



UNIVERSITÀ DEGLI STUDI ROMA TRE

Dipartimento di Ingegneria
Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea

Smart crawling di siti web strutturati

Laureando

Fabio Cibecchini

Matricola 462552

Relatore

Prof. Valter Crescenzi

Anno Accademico 2016/2017

Ai miei genitori

Introduzione

Il Web è oggi il principale mezzo di comunicazione di massa e rappresenta una fonte vasta e preziosa di informazioni: esso può essere esplorato e archiviato per diversi scopi mediante l'utilizzo dei crawler, software che analizzano ed estraggono il contenuto dei siti web in modo metodico e automatizzato. I motori di ricerca sfruttano i crawler per indicizzare e categorizzare milioni di siti al fine di restituire su richiesta un insieme di contenuti rilevanti per la chiave di ricerca. Applicazioni software confrontano i prezzi di milioni di prodotti per restituire agli utenti le offerte più convenienti. Software di business intelligence analizzano informazioni provenienti dai social network e da notizie on-line per supportare le decisioni strategiche delle aziende. Altre organizzazioni archiviano il contenuto dei siti per mantenere una biblioteca virtuale della storia del Web. Tuttavia, l'esplorazione efficiente di una fonte di informazioni così grande porta con sé diverse problematiche: i crawler devono scaricare pagine web ad un ritmo molto alto ma senza sovraccaricare i server web. Devono disporre di uno spazio di archiviazione adeguato a raccogliere tutte le pagine. Devono essere in grado, a seconda del caso d'uso, di riconoscere le zone di un sito di maggior interesse per il dominio applicativo e tenere aggiornate le informazioni estratte da esse. Devono inoltre catalogare i siti e le pagine secondo criteri che permettano un'estrazione efficace delle informazioni contenute nelle pagine HTML.

Questa tesi descrive una soluzione che affronta le sfide elencate: viene definito un algoritmo che inferisce il modello di navigazione di un sito web in base all'analisi strutturale di un numero limitato ma rappresentativo di pagine HTML. Il modello generato partitiona il sito in classi di pagine strutturalmente simili, descrivendo inoltre le proprietà dei collegamenti di navigazione (link) tra le varie classi. In base a questo algoritmo

viene definito un crawler che, dato in input un insieme di URL di siti web, in una prima fase genera in maniera scalabile il modello di ciascun sito. I modelli così inferiti vengono utilizzati, in una seconda fase, per eseguire un crawling intensivo e distribuito dei siti guidato dalla loro struttura.

I risultati sperimentali ottenuti mostrano che i modelli generati, uniti alla fase di crawling intensivo, sono adatti ad esplorare in maniera scalabile diversi siti web, con lo scopo di ottenere una suddivisione delle pagine archiviate in base alla loro struttura. Le pagine di una stessa classe condividono lo stesso template HTML e possono quindi essere usate per estrarre facilmente dei dati di interesse da esse. I gruppi di pagine scaricate possono inoltre essere compressi efficacemente per risparmiare lo spazio di archiviazione occupato. Inoltre i modelli inferiti, descrivendo i percorsi di navigazione dei siti, possono essere usati per studiare in maniera incrementale il cambiamento di un sito nel tempo, al fine di individuare le classi che hanno maggiori probabilità di essere state aggiornate tra un crawling e il successivo.

La tesi è organizzata in 5 capitoli e un'appendice, secondo la seguente struttura.

Il primo capitolo introduce il problema dell'estrazione di informazione dal Web: viene data una panoramica delle motivazioni alla base di questa disciplina e vengono illustrate le principali fasi e tecniche di recupero delle informazioni dal Web. Vengono poi descritti i principali problemi e le sfide nel campo dell'archiviazione e del recupero delle informazioni.

Il secondo capitolo presenta una panoramica dei lavori in letteratura correlati al tema del crawling efficiente di siti web strutturati e della estrazione scalabile di informazioni. Il terzo capitolo descrive la metodologia usata per generare un modello di navigazione di sito web: in particolare vengono descritte le regolarità delle pagine web sfruttabili per la classificazione, la struttura del modello generato e l'algoritmo di generazione in dettaglio.

Il quarto capitolo descrive nello specifico l'architettura del crawler realizzato per supportare la generazione dei modelli e per la successiva fase di crawling intensivo di siti web guidato dalla struttura.

Il quinto capitolo descrive gli esperimenti compiuti e i risultati ottenuti riguardanti la generazione scalabile di modelli di siti web. Vengono descritti il dataset utilizzato, le

metriche di valutazione e l'analisi dei risultati riportati.

Chiude la tesi un'appendice che descrive la sintassi della specifica di crawling (modello di navigazione) in formato CSV da utilizzare nella fase di crawling intensivo.

Indice

Introduzione	iii
Indice	vi
Elenco delle figure	x
Elenco delle tabelle	xii
1 Archiviazione del Web	1
1.1 Introduzione	1
1.2 Web scraping	2
1.2.1 Crawling	2
1.2.1.1 Qualità di un crawler	3
1.2.1.2 Casi d'uso	5
1.2.2 Estrazione dei dati	6
1.2.2.1 Generazione automatica di wrapper	6
1.3 Problemi	8
1.3.1 Sampling di pagine per la generazione di wrapper	8
1.3.2 Precisione del crawling	8
1.3.3 Crawling incrementale	9
1.3.4 Compressione	10
2 Lavori correlati	12
2.1 Clustering di pagine web	12
2.1.1 Crescenzi et al 2005 [5]	12

2.1.2	Blanco et al 2011 [2]	13
2.2	Generazione scalabile di wrapper	14
2.2.1	AAH [6]	14
2.2.2	DIADEM [9]	15
2.3	Crawling efficiente di siti web	16
2.3.1	IRobot [4]	16
2.3.2	ACEBot [7]	16
3	Modello di navigazione di un sito	18
3.1	Struttura delle pagine web	19
3.1.1	Struttura dei link	19
3.1.2	Struttura dei testi	22
3.1.3	Classificazione	25
3.2	Struttura del modello	25
3.2.1	Pagina e schema	25
3.2.1.1	Link Collection	25
3.2.1.2	Text Collection	28
3.2.2	Page Class	29
3.2.2.1	Definizione	29
3.2.2.2	Class Link	30
3.2.3	Grafo del sito web	31
3.3	Generazione del modello	32
3.3.1	Algoritmo	32
3.3.2	Selezione dei candidati	35
3.3.3	Rifinitura dei cammini	35
3.3.3.1	Ispezione	36
3.3.3.2	Reticolo dei cammini	36
3.3.4	Aggiornamento del modello	39
3.3.4.1	Peso dei cammini	40
3.3.4.2	Codifica del modello	45
3.3.5	Euristiche di navigazione	47

4 Architettura del crawler	48
4.1 Crawler distribuito	48
4.1.1 Requisiti	48
4.1.2 Diagramma delle classi di dominio	51
4.1.2.1 Visione	51
4.1.2.2 Classi	52
4.1.3 Modello ad attori	55
4.1.3.1 Visione	55
4.1.3.2 Attori	60
4.2 Generazione scalabile dei modelli	64
4.2.1 Requisiti	64
4.2.2 Diagramma delle classi di dominio	64
4.2.2.1 Visione	64
4.2.2.2 Classi	65
4.2.3 Modello ad attori	67
4.2.3.1 Visione	68
4.2.3.2 Attori	69
4.2.4 Persistenza dei modelli	71
5 Esperimenti e risultati	74
5.1 Dataset e descrizione	74
5.2 Metriche di valutazione	75
5.2.1 Qualità delle Page Class	75
5.2.2 Qualità dei Class Link	77
5.3 Risultati	78
Conclusioni e Sviluppi Futuri	81

A Specifica di crawling	84
A.1 Sintassi	84
A.1.1 Form-XPath	85
A.1.2 Class Link	85
A.1.3 Data Link	86
A.2 Esempio d'uso	87
Bibliografia	93

Elenco delle figure

1.1	Esempio di grafo diretto tra pagine web	3
3.1	Esempio di pagina HTML: lista delle proprietà in vendita	20
3.2	Esempio di pagina HTML: presentazione di tipologie di proprietà	21
3.3	Etichette nella pagina di una proprietà	23
3.4	Etichette nella pagina di una lista di proprietà	24
3.5	DOM della pagina p_1	26
3.6	DOM della pagina p_2	27
3.7	DOM con menu ad ancora variabile	28
3.8	DOM della pagina p_3	29
3.9	DOM della pagina p_4	29
3.10	Schemi delle pagine, delle Page Class e class link	31
3.11	DOM con elementi e attributi HTML	38
3.12	Reticolo dei cammini per l'ancora url_1	38
3.13	Reticolo L della link collection creata a partire da url_1	39
3.14	Cammini di sito e di Page Class nella pagina di un prodotto	41
3.15	Cammini di sito, di Page Class e singola pagina nella pagina di un prodotto	42
3.16	Cammini di sito e di Page Class nella pagina di una lista di prodotti . . .	43
4.1	Diagramma delle classi di dominio per la fase di crawling intensivo	51
4.2	Architettura ad attori del crawler	56
4.3	Gerarchia degli attori relativi a un singolo sito	57
4.4	Gerarchia degli attori distribuiti responsabili del fetching delle pagine . . .	59
4.5	Diagramma delle classi di dominio per la fase di generazione dei modelli . .	65

4.6	Gerarchia degli attori distribuiti responsabili della generazione dei modelli	68
4.7	Diagramma delle azioni asincrone compiute dall'algoritmo di modellazione	70
4.8	Particolare del modello inferito per http://www.wwagency.com	71
4.9	Menu di un sito web identificato dall'algoritmo di generazione del modello	72
4.10	Rappresentazione di un ClassLink di tipo Menu su Neo4J	73
A.1	Page Class home	89
A.2	Una pagina della Page Class perfumes	90
A.3	Una pagina della Page Class perfume, oscurata da una form	90
A.4	Una pagina della Page Class perfume, con i dati da estrarre	91
A.5	Una pagina della Page Class perfume: piramide olfattiva	91
A.6	Esempi di record estratti dal crawler	92

Elenco delle tavole

5.1	Risultati per i siti di e-commerce	79
5.2	Risultati per i siti di agenzie immobiliari	80
5.3	Risultati per i siti di noleggio di auto usate	80
5.4	Risultati per altri siti	80
A.1	Header della specifica dei Class Link	86
A.2	Esempio di specifica di Class Link	86
A.3	Header della specifica dei Data Link	86
A.4	Esempio di specifica di Data Link	87
A.5	Specifiche di crawling per https://olfattheque.com	88

Capitolo 1

Archiviazione del Web

Questo capitolo introduce il problema dell'estrazione di informazione dal Web. Nella prima sezione viene data una panoramica delle motivazioni alla base di questa disciplina. Nella seconda sezione vengono illustrate le principali fasi e tecniche di recupero delle informazioni dal Web. Nella terza sezione vengono descritti i principali problemi e le sfide nel campo dell'archiviazione e del recupero delle informazioni da siti web.

1.1 Introduzione

L'information retrieval (recupero delle informazioni) è un importante tema di ricerca in informatica. Esso consiste nella ricerca, memorizzazione e organizzazione di dati proveniente da enormi collezioni di documenti eterogenei presenti sul Web.

Nell'ultimo decennio il World Wide Web è diventato il principale mezzo di comunicazione di massa, grazie soprattutto all'avvento di piattaforme che permettono a tutti gli utenti della rete di pubblicare contenuti, come social network, blog, forum, encyclopedie on-line, ecc. In quest'ottica, a causa dell'esorbitante quantità, della mutevolezza e della dispersione dei dati a disposizione, sono nati negli anni molti sistemi informatici che sfruttano questi contenuti. Per esempio, i motori di ricerca catalogano e indicizzano documenti Web con lo scopo di preservarne i contenuti, permettendo agli utenti di effettuare ricerche su determinati argomenti navigando tra essi. Altri sistemi archiviano milioni di siti al fine di preservarne i contenuti per ragioni storiche, così che continuino ad essere disponibili per le generazioni future. Altri ancora fondono dati provenien-

ti da fonti eterogenee per estrapolare ed analizzare informazioni strategiche utili per incrementare il vantaggio competitivo delle aziende operanti in un certo settore.

1.2 Web scraping

Il processo di estrazione di informazione dal Web mediante l'utilizzo di software è detto Web scraping. Esso è composto di due fasi: (1) il *crawling*, che indica la tecnica con cui vengono raccolti, archiviati e indicizzati i documenti (tipicamente pagine HTML) contenenti informazioni di interesse, (2) l'*estrazione dei dati*, che indica il processo con cui i dati vengono estratti dalle pagine archiviate dal crawler.

1.2.1 Crawling

Lo scopo del crawling è solitamente quello di archiviare velocemente ed efficacemente il maggior numero di pagine web, insieme alla struttura dei link che le connette tra loro. Un *web crawler* (detto anche *robot* o *spider*) è un software che scarica pagine web. Nell'accezione più generica, il crawler prende in input una coda q di URL (detti *seed*), rimuove un URL dalla coda, scarica la pagina corrispondente, estraе tutti gli URL contenuti nella pagina, e inserisce in coda tutti gli URL le cui corrispondenti pagine non sono ancora state visitate; il processo descritto viene poi ripetuto. La coda, detta *frontiera degli URL*, rappresenta quindi, ad ogni passo i dell'algoritmo, lo stato corrente della visita e contiene tutti gli URL scoperti che devono ancora essere scaricati.

L'intero processo del crawler può essere visto come la visita del *grafo diretto del Web*, in cui ogni nodo rappresenta una pagina HTML e ogni arco diretto rappresenta un link da una pagina sorgente a una pagina destinazione, come descritto in figura 1.1

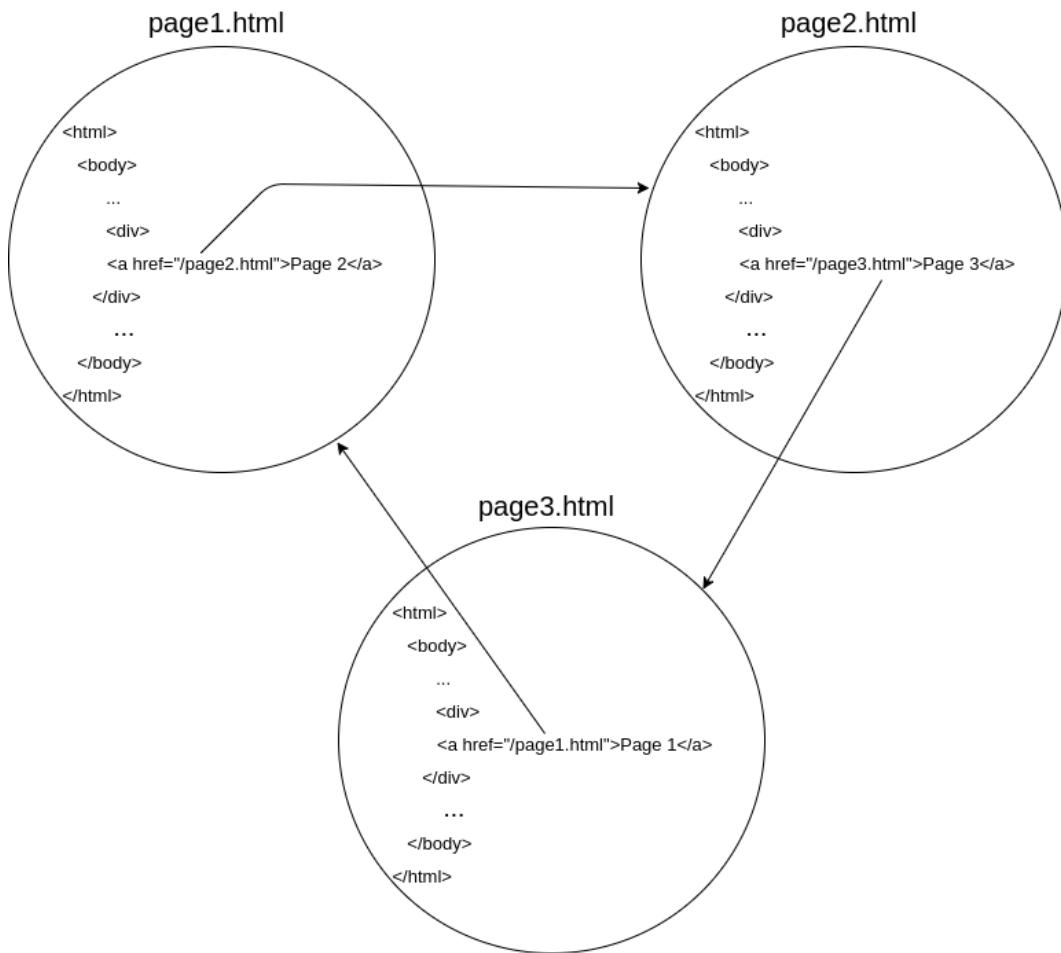


Figura 1.1: Esempio di grafo diretto tra pagine web

1.2.1.1 Qualità di un crawler

Sebbene l'algoritmo di crawling sia molto semplice, progettare un crawler professionale per uso commerciale richiede di tenere in considerazioni diversi attributi di qualità affinché esso sia performante ed efficace, con riferimento alla vasta dimensione e alla eterogenea natura del Web.

Politeness (Cortesia) Ogni web server utilizza politiche proprie per la gestione della banda utilizzabile dai suoi visitatori. Richieste consecutive da parte di uno stesso client potrebbero portare al sovraccarico del server, il che è considerato inaccettabile. I

web server dei siti web tendono quindi a bloccare i client che esibiscono questi comportamenti. Il crawler deve quindi essere progettato tenendo conto di questi limiti: da una parte è necessario garantire una visita efficiente del sito, nei termini degli obiettivi del crawler, dall'altra è necessario garantire il pieno rispetto degli interessi del web server a non essere sovraccaricato. Per queste ragioni, solitamente il crawler non dovrebbe mai scaricare simultaneamente pagine diverse di uno stesso sito.

Robustezza I web server utilizzano spesso delle trappole pensate appositamente per i crawler: alcune di queste generano dinamicamente pagine web all'interno del dominio, allo scopo di intrappolare il crawler nello scaricamento di queste pagine infinite. Altre utilizzano collegamenti ipertestuali, non visibili ad utenti umani, a pagine nascoste come esca per i crawler, al fine di identificare e bloccare le loro connessioni. Altre ancora nascondono il contenuto delle pagine costringendo il visitatore a compiere azioni (rispondere a una domanda, registrarsi al sito, ecc.) facile per un umano ma complicate per un programma automatico. Il crawler dovrebbe quindi essere il più possibile resiliente a questo tipo di resistenza da parte dei siti web.

Distribuzione Per migliorare le prestazioni il crawler dovrebbe essere distribuito, garantendo ad esempio che diversi host si occupino di scaricare siti diversi, ma anche pagine diverse dello stesso sito, per evitare di essere bloccati dai web server e ottimizzare l'uso della banda.

Efficienza Il crawler deve far uso di adeguati processori, memoria secondaria e banda per effettuare delle visite del Web efficienti.

Scalabilità Il crawler dovrebbe permettere la possibilità di aggiungere macchine per scalare sulla quantità di pagine di interesse, così come sfruttare appieno le architetture multi-core delle moderne CPU. Inoltre l'uso della memoria principale dovrebbe essere costante rispetto alla dimensione crescente della frontiera e dell'insieme di URL già scaricati (se non si intende rivisitarli).

Priorità dell'esplorazione Come specificato in 1.2.1, l'algoritmo di crawling è ri-conducibile alla visita di un grafo diretto. Il crawler dovrebbe quindi specificare e motivare la strategia con la quale intende effettuare l'esplorazione. Scelte classiche sono le visite in ampiezza e profondità, la prima delle quali può essere usata come rimedio alle trappole dei cicli di pagine infinite descritto nel paragrafo Robustezza. Altre scelte prevedono l'uso di code di priorità, in cui le pagine col contenuto informativo più alto, secondo metriche definite, vengono rimosse dalla frontiera e scaricate prima delle altre, indipendentemente dalla profondità delle pagine nel sito.

1.2.1.2 Casi d'uso

Oltre agli algoritmi e alle qualità elencate nei paragrafi precedenti, i crawler possono differire in molti altri aspetti anche a seconda del caso d'uso per cui li si intende utilizzare. Si possono identificare almeno i seguenti 3 casi d'uso:

Crawling ampio Il *crawling ampio* (*broad crawling*) è un tipo di crawling comune che copre la visita di un enorme (potenzialmente infinito) numero di domini. L'obiettivo è quindi l'archiviazione della maggiore porzione del Web possibile, dato il tempo, la banda e lo spazio a disposizione. I crawler che seguono questo caso d'uso esplorano il Web a partire da un numero limitato di siti in input, scoprendo nuovi domini esplorando i collegamenti esterni (interdominio) nelle pagine.

Crawling focalizzato Il *crawling focalizzato* (*focused crawling*) è un tipo di crawling di piccole o medie dimensioni che ha l'obiettivo di esplorare in maniera completa un numero limitato di siti web o argomenti. Possono far uso di tecniche di intelligenza artificiale, come i classificatori, per dare maggiore priorità nella frontiera agli URL che portano alle pagine più attinenti all'argomento di interesse, sfruttando come attributi gli elementi del *DOM* (*Document Object Model*) [21] e la struttura degli URL.

Crawling incrementale Il *crawling incrementale* (*incremental crawling*) è un tipo di crawling che ha l'obiettivo di mantenere aggiornata una collezione di pagine archiviate, riscaricando periodicamente le pagine per ottenerne i nuovi contenuti. Rispetto al crawling ampio e al crawling focalizzato, questo crawling non scarica istantaneamente

pagine web in un determinato periodo temporale, ma archivia diverse versioni di pagine in periodi differenti: deve perciò utilizzare una strategia per capire *quando* e *se* una pagina già archiviata ha cambiato il suo contenuto, associando una frequenza di visita, variabile nel tempo, ad ogni URL.

1.2.2 Estrazione dei dati

Il tema dell'estrazione si focalizza sulla trasformazione dei dati ottenuti dalle pagine di molti siti web, tipicamente contenuti in formato HTML, senza una struttura comune, in dati strutturati che possono essere analizzati e conservati in basi di dati locali.

L'estrazione dei dati viene tipicamente eseguita da programmi chiamati *wrapper*, che possono sfruttare diverse tecniche: espressioni regolari, espressioni XPath [20], micro-formati contenuti nell'HTML, programmi ad-hoc. In particolare, gli approcci basati su XPath ed espressioni regolari sfruttano le regolarità presenti nelle pagine web, che possono essere ridotte ad una rappresentazione ad albero del DOM.

1.2.2.1 Generazione automatica di wrapper

In questo contesto, la generazione automatica di wrapper è un problema di grande interesse: infatti la generazione manuale per ogni pagina, o gruppo di pagine, non è un approccio realizzabile su grande scala né scalabile, dato l'elevato numero di siti web che possono essere di interesse per un determinato dominio, e l'elevato numero di pagine che ogni sito può avere.

In generale, il problema può essere trattato da un punto di vista algoritmico grazie alla seguente assunzione: i siti web dinamici che collezionano ingenti quantità di informazioni vengono generati da basi di dati che incapsulano dati all'interno delle pagine HTML. Di conseguenza, diverse pagine condividono la stessa struttura (*template*) e possono essere trattate con lo stesso wrapper. Gli algoritmi di generazione automatica di wrapper prendono quindi in input insiemi di pagine conformi a un template comune, sfruttando regolarità e differenze nei DOM, per individuare autonomamente i campi di interesse e le regole per estrarli.

In letteratura esistono diversi approcci il cui obiettivo è scalare sul numero di wrap-

per generati per diversi siti web; essi si dividono principalmente in (1) *approcci non supervisionati* e (2) *approcci supervisionati*

Approcci non supervisionati Negli approcci non supervisionati i wrapper vengono generati a partire da pagine HTML non etichettate (prive di attributi che ne permettano la classificazione) sfruttando le regolarità presenti nel template. La parte di DOM riconducibile al template viene quindi filtrata, mentre i dati rimanenti vengono riallineati ed estratti. Questo tipo di approccio ha il vantaggio di non richiedere dati di addestramento, né alcun intervento umano, ed è per questo molto scalabile. Lo svantaggio è nella inclinazione a commettere errori in presenza di contenuti superflui al di fuori del template, come pubblicità innestate nelle pagine. Inoltre ai dati estratti dalle pagine non viene associata alcuna semantica. Ad esempio, estraendo una collezione di prezzi di prodotti in vendita su un sito di e-commerce, il wrapper generato non è in grado di associare l'etichetta **Prezzo** ai dati estratti. È richiesto inoltre un successivo intervento umano per identificare i dati rilevanti tra quelli estratti.

Approcci supervisionati Negli approcci supervisionati i wrapper vengono generati grazie all'uso di dati di addestramento in input, solitamente sotto forma di pagine web annotate, e del feedback di molti utenti non esperti. Avendo a disposizione una collezione di pagine in cui i dati di interesse sono evidenziati, è possibile generare automaticamente delle regole di estrazione (espressioni regolari o XPath) che estraggono quegli stessi dati, per poi riapplicarle ad altre pagine nuove. Le regole generate possono poi essere rifinite e migliorate grazie al feedback umano, ad esempio attraverso l'uso di piattaforme di crowdsourcing. I principali vantaggi di questo approccio sono la semantica associata ai dati estratti e la precisione delle regole di estrazione generate. Gli svantaggi sono la scalabilità limitata, la difficoltà nel collezionare molti dati di addestramento e la possibilità di errori nella validazione delle regole, dovuta al fatto che l'intervento umano richiede comunque conoscenze tecniche per essere efficace.

1.3 Problemi

Le tecniche di gestione dell'informazione su Web presentate nelle sezioni precedenti introducono un elevato numero di problematiche e sfide che devono essere prese in considerazione per effettuare uno scraping efficiente ed efficace di enormi collezioni di siti Web. Affrontare questi problemi è ancora più importante se le risorse di computazione a propria disposizione sono limitate e non confrontabili con quelle di un motore di ricerca professionale o di aziende leader del settore. In particolare, 4 sfide riguardano (1) la raccolta di *sampling* per la generazione di wrapper, (2) la precisione del crawling, (3) il crawling incrementale, (4) la compressione delle pagine archiviate.

1.3.1 Sampling di pagine per la generazione di wrapper

Come mostrato nella sottosezione 1.2.2.1, le tecniche di generazione automatica di wrapper richiedono in input un insieme omogeneo di pagine che condivida lo stesso template. La selezione di pagine strutturalmente simili e rappresentative di un insieme è un problema che può limitare fortemente la scalabilità della generazione: nella maggior parte dei casi un sampling manuale non è applicabile.

Si pensi ad esempio ad uno scenario in cui si vuole costruire una base di dati che raccolga specifiche di prodotti venduti su centinaia di migliaia di siti di e-commerce. Questi dati, sparsi, devono essere riconciliati e strutturati uniformemente affinché siano disponibili ad applicazioni terze o ad utenti di un motore di ricerca specializzato per questo dominio. In questo scenario collezionare diversi insiemi di pagine per un numero enorme di siti web strutturati diviene un'attività particolarmente complessa e non trattabile manualmente. Limitare l'estrazione a pochi enormi siti web col maggior numero di prodotti non rappresenta comunque una soluzione: analizzando il fenomeno della *long tail*, infatti, si può notare che il numero di prodotti presenti in molti siti web di piccole dimensioni è maggiore di quello presente in pochi siti web di grandi dimensioni.

