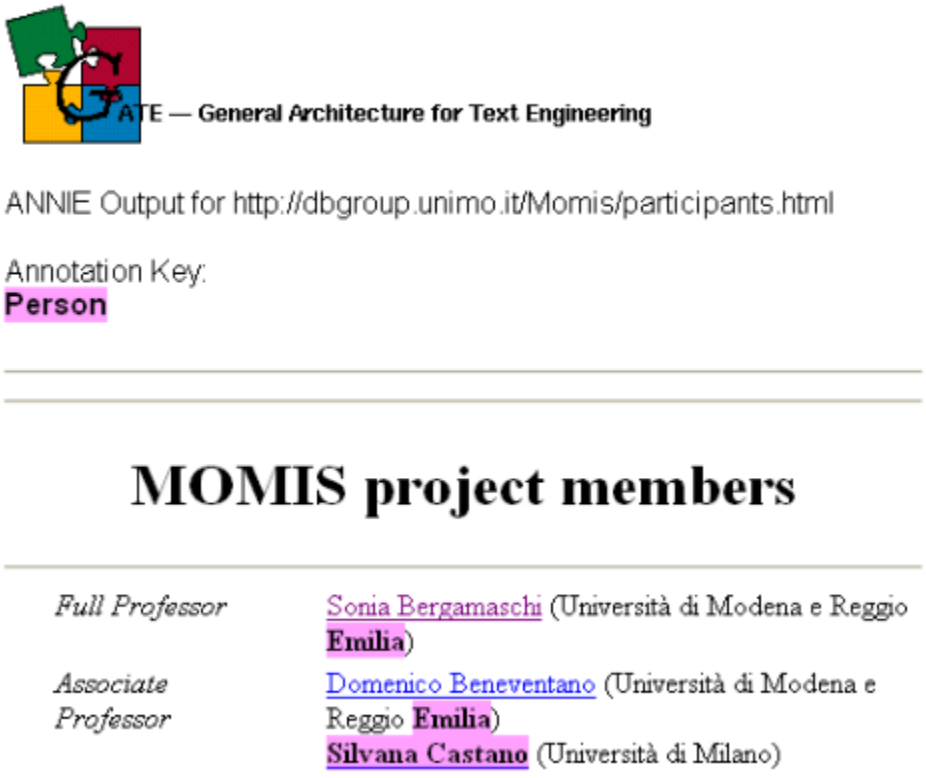

Figura A.1: Ricerca di Persone all'interno di *www.dbgroup.unimo.it / Momis / partecipants.html*

avanzato di retrieval di documenti, in cui la tradizionale ricerca per keywords viene affiancata da una innovativa ricerca per concetti. Ciò significa che l'architettura di KeyConcept è significativamente diversa rispetto a quella degli altri motori di ricerca (pag. 51) e, come si evince dalla Figura A.3, il tradizionale inverted index viene esteso per consentire il mapping fra i concetti ed i documenti.

