UM MODELO PARA PROTOTIPAGEM RÁPIDA DE APLICAÇÕES DE MINERAÇÃO NA WEB

ÁLVARO RODRIGUES PEREIRA JÚNIOR

UM MODELO PARA PROTOTIPAGEM RÁPIDA DE APLICAÇÕES DE MINERAÇÃO NA WEB

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

Orientador: Ricardo Baeza-Yates
Co-Orientador: Nivio Ziviani

Belo Horizonte
Outubro de 2008

ÁLVARO RODRIGUES PEREIRA JÚNIOR

A MODEL FOR FAST PROTOTYPING OF WEB MINING APPLICATIONS

Thesis presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

Advisor: Ricardo Baeza-Yates
Co-Advisor: Nivio Ziviani

Belo Horizonte
October 2008

Pereira Júnior, Álvaro Rodrigues
P436m A Model for Fast Prototyping of Web Mining Applications / Álvaro Rodrigues Pereira Júnior. — Belo Horizonte, 2008
xxxviii, 140 f. : il. ; 29cm

Tese (doutorado) — Universidade Federal de Minas Gerais
Orientador: Ricardo Baeza-Yates
Co-orientador: Nivio Ziviani

1. Mineração de dados (Computação) – Tese. 2. Banco de dados – Tese. 3. Algoritmos – Tese. 4. World Wide Web (Sistema de Recuperação de Informação) – Tese. I. Título.

CDU 519.6*72(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Um Modelo Para Prototipagem Rápida de Aplicações de Mineração na Web

ÁLVARO RODRIGUES PEREIRA JÚNIOR

Tese defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. RICARDO BAEZA-YATES - Orientador
YAHOO! Research Barcelona

PROF. NIVIO ZIVIANI - Co-orientador
Departamento de Ciência da Computação - UFMG

PROF. CARLOS ALBERTO HEUSER
Departamento de Informática - UFRGS

PROF. MARIANO P. CONSENS
University of Toronto

PROF. ALBERTO HENRIQUE FRADE LAENDER
Departamento de Ciência da Computação - UFMG

PROF. CLODOVEU AUGUSTO DAVIS JÚNIOR
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 31 de outubro de 2008.

To my love Thabata.

Acknowledgments

Finishing a Ph.D. thesis is something like being a twice-born. The difference is that at the second time you born already knowing. Overall, knowing that you could not have done everything alone. So many people have contributed to this achievement, and now I acknowledge at least part of them, the most important bodies for me during this important stage of my life.

I must be very luck to have had the chance to be supervised by Nivio Ziviani and Ricardo Baeza-Yates. My history with Nivio started at the beginning of my master course. I had no experience with research, I mean, I had no experience with anything I guess. Two years later, after a lot of work, I was finishing the master course and thinking of a Ph.D., in which Nivio had agreed to be my supervisor again. Nivio has a great history, a lot of experience, and I am happy for having learned so much with him.

I was about to finish the master course when I went to Santiago in Chile to present a paper. Ricardo Baeza-Yates, who had a long history of work collaboration with Nivio, saw what I was working on and proposed a research topic for my Ph.D thesis. Of course I did not understand anything he was talking about, but accepted the challenge. Four or five years later he saw his ideas become the WIM, which is the core of this thesis. Well, at least Ricardo saw "my interpretation of" his ideas become the WIM. Ricardo is another experienced researcher who complemented Nivio's experience for supervising my thesis with quality. These two guys have been important for me not only on the professional field but also as good friends.

Out of the technical sphere, my wife Stephanie Thabata has been the most important person during all these years. Actually, I was influenced by her for doing this course. She said I should try and she was the first person who believed I could do it, even before myself. She always demonstrated that this course was so important for her as it was for me. Of course I recognize that being a Ph.D. student's wife is not an easy task. She had to abdicate of too many things. Indeed, we got to have a lot of fun together, living in

different places and meeting many different people.

During the first half of my course my workplace was LATIN – LAboratory for Treating Information, in the Department of Computer Science (DCC) of the Federal University of Minas Gerais (UFMG). It was a place to share knowledge and I cannot forget the support all the colleagues gave me for all those years, which include the years I was there for my master course. Also my acknowledgments to the professors, technicians, and students I had the chance to cohabit in the DCC/UFMG.