1.3.2 Precisione del crawling

Come descritto in 1.2.1, un crawler generico effettua una visita dei siti web secondo una determinata strategia, aggiungendo alla frontiera gli URL rilevanti all'obiettivo

dell'archiviazione e terminando dopo una specifica soglia temporale o quando la frontiera è vuota.

Questo generico approccio non tiene conto della natura dei siti Web in questione e del loro modello di navigazione in fase di decisione della strategia da adottare. Il crawling di siti web strutturati, generati dinamicamente da applicazioni software, può quindi risultare inefficiente sotto diversi aspetti: in termini di uso della banda, a causa delle eccessive richieste HTTP inviate ai web server. In termini di spazio di archiviazione, a causa della possibile ridondanza di contenuto tra pagine diverse. In termini di accesso all'archivio, a causa della mancanza di un'organizzazione ragionata delle pagine HTML scaricate.

Si pensi ad esempio ad uno scenario in cui si vuole effettuare crawling di forum web con l'obiettivo di archiviare tutti i post degli utenti, al fine di migliorare la qualità delle risposte di un motore di ricerca a domande su un determinato argomento. Tipicamente i forum sono organizzati in liste di topic, ed è possibile ordinare gli stessi topic in diverse viste, corrispondenti a diverse pagine HTML, come ad esempio "ordina per data" o "ordina per autore". Un crawler generico, non conoscendo la natura dei collegamenti del sito, né il modello di navigazione del sito, è portato a scaricare pagine diverse che presentano il medesimo contenuto. Per lo stesso motivo, può essere portato a scaricare pagine non accessibili agli utenti non registrati, sprecando inutilmente banda e spazio. Conoscere la struttura di un sito web e i pattern di navigazione attraverso le pagine potrebbe quindi aiutare il crawler a capire quali sono le pagine web importanti da scaricare (per evitare i contenuti non accessibili o ridondanti) e quali link seguire.

1.3.3 Crawling incrementale

Il crawling incrementale, come descritto in 1.2.1.2, necessita di aggiornare periodicamente le pagine scaricate, richiedendo quindi un ulteriore controllo intelligente volto a ripianificare lo scaricamento di risorse già archiviate.

Questo tipo di crawling introduce nuove problematiche rispetto a quelle di un crawling *snapshot*: la frontiera non può filtrare gli URL già scaricati, ma piuttosto deve ripianificare il loro accesso seguendo determinate priorità. Devono essere stabilite euristiche per stabilire la frequenza di visita di ciascun URL. Altre euristiche sono necessarie per

permettere al crawler di comprendere se la pagina ha subito un cambiamento significativo rispetto all'ultima visita (cambiamenti nel template o nelle pubblicità non sono infatti interessanti a livello di contenuto). In generale, una strategia efficiente deve tener conto della probabilità che ha una determinata pagina di cambiare nel tempo. Si pensi ad esempio ad uno scenario in cui si vuole effettuare il crawling di un gran numero di siti di agenzie di stampa, al fine di archiviare il maggior numero possibile di notizie pubblicate per ricavare informazioni su determinati argomenti o entità. Riscaricare periodicamente l'intero sito è una soluzione semplice, ma se il periodo di attesa tra un crawling e l'altro non è ben calcolato, si rischia di perdere molte notizie che sono state pubblicate tra una pausa e la successiva, e che nel frattempo sono state rimosse dal sito. Inoltre, riscaricare completamente il sito porta a sprecare banda, tempo e spazio di archiviazione per le pagine che non sono cambiate. Conoscere la struttura del sito web e i pattern navigazionali, così come raggruppare diversi siti che condividono la frequenza di aggiornamento, può aiutare il crawler a decidere, in ogni momento, quali sono i siti con contenuto nuovo e quali aree di ciascun sito potrebbero essere state aggiornate.

1.3.4 Compressione

Il crawling di vaste porzioni del Web pone dei seri problemi sull'utilizzo efficiente dello spazio di archiviazione. Nel 2017 il peso medio di una singola pagina web ha superato i 3 MB¹. Per questo motivo, un crawling professionale solitamente non archivia un file per ogni pagina HTML. L'Internet Archive [15], specializzato nell'archiviazione dell'intero Web per fini storici, ha per questo ideato il formato Web ARChive (WARC), divenuto poi uno standard ISO². Questo formato specifica un metodo per aggregare diverse risorse digitali (pagine HTML, immagini, documenti, ecc.) in un unico file comprendente i loro metadati. È tradizionalmente usato per archiviare blocchi di pagine scaricate durante una sessione di crawling, allo scopo di risparmiare spazio in fase di salvataggio e organizzare i contenuti trovati per migliorare il tempo di accesso alle singole risorse. L'Internet Archive, tramite l'uso di file WARC di dimensioni massime

¹<http://httparchive.org/trends.php>

²<https://www.iso.org/standard/68004.html>

fisse (1 GB) e di algoritmi standard di compressione (*gzip*), ad Agosto 2014 utilizzava 9,6 PB (9'600'000 GB) di spazio, con un tasso di crescita stimato a 20 TB (20'000 GB) al mese³. Sebbene queste tecniche permettano di ridurre lo spazio di archiviazione, non traggono vantaggio dalla ridondanza dei template nelle pagine dei siti web strutturati, né tengono in considerazione i cambiamenti minori che una stessa pagina può subire nel corso del tempo.

Si pensi ad esempio ad uno scenario in cui si vuole effettuare il crawling di siti web strutturati generati dinamicamente da un'applicazione software: una normale archiviazione, anche tramite WARC, porta a salvare gli elementi di template dell'HTML di ciascuna pagina, sprecando spazio per informazioni ridondanti. Una conoscenza approfondita della struttura di tali siti potrebbe garantire una compressione focalizzata sui dati propri di una pagina, piuttosto che su quelli ridondanti condivisi da molte pagine dello stesso sito.

³<https://archive.org/web/petabox.php>

Capitolo 2

Lavori correlati

In questo capitolo viene presentata una panoramica dei lavori in letteratura correlati al tema del crawling efficiente di siti web strutturati e della generazione automatica di wrapper, di cui questa tesi fa parte. In particolare, vengono presentati lavori che descrivono tecniche di crawling ed estrazione di dati, suddivisi nelle seguenti categorie: tecniche **clustering di pagine web**, tecniche di **generazione scalabile di wrapper**, tecniche di **crawling efficiente di siti web**.

2.1 Clustering di pagine web

I lavori in questa categoria si concentrano sulla divisione delle pagine di un sito web in diversi gruppi (*cluster*): essi propongono tecniche di classificazione basate su attributi (*feature*) estraibili tramite l'analisi del DOM o del formato degli URL. L'obiettivo comune dei lavori è quello di scalare sul raggruppamento di pagine provenienti da molti siti web, che possano essere usate per generare automaticamente wrapper.

2.1.1 Crescenzi et al 2005 [5]

Questo lavoro presenta un sistema che, prendendo in input la homepage di un sito web (*entry point*), scopre automaticamente le principali classi di pagine esplorando solo una porzione del sito. In output viene prodotto un modello del sito composto da cluster di pagine web. Gli autori sostengono che il modello prodotto può essere usato come input

per la generazione automatica di wrapper, dato che ogni cluster esibisce la richiesta uniformità strutturale.

L'intuizione alla base della classificazione risiede nell'analisi della presentazione dei collegamenti all'interno delle pagine: l'osservazione chiave è che i link riflettono la regolarità della struttura. L'assunzione è che, in una pagina, un gruppo di link raggruppati insieme (ad esempio in una lista) punta a pagine dello stesso cluster. Ad esempio, una pagina web che descrive una squadra di calcio può avere al suo interno una lista di link che puntano alle pagine dei calciatori della squadra: queste pagine appartengono a un cluster, quello dei calciatori, poiché ogni pagina di calciatore esibisce probabilmente lo stesso template. Questi gruppi di link, dette *link collections*, vengono quindi utilizzate per caratterizzare la struttura di una pagina, e vengono esplorati per scoprire nuove pagine che a loro volta si assume facciano parte dello stesso cluster.

Il lavoro propone quindi una tecnica di clustering basata sull'esplorazione delle link collections, per la scoperta di nuovi gruppi, e sul raggruppamento delle pagine che espongono una struttura dei link nella pagina simili, in termini di link collections presenti nel DOM.

Sebbene le assunzioni di base mostrino che è possibile creare un modello di sito web composto da cluster di pagine, il lavoro presenta alcuni limiti. Non è possibile raggruppare con precisione pagine che non contengono link collections, perché verrebbero inserite tutte in unico cluster. Non viene data enfasi alla natura dei collegamenti tra i cluster, e le link collections che portano a pagine non strutturalmente simili vengono ignorate nella generazione del modello. Non si fa distinzione tra le link collections ricorrenti in tutte le pagine del sito e quelle caratteristiche di un solo cluster.

2.1.2 Blanco et al 2011 [2]

In questo lavoro viene presentato un algoritmo scalabile che crea cluster di pagine web utilizzando soltanto gli URL per caratterizzare la loro struttura. L'intuizione alla base del lavoro si basa sul fatto che la sola similarità tra diversi URL non è correlata alla similarità delle pagine corrispondenti; piuttosto, gli URL vengono considerati nella loro interezza al fine di scoprire i pattern che li caratterizzano. In particolare, gli autori presentano un framework che genera un insieme di script che fornisce la spiegazione

più semplice per il formato degli URL osservati. Gli URL vengono quindi analizzati e divisi in *token* al fine di essere ridotti in *script terms* e *data terms*, così da trovare un’ipotesi (insieme di script) che offre la spiegazione più semplice dei dati osservati (l’insieme di URL).

Questo tipo di approccio è particolarmente performante grazie al fatto che vengono utilizzati solamente gli URL per categorizzare le pagine, senza bisogno di effettuare analisi complesse del contenuto. Tuttavia, non sempre gli URL hanno un contenuto informativo abbastanza alto da giustificare la mancata esplorazione delle rispettive pagine: molti siti web infatti, sebbene generati dinamicamente, possono usare URL molto diversi tra loro per identificare pagine con lo stesso template, semplicemente perché la separazione può avvenire su base semantica piuttosto che strutture. Ad esempio, in un sito di ristoranti, le sezioni dedicate ai ristoranti di diversi paesi potrebbero avere URL completamente differenti tra loro ma template strutturale comune.

2.2 Generazione scalabile di wrapper

I lavori in questa categoria presentano tecniche per la generazione automatica di wrapper, concentrandosi in particolare sul problema di scalare il processo sull’intero Web. Queste tecniche hanno spesso bisogno di fare riferimento a Basi di Conoscenza (*Knowledge Bases*) o basi di dati preesistenti: si tratta infatti di tecniche supervisionate in cui l’aiuto dell’esperto umano viene sostituito da quello di ontologie apposite.

2.2.1 AAH [6]

Questo lavoro propone tecniche di crawling e generazione di wrapper efficienti dedicate all’ estrazione di dati da CMS (*content management systems*), strumenti software utilizzati sui web server per facilitare la gestione di contenuti dinamici. Viene descritto AAH (*Application-Aware Helper*), un modulo che assiste il processo di crawling identificando il tipo e la versione del CMS usato da un sito web in input, e suggerendo di conseguenza appropriate azioni di crawling (link da seguire, contenuti da estrarre). Per i CMS riconosciuti, il modulo identifica le aree ricche di contenuti, evita l’archiviazione di pagine ridondanti, e arricchisce le informazioni estratte con annotazioni semanti-

che. Per identificare i CMS usati, AAH è assistito da una Knowledge Base di CMS e applicazioni Web note, comprendente algoritmi per identificare tipi e versioni durante il crawling. L'algoritmo utilizzato prevede quindi, per ogni sito di interesse, l'identificazione dell'applicazione web, l'identificazione della versione usata, e la conseguente estrazione di contenuti e navigazione efficiente.

Sebbene il sistema effettui un crawling efficiente rispetto a quello di un crawler generico e permetta una generazione automatica di wrapper per siti diversi, il limite evidente è nella necessità di usare una Knowledge Base di CMS. Questo limita fortemente la scalabilità del sistema, e non ottiene alcun vantaggio da siti che utilizzano applicazioni web non note alla Knowledge Base.

2.2.2 DIADEM [9]

In questo lavoro viene descritto il progetto DIADEM (*Domain-Centric Intelligent automated Data Extraction Methodology*), un sistema di estrazione dati (*Data Extraction*) che utilizza delle ontologie di dominio per guidare il processo di estrazione di diversi siti strutturati relativi a diversi domini. Queste ontologie vengono manualmente create da esperti del dominio e descrivono il modo in cui le entità del dominio di interesse vengono presentate nel Web. L'ontologia identifica gli attributi attesi che vengono trovati sul Web, insieme al loro tipo e alla loro funzione di attributi opzionali o obbligatori. Grazie ad essa, prendendo in input un sito web da esplorare, DIADEM effettua il crawling del sito mediante la compilazione delle form e la navigazione dei link. Per trovare le pagine di dettaglio delle entità di interesse effettua un matching degli attributi e dei record contenuti nelle pagine HTML che mostrano liste di risultati. Il riconoscimento degli attributi nelle pagine è possibile grazie all'aiuto dell'ontologia. A questo punto il sistema genera un wrapper a partire da un insieme di *sample* di pagine con template comune, e usa il wrapper ottenuto per estrarre dati dalle pagine dello stesso tipo. Per effettuare la navigazione ed estrazione DIADEM utilizza un linguaggio specifico, OXPath [12], che definisce contemporaneamente i cammini di navigazione del sito e le regole di estrazione dei suoi contenuti.

Questa soluzione è altamente scalabile sul numero di siti web, ma è difficile da adattare ad un gran numero di domini, visto che richiede la creazione manuale di un'ontologia

per ognuno di essi.

2.3 Crawling efficiente di siti web

I lavori di questa categoria si concentrano sull'esplorazione efficiente di siti web: il loro obiettivo principale è quello di proporre strategie di navigazione alternative alla classica visita in ampiezza, che mirino a migliorare la qualità del contenuto archiviato, risparmiando nel contempo tempo e spazio evitando le aree dei siti meno interessanti, secondo determinate metriche.

2.3.1 IRobot [4]

In questo lavoro viene presentato IRobot, un crawler specializzato nel crawling efficiente di forum web. IRobot crea un modello (*sitemap*) del sito web in fase di crawling. La sitemap viene costruita scaricando dapprima alcune pagine in maniera casuale. Queste vengono poi raggruppate in cluster in base all'identificazione di regioni ripetitive nel DOM, in base al pattern degli URL e alla stima del contenuto informativo presente in esse. Una volta calcolata la sitemap, IRobot ottiene la struttura del forum. Il modello consiste in un grafo diretto in cui i vertici sono cluster di pagine e gli archi diretti sono link tra pagine dei cluster. Infine viene effettuata un'analisi dei cammini volta a identificare una navigazione ottima che consenta al crawler di evitare l'archiviazione di pagine non valide o ridondanti.

Questa tecnica permette un crawling efficiente tramite la creazione di modelli di forum in maniera non supervisionata. Tuttavia è una tecnica applicabile ai soli forum, poiché l'algoritmo di clustering si basa sull'identificazione di classi di pagine predefinite e tipiche di tutti e soli i forum. È inoltre poco efficace in presenza di forum che usano uno schema degli URL ambiguo e non standard, in cui è difficile identificare un gruppo di pagine omogeneo in base al pattern dell'indirizzo delle pagine.

2.3.2 ACEBot [7]

In questo lavoro viene presentato ACEBot (*Adaptive Crawler Bot for data Extraction*), un crawler guidato dalla struttura che sfrutta la struttura interna delle pagine e guida il

crawling in base all’importanza del contenuto. ACEBot prevede due fasi: nella prima, detta fase di apprendimento, costruisce una mappa dinamica del sito web, scaricando un numero limitato di pagine, e apprende una strategia di attraversamento sulla base dei percorsi di navigazione che portano alle informazioni più preziose. Nella seconda, ACEBot esegue un crawling completo del sito in base ai percorsi di navigazione trovati. Per la generazione della mappa, il sito viene ridotto a un grafo diretto in cui ogni nodo è una pagina e ogni arco diretto è un cammino, nel DOM della pagina sorgente, dalla radice al link (il tag HTML `<a>`) che porta alla pagina di destinazione. Ad ogni nodo (pagina) viene associato il numero di 2-grammi presenti nel DOM (una sequenza di 2 parole nell’HTML). In questo modo ACEBot è in grado di associare ad ogni percorso di navigazione tra due nodi p_1 e p_2 un punteggio, sulla base del numero totale di 2-grammi incontrati e del numero di pagine che devono essere scaricate per completare il cammino.

Questo approccio ha il vantaggio di inferire modelli di siti web che permettono una navigazione efficiente e limitata ai soli percorsi con contenuto informativo più alto in modo totalmente non supervisionato. Tuttavia, per effettuare un crawling ottimale su siti di grandi dimensioni, l’approccio ha bisogno di scaricare nella fase di apprendimento un numero di pagine fino al 66% superiore rispetto a quello necessario per gli approcci usati da altri crawler specializzati come IRobot 2.3.1 o da sistemi supervisionati come AAH 2.2.1.

Capitolo 3

Modello di navigazione di un sito

Come descritto nel capitolo 2, il problema della generazione automatica e scalabile di wrapper, così come quello del crawling efficiente di siti web, è stato affrontato in numerosi lavori, facendo uso di approcci sia supervisionati sia non supervisionati.

In questo capitolo viene descritta la metodologia usata da questo lavoro per effettuare un crawling intelligente e scalabile di siti web, riprendendo in parte il lavoro descritto in 2.1.1. Il sistema sviluppato, dato in input un insieme di entry point di siti web, crea per ciascuno di essi, scaricando un numero limitato di pagine, un modello di navigazione del sito in maniera non supervisionata e scalabile. Il modello di navigazione inferito è composto da cluster (dette *Page Class*) di pagine web e da collegamenti (detti *Class Link*) tra Page Class. Viene inoltre investigata la tipologia dei Class Link, al fine di ricostruire con precisione la topologia di ciascun sito. I modelli di navigazione inferiti vengono poi usati per effettuare un crawling intenso e scalabile di siti web. L'obiettivo è quello di generare su vasta scala modelli di navigazione di siti web strutturati, che possano essere utili per diversi compiti: (1) attribuire a ciascuna pagina una Page Class, al fine di generare sample utili alla creazione automatica di wrapper, (2) ricostruire la topologia del sito, al fine di usare il modello come base per identificare le regioni di un sito più frequentemente aggiornate (si confronti 1.3.3), (3) raggruppare le pagine strutturalmente simili, al fine di migliorare la compressione dell'archiviazione (si confronti 1.3.4).

Questo capitolo si limita alla descrizione della generazione dei modelli di siti web: nella prima sezione vengono descritte le regolarità delle pagine web sfruttabili per la classificazione. Nella seconda sezione viene descritto in dettaglio la struttura del modello generato. Nella terza sezione viene descritto in dettaglio l'algoritmo di generazione del modello.

3.1 Struttura delle pagine web

L'approccio alla base del raggruppamento delle pagine web di un sito si basa sull'identificazione di feature, all'interno del DOM delle pagine, che descrivono la loro struttura. Si consideri, come esempio, il sito di un'agenzia immobiliare inglese, White Walls¹, composto da circa 2000 pagine con informazioni e prezzi di case in vendita e in affitto. Queste pagine sono ben strutturate: per esempio tutte le pagine relative alle proprietà in vendita o in affitto mostrano lo stesso tipo di informazioni (indirizzo, prezzo, metri quadri, numero di stanze, ecc.), il che fa supporre che vengano generate con un formato standard da un'applicazione web. Allo stesso modo anche tutte le pagine che mostrano liste di proprietà in una certa area geografica condividono una struttura simile, diversa da quella delle singole proprietà. Inoltre le pagine contengono link tra di loro, in modo da proporre cammini di navigazione che rispecchiano le relazioni semantiche tra le entità descritte; per esempio, ciascuna pagina di case in affitto in un'area geografica contiene dei link che portano alle pagine delle singole proprietà.

La regolarità nei link e nelle informazioni testuali mostrate nelle pagine con struttura simile possono essere usate come feature di classificazione per raggruppare pagine simili in cluster omogenei.

3.1.1 Struttura dei link

La prima osservazione nello studio della regolarità delle pagine è che i link sono un'espressione concreta della struttura dei template di un sito web. Si consideri la pagina web in figura 3.1, che mostra una lista di proprietà in vendita nell'aera di Oxford. Come evidenziato, i link sono raggruppati in collezioni che esibiscono proprietà strutturali e

¹<http://www.wwagency.com>

di presentazione uniformi: questi gruppi vengono chiamati *link collection*. Solitamente i link nella stessa link collection conducono a pagine simili tra loro. I link della link collection nella parte alta della pagina conducono ad altre pagine di risultati che mostrano liste di proprietà, simili quindi a quella di partenza; i link della link collection nella parte centrale conducono a pagine di dettaglio delle proprietà, anche queste simili tra loro. Si può inoltre osservare che la stessa disposizione delle link collection è mantenuta in tutte le pagine dei risultati.

The screenshot shows a real estate search results page for properties in Oxford & Bicester. At the top, there's a navigation bar with links for SELLING, LETTING, VALUATION, EPC, ABOUT, CAREERS, LINKS, CONTACT, and AUCTION. Below that is a secondary navigation bar with 'Home' and 'Property Search'. On the right, there's a 'my White Walls' button. The main content area is titled 'Property in Oxford & Bicester search results'. It includes a back-link to 'Back to postcode area map', a dropdown for 'Sort by price high-low', and a checkbox for 'Tick to hide recently sold properties'. A red arrow points from the text 'pagine dei risultati' to the page navigation links at the bottom of the results. The results themselves are presented in three cards, each showing a thumbnail image, the property address ('Bagley Wood Rd, Kennington OX1'), a price ('£1,395,000'), and basic details like 'Bedrooms: 5', 'Bathrooms: 3', 'Area: Oxfordshire', and 'Type: House'. Each card also has a 'View full property details »' button. A second red arrow points from the text 'pagine delle proprietà' to one of these detailed property cards.

Figura 3.1: Esempio di pagina HTML: lista delle proprietà in vendita

Allo stesso modo, come mostrato in figura 3.2, le pagine di presentazione delle tipologie di proprietà mantengono la stessa caratteristica. Ogni pagina che presenta un tipo di proprietà (in vendita, in affitto) ha una link collection sopra alla mappa che porta a pagine di risultati di diverse aree geografiche.

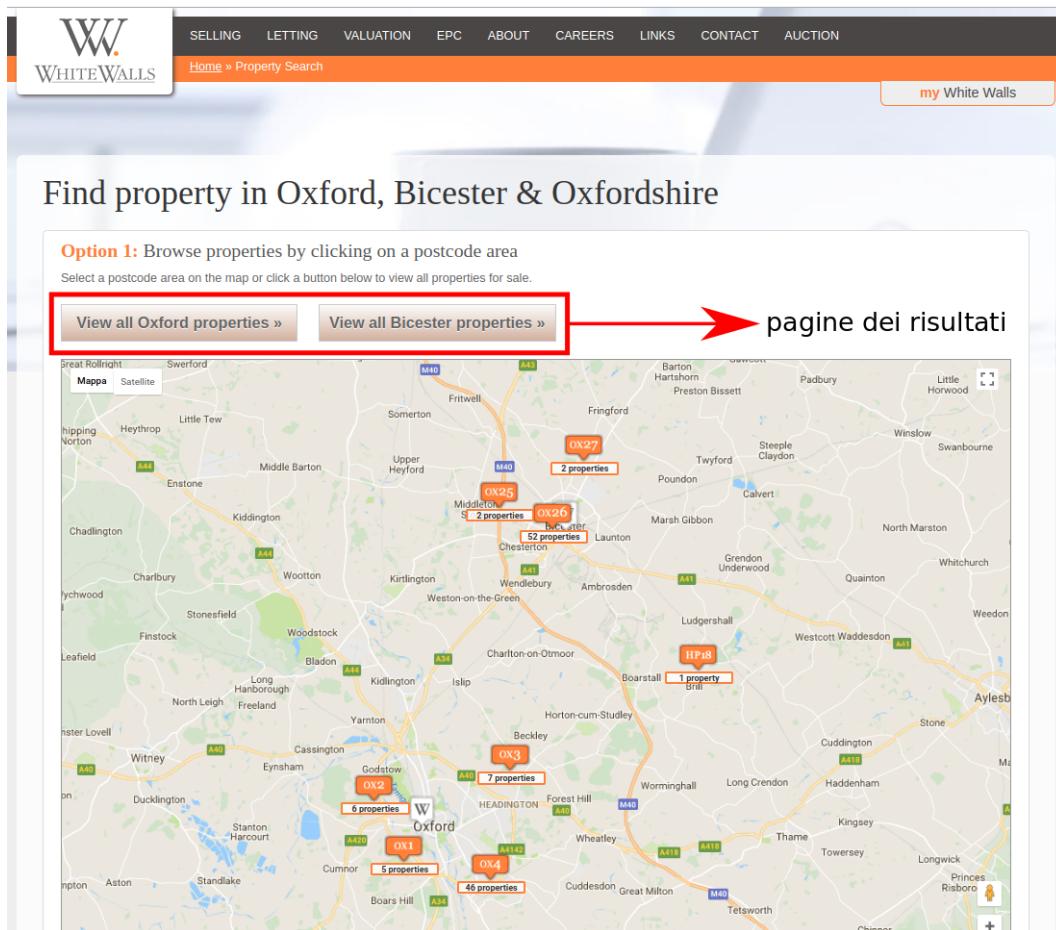


Figura 3.2: Esempio di pagina HTML: presentazione di tipologie di proprietà

Sulla base di queste argomentazioni si può quindi assumere che gruppi di link che condividono le stesse proprietà strutturali e di presentazione conducono a pagine con la stessa struttura. Nell'esempio precedente, ogni pagina che mostra liste di proprietà ha gruppi di link che conducono ad altre liste e ai dettagli delle proprietà.

3.1.2 Struttura dei testi

Le link collection sono uno strumento utile a caratterizzare pagine diverse con la stessa struttura, ma non sono l'unica forma di regolarità presente. Inoltre, possono esistere pagine prive di link. Per questi motivi possono essere presi in considerazione anche i testi fissi presenti in più pagine. Si consideri la pagina web in figura 3, che mostra i dettagli di una proprietà in affitto. Come evidenziato, la pagina presenta alcuni testi con proprietà comuni:

- **isolamento:** i testi vengono strutturati nel DOM e mostrati nella pagina in zone isolate dal resto.
- **brevità:** i testi sono brevi e hanno spesso il ruolo di etichette che identificano informazioni di interesse nella pagina.
- **unicità:** i testi compaiono una sola volta nella pagina.