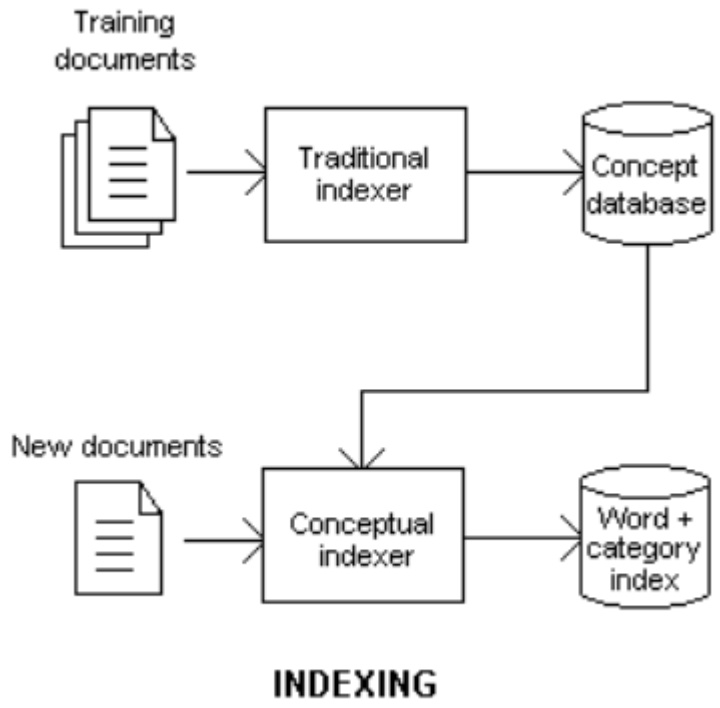

Figura A.2: Processo di indicizzazione di KeyConcept

La fase di *start-up* di KeyConcept è un processo estremamente critico, in cui vengono selezionati i concetti da rappresentare nell'indice. Attualmente KeyConcept indicizza circa 3000 concetti derivanti dai primi 3 livelli della tassomia proposta da ODP. Per ogni concetto, KeyConcept ottiene da ODP un numero fissato di documenti, ne realizza il merging e il *super-document* risultante viene analizzato ed indicizzato tramite la tradizionale tecnica TFI-DF. Un super-document è l'elemento rappresentativo di un concetto, ossia il centroide del relativo training set. L'indicizzazione realizzata da KeyConcept

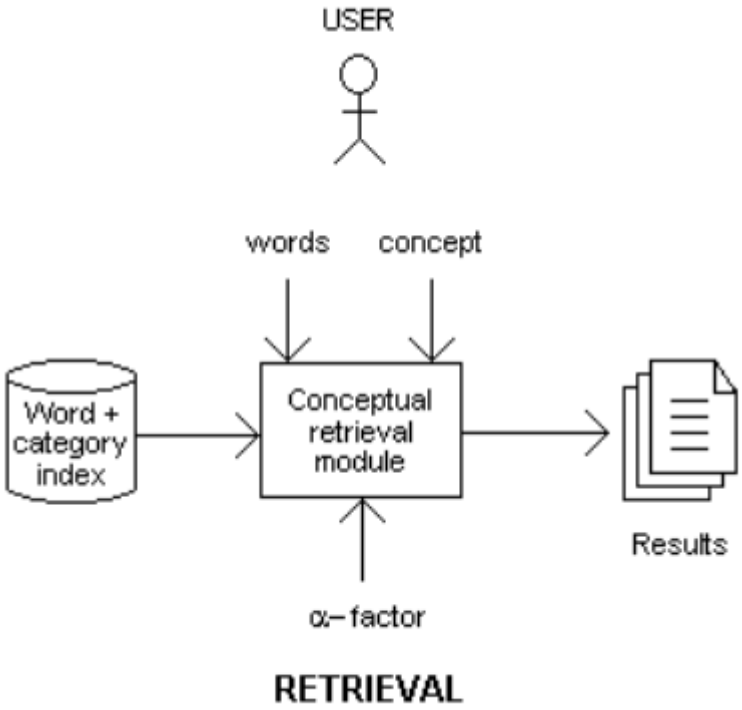

Figura A.3: Information Retrieval in KeyConcept

è in sintesi il calcolo della similarità fra i vari documenti nella collezione e i centroidi dei concetti.

La fase di retrieval vede l'utente interagire con il sistema fornendo ad esso queries contenenti parole, identificatori dei concetti e un valore compreso fra 0 e 1 detto $\alpha - factor$. Il fattore α indica l'importanza relativa del *concept match* rispetto al *keyword match*: se α vale 1 significa che la ricerca avviene solo tramite concetti, se vale 0 la ricerca avviene solo per keywords. Il fattore $\alpha - factor$ influisce sul punteggio dei documenti restituiti perchè interviene nella funzione di scoring nel seguente modo:

$$DocumentScore = (\alpha * ConceptScore) + ((1 - \alpha) * WordScore) \quad \text{(A.1)}$$

Bibliografia

[AGG 2001] C. C. Aggarwal and P. S. Yu: *On Effective Conceptual Indexing and Similarity Search in Text Data*, IBM Watson Research Center Report, 2001.

[ALB 99] R. Albert, H. Jeong and A. L. Barabasi: *Diameter of the World Wide Web*, Nature, vol. 401, pp 130-131, 1999.