During the second half of my Ph.D. course I was in Barcelona, Spain, at Yahoo! Research. It was a great personal experience, for the people I met and lived together. In terms of research experience, I feel that I lived the other side of the coin. The researchers at Yahoo!, and probably at any other IT company that supports research, must also be good engineers. Generally they have no students, so that they need to put their hands on and implement the solutions. This scenario showed me the importance of being also a good engineer, before being a good researcher. My acknowledgments to all the friends I made there.

My acknowledgments also to my thesis defense external committee, for their invested time and important comments made: prof. Carlos Alberto Heuser, prof. Mariano P. Consens, prof. Alberto Henrique Frade Laender, and prof. Clodoveu Augusto Davis Júnior.

Finally, my acknowledgments to my parents, Álvaro and Valéria. They have given me unconditional support to my life since I was born (for the first time), and without their care I would not be here writing the last words in this thesis.

Resumo

Mineração Web pode ser vista como o processo de encontrar padrões na Web por meio de técnicas de mineração de dados. Mineração Web é uma tarefa computacionalmente intensiva, e a maioria dos softwares de mineração são desenvolvidos isoladamente, o que torna escalabilidade e reusabilidade difícil para outras tarefas de mineração. Mineração Web é um processo iterativo onde prototipagem tem um papel essencial para experimentar com diferentes alternativas, bem como para incorporar o conhecimento adquirido em iterações anteriores do processo. O objetivo desta tese é o desenvolvimento de um modelo para prototipagem rápida em mineração Web, chamado WIM – *Web Information Mining*. A principal motivação para desenvolver o WIM é o fato de que seu modelo conceitual provê os seus usuários com um nível de abstração apropriado para prototipagem e experimentação durante a tarefa de mineração.

WIM é composto de um modelo de dados e de uma álgebra. O modelo de dados WIM é uma visão relacional dos dados Web. Os três tipos de dados existentes na Web, chamados de conteúdo, de estrutura e dados de uso, são representados por relações. Os principais componentes de entrada do modelo de dados WIM são as páginas Web, a estrutura de hiperlinks que interliga as páginas Web, e os históricos (*logs*) de consultas obtidos de máquinas de busca da Web. A programação WIM é baseada em fluxos de dados (*dataflows*), onde sequências de operações são aplicadas às relações. As operações são definidas pela álgebra WIM, que contém operadores para manipulação de dados e para mineração de dados. WIM está implementado contendo uma linguagem de programação declarativa provida por sua álgebra. A arquitetura do software WIM é apresentada, juntamente com suas questões de implementação, e projetos de arquiteturas alternativas são discutidos, sobre o qual uma versão futura do software WIM para escala industrial poderia ser implementada.

WIM é aplicado a um conjunto de cinco casos de uso reais em mineração Web, como uma maneira de demonstrar os recursos do WIM. O principal caso de uso, chamado Árvores

Genealógicas na Web, é um estudo de como o conteúdo da Web evolui com o tempo. Esse caso de uso foi escolhido para realização de uma análise completa dos resultados, os quais apresentam evidências de que parte dos usuários editores de conteúdo na Web realizam consultas em máquinas de busca para encontrar conteúdo e então republicar o que foi encontrado como resultado de consulta. A conclusão é que máquinas de busca influenciam o conteúdo da Web. A experimentação do WIM nos cinco casos de uso mostrou que o seu uso facilita significantemente a prototipagem rápida em mineração Web. O uso experimental da linguagem de programação WIM mostrou que ela reduz o tamanho do código escrito para uma aplicação em ordens de magnitude, quando comparada com implementações isoladas.

Abstract

Web mining can be seen as the process of discovering patterns from the Web by means of data mining techniques. Web mining is a computation-intensive task and most mining software is developed ad-hoc, which makes scalability and reusability difficult for other mining tasks. Web mining is an iterative process and prototyping plays an essential role in experimenting with different alternatives, as well as in incorporating knowledge acquired in previous iterations of the process. The objective of this thesis is the development of a model for fast Web mining prototyping, referred to as WIM – Web Information Mining. The main motivation for developing the WIM model is the fact that its underlying conceptual model provides its users with a level of abstraction appropriate for prototyping and experimentation during the Web mining task.