Questi testi presentano quindi proprietà strutturali e di presentazione uniformi, che si ripetono con le medesime caratteristiche in diverse pagine dello stesso tipo. Ad esempio, il gruppo di testi in basso a sinistra identifica un template che descrive le caratteristiche della proprietà in affitto. Questi gruppi di testi vengono chiamati *text collection*.

**St. Mary's Road
East Oxford OX4**

£3,500 per month

MORE PHOTOS COMING SOON Available six bedroom HMO house located on a popular road in East Oxford. Offering a semi-open plan modern kitchen providing access to garden. The house will be ready following conversion middle of January.

Located within easy reach of Templar Square shopping centre and Templar Retail Park. The property is also conveniently located for the Oxford BMW Mini plant. For those looking to commute the A34 and M40 are within easy reach.

Key Details

Property Type	House
Postcode Area	OX4
Bedrooms	6
Bathrooms	3
Furnishing	Furnished
Available from	16 Jul 2017
Area	East Oxford
Tenure	N/A
Sq feet	0

Property Location

Click the map below for an interactive view and to find nearby places.

Live Postcode Intelligence

Applicants looking to rent in OX4 through White Walls Agency

611	1 bed+	107
	2 bed+	92
	3 bed+	53
	Various	359

Interested in this property?

Rupert Lister

Call our Oxford Branch
Mon to Fri 8am – 7pm
Sat 10am – 3pm
01865 246 100
15 Hollybush Row
Oxford
OX1 1JH

My property value?

Arrange a viewing »

IMPORTANT NOTICE

1. White Walls Agency Limited for itself and for the vendor(s) or lessor(s) of this property whose agents they are, give notice that: 1) These particulars do not constitute any part of an offer or contract 2) None of the statements contained in these particulars as to the property are to be relied upon as statements or representations of fact. 3) Any intending purchaser or lessee must satisfy himself by inspection or otherwise as to the correctness of each of the statements contained in these particulars. 4) The Vendor(s)

Figura 3.3: Etichette nella pagina di una proprietà

Le stesse assunzioni possono essere fatte per la pagina in figura 4, che mostra una lista di proprietà in affitto: ciascuna pagina di questa categoria infatti presenta un testo singolo in alto, "Property Search". Questa caratteristica è quindi utile a identificare le pagine che mostrano liste di proprietà.

The screenshot shows a real estate website for 'WHITEWALLS'. The top navigation bar includes links for SELLING, LETTING, VALUATION, EPC, ABOUT, CAREERS, LINKS, CONTACT, and AUCTION. The 'Property Search' link is highlighted with a red box. A user icon 'my White Walls' is also visible.

The main content area displays 'Property in Oxford & Bicester search results'. It includes a sorting dropdown ('Sort by price high-low'), a checkbox for hiding recently let properties, and a 'Save search' button. Navigation links for 'PREVIOUS' (1, 2, 3, 4, 5, 6), 'NEXT', and '31 - 40 of 53 properties' are shown.

Three property listings are displayed:

- East Avenue, East Oxford OX4**
 A flat fronted, mellow brick Victorian cottage featuring two double bedrooms, two reception rooms, large garden and off street parking in the ever popular East Avenue. A beautiful cottage with a wealth of period features, original wood floors, fully fitted kitchen, separate utility area and lovely modern bathroom. You enter the [...]
[View full property details »](#)
- Kelburne Road, Cowley, Oxford OX4**
 A spacious three bedroom family home with the benefit of an open plan cream gloss kitchen to the dining area, running the width of the house to the rear. There is also a separate bay fronted reception room, modern family bathroom and off road parking to the front. The front door enters to [...]
[View full property details »](#)
- Empress Ct, Central Oxford OX1**
 A furnished, impressive one double bedroom second floor apartment situated within this up-market development moments from the many amenities, shops and public transport in central Oxford. The front door leads to the entrance hall with door to a semi-open plan living space and fitted kitchen, leading to balcony. One well proportioned double bedroom [...]
[View full property details »](#)

Figura 3.4: Etichette nella pagina di una lista di proprietà

3.1.3 Classificazione

Sulla base delle osservazioni descritte nelle sottosezioni precedenti, è possibile dedurre che:

- è ragionevole assumere che i link che condividono proprietà strutturali e di presentazione portano comunemente a pagine strutturalmente simili.
- l'insieme delle proprietà strutturali e di presentazione che caratterizzano i link e i testi fissi di una pagina possono essere usati per caratterizzare la struttura della pagina stessa. In altre parole, quando due o più pagine condividono le stesse proprietà caratterizzanti i link e i testi, è probabile che esse abbiano la stessa struttura. Negli esempi precedenti, le pagine che mostrano liste di proprietà presentano le medesime link collection e text collection, con le medesime proprietà di presentazione.

3.2 Struttura del modello

Viene ora descritto formalmente il modello a cui viene ridotta la struttura di un sito web, basato sull'idea che le proprietà strutturali e di presentazione associate ai link e ai testi di una pagina sono sufficienti a descrivere la pagina stessa.

3.2.1 Pagina e schema

Una pagina web viene rappresentata come un sottoinsieme di cammini dalla radice ai link e di cammini dalla radice ai testi nella rappresentazione ad albero del DOM, insieme agli URL e ai testi stessi.

3.2.1.1 Link Collection

Una link collection è una struttura composta da un cammino dalla radice a un'ancora², insieme agli URL che condividono quel cammino. Si prenda come esempio il DOM della pagina p_1 mostrato in figura 3.5. Questa pagina è composta da due link collection: $HTML - TABLE - TR - TD - A \{url_{B1}, url_{B2}\}$ e $HTML - UL - LI - A$

²I cammini vengono rappresentati come XPath applicate sul DOM della pagina.

$\{url_C1, url_C4\}$. Nell'esempio della pagina p_2 in figura 3.6 invece le link collection sono $HTML - TABLE - TR - TD - A \{url_B3, url_B4, url_B2\}$, $HTML - B - A \{url_B1\}$, $HTML - UL - LI - A \{url_C2\}$. A una link collection può essere associato anche un tipo, definito in base alla omogeneità della struttura delle pagine associate agli URL della collezione.

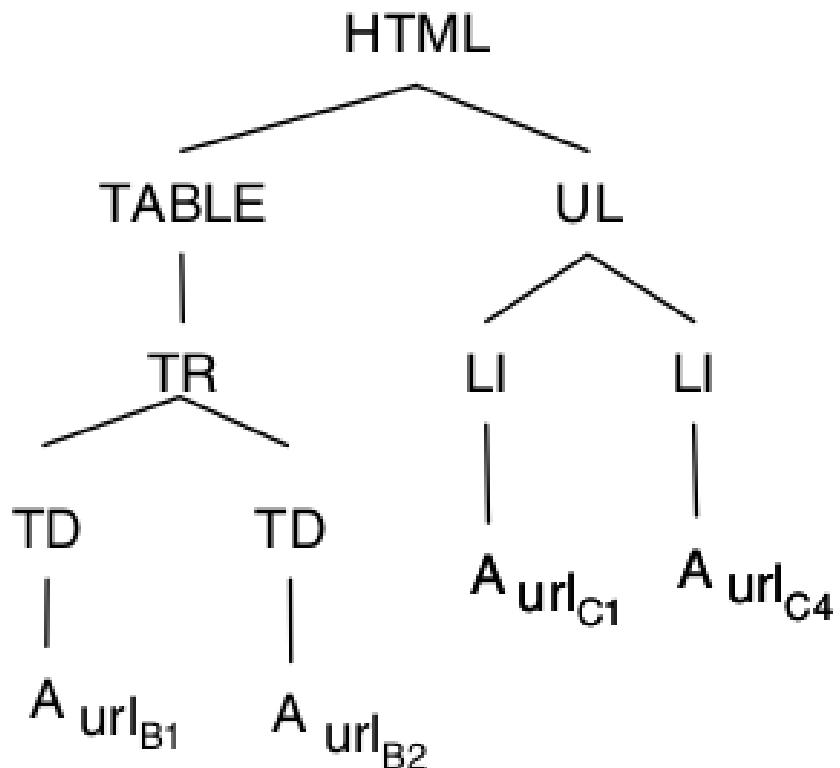
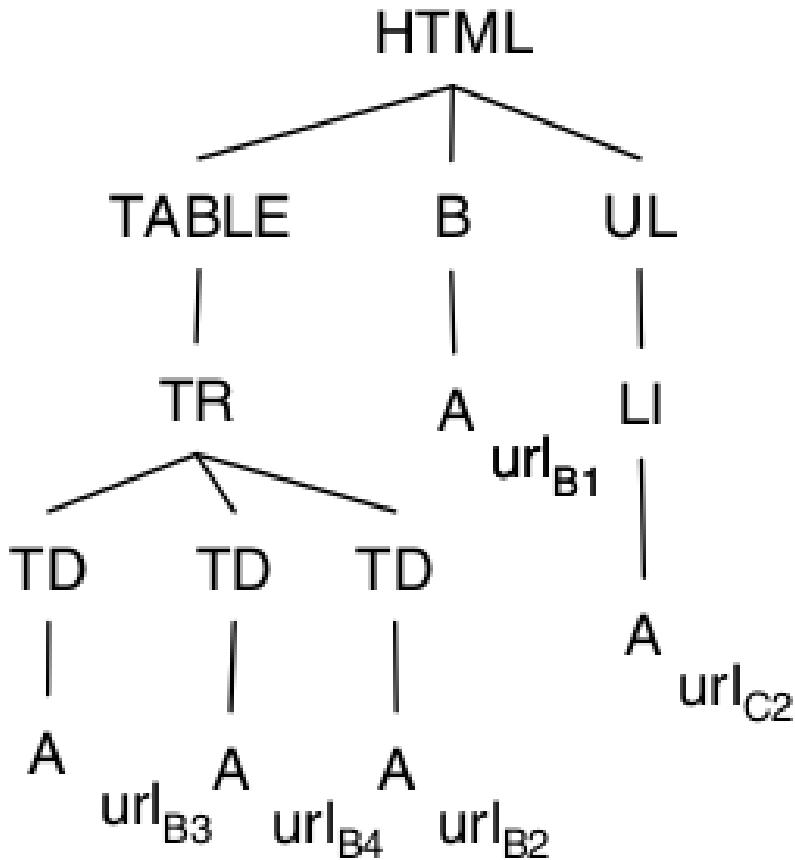


Figura 3.5: DOM della pagina p_1

Liste Le link collection di tipo lista portano a un gruppo di URL omogenei, appartenenti alla stessa classe. Nel DOM di figura 3.5, se $\{url_C1, url_C4\}$ portano a pagine strutturalmente simili, allora la link collection $HTML - UL - LI - A$ è una lista.

Figura 3.6: DOM della pagina p_2

Menu Le link collection di tipo menu portano a un gruppo di URL non omogenei, appartenenti a classi diverse. Nel DOM di figura 3.6, se $\{url_{B3}, url_{B4}, url_{B2}\}$ portano a pagine strutturalmente diverse, allora la link collection $HTML-TABLE-TR-TD-A$ è un menu.

Si noti inoltre che il tipo menu può essere suddiviso in due tipologie: menu *ad ancora fissa* e menu *ad ancora variabile*. Si prenda ad esempio il DOM di figura 3.7: la link collection $HTML - TABLE - TR - TD - A$ è presente anche in questa pagina ma porta ad URL diversi da quella precedente. In questo caso la link collection $HTML - TABLE - TR - TD - A$ è un menu ad ancora variabile. Al contrario, una link collection di tipo menu che conduce a URL non omogenei ma sempre uguali è allora un menu ad

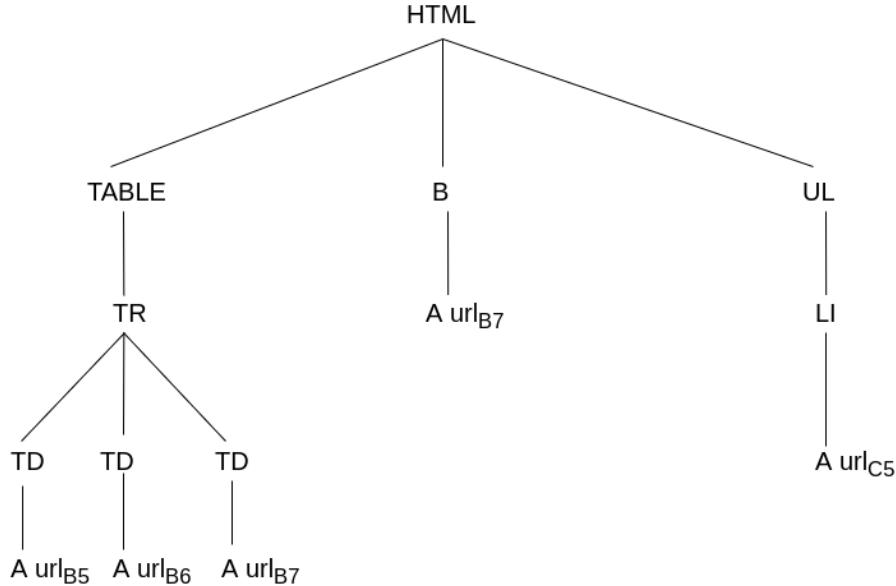


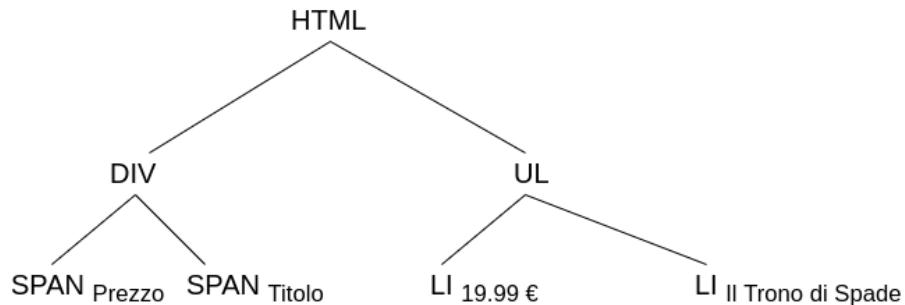
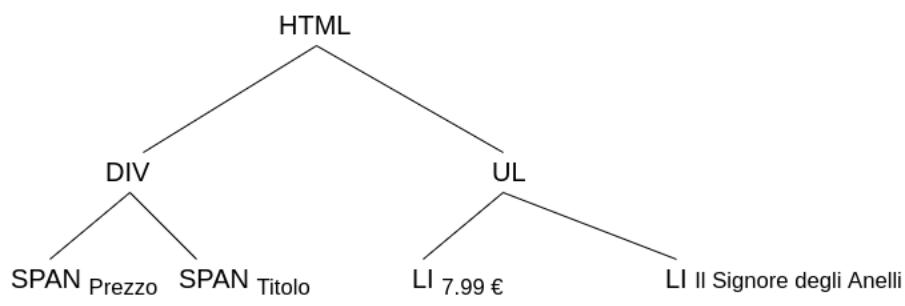
Figura 3.7: DOM con menu ad ancora variabile

ancora fissa.

Link singoli Le link collection di tipo singolo portano a un solo URL. Questa situazione è facilmente osservabile nei moderni siti web strutturati, nei quali link che rimandano ad esempio ad una pagina di login in un'area persona o alla homepage del sito hanno spesso una posizione di presentazione isolata dal resto dei collegamenti. Nel DOM di figura 3.6, la link collection con cammino $HTML - B - A$ è di tipo singolo.

3.2.1.2 Text Collection

Una text collection è una struttura composta da un cammino dalla radice a un elemento HTML contenente un testo, insieme ai testi che condividono quel cammino. Si prenda come esempio il DOM della pagina p_3 mostrato in figura 3.8. Questa pagina è composta da due text collection: $HTML - DIV - SPAN \{ "Prezzo", "Titolo" \}$ e $HTML - UL - LI \{ "19.99", "IlTronodiSpade" \}$. La pagina p_4 di figura 3.9 è invece composta dalle text collection $HTML - DIV - SPAN \{ "Prezzo", "Titolo" \}$ e $HTML - UL - LI \{ "7.99", "IlSignoredegliAnelli" \}$.

Figura 3.8: DOM della pagina p_3 Figura 3.9: DOM della pagina p_4

3.2.2 Page Class

In questo contesto, lo *schema di una pagina* è un’astrazione della pagina consistente in un insieme di cammini dalla radice a un elemento HTML (link o elemento contenente testo). Negli esempi precedenti, gli schemi delle pagine in figura 3.5, 3.6 e 3.8 sono: $\{HTML - TABLE - TR - TD - A, HTML - UL - LI - A\}$ per p_1 , $\{HTML - TABLE - TR - TD - A, HTML - B - A, HTML - UL - LI - A\}$ per p_2 , $\{HTML - DIV - SPAN, HTML - UL - LI\}$ per p_3 e p_4 .

3.2.2.1 Definizione

Sulla base della definizione di schema di una pagina, si definisce *Page Class* una collezione di pagine, e lo *schema di una Page Class* come l’unione degli schemi individuali delle pagine. Si noti che, mentre per i cammini delle link collection viene effettuata

una semplice unione, il cammino di una text collection entra invece a far parte dello schema solo se i testi associati alla text collection sono gli stessi in almeno due pagine della Page Class.

Per esempio, lo schema della Page Class comprendente le pagine di figura 3.5 e 3.6, p_1 e p_2 , ha il seguente schema: $\{HTML - TABLE - TR - TD - A, HTML - UL - LI - A, HTML - B - A\}$. Lo schema della Page Class comprendente le pagine di figura 3.8 e 3.9, p_3 e p_4 , è invece $\{HTML - DIV - SPAN\}$. In quest'ultimo caso viene infatti considerata solo la text collection che porta a una collezione di testi invarianti (nell'esempio, etichette indicanti i campi di un libro in vendita).

Intuitivamente, gli schemi di pagine strutturalmente simili si sovrappongono largamente, perciò lo schema della Page Class risultante dall'unione di esse non differirà molto dallo schema delle pagine singole.

3.2.2.2 Class Link

Data la Page Class C_A e uno dei cammini del suo schema pt , si considerino le link collection delle pagine in C_A associate con pt . Esiste allora un *Class Link* L tra C_A e la Page Class C_B se esistono link nelle link collection associate a pt che puntano a pagine di C_B . pt è allora il cammino di L . Ad L è inoltre associato un tipo (lista, menu, link singolo) direttamente derivato dal tipo associato alle link collection in C_A di cui pt è il cammino.

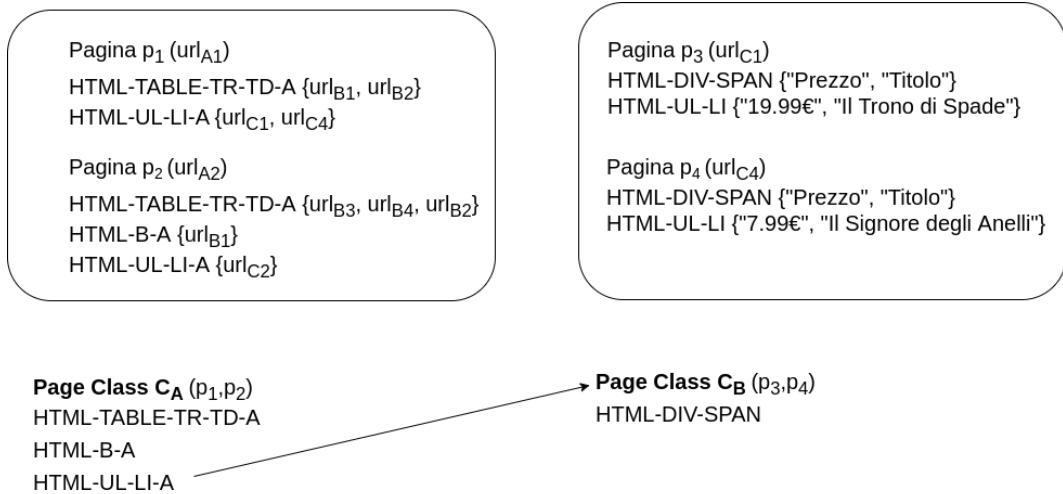


Figura 3.10: Schemi delle pagine, delle Page Class e class link

Nell'esempio in figura 3.10, C_A ha un Class Link verso C_B identificato da $HTML - UL - LI - A$. Essendo inoltre, in p_1 , la link collection $HTML - UL - LI - A$ $\{url_{C1}, url_{C4}\}$ di tipo lista (url_{C1} e url_{C4} sono infatti nella stessa Page Class), anche il Class Link identificato è di tipo lista.

3.2.3 Grafo del sito web

Un *modello di sito web* è allora un grafo diretto in cui i nodi sono Page Class e gli archi diretti sono Class Link. In quest'ottica un modello di sito web descrive i cammini di navigazione tra le Page Class. Si consideri il modello in figura 3.10: esso indica che per ogni pagina appartenente alla Page Class C_A , i link i cui cammini sono $HTML - UL - LI - A$ conducono a pagine appartenenti alla Page Class C_B . Intuitivamente, un modello ben costruito descrive un partizionamento omogeneo del sito web, tale che ogni Page Class contiene pagine strutturalmente simili.

3.3 Generazione del modello

La generazione automatica del modello avviene mediante un algoritmo volto a scaricare un numero massimo specificato in input di pagine. Le pagine vengono suddivise in classi in maniera incrementale, durante il crawling del sito. La qualità del modello generato è valutata con un approccio basato sulla teoria dell'informazione e sul machine learning, ispirato al *principio della minima lunghezza di descrizione* (MLD) [13].

3.3.1 Algoritmo

L'algoritmo 1 mostra lo pseudocodice della procedura di generazione del modello di un sito web. L'input dell'algoritmo è l'entrypoint del sito web (tipicamente l'URL della homepage), che diventa la prima pagina della prima classe del modello. Le link collection vengono quindi estratte dalla pagina e inserite in una coda di priorità. L'algoritmo prosegue l'iterazione fino a che non sono più presenti link collection in coda, o fino a quando un numero massimo di richieste HTTP al server è stato eseguito (ovvero, fino allo scaricamento di un numero massimo di pagine). Ad ogni iterazione (righe 5-39), l'algoritmo seleziona una link collection dalla coda e scarica un numero limitato di URL in essa. Si assume infatti che pochi URL di una collezione³ siano sufficienti a caratterizzarne la natura intera. Questo limite non viene tuttavia considerato se l'algoritmo identifica un menu come tipo della link collection (in questo caso tutti gli URL vengono scaricati).

Le pagine scaricate passano poi per tre fasi in sequenza:

- Nella prima fase, chiamata *selezione dei candidati* (righe 9-17), le pagine scaricate sono divise in gruppi in base al loro schema; successivamente i gruppi simili vengono uniti e si ottengono le classi candidate.
- Nella seconda fase, chiamata *ispezione dei candidati* (righe 19-25), viene indagata la natura della link collection, in base al numero di candidati creati e al numero di pagine scaricate: se la link collection punta a un gruppo di pagine non simili tra loro, l'espressività del cammino associato viene modificata al fine di trovare

³Negli esperimenti eseguiti sono stati scaricati 3 link non consecutivi per link collection.

link più simili tra di loro (i dettagli sono discussi più avanti nella sezione 3.3.3). Infine alla link collection viene associato un tipo (lista, menu o link singolo).

- Nella terza fase, chiamata *aggiornamento del modello* (righe 27-37), le classi candidate vengono usate per aggiornare il modello. L'obiettivo di questa fase è quello di capire se i candidati rappresentano nuove classi del modello o se devono essere accorpate a classi già calcolate e inserite nel modello. Infine vengono salvati i collegamenti tra la pagina che contiene la link collection corrente e le pagine scaricate.

Alla fine della terza fase, l'algoritmo ha generato una versione rifinita del modello. Le nuove link collection delle pagine scaricate vengono quindi aggiunte in coda e la prossima collezione viene processata.

Quando la condizione di terminazione del ciclo è raggiunta, vengono analizzati i collegamenti tra tutte le pagine del modello: in base a questa analisi vengono create le Page Class e i relativi Class Link, che formano in output il grafo di navigazione del sito.

Nelle sezioni successive vengono descritti nel dettaglio i tre passi dell'algoritmo.

Algorithm 1 GENERATE MODEL

Require: l_0 , entrypoint del sito web
Require: n , numero massimo di richieste HTTP
Require: dt , soglia per l'unione dei candidati
Ensure: $GraphM$, modello del sito web come grafo di navigazione

```

1:  $M = \emptyset$  // modello, insieme delle Page Class
2:  $Q = \{l_0\}$  // coda delle Link Collection
3:  $fetched = 0$  // numero di richieste HTTP
4: while  $Q \neq \emptyset$  and  $fetched < n$  do
5:    $lc \leftarrow top(Q)$  // Link Collection con priorità maggiore
6:    $pages \leftarrow top(lc)$  // download di un numero limitato di url nella Link Collection
7:    $fetched += size(pages)$ 
8:   // Selezione dei candidati
9:    $Groups = (G_1, \dots, G_k)$  // pagine in  $pages$  raggruppate per schema
10:  for  $i : 1..k$  do
11:    for  $j : k..i + 1$  do
12:      if  $distance(G_i, G_j) < dt$  then
13:         $collapse(G_i, G_j)$  // unisce i due gruppi
14:      end if
15:    end for
16:  end for
17:   $Candidates = (C_1, \dots, C_k)$  // le classi candidate finali di  $Groups$ 
18:  // Ispezione dei candidati
19:  if  $size(Candidates) > 1$  and  $shouldRefine(Candidates)$  then
20:     $lc' \leftarrow refine(lc)$  // rifinisce il cammino di  $lc$  sulla base del suo reticolo  $L$ 
21:     $add(Q, lc')$ 
22:     $continue$ 
23:  else
24:     $setType(lc)$  // stabilisce il tipo della Link Collection
25:  end if
26:  // Aggiornamento del modello
27:  for all  $C$  in  $Candidates$  do
28:     $M_{new} = M \cup \{C\}$ 
29:     $Models = list(M' = (M - C') \cup \{C \cup C'\} \text{ for any } C' \text{ in } M)$ 
30:     $M_{merged} = M'$  in  $Models$  with lowest  $MDL(M')$ 
31:    if  $MDL(M_{merged}) < MDL(M_{new})$  then
32:       $M = M_{merged}$ 
33:    else
34:       $M = M_{new}$ 
35:    end if
36:  end for
37:   $setLinks(lc, pages)$  // crea i link tra la pagina di  $lc$  e le pagine scaricate
38:   $lcs \leftarrow linkCollections(pages)$ 
39:   $addAll(Q, lcs)$ 
40: end while
41:  $GraphM \leftarrow graph(M)$  // crea il grafo delle Page Class
42: return  $GraphM$ 
```

3.3.2 Selezione dei candidati

L'input della fase di selezione dei candidati è un'insieme di pagine scaricate appartenenti alla link collection in esame. Questa fase partiziona le pagine in insiemi separati di classi candidate. Lo scopo è quello di tenere separate pagine strutturalmente diverse al fine di poter indagare la natura della link collection nella fase successiva.

L'algoritmo identifica prima le pagine con schema identico (riga 9). In una seconda fase, si cerca di unire gruppi diversi che presentano una struttura tuttavia simile. Questa situazione intermedia può presentarsi in casi in cui è presente un gruppo di link più corposo, quindi più rappresentativo della link collection, e gruppi più piccoli con pochi link, che potrebbero far riferimento a pagine con template diverso ma con semantica simile a quella del gruppi più grande.