[ALK 97] R. Al-halimi, R. Kazman: *Temporal Indexing Through Lexical Chaining*, WordNet: An electronic Lexical Database and some of its Applications, C. Fellbaum(ed.), MIT Press, 1997.

[AME 00] B. Amento, L. Terveen and W. Hill: *Does "Authority" mean quality? Predicting Expert Quality Ratings of Web Documents,* in Proceedings of the 23rd ACM SIGIR on Research and Development in Information Retrieval, pp. 296-303, 2000.

[ARA 01] A. Arasu, J. Cho, H. Garcia-Molina: A. Paepcke, S. Raghavan: *Searching the World Wide Web*, ACM Transactions on Internet Technology, 1(1), 2001, pp. 2-43.

[ARM 95] R. Armstrong, T. Joachims, D. Freitag and T. Mitchell: *WebWatcher: a Learning Apprentice for the World Wide Web,* in AAAI Spring Symposium on Information Gathering, Stanford, March 1995.

[ASH 97] N. Ashish, C. Knolock: *Wrapper generation for semi-structured internet sources*, In ACM SigMoD workshop on Management of Semistructured Data, 1997.

[ATT 98] G. Attardi, S. Di Marco, D. Salvi, F. Sebastiani: *Categorisation by Context*, in Workshop on Innovative Internet Informations Systems, 1998.

[BAR 97a] R. Barzilay: *Lexical Chain for Summarization* M.Sc degree of Ben-Gurion University of the Negev, 1997.

[BAR 97b] R. Barzilay, M. Elhadad *Using Lexical Chains for Text Summarization* , in ACL/EACL-97 summarization workshop, pages 10-18, Madrid, 1997.

[BAR 00] Z. Bar-Yossef, A.Berg, S. Chien and J.F.D. Weitz: *Approximating aggregate queries about web pages via random walks*, in Proceedings of the 26th International Conference on Very Large Database, 2000.

[BEL 02a] F. Bellifemine, G. Caire, T. Trucco e G. Rimassa: *Jade Programmer's Guide*, 15 July 2002.

[BEL 02b] F. Bellifemine, G. Caire, T. Trucco e G. Rimassa: *Jade Administrator's Guide*, 15 July 2002.

[BEN 97a] Domenico Beneventano, Sonia Bergamaschi, Claudio Sartori, and Maurizio Vincini: *ODB-QOPTIMIZER: A tool for semantic query optimization in oodb*, In Int. Conference on Data Engineering - ICDE97, 1997.

[BEN 97b] D. Beneventano, S. Bergamaschi, C. Sartori, and M. Vincini. *Odbtools: a description logics based tool for schema validation and semantic query optimization in object oriented databases*, In Sesto Convegno AIIA - Roma, 1997.

[BER 94] T. Berners-Lee, R. Cailliau, A. Luotonen, H.F. Nielsen and A.Secret: *The World-Wide Web*, Communications of the ACM. 37(8): 76-82, August 1994.

[BER 98a] S. Bergamaschi, S. Castano, S. Capitani di Vimercati, S. Montanari, and M. Vincini: *An intelligent approach to information integration*, in Proceedings of the International Conference in Formal Ontology in Information Systems (FOIS'98), Trento, Italy, june 1998.

[BER 98b] S. Bergamaschi, S. Castano, S. De Capitani di Vimercati, S. Montanari, and M. Vincini: *Exploiting schema knowledge for the integration of heterogeneous sources*, in Sesto Convegno Nazionale su Sistemi Evoluti per Basi di Dati - SEBDB98, Ancona, pages 103-122, 1998.

[BER 99] S. Bergamaschi, S. Castano, D. Beneventano, and M. Vincini: *Semantic integration and query of heterogeneous information sources*, in Journal of Data and Knowledge Engineering 1, 1999.

[BHA 99] K. Bharat and A. Broder: *Mirror, mirror on the web: A study of host pairs with replicated content*, in Proceedings of the 8th International World Wide Conference, 1999.

[BHA 98] K. Bharat, M. R. Henzinger: *Improved Algorithms for Topic Distillation in a Hyperlinked Environment*, 21st ACM SIGIR Conference on Research and Development in Information Retrieval, 1998.