WIM is composed of a data model and an algebra. The WIM data model is a relational view of Web data. The three types of existing Web data, namely Web content, Web structure and Web usage, are represented by relations. The main input components for the WIM data model are the Web pages, the hyperlink structure linking Web pages and the query logs obtained from Web search engines. WIM is implemented with a declarative programming language provided by its algebra. The WIM programming language is based on dataflows, where sequences of operations are applied to relations. The operations are defined by the WIM algebra, which contains operators for data manipulation and for data mining. We present the WIM software architecture, its implementation issues, and discuss alternative architecture designs on which a forthcoming industrial-scale WIM software version could be implemented.

We have applied WIM to a set of five real Web mining use cases, as a means to demonstrate the WIM features. The main use case, called Genealogical Trees on the Web, is a study of how Web content evolves in time. We have elected this use case to perform a complete analysis of its results, which present evidence that some Web publishers actually performed queries using search engines in order to find content and then republish

what was found as answer to the query. The conclusion is that search engines bias the content of the Web. The experimentation of WIM in five real use cases has been shown to significantly facilitate fast Web mining prototyping. Experimental use of the WIM programming language has shown that it reduces the code size written for an application by orders of magnitude when compared with ad-hoc implementations.

Resumo Estendido

Um Modelo para Prototipagem Rápida de Aplicações de Mineração na Web

Introdução

Mineração de dados na Web, ou simplesmente mineração Web, é o processo de descobrir informação útil em dados da Web, por meio de técnicas de mineração de dados [Liu, 2007; Chakrabarti, 2002]. O custo de encontrar a informação correta não está associado à falta de técnicas de mineração de dados, mas à complexidade de gerenciar os dados Web necessários, e ao uso eficiente e eficaz de técnicas conhecidas de mineração. Com o objetivo de reduzir significativamente o custo em minerar dados Web, de forma a fazer o processo de mineração mais acessível e de fácil uso, a tese aqui resumida representa uma das primeiras contribuições para a criação de uma *máquina de mineração* para a Web.

Mineração Web é um processo iterativo, no qual prototipagem tem um papel essencial para experimentar facilmente com diferentes alternativas, bem como para incorporar o conhecimento adquirido durante iterações anteriores do processo. Para facilitar prototipagem, um nível apropriado de abstração precisa ser provido ao usuário ou programador responsável pela tarefa de mineração. Motivado pela falta de um arcabouço para tal abstração, a parte central da tese compreende o projeto e desenvolvimento de um modelo para prototipagem intensiva em mineração Web.

O principal objetivo desta tese é o projeto e desenvolvimento do modelo WIM – *Web Information Mining*, um modelo implementado como um protótipo de software para prototipagem rápida de soluções de mineração Web. WIM inclui um modelo de dados formal e uma álgebra, que estão especialmente projetados para manipular e minerar os principais tipos de dados presentes na Web. Como um requisito para que no futuro o atual

protótipo WIM possa se tornar uma ferramenta de uso extenso, um projeto consistente é extremamente importante. Para tal, vasto conhecimento do mundo que envolve mineração Web é fundamental, de forma que outro objetivo da tese é também o estudo de um conjunto de problemas de mineração Web. WIM foi aplicado a um conjunto de cinco problemas em mineração Web.

Este artigo é um resumo em português da tese, e busca apresentar de uma forma geral e em alto nível o trabalho desenvolvido. Estão presentes na tese, mas não fazem parte do escopo deste resumo, os seguintes pontos: descrição dos estudos realizados para solução dos problemas em mineração Web apresentados na Seção , apresentação do modelo de dados formal WIM, apresentação sintática e semântica dos operadores da álgebra WIM, formalização e avaliação da álgebra, descrição da arquitetura e questões de implementação do protótipo. De toda forma, este resumo é completo o suficiente para mostrar os principais resultados gerados pela tese e sua importância.