Per unire gruppi simili ma non identici viene usata una funzione di distanza tra gli schemi di due classi. Coppie di classi che presentano una distanza minore di una soglia dt^4 vengono unite in una classe sola. Dal momento che l'obiettivo è quello di attrarre gruppi piccoli in quelli grandi, le classi vengono ordinate per cardinalità e vengono calcolate le distanze tra i gruppi più grandi e quelli più piccoli, iterativamente, fino a che non sono state confrontate tutte le coppie.

La distanza tra due schemi è definita come la cardinalità normalizzata dell'insieme della differenza simmetrica tra i due schemi. Detti G_i e G_j gli schemi dei gruppi i e j :

$$dist(G_i, G_j) = \frac{|(G_i - G_j) \cup (G_j - G_i)|}{G_i \cup G_j} \quad (3.1)$$

Si noti che se $G_i = G_j$ (schemi identici) allora $dist(G_i, G_j) = 0$; se $G_i \cap G_j = \emptyset$ (insiemi disgiunti) allora $dist(G_i, G_j) = 1$.

Questa fase produce gruppi di link più grandi, utili all'investigazione nella fase successiva.

3.3.3 Rifinitura dei cammini

Sebbene l'assunzione iniziale preveda che una link collection punti a pagine strutturalmente simili, questa condizione può essere violata: una link collection può infatti

⁴Negli esperimenti eseguiti è stato usato $dt = 0,2$

portare a un menu o ad una barra di navigazione intesa per condurre l'utente in aree strutturalmente diverse del sito. Ad esempio, osservando la figura 3.1 si può notare che i link nella parte superiore della pagina conducono ad aree distinte del sito. Allo stesso modo, il cammino usato per caratterizzare la link collection potrebbe semplicemente non indicare alcun template del sito, ma piuttosto essere il frutto di una espressività errata e non abbastanza corretta data al cammino stesso. Ad esempio, nella figura 3.1 è possibile che un cammino identificato punti sia ai link di paginazione sia a quelli dei dettagli delle proprietà.

Per risolvere questi problemi, la seconda fase prende in input i candidati creati e attua operazioni di ispezione e rifinitura dei cammini usati.

3.3.3.1 Ispezione

Nella fase di ispezione (riga 19) viene effettuata un'analisi del numero di candidati trovati. Se la link collection porta a più di un candidato, allora si modifica l'espressività del cammino associato, nella speranza che il cambiamento porti a individuare un gruppo di link omogeneo. Una nuova link collection, con espressività più specifica, viene quindi aggiunta in coda. Se il numero di gruppi rimane superiore ad uno, allora si considera la link collection di tipo menu: in questo caso viene effettuato un crawling di tutti i link associati, per permettere al crawler di raggiungere nuove zone del sito. Se la link collection porta a un solo link, si cerca anche in questo caso di modificare l'espressività del cammino, nella speranza che una espressività più generica porti a trovare un numero di link superiore.

Una volta completata la rifinitura, alla link collection viene associato un tipo in base al numero di classi trovate.

3.3.3.2 Reticolo dei cammini

Come definito in 3.2.1.1, una link collection è descritta da un cammino dalla radice a un'ancora nel DOM della pagina in analisi. Dal momento che la presentazione delle pagine dei moderni siti web è molto ricca, ciascun elemento HTML nel cammino è solitamente composto da diversi attributi con nome e valore. Si prenda come esempio il DOM mostrato in figura 3.11: ciascun cammino dalla radice ad un'ancora è composto

da elementi HTML comprendenti attributi. Ad esempio, il cammino esatto generato a partire dal link url_1 è $HTML - DIV[@CLASS = "List1"] - UL[@CLASS = "List1"] - LI - A$. Si noti che questo cammino descrive implicitamente una link collection che ha come URL associati l'insieme $\{url_1, url_2\}$. Allo stesso modo, usando una versione più generica del cammino, in cui ad esempio vengono specificati solo i nomi degli attributi degli elementi, senza valori, la link collection diventa la seguente: $HTML - DIV[@CLASS] - UL[@CLASS] - LI - A : \{url_1, url_2, url_3, url_4\}$.

L'*espressività del cammino* di una link collection è allora una versione di cammino definita dal livello di dettaglio nella specifica dei nomi e dei valori degli attributi degli elementi HTML che appartengono ad esso⁵. Sulla base di questa definizione, si definisce *reticolo dei cammini* di una link collection l'insieme delle espressività generabili per il cammino. Si prenda come esempio il reticolo mostrato in figura 3.12: esso mostra 5 diversi livelli di espressività del cammino dal link url_1 alla radice. Il livello 1 indica l'espressività più generica, mentre il livello 5 indica l'espressività più specifica. La figura 3.13 mostra le diverse versioni di link collection associabili al cammino di url_1 : ogni riga mostra una link collection generata selezionando un cammino all'i-esimo livello del reticolo.

⁵Si noti che ad ogni livello di espressività viene selezionato solo uno dei possibili cammini appartenenti al livello.

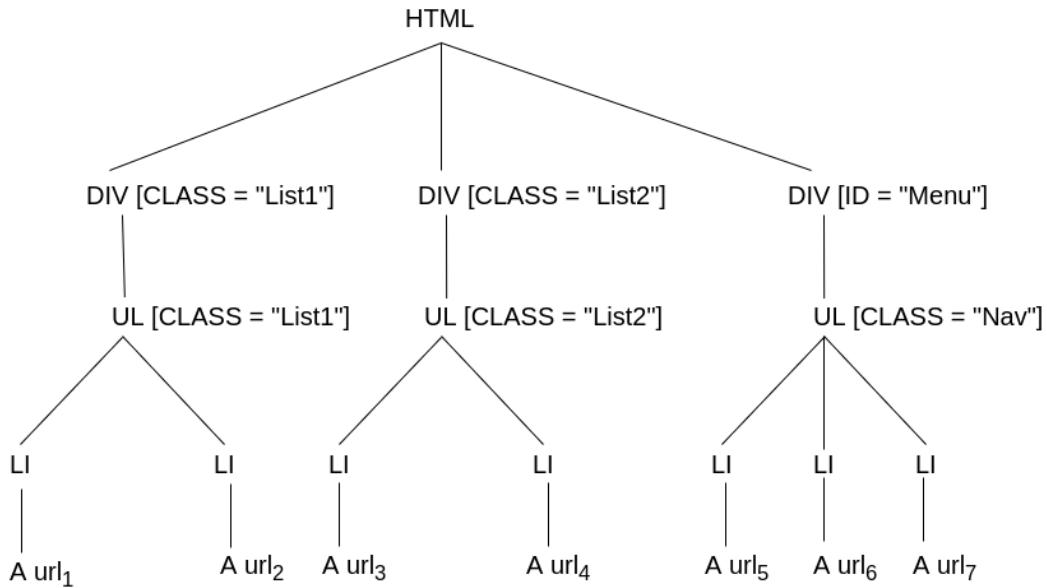
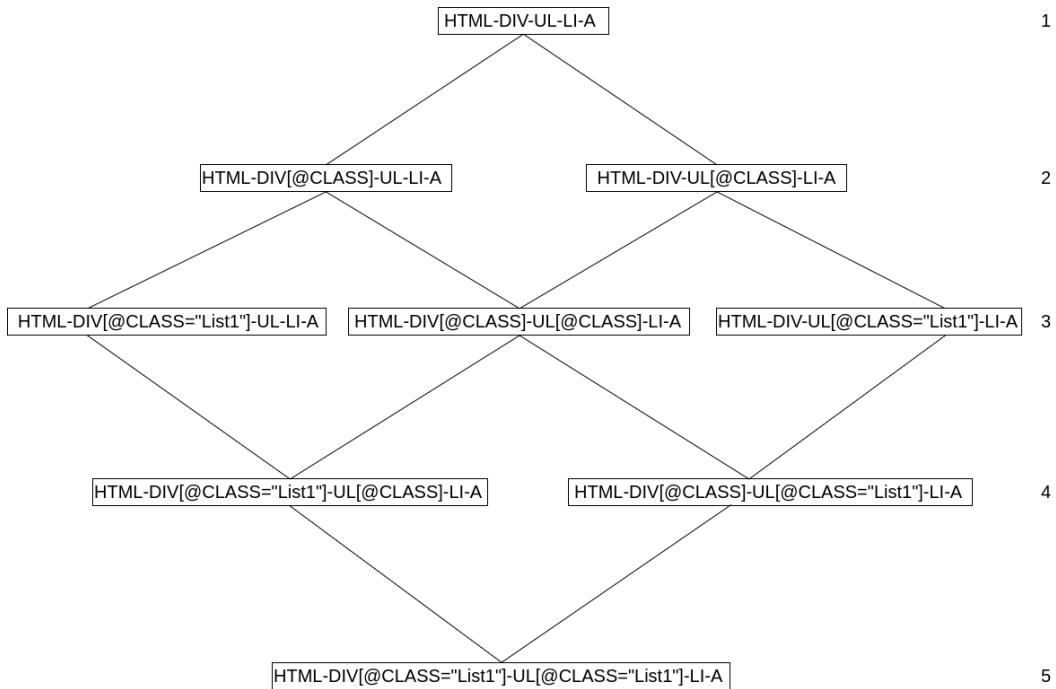


Figura 3.11: DOM con elementi e attributi HTML

Figura 3.12: Reticolo dei cammini per l'ancora *url*₁

L di lc₁

HTML-DIV-UL-LI-A : {url₁, url₂, url₃, url₄, url₅, url₆, url₇}

HTML-DIV[@CLASS]-UL-LI-A : {url₁, url₂, url₃, url₄}

HTML-DIV[@CLASS]-UL[@CLASS]-LI-A : {url₁, url₂, url₃, url₄}

HTML-DIV[@CLASS="List1"]-UL[@CLASS]-LI-A : {url₁, url₂}

HTML-DIV[@CLASS="List1"]-UL[@CLASS="List1"]-LI-A : {url₁, url₂}

Figura 3.13: Reticolo *L* della link collection creata a partire da *url*₁

3.3.4 Aggiornamento del modello

La terza frase prende in input le classi candidate generate dalla link collection rifinita e le usa per aggiornare il modello, così da creare una versione rifinita dello stesso. In questa fase l'obiettivo principale è capire se le classi candidate compaiono già nel modello o se rappresentano nuove classi.

Questa decisione viene effettuata creando un'insieme di possibili modelli candidati e scegliendo il migliore tra essi, seguendo un approccio basato sulla teoria dell'informazione e sul machine learning, ispirato al principio della minima lunghezza di descrizione (MLD). Intuitivamente, il principio della MLD asserisce che il modello migliore per descrivere un insieme di dati è quello che minimizza la somma di: (1) la lunghezza di una stringa di bit che codifica il modello, e (2) la lunghezza di una stringa di bit che codifica i dati in base al modello. Un modello è quindi tanto migliore quanto più breve è la sua descrizione.

La descrizione del modello e dei dati si basa principalmente sulla codifica dei cammini appartenenti allo schema di ciascuna classe e ciascuna pagina. Nei paragrafi successivi viene descritta (1) la modalità di attribuzione di un peso a ciascun cammino presente nel modello, (2) come questo peso viene utilizzato nella codifica del modello vera e propria.

3.3.4.1 Peso dei cammini

Come descritto 3.2.2, a ciascuna pagina e Page Class del modello viene attribuito uno schema composto da un’insieme di cammini. I cammini rappresentano quindi la struttura di una pagina e di una Page Class. I moderni siti web strutturati, solitamente generati da applicazioni web, presentano elementi strutturali e di presentazione che si ripetono in diverse pagine del sito: si pensi ad esempio ad un menu di navigazione o ad un collegamento alla homepage del sito, che vengono spesso mostrati in tutte le pagine. Allo stesso modo esistono elementi di presentazione che si ripetono all’interno delle sole pagine di una Page Class, e che quindi caratterizzano il template della Page Class stessa. La frequenza di questi elementi strutturali nelle pagine, formalizzati nel modello come cammini di link collection e text collection, è utile a identificare i cammini associabili a diversi livelli di astrazione: cammini di sito, di Page Class e di singola pagina.

Si prenda come esempio la figura 3.14, che mostra la pagina di dettaglio di un prodotto in un sito web di e-commerce⁶: le link collection in blu, caratterizzate da un cammino, rappresentano elementi strutturali presenti in tutte le pagine del sito. Si tratta infatti di menu di navigazione tra le diverse aree del portale, quindi sono *cammini di sito*. Le link e text collection in rosso, invece, rappresentano gli elementi strutturali comuni a ciascuna pagina che mostra i dettagli di un prodotto: sono quindi *cammini di Page Class*. Nella figura 3.15 viene mostrata la pagina di un altro prodotto: come si può notare, i cammini blu e rossi sono i medesimi. La link collection al centro della pagina, in grigio, rappresenta invece un link caratterizzato da un cammino non presente nel DOM del prodotto precedente. È quindi un *cammino di singola pagina*. Infine, nella figura 3.16 viene mostrata una pagina che mostra elenchi di prodotti, facente parte quindi di un’altra Page Class: come si può notare, le link collection in blu presentano la stessa struttura delle pagine precedenti. Le link collection in rosso sono invece composto da cammini di Page Class, presenti nelle altre pagine della Page Class dei risultati ma non in quella dei dettagli di prodotto.

⁶<http://www.abtvaultin.com>

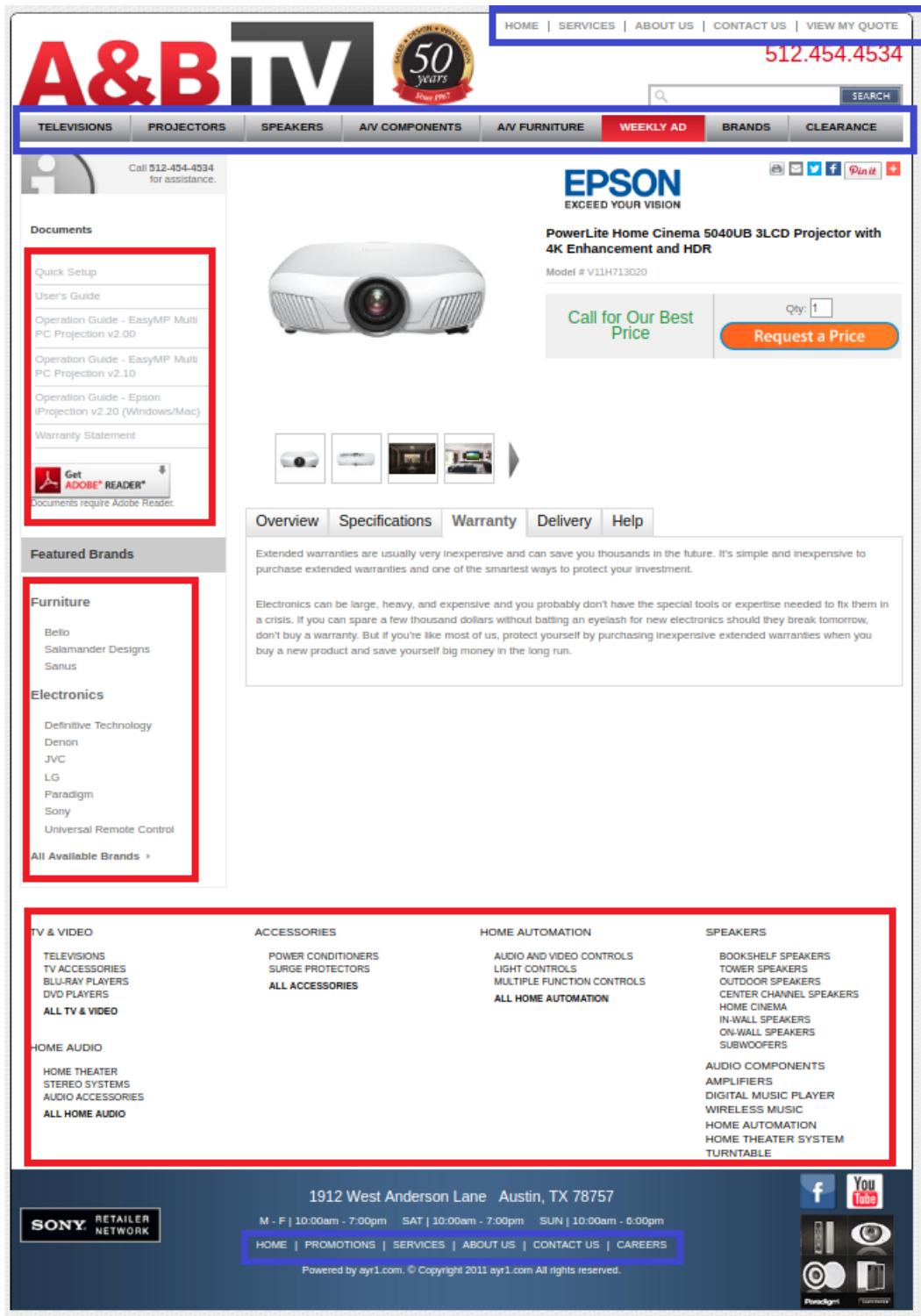


Figura 3.14: Cammini di sito e di Page Class nella pagina di un prodotto

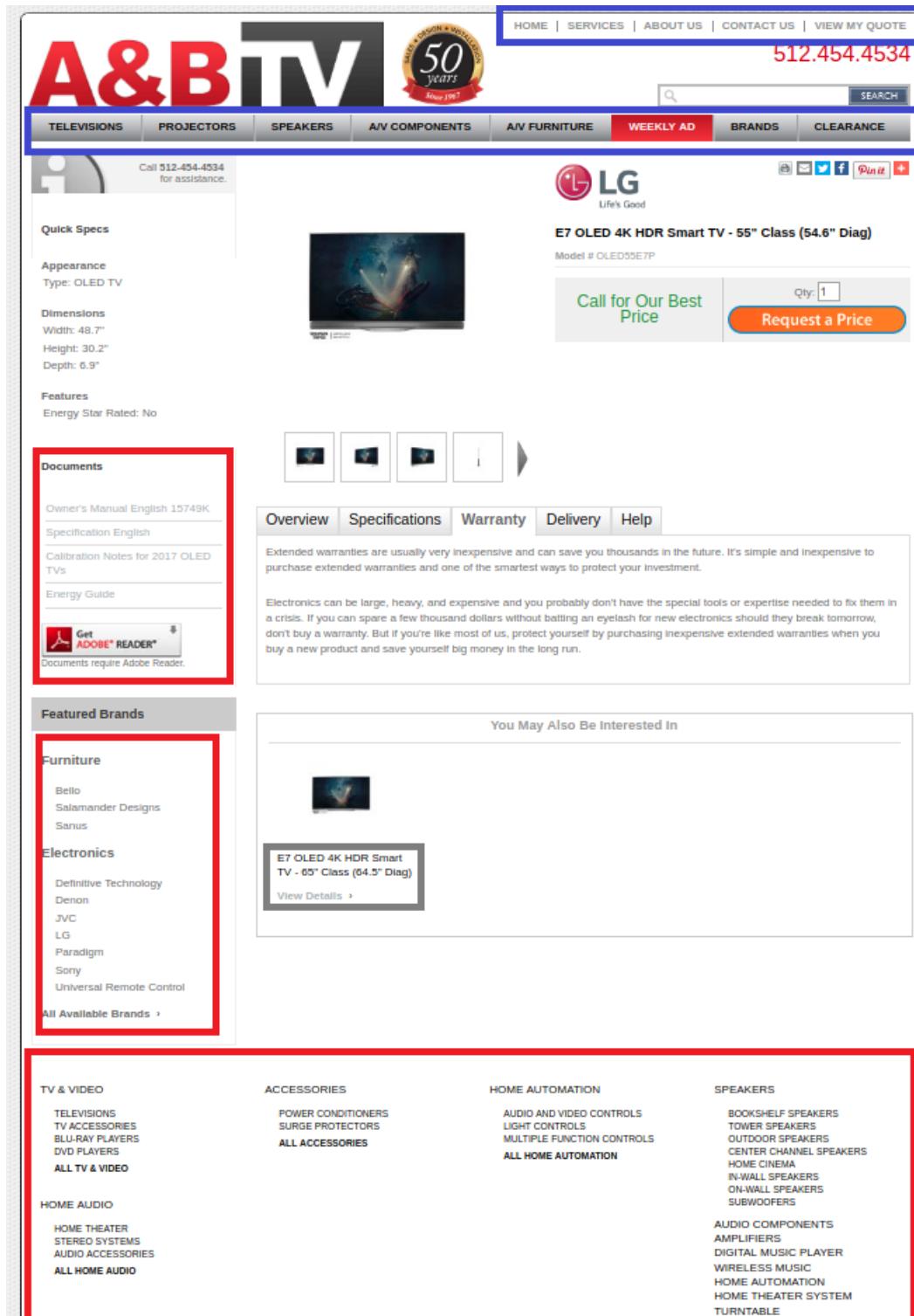


Figura 3.15: Cammini di sito, di Page Class e singola pagina nella pagina di un prodotto

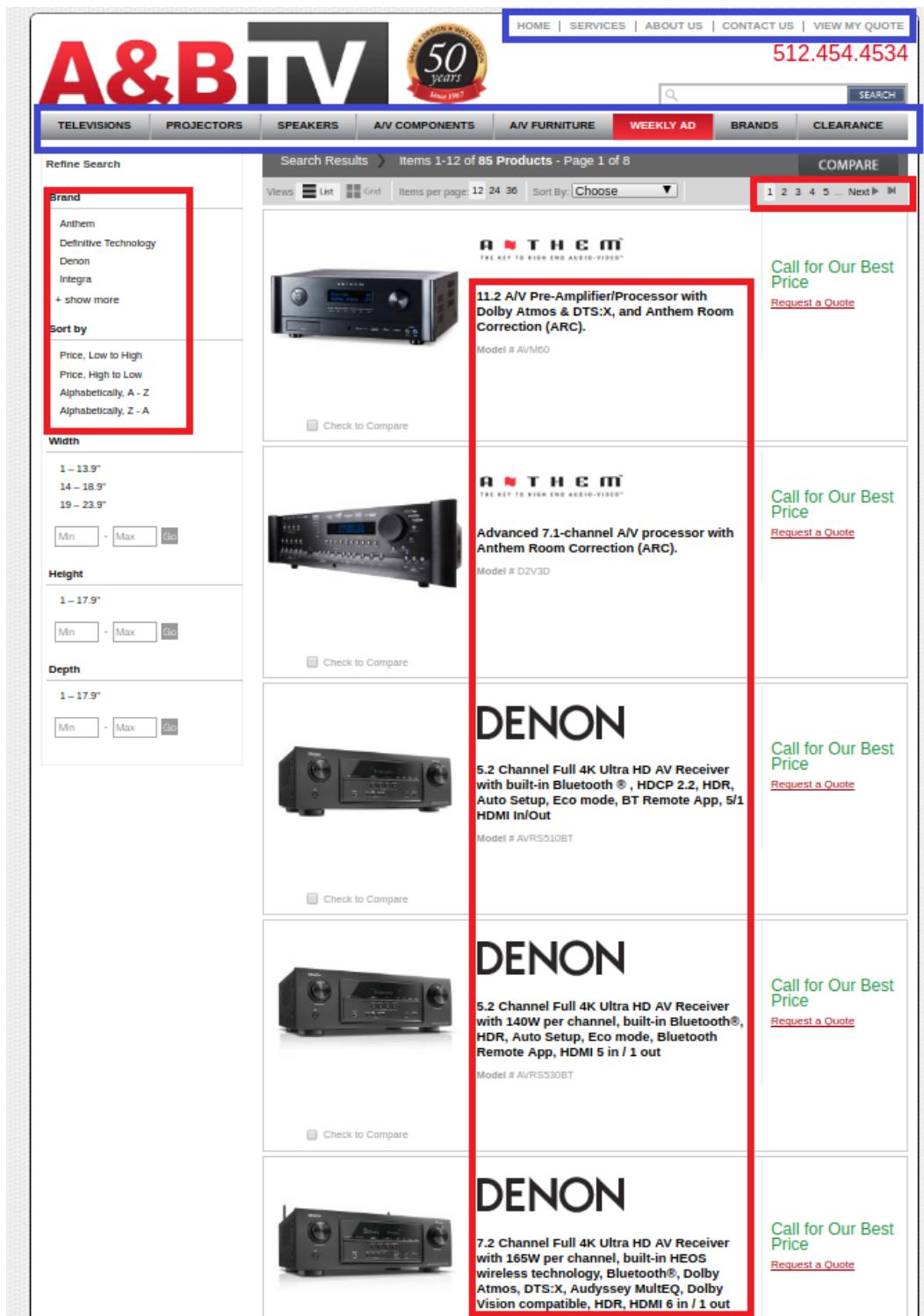


Figura 3.16: Cammini di sito e di Page Class nella pagina di una lista di prodotti

In base a queste considerazioni, è possibile attribuire un peso a ciascun cammino appartenente allo schema di una Page Class. Il calcolo del peso è basato sulla funzione *Tf-Idf*, usata nell'area del recupero delle informazioni per misurare l'importanza di un termine rispetto ad un documento o ad una collezione di documenti⁷.

Dato un cammino cp appartenente allo schema S di una Page Class C del modello M e detto P l'insieme delle pagine di M :

$$tf(cp, C) = \frac{|\{p \mid p \in C, cp \in p\}|}{|C|} \quad (3.2)$$

$$idf(cp) = \ln \left(\frac{|P|}{|\{p \mid p \in P, cp \in p\}|} \right) \quad (3.3)$$

$$Tf-Idf(cp, C) = tf(cp, C) \times idf(cp) \quad (3.4)$$

$$weight(cp, C) = \frac{1}{Tf-Idf(cp, C) + 1} \quad (3.5)$$

Intuitivamente, l'indice *tf* (frequenza) di un cammino è tanto più alto quanto maggiore è la sua presenza negli schemi delle pagine di una Page Class. L'indice *idf* (inverso della frequenza) è invece tanto più alto quanto minore è la sua presenza negli schemi delle pagine del modello intero. Di conseguenza, l'indice *Tf-Idf* è più alto per quei cammini che sono molto frequenti all'interno di una sola Page Class (cammini di Page Class), mentre è più basso per i cammini molto frequenti in tutte le Page Class (cammini di sito) o in poche pagine di una Page Class (cammini di singola pagina).

Il peso per la codifica del cammino cp di una Page Class C (mostrato nell'equazione 3.5) è allora tanto più basso quanto è più alto il suo $Tf-Idf(cp, C)$, poiché un *Tf-Idf* alto indica un cammino proprio della Page Class. Nelle pagine delle figure 3.14 e 3.15 ad esempio, i cammini delle link e text collection in rosso avranno un peso basso per quella Page Class.

⁷<https://nlp.stanford.edu/IR-book/html/htmledition/tf-idf-weighting-1.html>

3.3.4.2 Codifica del modello

Nel contesto di questo algoritmo, la codifica del modello avviene attraverso la codifica dei cammini degli schemi di ciascuna Page Class. La codifica dei dati avviene attraverso la codifica delle pagine visitate dall'algoritmo. In questo contesto, il principio di MLD viene usato per quantificare quanto bene si adatta un insieme di pagine web nel modello.