[BOL 98] K. D. Bollacker, S. Lawrence, C. Lee Giles: *CiteSeer: An Autonomous Web Agent for Automatic Retrival and Identificaion of Interesting Publications*, published in 2nd International ACM Conference on Autonomous Agents, pp. 116-123, ACM Press, May, 1998.

[BRI 92] E. Brill: *A Simple Rule-Based Part of Speech Tagger*, in Proceedings of the Third Conference on Applied Natural Language Processing, ACL, 1992.

[BRI 94] E. Brill: Some Advances in Transformation-Based Part of Speech Tagging, in Proceedings of the 12th National Conference on Artificial Intelligence, AAAI-94, 1994.

[BRI 98] S. Brin, L. Page: *The Anatomy of a Large-Scale Hypertextual Web Search Engine*, in Computer Networks and ISDN Systems, vol. 30, pp. 107,117, 1998.

[BRO 98] A. Z. Broder, K. Bharat: *A technique for measuring the relative size and overlap of public Web Search Engines,* in Proceedings of the WWW7 Conference, 1998.

[BRO 00] A. Z. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. S. Tomkins and J. Wiener: *Graph structure in the web: experiments and models*, in Proceedings of the 9th WWW Conference, 2000.

[CAS 97] S. Castano and V. De Antonellis: *Deriving global conceptual views from multiple information sources*, In preProc. of ER'97 Preconference Symposium on Conceptual Modeling, Historical Perspectives and Future Directions, 1997.

[CHA 94] S. Chawathe, Garcia Molina, J. Hammer, K. Ireland, Y. Papakostantinou, J. Ullman, and J. Widom: *The TSIMMIS project: Integration of heterogeneou information sources*, in IPSJ Conference, Tokio, Japan, 1994.

[CHA 98] S. Chakrabarti, B. Dom, P. Raghavan, S. Rajagopalan, D. Gibson and J. Kleinberg: *Automatic Resource Compilation by Analyzing Hyperlink Structure and Associated Text,* in Proocedings of the 7th World Wide Web Conference, 1998.

[CHA 99] S. Chakrabarti, M. van der Berg, and B. Dom: *Focused crawling: a new approach to topic-specific web resource discovery*, in Proc. of the 8th International World-Wide Web Conference (WWW8), 1999.

[CHA 01] M. Chau and H. Chen: *Personalized and Focused Web Spiders*, to appear in Zhong, Liu and Yao editors, Web Intelligence, Springer-Verlang, 2001.

[CHA 02] M. Chau, D. Zeng, H. Chen, M. Huang, D. Hendriawan: *Design and evalutation of a multi-agent collaborative Web mining system*, Decision Support System, Elsevier Science, 2002.

[CHE 96] F. C. Cheong: *Internet Agents: Spiders, Wanderers, Brokers and Bots,* New Riders Publishing, Indianapolis, Indiana, USA, 1996.

[CHE 98] H.Chen, Y.M. Chung, M. Ramsey, C.C. Yang: *An Intelligent Personal Spider (Agent) for Dynamic Internet/Intranet Searching*, J. Am. Soc. Inf. Sci., pp. 604-618, 49(7), 1998.

[CHO 98] J. Cho, H. Garcia-Molina, L.Page: *Efficient crawling through URL ordering*, in Proc. of the Seventh World-Wide Web Conference, 1998.

[CHO 00a] J. Cho and H. Garcia-Molina: *The Evolution of the Web and Implications for an Incremental Crawler*, in Proceedings of the 26th VLDB Conference, Cairo, Egypt, 2000.

[CHO 00b] J. Cho, H. Garcia-Molina: *Synchronizing a database to improve freshness*, in Prooceedings of the 2000 ACM SIGMOD, 2000.

[CUN 96] H. Cunningham, Y. Wilks, R. J. Gaizauskas: *GATE - a General Architecture for Text Engineering*, in Proceedings of COLING-96, Copenhagen, 1996.

[DEB 94] P. M. E. De Bra, R. D. J. Post: *Information Retrieval in the World-Wide Web: making client-based searching feasible*, Computer Networks and ISDN Systems, vol 27, pp. 183-192, Nov. 1994.