A Seção apresenta um conjunto de aplicações em mineração Web nas quais o WIM é capaz de implementar soluções. Para algumas aplicações, ambas soluções, isolada e usando WIM, foram desenvolvidas. A Seção 3 apresenta, de uma forma geral, os principais conceitos presentes no modelo WIM, com ilustrações reais de uso. Finalmente, as conclusões, bem como uma discussão sobre os trabalhos atuais e futuros, gerados como resultados diretos da tese, são apresentados na Seção .

Aplicações em Mineração Web Desenvolvidas

Para que se entenda os tipos de cenários nos quais o WIM pode ser empregado, esta seção apresenta uma série de aplicações já estudadas através do WIM. A principal aplicação estudada na tese é uma análise da evolução do conteúdo da Web de acordo com o tempo [Baeza-Yates et al., 2008]. O programa WIM identifica documentos pais, que são fontes de cópias, e documentos filhos, que são documentos mais recentes contendo parte do conteúdo de documentos velhos. Problemas como detecção de duplicatas para filtragem, comparação de URLs, associação de um único pai a cada filho, são tratados nesse estudo e implementados tanto pelo WIM quanto de forma isolada, para comparação das duas soluções de implementação.

Como resultado do trabalho de mineração de dados realizado neste estudo [Baeza-Yates et al., 2008], foi identificado o padrão de cópias parciais de documentos na Web. Como principais conclusões, estimou-se que quase 25% dos novos documentos na Web têm trechos de documentos previamente existentes, e que cópias de sites externos (em

relação ao site do documento filho) acontecem mais frequentemente que cópias do mesmo site. Isso demonstra que usuários copiam mais conteúdo de outros do que os seus próprios conteúdos, que estariam no mesmo site. O estudo também relacionou o papel de máquinas de busca no comportamento de usuários ao reutilizar conteúdo previamente publicado na Web. Como forte conclusão, foi verificado que documentos que aparecem como resultado de consultas no topo de máquinas de busca são muito mais usados como fonte de cópia do que outros documentos, o que indica que máquinas de busca influenciam no próprio conteúdo de novos documentos gerados na Web.

Uma outra aplicação desenvolvida foi um estudo de duplicatas na Web [Baeza-Yates et al., 2007b]. Apesar de vários autores afirmarem que o percentual de duplicatas (incluindo conteúdo similar – *near-duplicates*) na Web é em torno de 20% a 30% [Bharat and Broder, 1999; Fetterly et al., 2003], foi verificado que este percentual é de fato muito maior, entre 35% e 45%. A diferença nos resultados é porque a avaliação feita por outros autores é baseada em coleções Web geradas a partir de lista de links encontradas em cada documento. Neste caso, no momento da coleta, cada nova URL encontrada como link em páginas anteriormente coletadas é inserida em uma fila para coleta, e logo também coletada e incluída na base de dados Web.

O que foi identificado nos experimentos é que documentos duplicados, em geral, não possuem outros documentos com links para eles (*inlinks*). Isso significa que muitas duplicatas não são encontradas pelos métodos tradicionais de coleta. Como foi usada uma coleção da Web chilena, onde todos os domínios registrados foram usados como semente para inicializar o coletor, todos os sites sob o domínio `.cl` tinham representantes na base de dados, tornando o estudo muito mais abrangente. Dessa forma, identificou-se que a Web possui cerca de 50% mais documentos duplicados do que previamente reportado em outros experimentos.

Além das aplicações em mineração Web apresentadas acima, foi iniciado o estudo de quatro outras aplicações, que também são usadas na tese como casos de uso para validar o modelo WIM, e serão introduzidas a seguir.

Também relacionada ao estudo de duplicatas, foi implementada uma solução WIM para avaliar como links para novos documentos cresce com o tempo, tanto para novos documentos que são duplicados de documentos antigos, quanto para novos documentos com novo conteúdo. Foi identificado que o número de links para novos documentos com conteúdo novo cresce significativamente mais do que o número de links para novos documentos com conteúdo duplicado. Essa conclusão pode ser mais explorada para classificar

documentos a serem coletados, dado que nem todas as URLs descobertas em máquinas de busca comerciais podem ser coletadas. Considerar como heurística para coleta a evolução dos links para uma dada URL pode ser importante para economizar recursos e dirigir de forma eficiente a lista de URLs a serem coletadas e então indexadas por máquinas de busca.