Si consideri ora la parte (1) della codifica, che rappresenta il costo del modello. Dal momento che il modello può essere descritto dall'insieme degli schemi delle sue Page Class, il suo costo può essere definito come il costo della concatenazione pesata dei cammini appartenenti allo schema di ciascuna classe.

Detto M il modello, C_i l'i-esima Page Class di M , cp_j il j-esimo cammino dello schema di C_i :

$$enc(M) = \sum_{i=0}^n enc(C_i) \quad (3.6)$$

$$enc(C) = \sum_{j=0}^m weight(cp_j, C) \quad (3.7)$$

Si prenda come esempio un modello M composto da due Page Class, tale che $M = \{C_A, C_B\}$, assumendo che: (i) C_A ha uno schema con tre cammini $\{cp_1, cp_2, cp_3\}$ e C_B ha uno schema con un cammino $\{cp_4\}$ e (ii) le Page Class contengono le seguenti pagine: $C_A = \{p_1\}$ e $C_B = \{p_2, p_3\}$. Allora $enc(M)$ è uguale a $weight(cp_1, C_A) + weight(cp_2, C_A) + weight(cp_3, C_A) + weight(cp_4, C_B)$.

La parte (2) della codifica rappresenta il costo dell'insieme delle pagine visitate in base alla Page Class alle quali appartengono. Dal momento che una pagina è descritta da un'insieme di text e link collection, il calcolo della codifica di ciascuna pagina p nella Page Class C avviene nel seguente modo: per ogni cammino nello schema di p viene considerato il suo peso rispetto a C , moltiplicato per una costante c_{index} (intuitivamente, se un cammino di p è già stato codificato nello schema C , si considera il costo della codifica del suo solo indice). Per ogni cammino in C non presente in p viene considerato il suo peso rispetto a C , moltiplicato per una costante c_{miss} (questa costante è maggiore

di c_{index} , perché in questo caso il cammino intero deve essere codificato)⁸. Si considera infine il costo della codifica di ciascun URL presente in p .

Detto M il modello, D l'insieme di pagine visitate, p_j la j-esima pagina della i-esima Page Class C_i , cp_k il k-esimo cammino dello schema di p_j :

$$enc(D|M) = \sum_{i=0}^n \sum_{j=0}^m enc(p_j|C_i) \quad (3.8)$$

$$\begin{aligned} enc(p|C) = & |\{url \mid url \in p\}| + \sum_{k=0}^n c_{\text{index}} \times weight(cp_k, C), cp_k \in p, cp_k \in C \\ & + \sum_{k=0}^m c_{\text{miss}} \times weight(cp_k, C), cp_k \notin p, cp_k \in C \end{aligned} \quad (3.9)$$

Riprendendo il modello M dell'esempio precedente, si supponga che la pagina p_1 sia descritta nel seguente modo:

$$p = \{cp_1(url_{11}, url_{12}), cp_3(url_{31}, url_{32}, url_{33})\}.$$

Allora il costo della sua codifica rispetto alla Page Class C_A è:

$$enc(p_1, C_A) = 5 + c_{\text{index}} \times weight(cp_1, C_A) + c_{\text{index}} \times weight(cp_3, C_A) + c_{\text{miss}} \times weight(cp_2, C_A)$$

Per concludere, la MLD del modello M è data da:

$$MLD(M) = enc(M) + enc(D|M) \quad (3.10)$$

Per ogni passo dell'algoritmo, quindi, la fase di aggiornamento del modello confronta ciascuna classe candidata C' con le classi $\{C_1, \dots, C_n\}$ del modello, calcolando il costo della MLD per ciascun modello ottenuto accorpando C' ad ogni C_i . Il modello di costo minimo viene quindi confrontato con quello composto dalle classi $C' \cup \{C_1, \dots, C_n\}$. Tra queste due versioni viene quindi scelta quella con costo più basso.

⁸Negli esperimenti sono stati usati $c_{\text{miss}} = 1$, $c_{\text{index}} = 0,8$

3.3.5 Euristiche di navigazione

La coda in cui vengono inserite le link collection da esplorare viene ordinata utilizzando euristiche volte a incrementare la qualità del modello prodotto con il minor numero di pagine possibili. Il primo obiettivo è quello di condurre il crawler verso regioni inesplorate del sito, al fine di creare nuove classi e collegamenti nel grafo di navigazione; il secondo obiettivo è quello di migliorare l'affidabilità delle classi prodotte visitando un alto numero di pagine che appartengono a classi già identificate.

I due obiettivi sono quindi in contrasto: infatti il primo permette di avere un modello più completo, mentre il secondo permette di avere un modello più affidabile. Questi due obiettivi si traducono in due strategie di navigazione differenti. La prima, chiamata *navigazione densa*, assegna priorità maggiore alle link collection che contengono molti link rispetto al numero totale di link nella classe di appartenenza. Questo perché le link collection più grandi hanno maggiore probabilità di portare a liste di pagine strutturalmente simili e che contengono informazioni rilevanti per il dominio di interesse del sito web. La seconda, chiamata *navigazione sparsa*, assegna priorità maggiore alle link collection più piccole, affinché il crawler possa raggiungere per prime zone inesplorate del sito.

Capitolo 4

Architettura del crawler

Questo capitolo descrive nello specifico l'architettura del crawler realizzato per supportare la generazione dei modelli descritti nel capitolo 3 e per la successiva fase di crawling intensivo di siti web guidato dalla struttura.

Nella prima sezione viene descritta l'architettura del crawler distribuito usato per effettuare il crawling intensivo di siti web: il crawler prende in input una specifica di navigazione, descritta sotto forma di modello del sito web (si confronti la sezione 3.2.3), e fornisce in output le pagine organizzate per Page Class. Nella seconda sezione viene descritta l'architettura della fase preliminare di crawling, specializzata nella generazione scalabile di modelli di siti web, secondo le metodologie descritte nel capitolo 3.

4.1 Crawler distribuito

4.1.1 Requisiti

Come descritto nel capitolo 1, un crawler generico effettua una visita in ampiezza o profondità del Web non associando alcuna informazione strutturale agli URL scoperti, se non il dominio di appartenenza. Il crawler realizzato in questo lavoro effettua un crawling focalizzato (cfr. 1.2.1.2) ed è guidato dalla struttura dei siti web presi in analisi: il crawling rimane quindi confinato ai siti web dati in input, ma tiene conto della Page Class nel modello del sito di appartenenza di ciascun URL per associare informazioni strutturali a ciascuna pagina web scaricata.

Requisiti funzionali Il crawler deve rispettare i seguenti requisiti funzionali:

- Data una specifica di crawling, sotto forma di modelli di siti web, il crawler deve navigare i siti attenendosi al modello di navigazione del sito.
- Il crawler deve poter salvare, per ciascun URL scaricato, la rispettiva pagina HTML e associare al file il sito e il nome della Page Class di appartenenza.
- Il crawler deve poter estrarre dei dati dalle pagine, utilizzando wrapper specificati in input e associati a determinate Page Class.
- Il crawler deve poter compiere azioni sulle pagine di determinate Page Class, come ad esempio il riempimento automatico di form, secondo regole date in input, per mostrare elementi nascosti nella stessa pagina.
- Il crawler deve poter navigare anche tramite il riempimento delle form: deve essere possibile specificare un Class Link tra due Page Class in base al completamento di una form, piuttosto che in base a un cammino. Ad esempio, data l'homepage di un motore di ricerca, il crawler deve poter raggiungere una pagina che mostra risultati di ricerca inserendo un testo, dato in input, nella barra di ricerca.
- Il crawling di un sito web deve poter essere interrotto in qualsiasi momento, permettendo altresì di riprendere il crawling in un secondo momento ripristinando lo stato della visita al momento dell'interruzione, senza necessità di ricominciare dall'inizio.

Requisiti di qualità Il crawler deve rispettare i seguenti requisiti di qualità:

- Il crawler deve poter scalare sul numero di siti web dati in input, distribuendo il crawling di diversi siti web su diversi host.
- Il crawler deve poter scalare sul numero di pagine di un singolo sito web, distribuendo il crawling di un sito su diversi host. Da una parte l'obiettivo è quello di guadagnare spazio e tempo, distribuendo sia le risorse di calcolo impiegate dai thread dei singoli host per un'applicazione I/O bound, sia i dati scaricati (le pagine

HTML). Dall'altra, si vuole evitare che il crawler venga bloccato dal server web a causa di un eccessivo numero di richieste provenienti da uno stesso indirizzo IP.

- L'utilizzo della memoria principale deve essere costante rispetto al numero di pagine raggiunte dal crawler: il crawler dovrebbe salvare su file le pagine raggiunte (caching) e ripristinarne il contenuto solo nel momento e per il tempo necessario.
- L'utilizzo della memoria deve essere costante rispetto alla dimensione della frontiera: il crawler deve persistere in memoria secondaria una parte degli URL nella frontiera, dando una dimensione massima fissa agli URL presenti in memoria principale.

4.1.2 Diagramma delle classi di dominio

Viene ora presentata la modellazione di dominio progettata per rappresentare i concetti fondamentali della fase di crawling intensivo, e rispettarne i requisiti funzionali.

4.1.2.1 Visione

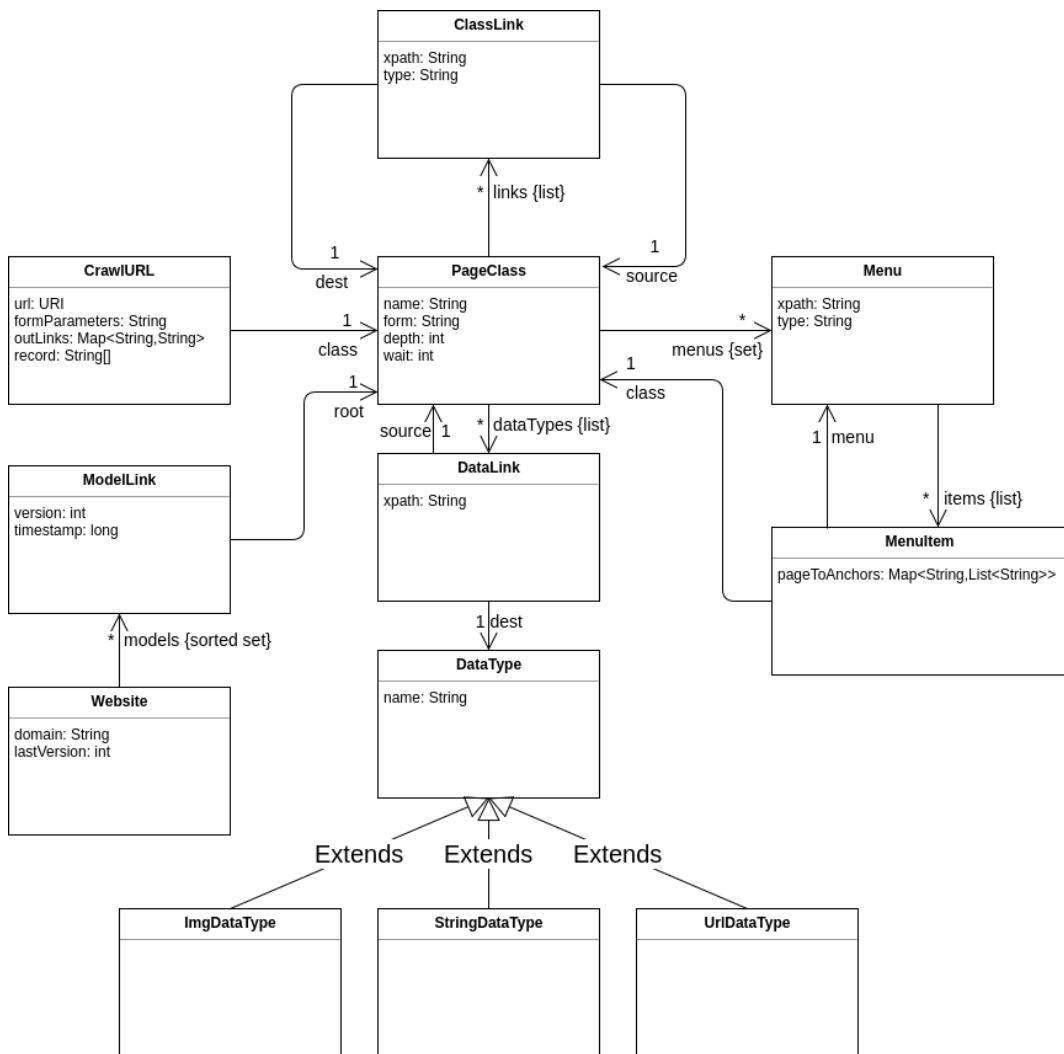


Figura 4.1: Diagramma delle classi di dominio per la fase di crawling intensivo

La figura 4.1 mostra il diagramma delle classi di dominio per la fase di crawling intensivo. La modellazione è pensata per permettere al crawler di associare a ciascun URL scaricato una Page Class che presenta determinate caratteristiche omogenee in termini di collegamenti di navigazione, con rispettiva tipologia, e di estrazione dei dati di interesse.

4.1.2.2 Classi

CrawlURL Un CrawlURL rappresenta lo stato della visita di un URL. Ciascun CrawlURL è associato a una Page Class, stabilita in base alla XPath del Class Link che ha portato all'estrazione di quell'URL. Ad esempio, data una pagina p della Page Class C_1 e un ClassLink cl tra C_1 e C_2 descritto da una XPath xp , gli URL estratti da p applicando la XPath xp di cl sono trasformati in oggetti CrawlURL associati alla Page Class C_2 .

Durante il crawling, un CrawlURL ha diversi stati:

- **Non scaricato:** quando si trova in coda nella Frontier
- **Salvato:** quando è stato scaricato e la pagina HTML associata è stata salvata su disco
- **Estratto:** quando dalla sua pagina sono stati estratti i suoi link uscenti ed eventuali dati estratti applicando un wrapper sulla pagina. In particolare, le informazioni estratte dal CrawlURL vengono memorizzate dal CrawlURL stesso fino a quando il suo processamento non è completo.
- **Esplorato:** quando l'esplorazione è stata completata e il CrawlURL non è più di interesse per il crawler.

PageClass La classe PageClass è la rappresentazione progettuale dell'omonimo concetto espresso nella sezione 3.2.2. Rappresenta un nodo di navigazione del grafo del sito web e comprende metodi che consentono l'esplorazione del sito tramite la navigazione dei ClassLink e delle form, così come l'estrazione di dati tramite l'utilizzo di wrapper da applicare sulle pagine degli URL che fanno parte del nodo. Durante il crawling, l'istanza di PageClass associata a ciascun URL viene quindi usata per indirizzare la

navigazione del crawler verso determinate aree del sito.

Ad una PageClass vengono inoltre attribuiti diverse variabili d'istanza:

- **depth**: profondità della PageClass in base alla sua distanza minima dalla PageClass di partenza (quella di cui fa parte l'homepage del sito).
- **wait**: un tempo di attesa che il crawler deve rispettare tra due esplorazioni di CrawlURL appartenenti alla stessa PageClass.
- **form**: rappresenta una XPath che, applicata alle pagine di questa PageClass, identifica un nodo nel DOM contenente una form. Oltre alla XPath, la stringa contiene anche i parametri da inserire nella form per completarla. Questo parametro identifica quindi un'azione da compiere sulle pagine di questa PageClass, al fine ad esempio di mostrare al crawler dei dati altrimenti nascosti.

ClassLink La classe ClassLink rappresenta un collegamento tra due PageClass; è la rappresentazione progettuale dell'omonimo concetto espresso nella sezione 3.2.2.2. Un ClassLink è identificato dalle PageClass di partenza e destinazione, insieme alla XPath di navigazione usata dal crawler per passare da una classe all'altra. I ClassLink definiscono anche il tipo del collegamento; si noti che questa classe definisce i collegamenti di tipo lista e link singolo, mentre i collegamenti di tipo menu vengono delegati alla classe Menu. Inoltre un ClassLink può essere anche di tipo form, per indicare che la navigazione verso la PageClass destinazione avviene tramite la compilazione dei parametri (specificati nella XPath di navigazione) di una form nelle pagine.

Menu e MenuItem La classe Menu rappresenta un menu di navigazione di un sito web. In particolare, rappresenta il collegamento tra una PageClass di partenza e una lista di PageClass destinazioni differenti. Un menu è infatti definito come una lista di link, all'interno di una pagina web, che portano ad aree del sito strutturalmente diverse tra loro. In fase di modellazione un menu è associato ad una PageClass e contiene una lista di MenuItem che conducono a loro volta a una PageClass ciascuno, insieme alle ancore dei link che fanno riferimento a quella PageClass di destinazione.

In particolare, un Menu contiene le seguenti variabili d'istanza:

- **xpath**: identifica la XPath che porta al menu nelle pagine associate alla PageClass di partenza.
- **type**: rappresenta il tipo di menu. In particolare, un menu che presenta sempre gli stessi link negli elementi della lista è detto *ad ancora fissa*, mentre un menu che presenta sempre link diversi in ogni pagina in cui compare è detto *ad ancora variabile*.

DataType e DataLink La classe DataType rappresenta un wrapper utile ad estrarre un record di dati dalle pagine appartenenti alla PageClass a cui il DataType è associato. In particolare un DataType identifica un nodo nel DOM delle pagine di una PageClass, raggiungibile tramite una XPath definita nella classe DataLink, che mette in collegamento una PageClass con il tipo di dato da estrarre. Le XPath dei DataLink si differenziano da quelle dei ClassLink nel fatto che non sono utilizzate per navigare il sito web ma per estrarre dati da esso.

DataType è una classe astratta: mentre il DataLink identifica la XPath usata per raggiungere il nodo di interesse nel DOM, le implementazioni concrete che estendono DataType specificano le trasformazioni da eseguire sul nodo selezionato per estrarre il dato di interesse. Il diagramma di figura 4.1 mostra implementazioni concrete per l'estrazione di immagini, stringhe di testo e URL dal DOM. Si noti che altre implementazioni possono essere aggiunte al modello senza necessità di modificare l'architettura del crawler.

Un DataType è inoltre identificato da un nome, che rappresenta la semantica del dato estratto. Si prenda come esempio l'estrazione di un prezzo da pagine appartenenti a una PageClass di dettagli di un prodotto. Associando il nome *Prezzo* al DataType, tutti i prezzi estratti dai DOM avranno questa etichetta associata.

Website e ModelLink La classe Website rappresenta un contenitore di modelli per uno stesso sito web. Il crawler permette infatti di archiviare diverse versioni di un modello di sito web, generate in periodi diversi, al fine di studiarne i cambiamenti strutturali. La classe ModelLink rappresenta la versione di un modello e contiene informazioni riguardanti l'identificativo (un numero progressivo) e il timestamp del

momento in cui la versione del modello è stata generata. Ciascun ModelLink mantiene un’associazione con la PageClass di partenza della versione (la classe contenente la homepage), usata per iniziare il crawling del sito.

4.1.3 Modello ad attori

In questa sezione viene illustrata l’architettura del crawler vero e proprio, mostrando in particolare il funzionamento dei moduli che permettono il rispetto dei requisiti funzionali e di qualità elencati nella sezione 4.1.1.

In particolare il crawler è progettato tramite l’uso di Akka [18], un toolkit allo stato dell’arte per lo sviluppo di applicazioni concorrenti, altamente distribuite e tolleranti ai guasti, basato sulla teoria del modello ad attori. Questo approccio prevede l’utilizzo di *attori* come primitive universali per il calcolo concorrente. Ogni attore infatti possiede un proprio flusso di esecuzione con accesso esclusivo al suo stato, e comunica con gli altri attori scambiando *messaggi*: questi vengono inviati in maniera non bloccante e ricevuti in un buffer locale di processamento da cui l’attore destinatario effettua un’estrazione bloccante. Il modello è quindi descrivibile come un ciclo di processamento asincrono di messaggi da parte di diversi attori.

4.1.3.1 Visione

L’algoritmo principale di crawling prevede i seguenti passi in un ciclo iterativo:

- Scelta di un CrawlURL dalla Frontiera, sulla base della PageClass di appartenenza.
- Scaricamento della pagina HTML associata al CrawlURL tramite una richiesta HTTP al server relativo.
- Salvataggio della pagina HTML su disco.
- Estrazione dei link uscenti tramite l’applicazione delle XPath di navigazione della PageClass sul DOM della pagina.
- Attribuzione della relativa PageClass ad ogni link estratto.

- Estrazione dei dati dalla pagina tramite l'applicazione delle XPath di estrazione dei DataType associati alla PageClass.
- Salvataggio dei dati estratti e delle informazioni relative al CrawlURL completato.
- Inserimento in Frontiera dei nuovi URL.

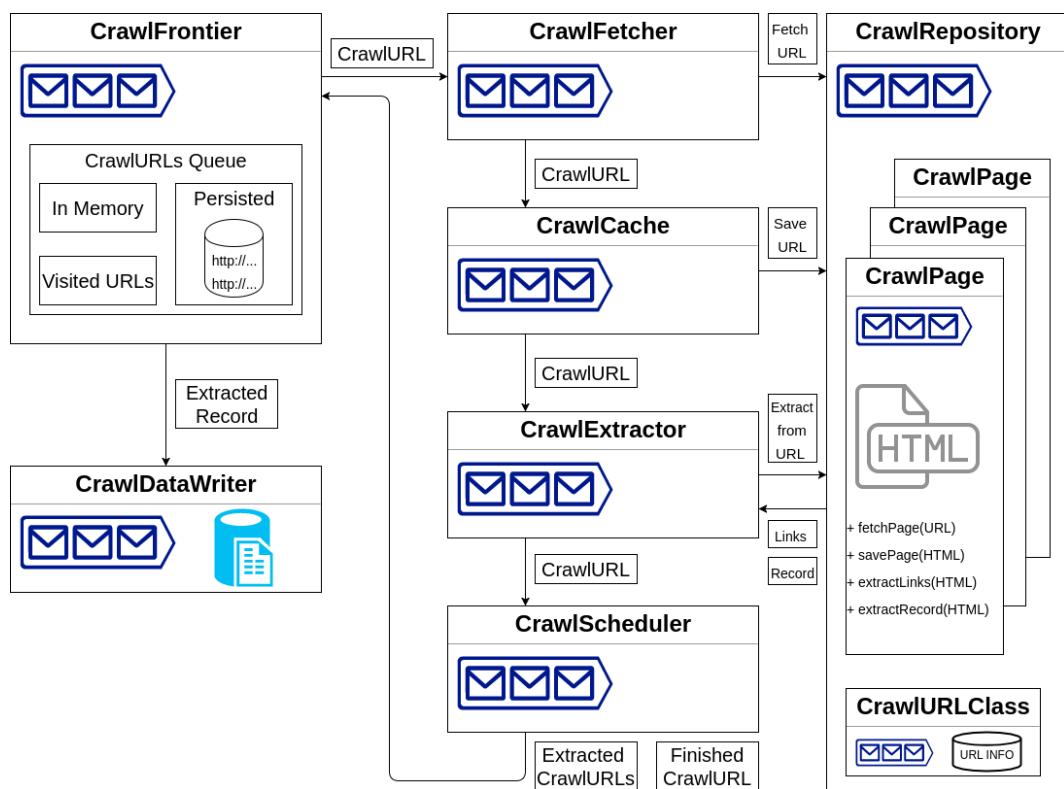


Figura 4.2: Architettura ad attori del crawler

La figura 4.2 mostra in dettaglio la realizzazione nel modello ad attori dell'algoritmo di crawling descritto sopra. Ciascun componente del crawler è un attore che esegue richieste ricevute tramite l'invio asincrono di messaggi ed invia risposte e richieste a sua volta tramite messaggi. In particolare, gli attori nella parte centrale dello schema processano in maniera sequenziale un CrawlURL, aggiornando ciascuno lo stato della sua visita. Le azioni vere e proprie sulla pagina, come l'invio delle richieste HTTP al server e l'applicazione delle XPath sul DOM, vengono effettuate invece da attori responsabili

di una singola pagina noti a **CrawlRepository**, un controller che ha lo scopo di ricevere richieste di azioni sugli URL e di inviarle ai relativi attori. La Frontiera, nella parte sinistra dello schema, si occupa invece di mantenere in coda i nuovi URL scoperti ma non ancora visitati, e delega ad un attore **CrawlDataWriter** il salvataggio dei dati estratti dai CrawlURL le cui visite sono state completate.