[DIL 00] M. Diligenti, F. M. Coetzee, S. Lawrence, C. L. Giles and M. Gori: *Focused Crawling Using Context Graphs*, International Conference on Very Large Databases, VLDB 2000, Cairo, Egypt, pp. 527-534, 2000.

[DIL 02] M. Diligenti, M. Gori, M. Maggini: *Web Page Scoring Systems for Horizontal and Vertical Search*, in Proceedings of the WWW2002, Honolulu, USA, 2002.

[DRE 97] A. E. Howe and D. Dreilinger: *Experiences with Selecting Search Engines using MetaSearch,* ACM Transactions on Information Systems, vol 15, no. 3, July 1997.

[EDM 69] H. Edmunson: *New methods in automatic extracting*, Journal of the ACM 16(2), pp. 264-285, 1969.

[EIC 95] D. Eichmann: *Ethical Web Agents*, in Computer Networks and ISDN Systems, vol. 28(1,2), pp.127,136, 1995.

[EST 01] M. Ester, M. GroSS, H.P. Krieger: *Focused Web Crawling: A Generic Framework for Specifying the User Interest and for Adaptive Crawling Strategies* , University of Munich, Technical Report, 2001.

[FIE 98] J. Fiedler and J. Hammer: *Using the Web Efficiently*, UF Technical Report TR98-007, 1998.

[FRE 98] D. Freitag: *Information Extraction from HTML: Application of a General Machine Learning Approach*, In Prooceedings of the Fifteenth national Conference on Artificial Intelligence, Madison WI, 1998, AAAI Press.

[FUE 02] M. Fuentes, H. Rodríguez: *Using cohesive properties of text for Automatic Summarization*, in Jotri'02, 2002.

[GAU 96] S. Gauch, G. Wang and M. Gomez: *ProFusion: intelligent fusion of multiple, distributed search engines,* Journal of Universal Computer Science, 2(9), 1996.

[GAU 99] Y. Fan and S. Gauch: *Adaptive Agents for Information Gathering from Multiple, Distributed Information Sources,* in Proceedings of the AAAI Symposium on Intelligent Agent in CyperSpace, Stanford University, March 1999.

[GAU 02] S. Gauch, J. M. Madrid: *Incorporating Conceptual Matching in Search*, Technical Report, Kansas University, 2002.

[GIL 96] J.Gilarranz, J. Gonzalo, and F. Verdejo: *Using the eurowordnet multilingual semantic database*, In Proc. of AAAI-96 Spring Symposium Cross-Language Text and Speech Retrieval, 1996.

[GRE 99] S.J. Green: *Building Hypertext Links By Computing Semantic Similarity*, IEEE Transactions on Knowledge and Data Engineerign, Vol. 11, No. 5, September/October 1999.

[GUA 97] N. Guarino: *Semantic matching: Formal ontological distinctions for information organization, extraction and integration*, Technical Report, Summer School on Information Extraction, Frascati, Italy, July 1997.

[GUI 02] V. Guidetti: *Intelligent Information Integration systems: extending lexicon ontology*,Università degli Studi di Modena e Reggio Emilia, 2002.

[HAC 84] R. Hasan: *Reading Comprehension*, 1984.

[HAS 84] R. Hasan: *Coherence and Cohesive harmony*, in J. Flood ed, Understanding Reading Comprehension, pp. 181-219, IRA: Newark, Delaware, 1984.

[HEN 01] J. Hendler, O. Lassila, and T. Berners-Lee: *The semantic web, a new form of web content that is meaningful to computers will unleash a revolution of new possibilities* , Scientific American, May 2001.

[HER 98] M. Hersovici, M. Jacovi, Y. S. Maarek, D. Pelleg, M. Shtalhaim, S. Ur: *The shark-search algorithm - An application: tailored Web site mapping*, in Proceedings of the 7th World Wide Web Conference (WWW7), Brisbane (Australia), 1998.

[HEY 99] A. Heydon and M. Najork: *Mercator: A Scalable, Extensible Web Crawler*, WWW 2(4), 1999.