O WIM também foi aplicado aos outros tipos de dados encontrados na Web. Particularmente, uma aplicação bastante interessante consiste no cálculo de uma medida de relevância para documentos Web baseada no grafo de cliques de usuários. A medida se baseia na intuição de que o fluxo de cliques de um usuário em uma dada sessão de busca, na maioria das vezes, indica que os documentos clicados mais no final da sessão tendem a ser mais relevantes para aquela consulta do que os documentos clicados no início.

A solução WIM minera os dados de uso de forma a compor um grafo representando a ordem de cliques dentro de uma sessão. Logo, o *Pagerank* de cada documento do grafo é calculado, de forma a associar um *Pagerank* de uso a cada documento. Como um exemplo do grafo, considere que um usuário clicou em uma página A, depois em B e em seguida em C. O grafo é composto de um link de A para B e outro de B para C. Como C foi o último documento clicado nessa sessão, um auto link em C deve ser incluído, de forma a forçar um maior valor de relevância para o último documento clicado na sessão. Um único grafo, possivelmente com vários componentes conectados (florestas), é gerado para todos os documentos clicados, como resultado do processamento de todas as sessões de um histórico de uso.

Para este trabalho foram utilizadas duas bases de dados de uso. A primeira, da máquina de busca Todocl[1], com alguns milhões de cliques; e a segunda, do Yahoo! (do Reino Unido – `.uk`), com 22 milhões de cliques. O *Pagerank* real, utilizado para comparação com o *Pagerank* de uso, foi tomado de uma base de dados do Reino Unido contendo 77 milhões de documentos. A solução WIM inclui todas as etapas do processamento: desde a abertura dos arquivos com os dados de uso, a criação do grafo, até a associação entre documentos da coleção mencionada e os documentos clicados na base de dados de uso.

Outro problema envolvendo os dados de uso foi o estudo da intenção do usuário ao realizar uma busca. Basicamente, existem três possíveis intenções gerais: informacional, quando o usuário busca por informação que pode aparecer em diferentes documentos; navegacional, quando o usuário já sabe onde clicar mas faz a consulta como um atalho, resultando em somente um clique; e transacional, quando o usuário busca realizar uma transação, como realizar uma compra ou fazer um *download*.

[1] www.todocl.cl

A aplicação WIM desenvolvida para este problema busca classificar automaticamente, sem nenhuma intervenção do usuário, consultas como informacional, navegacional, ou com as duas intenções. O diferencial deste trabalho, além de não demandar dados de humanos para treino, é a proposta de que para um conjunto representativo de consultas, diferentes usuários realizando a mesma consulta podem ter diferentes intenções. Por exemplo, ao buscar pelo nome de uma pessoa, pode-se estar interessado na página pessoal da pessoa, ou diferentes opiniões sobre a mesma. Com este trabalho, também se busca demonstrar que consultas transacionais não são importantes de serem identificadas por máquinas de busca, uma vez que o sistema de busca não pode tirar nenhum proveito da característica transacional de uma consulta. Por outro lado, consultas comerciais têm um impacto importante e devem ser identificadas e tratadas com precisão em um sistema de busca, uma vez que gera receita para o negócio.

Por fim, o WIM foi utilizado efetivamente para selecionar documentos para avaliação de relevância, em uma iniciativa para gerar uma coleção pública de referência para pesquisa em aprendizado de *ranking* de sistemas de busca. Tal avaliação de relevância foi feita para os dados da máquina de busca Todocl. Essa coleção deve ser importante no meio acadêmico, uma vez que dados de uso, que são de difícil acesso por estarem em domínio de organizações privadas, também serão disponibilizados. O WIM foi utilizado para selecionar os documentos de acordo com métodos de recuperação de informação como TF-IDF e BM-25, baseados em *Pagerank* e outras medidas de relevância.