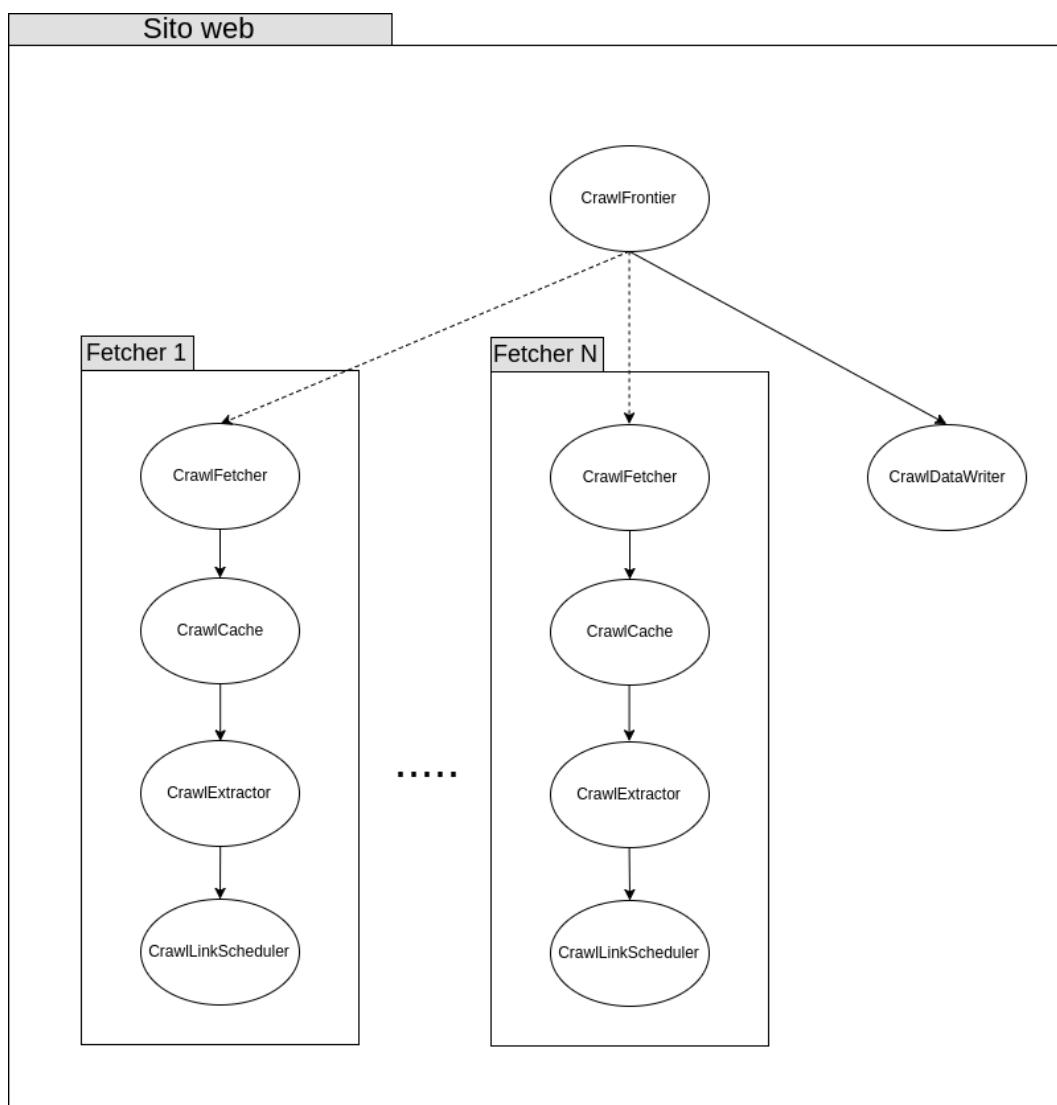


Figura 4.3: Gerarchia degli attori relativi a un singolo sito

In figura 4.3 viene mostrata la gerarchia degli attori responsabili di aggiornare lo stato della visita dei CrawlURL. Si noti che lo schema rappresenta due diversi livelli di distribuzione dei componenti software: una **CrawlFrontier**, insieme ai suoi attori figli, gestisce lo stato della visita di un singolo sito web. Il crawling di diversi siti web porta alla generazione di una nuova *istanza* di gerarchia per ognuno di essi. Ogni gerarchia di sito web può essere creata su diversi host specificati in un file di configurazione. Il secondo livello di distribuzione è nella catena di processamento dei CrawlURL: come si può notare in figura, la CrawlFrontier crea una catena di processamento di CrawlURL per ogni fetcher che si intende utilizzare nel crawling. Ad esempio, se si vuole eseguire il crawling di un sito web con 4 differenti host, per evitare di essere bloccati dal server web a causa dell'eccessivo numero di richieste, il crawler crea 4 catene di processamento. Si noti che alla CrawlFrontier, singola per ogni sito, è delegata la responsabilità di coordinare le diverse catene e restituire a ciascuna di esse il prossimo CrawlURL da processare. Si noti inoltre che il **CrawlDataWriter**, responsabile del salvataggio degli eventuali record estratti dai CrawlURL, è unico e vive nello stesso host della CrawlFrontier: questo permette al sistema di riconciliare i dati estratti da host distribuiti e salvarli in un'unica macchina al fine di potervi accedere da un file system unico.

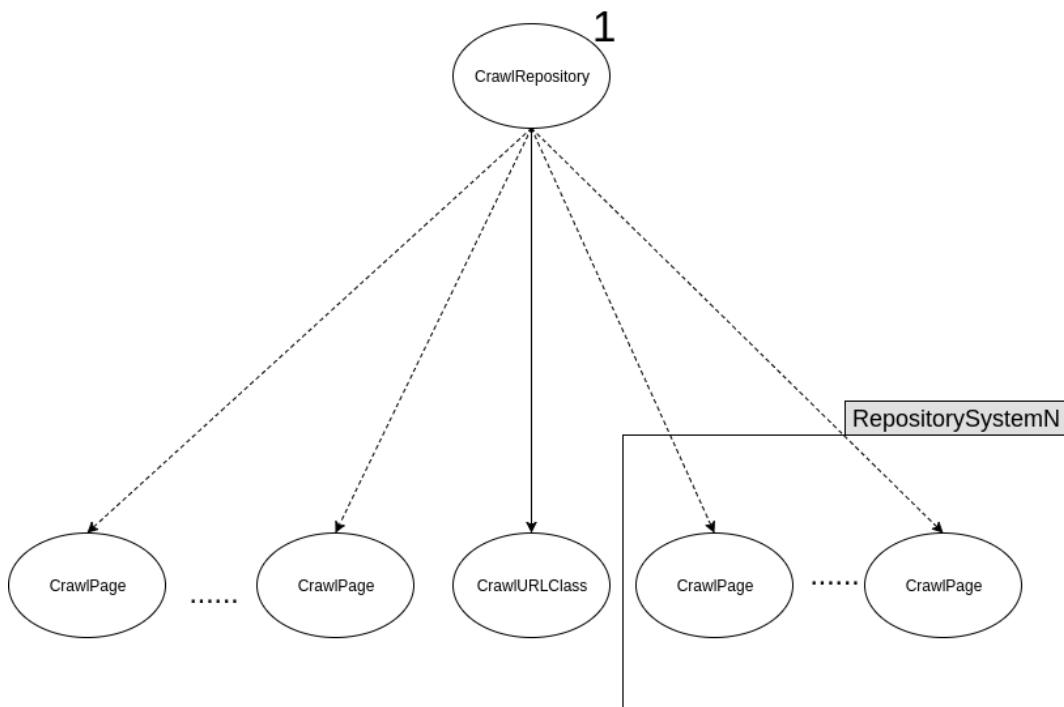


Figura 4.4: Gerarchia degli attori distribuiti responsabili del fetching delle pagine

In figura 4.4 viene mostrata la gerarchia degli attori responsabili di effettuare concreteamente le azioni richieste sulle pagine web scaricate. Il **CrawlRepository** è una singleton che crea e mantiene in memoria un attore **CrawlPage** per ogni pagina scaricata. In particolare, le **CrawlPage** vengono distribuite equamente sugli host specificati, al fine di garantire che il fetching (scaricamento) delle pagine avvenga in maniera distribuita. Infine, un attore singleton **CrawlURLClass** scrive sul file system dell'host master (specificato in un file di configurazione) le informazioni relative ai **CrawlURL** completati.

4.1.3.2 Attori

CrawlFetcher Il CrawlFetcher è l'attore responsabile di richiedere i CrawlURL alla CrawlFrontier. Dopo aver ricevuto il CrawlURL, il fetcher invia al CrawlRepository la richiesta di fetching vera e propria, inviando tramite messaggio asincrono l'URL di interesse ed eventuali parametri, se il modello di crawling prevede il completamento di una form per quell'URL. Il CrawlFetcher viene notificato dal CrawlRepository quando la richiesta è stata completata: se il fetching è andato a buon fine, lo stato del CrawlURL viene aggiornato ed è inviato al CrawlCache per il successivo processamento. Se il fetching della pagina ha generato un errore, il CrawlFetcher ritenta la richiesta di fetching per un numero di volte specificato in input in un file di configurazione. Se non è stato possibile scaricare il CrawlURL, il CrawlFetcher richiede alla frontiera il prossimo URL.

CrawlCache Il CrawlCache è l'attore responsabile di richiedere il salvataggio su disco dei CrawlURL ricevuti nel buffer dei messaggi. Questo attore invia la relativa richiesta al CrawlRepository, che la inoltra alla CrawlPage di competenza. Se il salvataggio è andato a buon fine, il CrawlCache aggiorna lo stato del CrawlURL e lo inoltra al CrawlExtractor, altrimenti richiede la distruzione della CrawlPage associata e rimane in attesa del prossimo URL.

CrawlExtractor Il CrawlExtractor è l'attore responsabile di richiedere l'estrazione dei link uscenti e dei dati dal CrawlURL ricevuto nel buffer dei messaggi. L'attore invia la richiesta al CrawlRepository, che la inoltra alla CrawlPage di competenza. L'attore viene notificato quando l'estrazione è stata completata, ricevendo le informazioni estratte. Il CrawlExtractor aggiorna quindi lo stato del CrawlURL e lo invia al CrawlScheduler, rimanendo poi in attesa del prossimo URL.

CrawlScheduler Il CrawlScheduler è l'attore responsabile di inviare alla CrawlFrontier i link estratti dal CrawlURL ricevuto, insieme al CrawlURL stesso. L'attore informa inoltre il CrawlRepository che la visita del CrawlURL è terminata, affinché la collegata CrawlPage venga distrutta. L'attore rimane quindi in attesa del prossimo URL.

CrawlRepository Il CrawlRepository è un attore singleton dell’intero sistema, ed è responsabile della creazione delle CrawlPage, che sono gli attori responsabili di eseguire azioni su un determinato URL. Il CrawlRepository, in qualità di controller, riceve richieste dagli attori della catena di processamento dei CrawlURL, creando gli attori responsabili CrawlPage ed inoltrando loro le richieste ricevute. È inoltre responsabile di distruggere le CrawlPage quando richiesto dagli attori della catena.

CrawlPage L’attore CrawlPage è responsabile di eseguire tutte le azioni richieste su un singolo URL, corrispondente a una pagina HTML. Contiene la logica delle seguenti operazioni:

- **fetchPage:** dato un URL, l’attore effettua una richiesta HTTP al server web del sito e scarica la pagina. La richiesta HTTP viene delegata ad un client web creato con **HtmlUnit** [11], un browser GUI-less per Java.

Se insieme all’URL l’attore riceve anche dei parametri (coppie nome valore), questi vengono inclusi nella richiesta, per consentire la navigazione nel sito verso un URL di cui non si conosce l’indirizzo. Infatti in questo caso verrà inviata una richiesta HTTP composta da un URL e da una lista di parametri, e la risposta del server produrrà una pagina con un nuovo URL.

Se insieme all’URL viene inviata una XPath indicante una form e dei dati da inserire, l’attore effettua prima una richiesta HTTP per scaricare l’URL, e poi applica questa XPath per completare una form nella pagina scaricata.

La pagina finale scaricata viene salvata in memoria.

- **savePage:** l’attore effettua il salvataggio su disco della pagina in memoria, comprensivo di tutte le immagini presenti.
- **extractLinks:** l’attore applica le XPath di navigazione per estrarre i link uscenti dalla pagina e le invia all’attore che ha fatto la richiesta. Nel caso di XPath che indicano una form e i dati per completarla, al posto di un link uscente (che non è presente) viene restituito l’URL della pagina attuale seguito dai dati da inviare al server: in questo modo la prossima richiesta di fetching, composta da URL e dati, condurrà alla pagina successiva.

- **extractRecord:** l'attore applica le XPath di estrazione sulla pagina, estraе i dati di interesse mediante i wrapper specificati nei DataType associati alla PageClass della pagina e li invia all'attore che ha fatto la richiesta.

CrawlFrontier L'attore CrawlFrontier rappresenta la frontiera degli URL di un singolo sito web. La sua responsabilità è quella di indirizzare la navigazione del crawler, facendo rispettare le soglie di attesa tra richieste consecutive definite nelle PageClass, e garantendo che non vi siano cicli nella navigazione (ogni URL deve essere scaricato una volta sola). La CrawlFrontier decide inoltre la strategia di navigazione del sito.

Di seguito vengono elencate le principali funzioni e caratteristiche della frontiera:

- **Coda:** la coda degli URL della frontiera è divisa in due parti. Una parte degli URL, composta dai più rilevanti secondo la strategia di navigazione, viene tenuta in una coda in memoria principale di dimensione fissa e configurabile. Quando la coda in memoria è piena, i CrawlURL meno rilevanti vengono rimossi dalla coda e inseriti in una coda persistita. La coda persistita¹ mantiene le informazioni sull'URL² e sulla PageClass di appartenenza.
- **URL visitati:** per evitare che il crawler visiti un URL già scaricato, la frontiera mantiene in memoria una hash table degli URL visitati. Se un nuovo URL ricevuto dal CrawlScheduler è già presente nella hash table viene scartato, altrimenti viene inserito in coda e nella hash table. Per risparmiare lo spazio occupato in memoria, la hash table computa e memorizza una checksum del percorso relativo dell'URL piuttosto che l'URL completo.
- **Strategia di navigazione:** la frontiera ordina i CrawlURL da scaricare secondo una coda di priorità basata sulle proprietà delle PageClass associate agli URL. In particolare, i CrawlURL sono ordinati in base alla distanza della loro PageClass dalla PageClass radice. Minore è la distanza dalla PageClass radice (quella di cui fa parte l'homepage), maggiore è la loro priorità. Questa strategia viene usata considerando come euristica il fatto che le pagine delle classi più vicine

¹La coda persistita è implementata come un semplice file CSV.

²Dal momento che la coda contiene gli URL di un solo dominio, la coda persistita memorizza solo il percorso relativo dell'URL.

all'entrypoint del sito hanno maggiori probabilità di contenere informazioni di maggiore interesse per il dominio del sito in questione.

- **Ripristino della frontiera:** la frontiera è in grado di salvare e ripristinare lo stato della sua coda, al fine di supportare lo scenario in cui un crawling viene interrotto e ripreso in un secondo momento, ripartendo dall'ultimo punto raggiunto. Questo viene garantito mediante l'utilizzo del pattern CQRS (Command Query Responsibility Segregation)³: quando la frontiera compie un'azione su un URL (inserimento in coda, rimozione dalla coda), questa viene memorizzata in un database NoSQL. In questo modo, ogni volta che la CrawlFrontier, identificata da un id, viene ricreata dal sistema, vengono anche riapplicate tutte le azioni di inserimento e rimozione effettuate dalla frontiera. In seguito a questa fase iniziale di ripristino lo stato corrente della frontiera è ripristinato e il crawling può ricominciare da dove è stato interrotto.

CrawlDataWriter Il CrawlDataWriter è l'attore responsabile di salvare su file i dati estratti dai CrawlURL processati dal crawler. Quando la visita di un CrawlURL è stata completata, la frontiera invia a questo attore il record contenente i dati estratti da un URL, che li salva in formato CSV. In particolare, il CrawlDataWriter crea un file per ogni PageClass del sito, al fine di raggruppare insieme dati appartenenti alle pagine con la stessa struttura.

CrawlURLClass L'attore CrawlURLClass, singleton nel sistema, è responsabile di salvare le informazioni riguardanti ogni URL scaricato. In particolare, per ogni sito web, crea un CSV di triple (URL, PageClass, File) al fine di tenere traccia dello schema intenzionale del modello di navigazione, potendo così attribuire a ciascuna pagina scaricata la classe di appartenenza.

³<https://martinfowler.com/bliki/CQRS.html>

4.2 Generazione scalabile dei modelli

4.2.1 Requisiti

La generazione dei modelli di siti web segue le indicazioni metodologiche e algoritmiche descritte nel capitolo 3. I requisiti principali di questa fase di crawling preliminare prevedono che i modelli di siti web siano generabili in maniera concorrente e distribuita: deve essere possibile effettuare il crawling di modellazione di diversi siti web in maniera parallela, permettendo anche la distribuzione dei crawler che eseguono l'algoritmo su diversi host.

4.2.2 Diagramma delle classi di dominio

Si noti che l'architettura a supporto di questa fase è diversa da quella della fase di crawling intensivo, ed è complementare ad essa. Per questo motivo la modellazione di dominio presenta concetti confinati alla fase di generazione dei modelli, i cui output vengono poi utilizzati nella fase di crawling intensivo.

4.2.2.1 Visione

La figura 4.5 mostra il diagramma delle classi di dominio per la fase di generazione dei modelli di sito. La modellazione di dominio è pensata per riflettere i concetti espressi nel capitolo precedente. In particolare, le **ModelPageClass** rappresentano le Page Class inferite dall'algoritmo, e vengono successivamente trasformate in oggetti **PageClass** per essere usati nella fase di crawling intensivo.

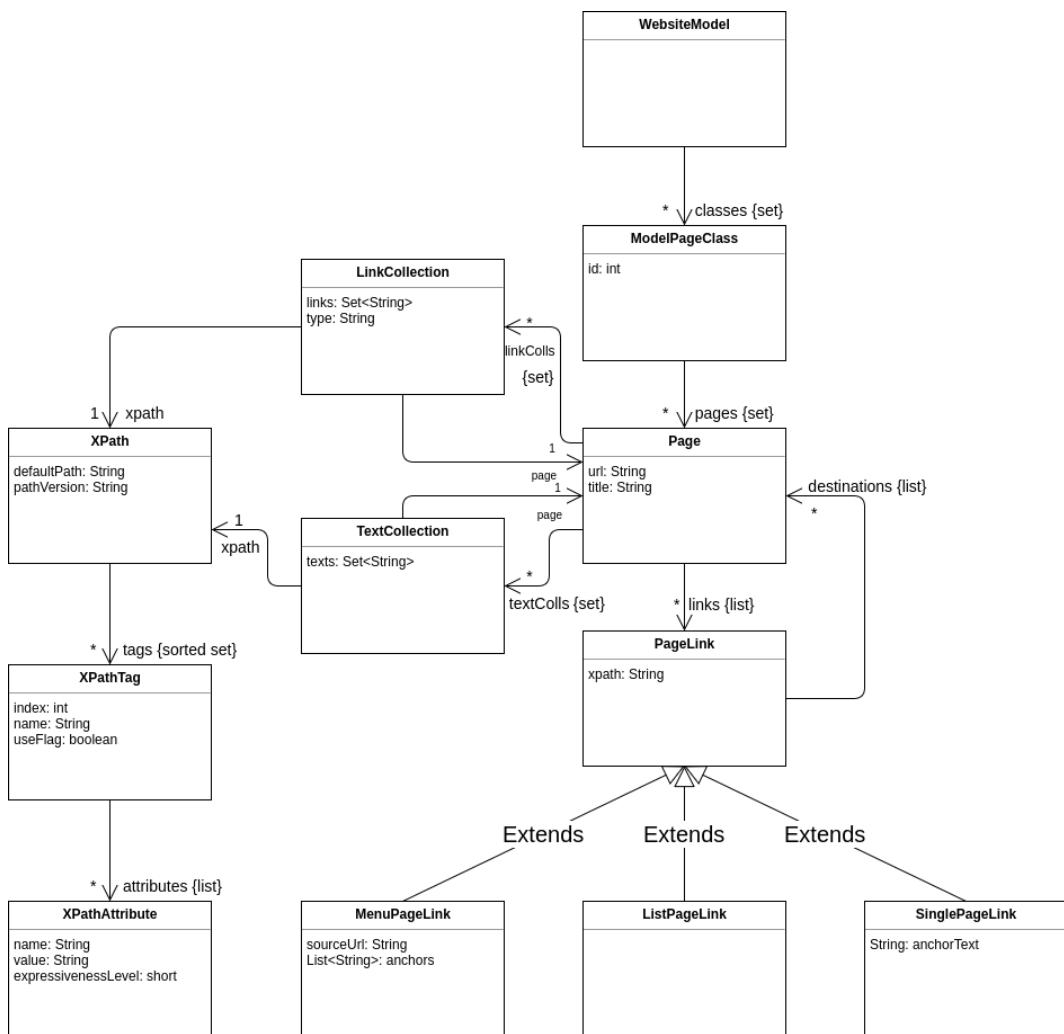


Figura 4.5: Diagramma delle classi di dominio per la fase di generazione dei modelli

4.2.2.2 Classi

Page e PageLink La classe **Page** rappresenta una pagina web ed è descritta come un insieme di **LinkCollection** e **TextCollection**. Questa classe contiene la logica per la creazione delle collezioni a partire dall'analisi del DOM della pagina: le XPath vengono generate considerando, per ogni link e per ogni testo⁴, il cammino dei tag dal link/testo fino alla radice (il tag HTML).

⁴Come euristica vengono considerati solo i testi con meno di 64 caratteri.

Ogni Page mantiene inoltre in memoria i collegamenti trovati con altri oggetti Page tramite la classe PageLink: quest'ultimo rappresenta una XPath, nella pagina di partenza, che punta a dei link (uno o molti) che rappresentano altri oggetti Page. PageLink è una classe estratta e le classi che la estendono rappresentano la tipologia del link: lista, menu, o singleton (link singolo).

Le implementazioni concrete della classe PageLink inoltre contengono la logica di creazione di un ClassLink tra due PageClass: a seconda delle **ModelPageClass** associate alla Page sorgente e alle Page destinazioni e al tipo del PageLink, viene inferito il collegamento tra le PageClass collegate. Ad esempio, si considerino la Page P_1 con un ListPageLink lp verso le Page $\{P_2, P_3, P_4\}$, e le PageClass C_1 e C_2 , tale che $P_1 \in C_1$ e $\{P_2, P_3, P_4\} \in C_2$. Allora lp inferisce e crea un **ClassLink** di tipo lista tra C_1 e C_2 .

LinkCollection e TextCollection Le classi LinkCollection e TextCollection sono la rappresentazione software dei concetti descritti nelle sezioni 3.2.1.1 e 3.2.1.2; mantengono in memoria le informazioni riguardanti gli URL e i testi associati, e sono collegati a un oggetto XPath che rappresenta la versione del cammino utilizzato dalle collezioni. Una Link Collection può essere rifinita a tempo dinamico variando l'espressività della sua XPath.

XPath La classe XPath rappresenta un reticolo di cammini dalla radice a un tag contenente un link o un testo. È la rappresentazione software del concetto di reticolo espresso nella sezione 3.3.3.2. La XPath viene generata percorrendo a ritroso il cammino dal tag finale di interesse alla radice, salvando per ogni tag tutti gli attributi comprensivi di nome e valore. La variabile d'istanza *defaultPath* rappresenta la versione di XPath utilizzata di default dal sistema, rappresentante un livello intermedio nel reticolo in cui vengono utilizzati tutti i tag del cammino e il nome del primo attributo di ciascun tag. Quando viene effettuata una rifinitura, la nuova versione della XPath viene generata variando, per ogni richiesta, il livello di espressività di un singolo attributo e viene salvata nella variabile *pathVersion*. Le classi collegate **XPathTag** e **XPathAttribute** modellano il livello di espressività di ciascun componente della XPath. Si noti che è anche possibile rimuovere un tag dal cammino per generare una XPath meno espresiva. Ad esempio, una XPath $//div/ul/li/a$ può essere decrementata di un livello di

espressività generando la XPath `//ul/li/a`, che identifica tutti i tag *UL* nel DOM anche non figli di un *DIV*.

ModelPageClass La classe ModelPageClass rappresenta una PageClass in fase di modellazione e definisce la logica per la creazione dello schema della classe e il contenimento delle Page appartenenti ad essa. In particolare lo schema di una ModelPageClass comprende una TextCollection solo se essa compare in almeno due Page del gruppo e se i suoi testi sono uguali in entrambe.

WebsiteModel La classe WebsiteModel rappresenta il modello di un sito web e comprende l'insieme delle ModelPageClass inferite dall'algoritmo. Il WebsiteModel è responsabile di trasformare, al termine dell'algoritmo, gli oggetti ModelPageClass in oggetti PageClass, tramite l'analisi dei PageLink delle Page del modello. Il grafo di navigazione, composto da PageClass e ClassLink, viene quindi restituito in output per essere salvato su disco o usato nella fase di crawling intensivo.

4.2.3 Modello ad attori

In questa sezione viene mostrata l'architettura dell'algoritmo di generazione dei modelli, e in particolare dei moduli che permettono il rispetto dei requisiti specificati nella sezione 4.2.1: anche questa fase è realizzata tramite l'uso di Akka e del modello ad attori al fine di ottenere i livelli di scalabilità e di concorrenza desiderati.

4.2.3.1 Visione

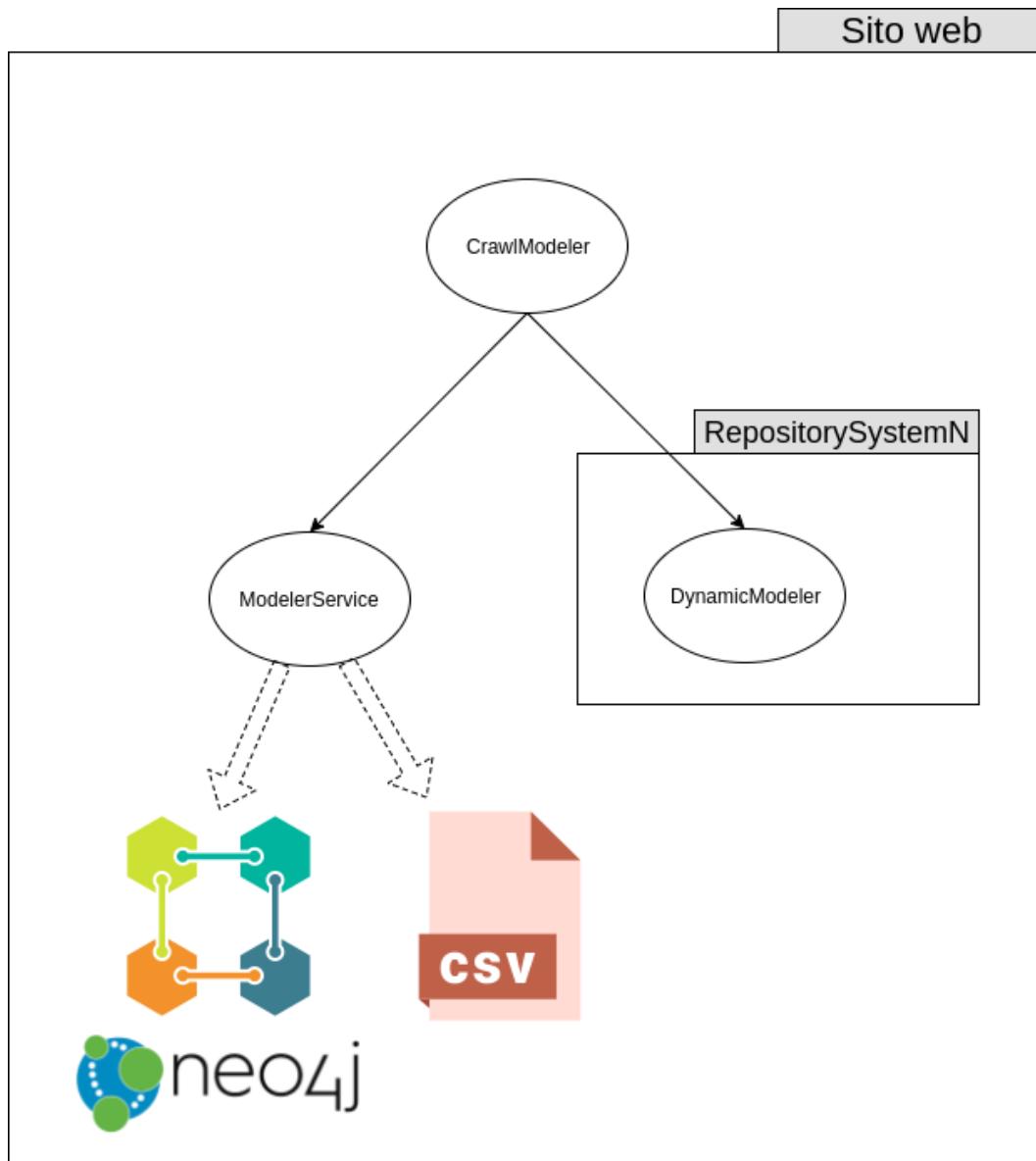


Figura 4.6: Gerarchia degli attori distribuiti responsabili della generazione dei modelli

La figura 4.6 mostra la gerarchia degli attori responsabili della generazione del caricamento dei modelli dei siti web. Il **CrawlModeler** è l'attore che si occupa di coordinare le attività di **generazione** e **persistenza/caricamento** dei modelli di un

singolo sito web. Il **DynamicModeler** è l'attore che implementa l'algoritmo di inferenza dei modelli descritto in 3.3.1. Si noti che nel caso della generazione dei modelli di più siti web in parallelo, i DynamicModeler vengono distribuiti sui diversi host disponibili dati in input, al fine di garantire maggiori prestazioni e tempi di esecuzione scalabili. Quando i DynamicModeler hanno generato il grafo delle PageClass, questi vengono inviati agli attori **ModelerService** (in esecuzione sull'host master) e salvati su disco, in formato CSV e in una base di dati a grafo, Neo4J⁵.