[HHC 76] M. A. K. Halliday and R. Hasan: *Cohesion in English*, English Language Series, Longman, 1976.

[HHG 85] M. A. K. Halliday and R. Hasan: *An Introduction to Functional Grammar*, Edward Arnold, London, 1985.

[HIR 98] G. Hirst, D. St-Onge: *Lexical chains as representations of context for the detection and correction of malapropisms* WordNet: An electronic lexical database, Christiane Fellbaum(editor), Cambridge, MA: The MIT Press, 1998.

[HOE 91] M. Hoey: *Patterns of Lexis in Text.* Oxford University Press, Oxford, 1991.

[HOV 97] E. Hovy and C. Lin: *Automated text summarization in summarist*, in ACL/EACL-97 Workshop on Intelligent Scalable Text Summarization, pp. 18-24. Association for Computational Linguistics and the European Chapter of the Association for Computational Linguistics, 1997.

[HOW 97] A. E. Howe and D. Dreilinger: *SavvySearch: A MetaSearch Engine that Learns which Search Engines to Query,* AI Magazine, 18(2), 1997.

[HTML 99] D. Raggett, A. LeHors, I.Jacobs: *HTML 4.01 Specification*, http://www.w3.org/TR/1999/REC-html401-19991224.

[HUL 95] R. Hull, R. King et altri: *Arpa I^3 reference architecture*, 1995. Available at http://www.isse.gmu.edu/I3_Arch/ index.html.

[KAM 03] S. Kamvar, T. H. Haveliwala, C. Manning, G.H. Golub: *Extrapolation Methods for Accelerating PageRank Computation*, Proceedings of the WWW2003 Conference, Budapest, 2003.

[KLE 98] J. Kleinberg: *Authoritative sources in a hyperlinked environment,* in Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, 1998.

[KUM 00] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, E. Upfal and A. S. Tomkins: *The Web as a graph,* in Proceedings of the 19th ACM SIGACT-SIGMOD-AIGART Symposium on Principles of Database Systems, PODS, 2000.

[KUP 95] J. Kupiec, J. Pedersen and F. Chen: *A trainable document summarizer*, in Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 68-73, Seattle, Washington 1995.

[KUS 97] N. Kushmerick: *Wrapper induction for information extraction*, PhD Thesis, University of Washington, 1997, Tech. Report UW-CSE-97-11-04.

[LAB 99] Y. Labrou, T. Finin: *Yahoo! as an Ontology - Using Yahoo! Categories to Describe Documents*, in Proceedings of the Eight International Conference of Information Knowledge Management (CIKM '99), 1999.

[LAW 98a] S. Lawrence and C. Lee Giles: *Searching the World Wide Web*, Science 280, 5360, 1998.

[LAW 98b] S. Lawrence, C. Lee Giles: *Context and page analysis for improved web search*, in IEEE Internet Computing, July/August 1998, pp. 38-46.

[LAW 99a] S. Lawrence and C. Lee Giles: *Accessibility of information on the web*, Nature, vol. 400, pp. 107-109, 1999.

[LAW 99b] S. Lawrence, C. Lee Giles, K. Bollacker: *Digital Libraries and Autonomous Citation Indexing*, IEEE Computer, Volume 32. Number 6, pp. 67-71, 1999.

[LEE 98a] C. Lee Giles, K. D. Bollacker, S. Lawrence: *CiteSeer: An Automatic Citation Indexing System*, In Digital Libraries 98 - Third ACM Conference on Digital Libraries, I. Witten, R. Ahscyn, F. Shipman III (eds.), ACM Press, New York, pp 89-98, 1998.

[LEV 66] I. V. Levenshtein: *Binary Codes capable of correcting deletions, insertions, and reversals*, in Cybernetics and Control Theory, 10(8):707Ű710, 1966.

[LIE 95] H. Lieberman: *Letizia: an Agent that Assists web Browsing*, in Proceedings of the International Joint Conference on Artificial Intelligence, Montreal, Canada, 1995.

[LUG 03] L. Lugli: *Integrazione di Sorgenti HTML in MOMIS: Analisi Comparativa degli Strumenti Esistenti* , Tesi di Laurea, Università degli Studi di Modena e Reggio Emilia, 2003.