4.2.3.2 Attori

CrawlModeler L'attore CrawlModeler è responsabile della coordinazione delle azioni di generazione del modello e di persistenza e caricamento dei modelli salvati. A seconda delle indicazioni date in un file di configurazione genera gli attori indicati per generare un nuovo modello o caricarne uno già salvato. Le PageClass generate o caricate vengono quindi inviate alle CrawlFrontier per la fase di crawling intensivo.

DynamicModeler L'attore DynamicModeler implementa l'algoritmo di generazione del modello di un sito web. Al fine di permettere al motore di esecuzione di alternare la generazione del modello con quello di altri siti web, ciascuna azione dell'algoritmo viene eseguita in modalità asincrona. La figura 4.7 rappresenta la schematizzazione dei passi dell'algoritmo. Si noti che ciascun ovale rappresenta un'azione sincrona compiuta dall'algoritmo: le frecce di collegamento con i rettangoli rappresentano invece messaggi inviati dall'attore a se stesso al fine di aggiornare il suo stato corrente. Questo garantisce, tramite l'invio di messaggi, che il sistema possa processare parallelamente diversi DynamicModeler dello stesso host, decidendo in ogni momento quale messaggio processare e quindi quale DynamicModeler far avanzare. Le frecce tratteggiate indicano passi opzionali dell'algoritmo, come la rifinitura delle XPath, l'avviso di scaricamento intero della LinkCollection se essa è un menu, e la selezione della prossima collezione se quella corrente non ha trovato URL validi. Quando la condizione di terminazione viene raggiunta, il DynamicModeler passa all'azione di finalizzazione del modello inferito, che consiste nella creazione del grafo delle PageClass a partire dalle ModelPa-

⁵<https://neo4j.com>

geClass create nel WebsiteModel. Il grafo finale viene quindi inviato al CrawlModeler per l'archiviazione e il crawling intensivo.

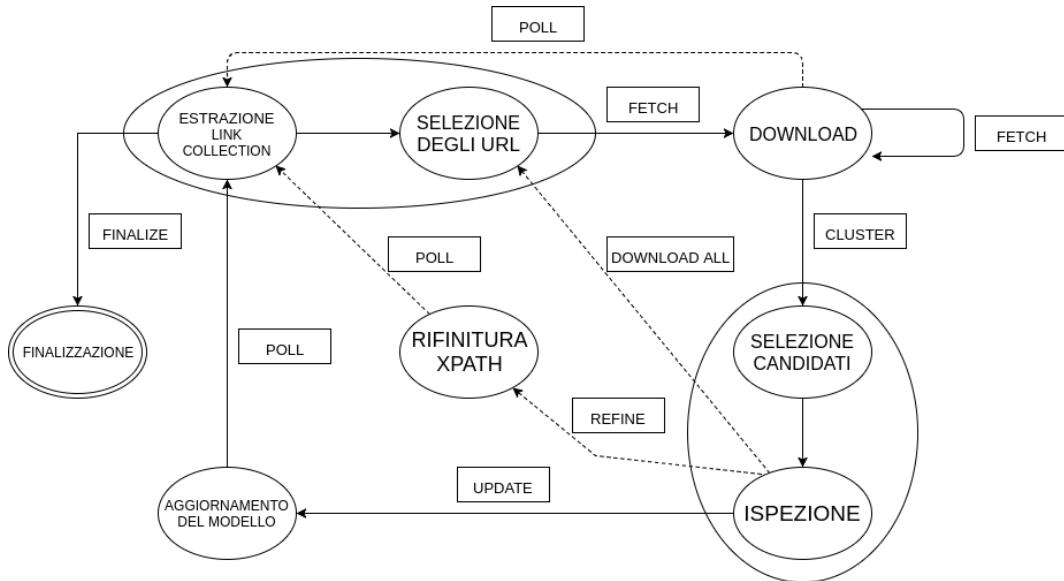


Figura 4.7: Diagramma delle azioni asincrone compiute dall'algoritmo di modellazione

ModelerService L'attore ModelerService è responsabile del salvataggio e del caricamento del modello di un sito web. Gestisce operazioni per la persistenza del modello ricevuto in input, sotto forma di grafo di PageClass, in formato CSV e su Neo4J. In particolare ModelerService salva informazioni riguardanti il *timestamp* al momento del salvataggio e un numero progressivo indicante la versione del modello ricevuto. Queste informazioni possono essere poi usate per stabilire quale versione di modello caricare per il crawling intensivo.

Oltre ai modelli inferiti dall'algoritmo, questo attore può caricare modelli di navigazione specificati in un file CSV secondo lo stesso formato: le regole specificate nel file vengono usate per ricreare un grafo di PageClass che viene inviato alla CrawlFrontier collegata e usato per la fase di crawling intensivo.

4.2.4 Persistenza dei modelli

I modelli inferiti vengono salvati in una base di dati a grafo per permettere interrogazioni volte a studiare determinate aree di un sito web e l'evoluzione dei Class Link tra esse, tramite il confronto di versioni differenti di uno stesso sito. In figura 4.8 viene mostrata una parte del modello di navigazione calcolato per il sito di un'agenzia immobiliare. In particolare, la Page Class in alto a sinistra rappresenta l'homepage del sito: da essa è possibile navigare verso la Page Class al centro della figura, che rappresenta la classe dei risultati di una ricerca. Come si può notare, essa presenta un Class Link di tipo lista verso se stessa (indicante i link di paginazione verso i risultati successivi) e una serie di Class Link di tipo lista e singleton verso la Page Class dei dettagli delle proprietà.

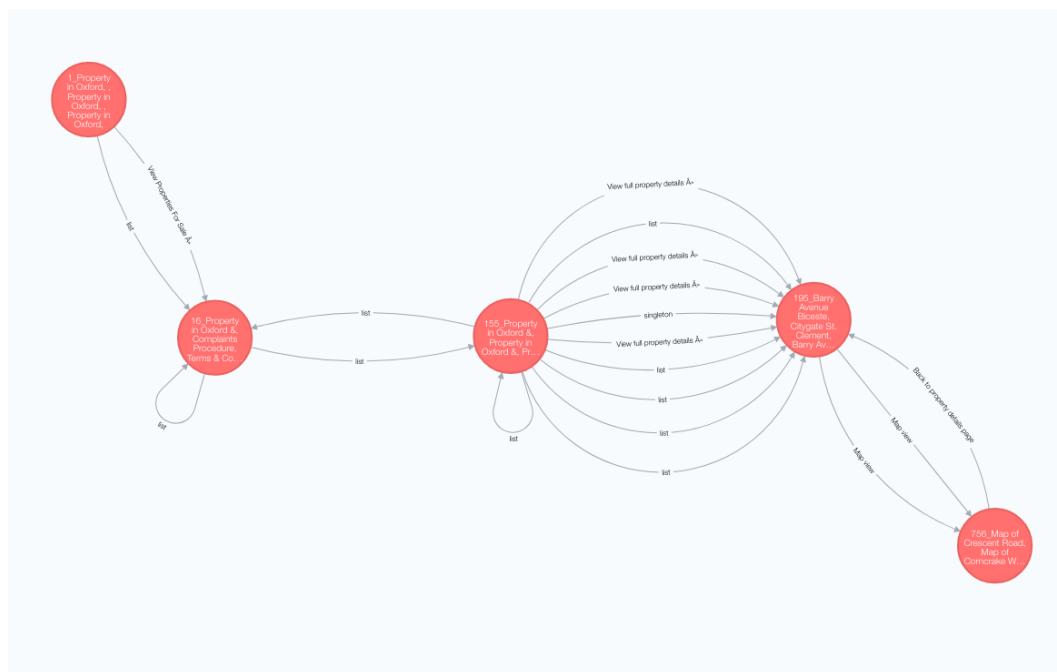


Figura 4.8: Particolare del modello inferito per <http://www.wwagency.com>

La figura 4.9 mostra in rosso una link collection indicante un menu nella homepage di un sito web; la figura 4.10 mostra la rappresentazione dello stesso menu nel modello inferito dall'algoritmo. Come si può notare, la Page Class in alto rappresenta la classe della pagina nella figura precedente. Il nodo centrale rappresenta il menu, identificato dalla XPath mostrata in basso. Infine i collegamenti uscenti dal menu rappresentano le Page Class destinazioni, che contengono rispettivamente le pagine *Short Lets* e *Contact*⁶

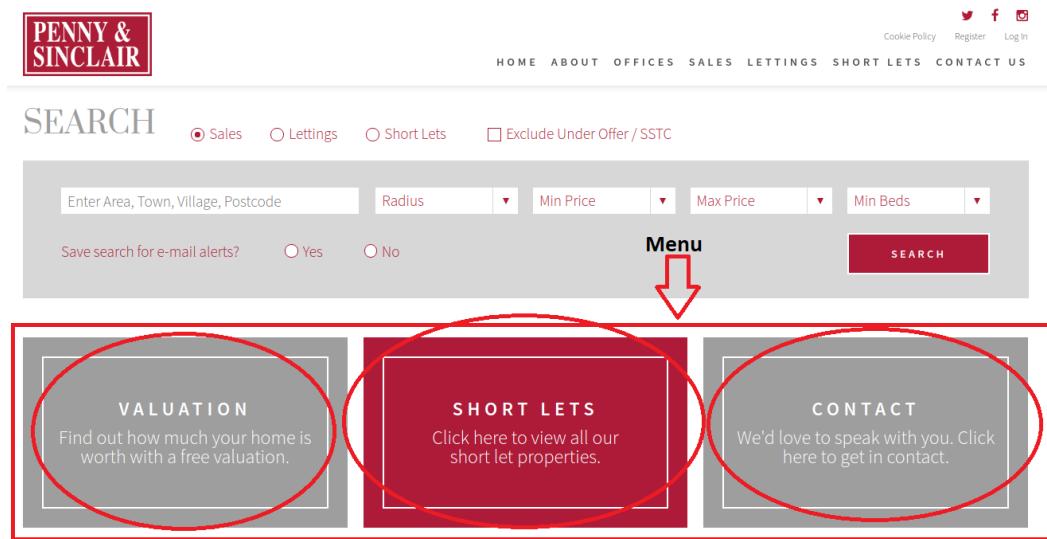


Figura 4.9: Menu di un sito web identificato dall'algoritmo di generazione del modello

⁶Il link *Valuation* non viene considerato nel modello, in quanto è un redirect alla stessa pagina della figura.

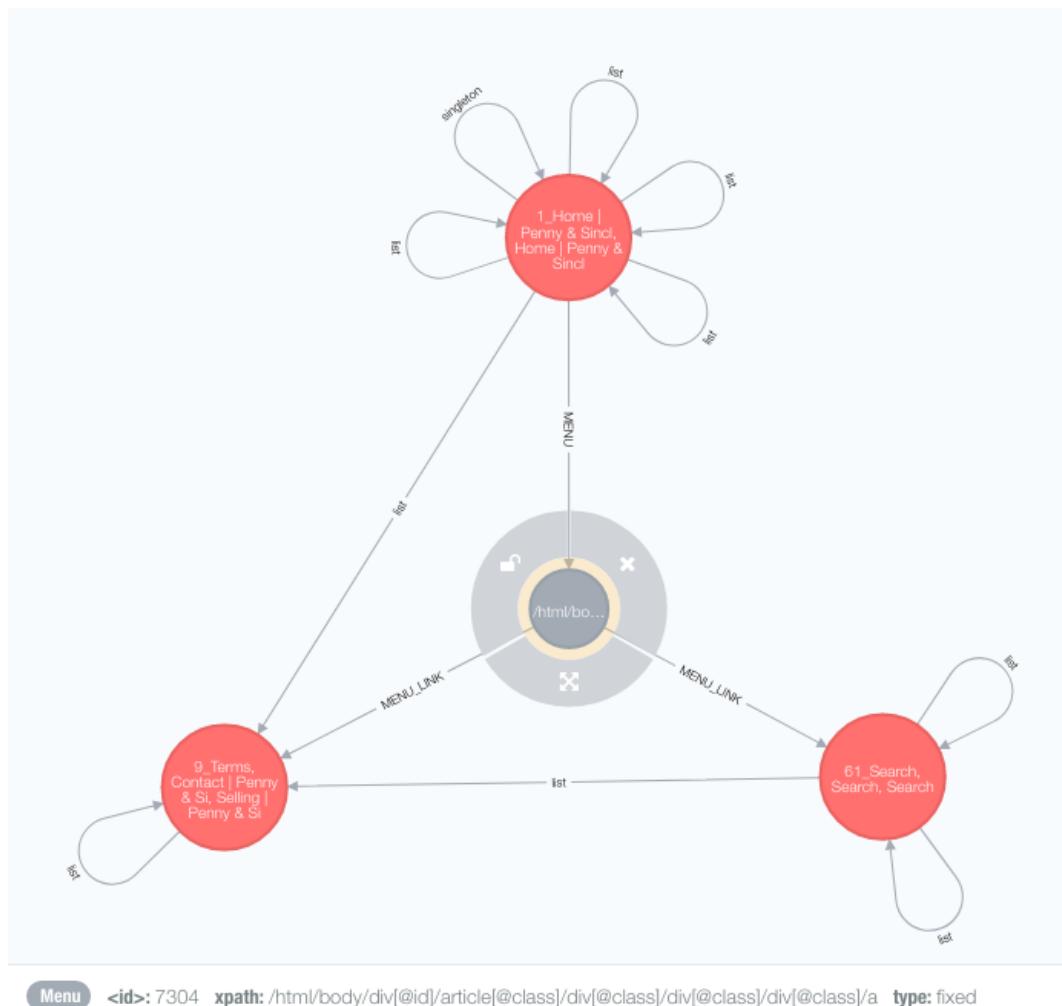


Figura 4.10: Rappresentazione di un ClassLink di tipo Menu su Neo4J

Capitolo 5

Esperimenti e risultati

Questo capitolo descrive gli esperimenti compiuti e i risultati ottenuti riguardanti la generazione scalabile di modelli di siti web. Nella prima sezione viene descritto il dataset utilizzato per gli esperimenti nella fase preliminare e finale di sviluppo dell'algoritmo. Nella seconda sezione vengono descritte le metriche di valutazione utilizzate per valutare l'efficacia della creazione dei modelli di navigazione. Nella terza sezione vengono riportati e analizzati i risultati ottenuti su siti web reali.

5.1 Dataset e descrizione

Nella fase preliminare di implementazione dell'algoritmo per la fase di generazione dei modelli dei siti web sono stati condotti esperimenti utilizzando 4 siti locali di test creati a mano: un sito per il testing della corretta generazione delle Page Class, due siti per il test della corretta rifinitura delle XPath, e un sito per il test della corretta individuazione dei *cammini di sito* (cfr. 3.3.4.1).

Nella fase di realizzazione finale l'algoritmo è stato invece testato su 25 siti reali: per ciascuno di essi è stato definito a mano un modello di navigazione (*golden set*) usato come base per valutare l'efficacia dell'algoritmo. In particolare, i siti web per il testing sono stati selezionati secondo i seguenti criteri:

- Esistono evidenze del fatto che il sito web è generato automaticamente da un'applicazione web.

- Le Page Class del sito sono facilmente identificabili, ed è possibile associare a ogni pagina una Page Class sulla base del suo URL.
- La maggior parte delle pagine del sito è raggiungibile attraverso link di navigazione.

Per ciascun sito web, il golden set è stato realizzato con le seguenti modalità:

- Sono state considerate le Page Class del sito con il maggior numero di pagine, o che rappresentano il contenuto informativo più interessante per il dominio del sito (classe dei prodotti per un sito di e-commerce, classe delle proprietà per un'agenzia immobiliare, ecc.)
- Per ogni Page Class individuata è stata definita un'espressione regolare che identifica tutti e soli gli URL di quella Page Class.
- Sono stati definiti i Class Link di navigazione tra le Page Class individuate, specificando anche il tipo di collegamento che lega le classi (lista, menu, link singolo).

I siti del dataset utilizzato sono stati selezionati dai seguenti domini: **agenzie immobiliari, vendita e noleggio di automobili usate, e-commerce**, più altri siti provenienti da domini vari.

5.2 Metriche di valutazione

5.2.1 Qualità delle Page Class

Per valutare la qualità delle Page Class prodotte dall'algoritmo, il modello del sito web è stato generato dall'algoritmo usando un numero limitato n di pagine. È stata quindi valutata la suddivisione effettuata di tutte le pagine nelle Page Class prodotte. Per ogni Page Class computata è stata quindi calcolata la *F-measure*, l'*Entropia* e la *Purezza*. La definizione di tali misure è descritta di seguito.

Detto S l'insieme delle pagine scaricate per la generazione del modello, siano C_1, C_2, \dots, C_n le Page Class corrette definite dal golden set, e siano $\hat{C}_1, \hat{C}_2, \dots, \hat{C}_m$ le Page Class computate dall'algoritmo. La *precisione* e la *recall* di ogni classe \hat{C}_j è definita come:

$$P(C_i, \hat{C}_j) = \frac{|C_i \cap \hat{C}_j|}{\hat{C}_j} \quad (5.1)$$

$$R(C_i, \hat{C}_j) = \frac{|C_i \cap \hat{C}_j|}{C_i} \quad (5.2)$$

Per calcolare questi termini, i risultati sono organizzati in una *matrice di confusione*: ogni elemento e_{ij} della matrice rappresenta il numero di pagine di C_i che sono state assegnate a \hat{C}_j . La precisione e la recall di una classe computata \hat{C}_j rispetto a una classe corretta C_i è allora ridefinita come:

$$P(C_i, \hat{C}_j) = \frac{e_{ij}}{\sum_{k=1}^n e_{kj}} \quad (5.3)$$

$$R(C_i, \hat{C}_j) = \frac{e_{ij}}{\sum_{k=1}^m e_{ik}} \quad (5.4)$$

La F-measure è la media armonica di precisione e recall. Data una classe corretta C_i e una classe computata \hat{C}_j :

$$F(C_i, \hat{C}_j) = \frac{2P(C_i, \hat{C}_j)R(C_i, \hat{C}_j)}{P(C_i, \hat{C}_j) + R(C_i, \hat{C}_j)} \quad (5.5)$$

Per ogni classe corretta C_i , la sua F-measure è definita come:

$$F_i = \max_{j=1, \dots, m} \left(\frac{2P(C_i, \hat{C}_j)R(C_i, \hat{C}_j)}{P(C_i, \hat{C}_j) + R(C_i, \hat{C}_j)} \right) \quad (5.6)$$

La F-measure dell'intero modello del sito web è definita come:

$$F^* = \sum_{i=1}^n F_i \times \frac{|C_i|}{\bigcup_{k=1}^n C_k} \quad (5.7)$$

La F-measure è quindi compresa nell'intervallo $[0,1]$: più è alta, tanto è migliore la qualità delle Page Class.

L'Entropia di una classe \hat{C}_j è definita come la coesione degli elementi all'interno del cluster, ed è definita in funzione della precisione $P(C_i, \hat{C}_j)$ su tutte le classi C_i :

$$H(\hat{C}_j) = - \sum_{i=1}^n P(C_i, \hat{C}_j) \log(P(C_i, \hat{C}_j)) \quad (5.8)$$

L'Entropia dell'intero modello del sito web è definita come:

$$H = - \sum_{j=1}^m H(\hat{C}_j) \times \frac{|\hat{C}_j|}{\bigcup_{k=1}^m \hat{C}_k} \quad (5.9)$$

In questo caso, una Entropia minore indica un cluster migliore.

Ogni classe computata può contenere pagine di diverse classi corrette. La Purezza è definita come il rapporto tra la classe più grande rappresentata nella classe computata e la dimensione della classe computata stessa. Intuitivamente, una classe computata "pura" contiene elementi di una sola classe corretta. La purezza misura quindi la dimensione del sottoinsieme della classe dominante. La Purezza è definita come:

$$P(\hat{C}_j) = \frac{1}{|\hat{C}_j|} \max_{k=1, \dots, n} e_{jk} \quad (5.10)$$

Similmente all'Entropia, la Purezza dell'intero modello è calcolata come la somma pesata della Purezza di ciascuna classe computata.

5.2.2 Qualità dei Class Link

Per valutare la qualità dei Class Link inferiti tra le Page Class individuate dall'algoritmo (fondamentali per la fase di crawling intensivo), sono state considerate le classi computate più vicine alle classi corrette del golden set, ed è stata verificata la presenza tra una coppia di classi computate dei Class Link specificati nel golden set, identificati dalle classi sorgente e destinazione e dal tipo del collegamento (lista, menu, link singolo). Per ogni coppia di Page Class è stata quindi calcolata la *Recall* dei ClassLink. La definizione di questa misura è descritta di seguito.

Siano C_1, C_2, \dots, C_n le Page Class corrette definite dal golden set, e siano $\hat{C}_1, \hat{C}_2, \dots, \hat{C}_m$ le Page Class computate dall'algoritmo. Siano dette (C_i, \hat{C}_j) le coppie di classi tali

che \hat{C}_j è la classe computata che massimizza la F-measure di C_i , secondo l'espressione 5.6. Intuitivamente, la classe computata \hat{C}_j è quella più simile alla classe corretta C_i . Allora, detti CL_i i Class Link di C_i e \hat{CL}_j i Class Link di \hat{C}_j :

$$R(CL_i, \hat{CL}_j) = \frac{|CL_i \cap \hat{CL}_j|}{|CL_i|} \quad (5.11)$$

Si noti che due ClassLink $cl_j \in \hat{CL}_j$ e $cl_i \in CL_i$ sono equivalenti se la classe computata di destinazione di cl_j è quella che massimizza la F-measure della classe corretta di destinazione di cl_i .

La Recall dei Class Link dell'intero modello è calcolata come la somma pesata della Recall dei Class Link di ciascuna coppia di classi.

5.3 Risultati

Gli esperimenti sono stati condotti generando i modelli con al più **200** pagine. Come strategia di navigazione è stata usata una politica di *navigazione densa* (cfr. 3.3.5).

Le tabelle seguenti mostrano i risultati della valutazione della qualità delle Page Class e dei Class Link prodotti per ciascun sito web dei domini considerati. Per ogni sito viene indicata la F-measure (F^*), l'Entropia (H), la Purezza (P) e la Recall dei Class Link (R). Viene inoltre specificato il numero di pagine totali del sito web.

I risultati ottenuti mostrano che la tecnica di clustering genera dei modelli di navigazione accettabili, e che 200 pagine sono in molti casi sufficienti a permettere al crawler di scoprire le Page Class principali dei siti. In particolare la Purezza dei siti è generalmente alta, il che indica che le Page Class computate sono rappresentative delle Page Class espresse nel golden set.

Per quanto riguarda la Recall dei Class Link, essa mostra una qualità più bassa rispetto a quella delle Page Class create. Questo potrebbe portare il crawler ad avere una scarsa Recall delle pagine totali del sito web nella fase di crawling intensivo. In particolare, questo porta ad ipotizzare che per generare un modello che sia in grado di descrivere la navigazione completa del sito è necessario analizzare un numero superiore di pagine durante la generazione automatica, rispetto a quello utilizzato negli esperimenti eseguiti.

Si noti dalla tabella 5.2 che i risultati migliori sono stati ottenuti nel dominio delle agenzie immobiliari: questo può essere dovuto in generale al fatto che i siti in questione sono più piccoli di quelli degli altri domini, hanno un minor numero di Page Class e presentano spesso un template con una struttura di navigazione standard, molto simile tra sito e sito.

I risultati sul dominio dei siti di e-commerce mostrato in tabella 5.1 presenta una F-measure e una Recall dei Class Link accettabile, ma inferiore rispetto al precedente dominio. Questo può essere dovuto al fatto che questa tipologia di sito presenta ovviamente molte più pagine ed ha un template meno prevedibile. Inoltre, le pagine relative ai dettagli di un prodotto tendono spesso a contenere diverse link collection opzionali, come ad esempio dei link a prodotti correlati o a prodotti venduti insieme a quello descritto nella pagina: dal momento che le link collection sono uno degli strumenti principali per descrivere una pagina, questo può portare l'algoritmo a dividere i prodotti in diverse Page Class.

Sito	F^*	H	P	R	pagine totali
http://www.abtvaultin.com	0,76	0,04	0,94	0,77	4200
http://www.algogo.com	0,83	0,13	0,95	0,81	2570
http://www.butterflyphoto.com	0,73	0,02	0,98	0,72	23500
http://www.manventureoutpost.com	0,83	0,08	0,84	0,81	52300
https://fumfie.com	0,81	0,13	0,91	0,7	15700
https://tinkerwatches.com	0,81	0,3	0,69	1	347
https://www.deliceville.com	0,85	0,02	0,98	0,87	3910
https://www.lanuovaerapietre.com	0,95	0,1	0,92	0,91	1590
https://www.percys.com	0,91	0,06	0,96	0,74	9540

Tabella 5.1: Risultati per i siti di e-commerce

Sito	F*	H	P	R	pagine totali
http://gssproperty.com	0,82	0,04	0,82	0,92	502
http://morganwilliams.com	0,85	0,04	0,95	0,6	1240
http://www.aredhouse.co.uk	0,86	0,13	0,86	0,32	1490
http://www.hunterfrench.co.uk	0,8	0,02	0,99	0,64	402
http://www.pennyandsinclair.co.uk	0,94	0,03	0,97	0,74	590
http://www.samuelsagents.co.uk	0,93	0	1	0,8	1780
http://www.vickery.co.uk	0,87	0,28	0,85	0,28	777
http://www.wwagency.com	0,91	0,09	0,9	0,86	1980
https://beville.co.uk	0,8	0,27	0,67	0,65	330
https://www.arnoldskeys.com	0,92	0,09	0,85	0,8	5190

Tabella 5.2: Risultati per i siti di agenzie immobiliari

Sito	F*	H	P	R	pagine totali
http://auto100.co.uk	0,91	0	1	0,58	1250
http://dovehousecars.com	0,86	0	1	1	3110
http://www.7starcarsales.co.uk	0,89	0,1	0,89	0,63	340
http://www.portfield-subaru.co.uk	0,68	0,25	0,32	1	680

Tabella 5.3: Risultati per i siti di noleggio di auto usate

Sito	F*	H	P	R	pagine totali
http://www.thomson.co.uk	0,51	0,62	0,54	0,22	42100
http://www.artcyclopedia.com	0,87	0,09	0,88	0,33	125000

Tabella 5.4: Risultati per altri siti

Conclusioni e Sviluppi Futuri

In questa tesi è stato presentato un algoritmo di generazione automatica di modelli di siti web, ispirato dalle metodologie descritte in [5], e l'architettura di un crawler che genera in maniera scalabile tali modelli e li usa per effettuare un crawling intensivo e distribuito dei siti dati in input.

Il modello di navigazione è presentato come un grafo diretto in cui i nodi rappresentano le Page Class del sito, e gli archi diretti rappresentano i link di navigazione tra le Page Class. La similarità strutturale tra le pagine è definita sulla base alle proprietà dei cammini identificati nel DOM, in particolare sulla base della struttura dei link e dei testi invarianti nell'HTML. Lo studio del reticolo dei cammini delle link collection permette inoltre di ridefinire a tempo dinamico l'espressività delle XPath usate per descrivere i collegamenti tra le Page Class, al fine di descrivere dei percorsi di navigazione più efficaci. L'attribuzione di un peso alle XPath degli schemi delle pagine permette inoltre di ottenere una conoscenza più profonda del template HTML, al fine di distinguere tra template comuni a tutto il sito, propri di una Page Class e propri di una singola pagina, e creare così dei cluster più precisi.

La fase di crawling intensivo è avvantaggiata dalla conoscenza del modello di navigazione prodotto nella fase preliminare poiché descrive la struttura dei percorsi di navigazione e permette al crawler di classificare efficacemente un gran numero di pagine web. Nella fase intensiva il crawler è in grado di scalare sull'attività di archiviazione mediante un duplice livello di distribuzione: diversi siti possono essere esplorati da host diversi, e uno stesso sito può essere esplorato da host diversi al fine di mantenere alta

la frequenza di pagine scaricate, senza essere bloccati dal server web.

I risultati mostrati evidenziano che i modelli generati possono essere utilizzati per eseguire sessioni di crawling efficienti scalando sul numero di siti visitati: in particolare, i modelli generati e le pagine scaricate nel crawling intensivo possono essere usati per (1) generare wrapper su grandi insiemi di pagine strutturalmente simili, (2) guidare il crawler verso aree del sito (Page Class) di particolare interesse per un dominio applicativo, evitando le aree meno importanti, (3) migliorare la compressione delle pagine archiviate, estraendo per ogni Page Class il template comune così da poter salvare per ogni pagina solo il contenuto particolare, (4) studiare i cambiamenti strutturali dei siti web nel tempo, tramite l'analisi dei cambiamenti nei modelli di navigazione inferiti.

Sviluppi futuri Il crawler progettato permette il clustering e la navigazione efficiente di siti web, ma diverse problematiche rimangono ancora insolute.

Un primo problema è che non sempre il modello di navigazione generato può essere usato nel crawling intensivo per archiviare tutte le pagine presenti: infatti, specie per i siti più grandi, molte Page Class potrebbero non essere state scoperte. Allo stesso modo i Class Link prodotti potrebbero limitarsi a descrivere link che portano a pagine già esplorate, limitando fortemente la completezza dell'esplorazione. Si pensi ad esempio, in un sito di e-commerce, alla Page Class che descrive una lista di prodotti di una categoria. Se il Class Link che porta alla paginazione, e quindi all'esplorazione delle pagine di risultati successivi, non viene trovato, il crawler non archiverà i prodotti successivi.

Un secondo problema è nella rifinitura del cammino delle XPath in fase di modellazione. Le XPath delle link collection di una pagina descrivono la sua struttura. Tuttavia, se esse vengono rifinite a tempo dinamico, anche lo schema della pagina viene modificato. Questo potrebbe portare l'algoritmo a dividere in Page Class diverse altre pagine che presentano le stesse XPath della precedente in una versione non ancora rifinita, poiché i loro link uscenti non sono ancora stati analizzati.

Un terzo problema è nella semantica delle Page Class. L'algoritmo infatti separa le pagine in base alla loro struttura, ma non viene fatta alcuna considerazione riguardo

la loro semantica. Ad esempio, in un sito di un giornale, le notizie di sport e politica possono avere la stessa struttura nel DOM. In questo caso il crawler crea una Page Class sola per entrambe le categorie: un'analisi semantica dei contenuti potrebbe portare alla loro divisione, per definire un modello di navigazione che tenga conto anche della semantica dei cluster. Un quarto problema è nella navigazione di siti web che usano principalmente form, e non link, per collegare tra loro diverse pagine. L'algoritmo di generazione dei modelli non è infatti in grado di inferire collegamenti tra Page Class collegate tramite form, né è in grado in primo luogo di visitare pagine non raggiungibili tramite link.

Un quinto problema è nell'uso della memoria usata dalla frontiera degli URL nella fase di crawling intensivo. Il crawler inserisce in una coda in memoria un numero massimo fisso di URL, spostando quelli meno importanti in una coda persistita che viene svuotata quando lo spazio in memoria torna ad essere disponibile. Tuttavia la dimensione della coda in memoria è fissa per tutto il crawling: un tuning dinamico della dimensione, basato ad esempio sul numero medio di URL in coda, potrebbe invece migliorare l'uso della memoria e l'efficienza della persistenza degli URL in eccesso.

Appendice A

Specific di crawling

Questa appendice descrive la sintassi della specifica di crawling in formato CSV da utilizzare nella fase di crawling intensivo. La specifica può essere data in input al crawler insieme all'URL del sito di interesse. Si noti che i modelli di navigazione di sito web generati automaticamente rispettano lo stesso formato. Pertanto, il crawling intensivo può essere eseguito utilizzando un modello inferito dall'algoritmo oppure fornendo manualmente una specifica che rispetti la sintassi descritta.

L'appendice è organizzata come segue: nella prima sezione viene descritta la sintassi della specifica. Nella seconda sezione viene presentato un esempio d'uso di crawling intensivo su un sito web reale con una specifica che prevede la navigazione e l'estrazione di dati dalle pagine visitate.

A.1 Sintassi

Una specifica di crawling rappresenta un modello di navigazione di sito web: essa prevede che le pagine del sito di interesse vengano intese come membri appartenenti a delle Page Class. La specifica definisce delle XPath di navigazione (Class Link) e di estrazione (Data Link) per recuperare dati di interesse dalle pagine di Page Class specifiche. Si noti in particolare che una XPath di navigazione o estrazione può essere definita come Form-XPath: un tipo di XPath che identifica una form in una pagina e contiene i dati per compilarla.

A.1.1 Form-XPath

Una Form-XPath è un tipo particolare di XPath che identifica una form in una pagina web e contiene i campi e i dati necessari a compilarla. Le Form-XPath possono essere specificate per compiere azioni sulla pagina (descritte in A.1.3) o per navigare nel sito (A.1.2).

Le Form-XPath devono essere definite con la seguente sintassi:

$$\text{XPath}\{\text{Form}\} [, \text{XPath}\{\text{Campo}\} : \text{string}] *$$

dove $\text{XPath}\{\text{Form}\}$ è una XPath che identifica una form nel DOM della pagina, $\text{XPath}\{\text{Campo}\}$ è una XPath che identifica un campo di testo appartenente alla form (nel percorso dal nodo form al nodo campo), string è una stringa di testo da inserire nel campo selezionato. L'operatore * indica che la struttura tra parentesi quadre può essere ripetuta zero o più volte.

A.1.2 Class Link

La tabella A.1 mostra l'intestazione della specifica di un Class Link: i campi **Sorgente** e **Destinazione** specificano delle etichette associate rispettivamente alla Page Class sorgente e di destinazione del Class Link. Il campo **Tipo** prevede la dicitura *link*. Il campo **XPath** specifica la XPath da usare per identificare il Class Link nella pagina appartenente alla classe Sorgente. Il campo **Sottotipo** specifica il sottotipo del Class Link e può assumere i seguenti valori:

- **singleton**: indica che il Class Link individua un link singolo verso la Destinazione.
- **list**: indica che il Class Link individua una lista di link con la stessa Destinazione.
- **menu**: indica che il Class Link individua uno o più link verso una Destinazione, ma si trova all'interno di un menu.
- **form**: indica che il Class Link individua una form nella pagina della classe Sorgente che, se compilata, conduce alla Destinazione. Se si usa questo sottotipo, allora nel campo XPath si deve inserire una Form-XPath.

Sorgente	Tipo	XPath	Destinazione	Sottotipo
----------	------	-------	--------------	-----------

Tabella A.1: Header della specifica dei Class Link

La tabella A.2 mostra una specifica di estrazione di esempio con due Class Link, di tipo form e singleton.

class1	link	//form[@id="global-search-form"],./input[@id="global-search"]:all	class2	form
class2	link	//span[@class="active"]/../following-sibling::li[1]/a	class2	singleton

Tabella A.2: Esempio di specifica di Class Link

A.1.3 Data Link

La tabella A.3 mostra l'intestazione della specifica di un Data Link: il campo **Sorgente** specifica la Page Class su cui eseguire l'estrazione. Il campo **Tipo** indica il tipo di dato da estrarre e può assumere i seguenti valori:

- **string**: è usato per estrarre stringhe di testo.
- **url**: è usato per estrarre URL.
- **img**: è usato per estrarre indirizzi di immagini.
- **form**: è usato per completare una form all'interno della pagina, al fine di modificare il contenuto della pagina stessa. Se si usa questo tipo, nel campo XPath deve essere specificata una Form-XPath.

Il campo **XPath** specifica la XPath da usare per identificare il nodo da estrarre nel DOM. Il campo **Nome** specifica l'intestazione da utilizzare come etichetta del dato estratto.

Sorgente	Tipo	XPath	Nome
----------	------	-------	------

Tabella A.3: Header della specifica dei Data Link

La tabella A.4 mostra una specifica di estrazione di esempio con tre Data Link, di tipo string, img e url.

detail	string	//h1/text()	company name
detail	img	//img[@alt='logo']	logo
detail	url	//div[@class='rdbx']//div[@style='padding-left:10px;']/text()	website

Tabella A.4: Esempio di specifica di Data Link

A.2 Esempio d'uso

Viene ora illustrato un esempio d'uso di crawling intensivo su un sito web reale. L'esempio mostra le azioni di crawling eseguite sul sito <https://olfatheque.com>, un motore di ricerca specializzato sui profumi. La specifica utilizzata in input è mostrata in tabella A.5, ed ha lo scopo di indirizzare il crawler verso le pagine di dettaglio di ciascun profumo al fine di estrarre i dati di interesse per ognuno di essi, come il nome e la piramide olfattiva. Si noti in particolare che la Page Class **perfumes** rappresenta le pagine che contengono liste di profumi: la specifica definisce infatti delle XPath di navigazione per raggiungere la Page Class **perfume** dei singoli profumi e per raggiungere le altre liste di profumi (Class Link di tipo list nella tabella). Si noti inoltre che da un'analisi del sito si può osservare che le pagine di dettaglio dei profumi (Page Class **perfume**) sono oscurate e rimandano ad una pagina di login: la specifica prevede quindi che il crawler, tramite una Form-XPath, inserisca dei dati per effettuare il login, al fine di avere accesso alle reali sembianze delle pagine di interesse.

home	link	//div[@id="sidebar-menu-content"]/ul/li[3]/a	perfumes	singleton
perfumes	link	//span[@class="active"]/../following-sibling::li[1]/a	perfumes	singleton
perfumes	link	//div[@id="result"]/div[@class]/div[@class]/h3/a	perfume	list
perfume	form	//form[@id="form"],./div/input[1]:user,./div/input[2]:pass		
perfume	string	//div[@class="fiche-title"]/h1/text()	Name	
perfume	string	//div[@id="recap"]/p[@class="famille"]/a/text()	Family	
perfume	string	//div[@id="recap"]/p/span[1]../a/text()	Facets	
perfume	string	//div[@id="recap"]/p/span[2]../a/text()	Perfumer	
perfume	string	//div[@id="recap"]/p/span[3]../a/text()	Creative studios	
perfume	string	//div[@id="recap"]/p/span[4]../a/text()	Creation date	
perfume	string	//div[@id="recap"]/p/span[5]../a/text()	Collection	
perfume	string	//div[@class="note note-tete"]/p/a/text()	Top notes	
perfume	string	//div[@class="note note-coeur"]/p/a/text()	Heart notes	
perfume	string	//div[@class="note note-fond"]/p/a/text()	Background Notes	

Tabella A.5: Specifica di crawling per <https://olfatheque.com>

Di seguito vengono mostrate alcune pagine di esempio appartenenti alle Page Class descritte nella specifica. I riquadri rossi evidenziano i link identificati dalle XPath di navigazione. I riquadri blu identificano i dati da estrarre identificati dalle XPath di estrazione.

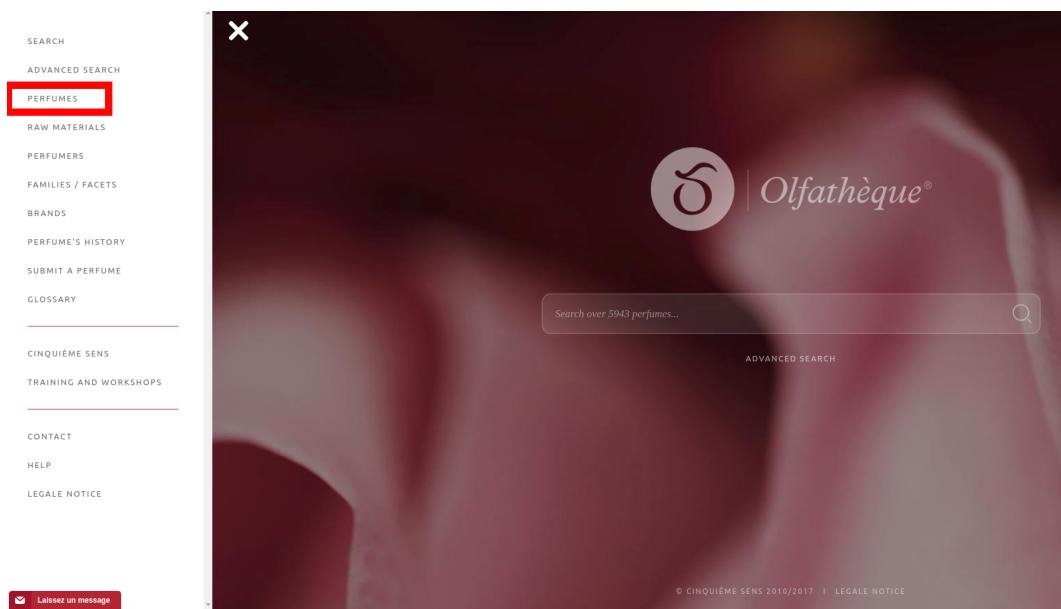


Figura A.1: Page Class home

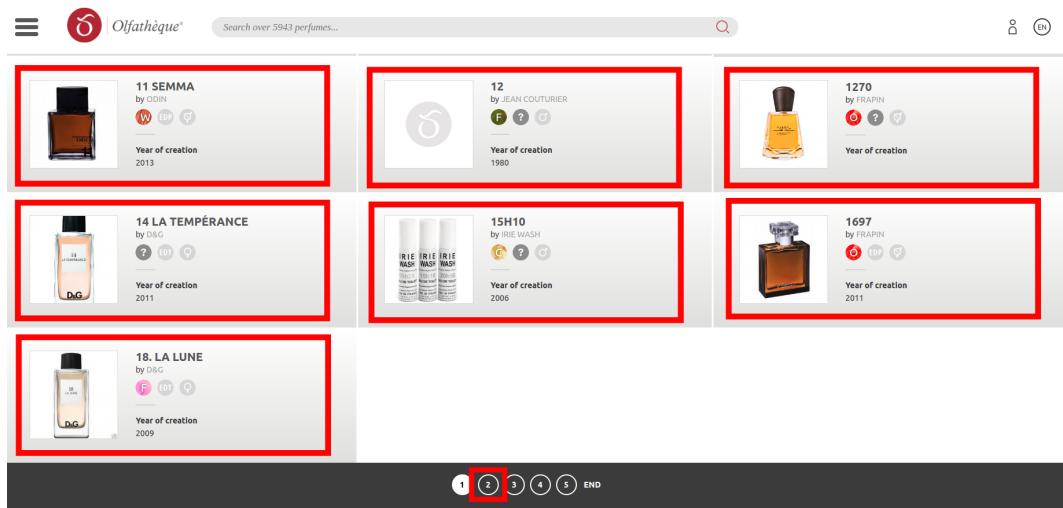


Figura A.2: Una pagina della Page Class perfumes

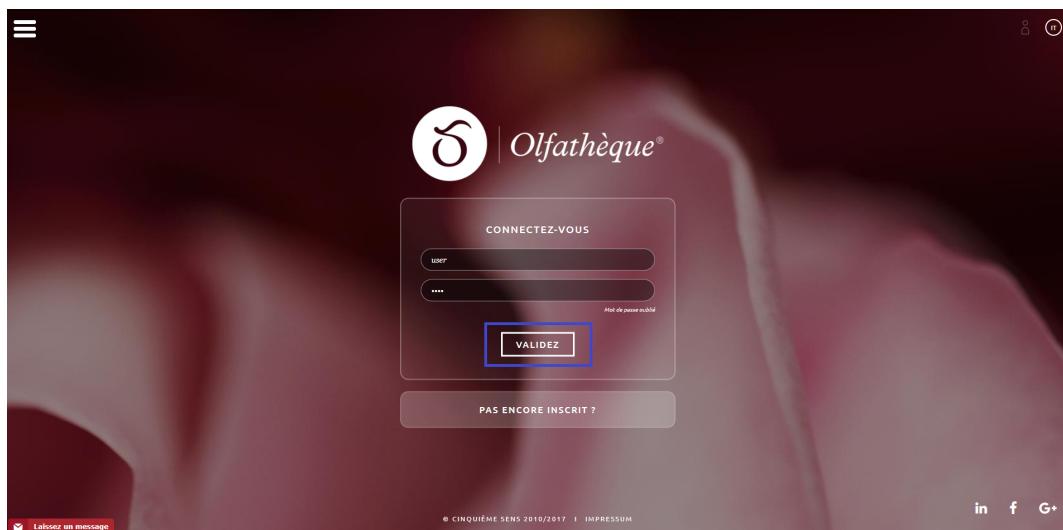


Figura A.3: Una pagina della Page Class perfume, oscurata da una form

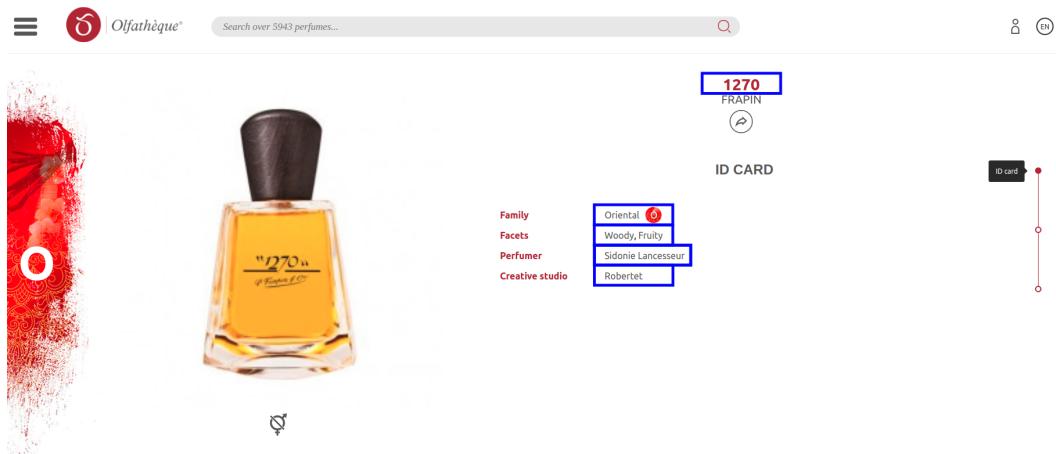


Figura A.4: Una pagina della Page Class perfume, con i dati da estrarre

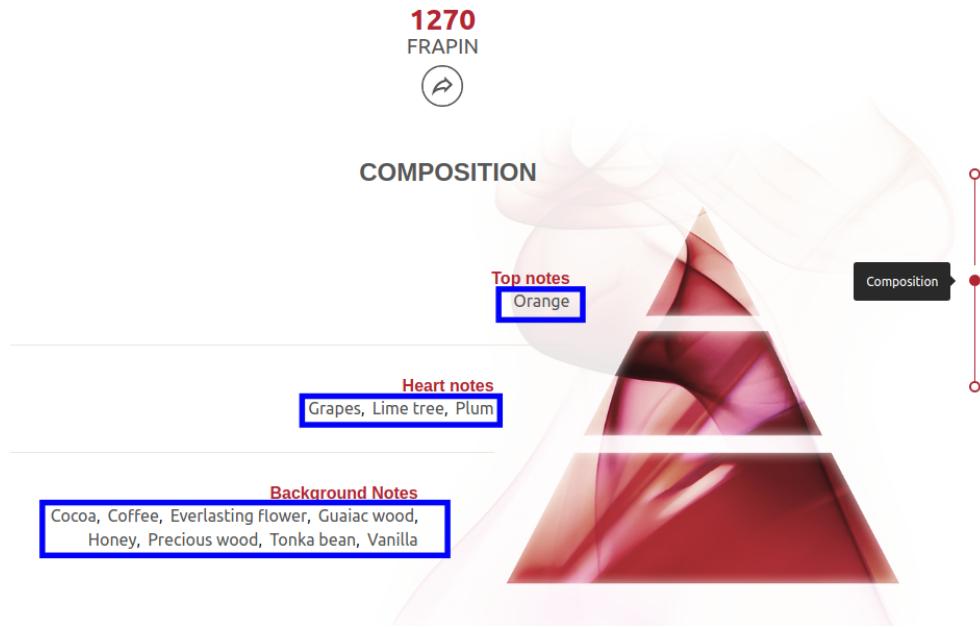


Figura A.5: Una pagina della Page Class perfume: piramide olfattiva

Il crawling di esempio, guidato dalla specifica mostrata, ha portato all'estrazione dei dati di 5903 profumi presenti sul sito. La figura A.6 mostra alcuni record di esempio del CSV estratto.

Name	Family	Facets	Perfumer	Creative studio	Creation	Collection	Top notes	Heart notes	Background Notes
Jasmin noir	Oriental	Woody Floral	Carlos Benaim Sophie Li IFF		2008		Tangerine	Gardenia Jasmin Tonka bean	
The Beat intense elixir	Floral	Powdery			2008				
Prelude to Love Invitation	Floral	Powdery Green	Calice Becker	Givaudan	2008		Bergamot Bitter Green note Nero Orris		
Gipsy Water	Amber	Woody Citrus	Jérôme Epinette	Robertet	2008		Bergamot Junipe Frankincense Or Amber Sandalwood		
Pulp	Citrus	Fruity Green	Jérôme Epinette	Robertet	2008		Bergamot Lemor Apple Elemi Fig Cedar wood Praline		
Green	Floral	Citrus Powdery Green	Jérôme Epinette	Robertet	2008		Orange Petit grai Honeysuckle Jas Tonka bean White musk		
Chembur	Woody	Leather Spicy	Jérôme Epinette	Robertet	2008		Bergamot Elemi Ginger Nutmeg Labdanum White musk		
Rose Noir	Chypre	Floral Musky	Jérôme Epinette	Robertet	2008		Grapefruit Freesia		White musk
POP femme	Oriental	Fruity	Michel Almairac	Robertet	2005				
Amor Amor Tentation	Floral	Woody Fruity Amber	Honorine Blanc	Firmenich	2008		Tangerine Jasmine	Cedar wood Vanilla	
Amor pour Homme Tenta	Oriental	Woody Fruity Tasty	Alberto Morillas Jacques	Firmenich	2008		Ginger Tangerine Blackcurrant Jas	Cedar wood Frankincense	

Figura A.6: Esempi di record estratti dal crawler

Bibliografia

- [1] Adar, E., Teevan, J., Dumais, S. T., and Elsas, J. L. The web changes everything: Understanding the dynamics of web content. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining* (New York, NY, USA, 2009), WSDM '09, ACM, pp. 282–291.
- [2] Blanco, L., Dalvi, N. N., and Machanavajjhala, A. Highly efficient algorithms for structural clustering of large websites. In *Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011* (2011), pp. 437–446.
- [3] Boldi, P., Marino, A., Santini, M., and Vigna, S. Bubing: Massive crawling for the masses. *CoRR abs/1601.06919* (2016).
- [4] Cai, R., Yang, J., Lai, W., Wang, Y., and Zhang, L. iRobot: an intelligent crawler for web forums. In *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008* (2008), pp. 447–456.
- [5] Crescenzi, V., Merialdo, P., and Missier, P. Clustering web pages based on their structure. *Data Knowl. Eng.* 54, 3 (Sept. 2005), 279–299.
- [6] Faheem, M., and Senellart, P. *Intelligent and Adaptive Crawling of Web Applications for Web Archiving*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 306–322.
- [7] Faheem, M., and Senellart, P. *Adaptive Web Crawling Through Structure-Based Link Classification*. Springer International Publishing, Cham, 2015, pp. 39–51.

- [8] Ferragina, P., and Manzini, G. On compressing the textual web. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining* (New York, NY, USA, 2010), WSDM '10, ACM, pp. 391–400.
- [9] Furche, T., Gottlob, G., Grasso, G., Guo, X., Orsi, G., Schallhart, C., and Wang, C. Diadem: Thousands of websites to a single database. *Proc. VLDB Endow.* 7, 14 (Oct. 2014), 1845–1856.
- [10] Gao, K., Wang, W., and Gao, S. Modelling on web dynamic incremental crawling and information processing. In *2013 5th International Conference on Modelling, Identification and Control (ICMIC)* (2013).
- [11] GARGOYLE SOFTWARE. HtmlUnit - GUI-Less browser for Java programs. <http://htmlunit.sourceforge.net>, 2017.
- [12] Grasso, G., Furche, T., and Schallhart, C. Effective web scraping with oxpath. In *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013, Companion Volume* (2013), pp. 23–26.
- [13] Grünwald, P. A tutorial introduction to the minimum description length principle. *CoRR math.ST/0406077* (2004).
- [14] Manning, C. D., Raghavan, P., and Schütze, H. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.
- [15] Mohr, G., Kimpton, M., Stack, M., and Ranitovic, I. Introduction to Heritrix, an archival quality web crawler. In *Proceedings of the 4th International Web Archiving Workshop IRAW'04* (Bath, UK, July 2004).
- [16] Najork, M., and Heydon, A. *High-Performance Web Crawling*. Springer US, Boston, MA, 2002, pp. 25–45.
- [17] Qingzhao, T., and Prasenjit, M. Clustering-based incremental web crawling. *ACM Trans. Inf. Syst.* 28, 4 (Nov. 2010), 17:1–17:27.
- [18] Roestenburg, R., Bakker, R., and Williams, R. *Akka in Action*. Manning Publications Co., 2017.

- [19] Sigurðsson, K. Incremental crawling with Heritrix. In *In IWAW* (2005).
- [20] W3C RECOMMENDATION. XML Path Language (XPath). <https://www.w3.org/TR/xpath>, 1999.
- [21] W3C RECOMMENDATION. Document Object Model (DOM) Level 3 Core Specification. <https://www.w3.org/TR/DOM-Level-3-Core>, 2004.