[LUH 58] H. Luhn: *The automatic creation of literature abstracts*, IBM Journal of Research and Development 2(2), 1958.

[MAE 01] A. Maedche and S. Staab: *Comparing Ontologies - Similarity Measures and a Comparison Study*, Internal Report of the Institute AIFB, University of Karlsruhe, 2001.

[MAE 03] A. Maedche and B. Motik and L. Stojanovic and R. Studer and R. Volz: *An Infrastructure for Searching, Reusing and Evolving Distributed Ontologies*, WWW 2003, Hungary, 2003.

[MAH 01] J. Mahadavan, P. A. Bernstein, E. Rahm: *Generic Schema Matching with Cupid*, in Proceedings of the 27th VLDB Conference, Roma, 2001.

[MAY 02] Maynard, K. Bontcheva, V. Tablan. *GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications*, in Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02). Philadelphia, July 2002.

[MEL 01] S. Melnik, H. Garcia-Molina, E. Rahm: *Similarity Flooding: A Versatile Graph Match Algorithm*, Technical Report, University of Stanford, 2001.

[MEN 02] W. Meng, C. Yu, K. Liu: *Building Efficient and Effective MetaSearch Engines,*ACM Computing Surveys (CSUR), v.34 n.1, p.48-89, March 2002.

[MIL 90a] G. Miller, R. Beckwith, C. Fellbaum, D. Gross, K.Miller: *Introduction to Wordnet: An on-line lexical database*, International Journal of Lexicography(special issue), 3(4): 235-312,1990.

[MIL 90b] G. Miller et. altri: *Five papers on WordNet*, in Cognitive Science Laboratory Technical Report, Princeton University, 1990.

[MIT 98] M. Craven, D. DiPasquo, D.Freytag, A. McCallum, T. Michell, K. Nigam, S. Slatery: *Learning to Extract Symbolic Knowledge from the World Wide Web*, in Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98), 1998.

[MOR 91] J. Morris and G. Hirst: *Lexical cohesion computed by thesaural relations as an indicator of the structure of text*, Computational Linguistics, vol. 17, n. 1, pp. 21-48, 1991.

[NAJ 01] M. Najork, J.L. Wiener: *Breadth-First Search Crawling Yields High-Quality Pages*, in WWW10, May 2-5, Hong Kong, 2001.

[NAT 02] E. Natalini: *Progetto e sviluppo di un agente hunter per la ricerca e l'archiviazione di nuove sorgenti informative*,Università degli Studi di Modena e Reggio Emilia, 2002.

[OKA 96] M. Okamura: *A Language Analysis Method for Spoken Dialogue Understanding*, Research on Understanding and Generating Dialogue by Integrated Processing of Speech, Language and Concept, 1996.

[OKA 94] M. Okamura, T. Honda *Word sense disambiguation and text segmentation based on lexical cohesion*,Proceedings of the Fifteenth International Conference on Computational Linguistics (COLING-94), volume 2, 755-761, 1994.

[ONO 94] K. Ono, K. Sumita, S. Miike: *Abstract generation based on rhetorical structure extraction*, in Proceedings of the 15th International Conference on Computational Linguistics (COLING-94), vol 1, pp. 344-348, Association for Computational Linguistics, Kyoto, Japan, 1994.

[PAG 98] L. Page, S. Brin, R. Motwani and T. Winograd: *The PageRank Citation Ranking, Bringing Order to the Web*, Stanford Digital Libraries Working Pager, 1998.

[PAY 90] C. Payce: *Constructing literature abstracts by computer: Techniques and prospects*, Information Processing and Management 26(1), pp. 171-186, 1990.

[POR 80] M. F. Porter: *An algorithm for suśx stripping*, Program 14(3), 130-137, 1980.

[RAG 01] S. Raghavan and H. Garcia-Molina: *Crawling the Hidden Web*, in Proceedings of the 27th VLDB Conference, Roma, 2001.

[RAT 96] A. Ratnaparkhi: *A Maximun Entropy Part-Of-Speech Tagger*, in Proceedings of the Empirical Methods in Natural Language Processing Conference, May 17-18, 1996, University of Pennsylvania.

[RIB 98] B. Ribeiro-Neto and A. Barbosa: *Query performance for tightly coupled distributed digital libraries,* In Proceedings of the Third ACM International Conference on Digital Libraries, pp. 182-190, 1998.

[ROG 77] P. M. Roget: *Roget's International Thesaurus, Fourth Edition*, Harper and Row Publishers Inc., 1977.

[SAL 89] G. Salton: *Automatic Text Processing*, Addison Wesley, Massachusetts, 1989.

[SAN 90] B. Santorini: *Part-of-Speech Tagging Guidelines for the Penn Treebank Project*, The Penn Treebank Project, 1990.

[SCH 95] H. Schutze, D. A. Hull and J. O. Pedersen: *A comparison of classifiers and document representation for the routing problem,* in Proceedings of the SIGIR-95, 18th ACM International Conference on Research and Development in Information Retrieval, Seattle, pp. 229-237, 1995.

[SEL 07] E. Selberg and O. Etzioni: *The MetaCrawler Architecture for Resource Aggregation on the Web,* IEEE Expert, January/February, 11-14, 1997.

[SHE 00] C. Sherman: *Humans do it better inside the open directory project*, Online, Luglio/Agosto 2000, pp. 43-50.

[SIL 02] H. G. Silber, K. F. McCoy: *Efficiently Computed Lexical Chains as an Intermediate Representation for Automatic Text Summarization*, Computational Linguistics, vol. 28, 2002.

[SOD 97] S. Soderland: *Learning to Extract Text-Based Information from the World Wide Web* , in Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining, 1997.

[SPA 93] K. Sparck Jones: *What might be in a summary?*, in G. Knorz, J. Krause and C. Womser-Hacker eds, Information Retrieval '93: von der modellierung zur anwendung. Konstanz: Universitatsverlag Konstanz, pp. 9-26, 1993.

[STA 97] M. A. Stairmand, W. J. Black: *Conceptual and Contextual Indexing using Word-Net derived Lexical Chain*, in Proceedings of BCS IRSG Colloquium, pp. 47-65, 1997.

[WIE 92] G. Wiederhold. Mediators in the architecture of future information systems. IEEE Computer, 25:38-49, 1992.

[WIE 96] G. Wiederhold et altri: *Integrating Artificial Intelligence and DataBase Technology*, Journal of Intelligent Information Systems, volume 2/3, June 1996.

[WOO 95] M. Wooldridge and N.R. Jennings: *Intelligent agents: Theories and practice* in The Knowledge Engineering Review, 10(2):115(152), 1995.

[YU 01] C. C. Aggarwal, F. Al-Garawi, P. S. Yu: *Intelligent Crawling on the World Wide Web with Arbitrary Predicates*, in Proceedings of the WWW10 Conference, Hong Kong, 2001.

[YUW 95] B. Yuwono, S.L. Lam, J. H. Ying, D. L. Lee: *A World Wide Web Resource Discovery System*, the 4th International WWW Conference, Boston, 1995.

[1] *The Deep Web: Surfacing Hidden Value*, available on-line, http://www.completeplanet.com/Tutorial/DeepWeb.

[2] *Web surpasses one billion documents: Inktomi/NEC press release*, http://www.inktomi.com.

[3] *http://www.searchenginewatch.com.*

[4] *http://www.profusion.com.*

[5] *http://www.robotstxt.com.*

[6] *http://www.allsearchengines.com.*

[7] *http://www.yahoo.com.*

[8] *http://www.altavista.com.*

[9] *http://www.WebCrawler.com.*

[10] *http://www.MetaCrawler.com.*

[11] *http://www.google.com.*

[12] *http://dmoz.org.*

[13] *http://www.directHit.com.*

[14] *http://www.arianna.it.*

[15] *htttp://citeseer.nj.nec.com/cs*

[16] *http://www.cs.washington.edu/research/ahoy*

[17] The GATE Web Demo: *http://www.gate.ac.uk/demos/index.html*

[18] ANNIE Web Interface: *http://gate.ac.uk/annie/index.jsp*

[19] KAON: *http://kaon.semanticweb.org*