Modelo de Dados e Álgebra

Nessa seção os conceitos mais importantes do modelo WIM serão apresentados, e um exemplo será apresentado como ilustração. Muitos conceitos são adaptados da literatura de bancos de dados relacionais [Codd, 1970].

O modelo de dados WIM foi projetado e desenvolvido observando-se as seguintes propriedades:

- Realidade: o modelo não é somente abstrato, e implementações reais devem ser possíveis (para esta tese, um protótipo foi implementado).

- Simplicidade: deve ser de fácil entendimento, para facilitar implementações futuras do modelo.

- Extensibilidade: para permitir que novas tarefas de mineração de dados não implementadas até então possam ser incluídas no futuro, e ainda para permitir escalabilidade se usado em aplicações industriais.
- Representatividade: o modelo tem de representar eficientemente e eficazmente os dados presentes na Web.
- Composicionalidade: essa é uma propriedade de álgebras que garante que saídas de operações possam ser usadas em outras operações. Essa propriedade é importante porque o WIM não é somente uma coleção de algoritmos de mineração de dados, mas um modelo de dados completo com uma álgebra de grande utilidade.
- Aplicação: Além de poder ser aplicado efetivamente a dados da Web, é interessante que versões mais avançadas permitam mineração em outras bases de dados que também tenham grafos.

O WIM se baseia no conceito de relação, onde dois tipos são definidos: *relações de nodos* e *relações de links*. Relações de nodos existem para representar vértices (nodos) de grafos, como páginas da Web, termos de um documento, ou consultas ou sessões de um histórico de consultas da Web. Relações de links existem para representar arestas (links) de grafos, como links entre páginas da Web, distância entre termos de um documento, similaridade de consultas, ou cliques de um histórico de consultas.

O conceito de relação de nodos é uma generalização do conceito de relação do modelo relacional. Ao invés de permitir vários atributos em uma relação, somente dois atributos são aceitos. O primeiro atributo representa a chave primária, e é chamado de K (*Key*). O segundo atributo representa um valor associado à chave, e é chamado de V (*Value*).

Por outro lado, relações de links possuem três atributos. O primeiro atributo representa nodos de início de um link, e é chamado de S (*Start*). O segundo atributo representa nodos de fim de um link, e é chamado de E (*End*). Individualmente, S e E representam nodos do grafo. Juntos, eles representam arestas do grafo. O terceiro atributo representa um valor, ou rótulo, associado à aresta, e é chamado de L (*Label*).

Apesar da restrição ao número de atributos em relações, WIM possui um mecanismo para associar atributos de diferentes relações, a partir de seus atributos chaves. *Conjuntos de Relações* são usados para este propósito. Em termos gerais, cada conjunto de relações possui um *conjunto cheio de valores chaves*, de forma que qualquer relação em uma base de dados WIM pertence a este conjunto de relações se todos valores de chaves da relação são encontrados no conjunto cheio de valores chaves do conjunto de relações.

Um conjunto de relações não pode possuir relações de diferentes tipos. Dessa forma, pode-se também concluir que conjuntos de relações têm tipos: *conjunto de relações de nodos* e *conjunto de relações de links*. Para o último, o conjunto cheio de valores chaves é visto como a união dos valores dos atributos de início e de fim.

Dois conjuntos de relações de diferentes tipos (nodo e link) são chamados de *compatíveis*, quando o conjunto cheio de valores chaves do conjunto de relações de links é igual ou é subconjunto do conjunto cheio de valores chaves do conjunto de relações de nodos. O conceito de compatibilidade também pode existir entre um conjunto de relações de nodo e os atributos ou de início ou de fim (independentemente) do conjunto de relações de links. Este conceito é importante para representar grafos bipartidos.

O modelo WIM também possui uma álgebra, que define *operações* sobre relações. A álgebra WIM faz uso de todos os conceitos apresentados até aqui para que as saídas das operações possam ser reutilizadas como novas relações, fazendo com que o modelo tenha a propriedade de composicionalidade. Dessa forma, operações WIM retornam novas relações que vão pertencer ao mesmo conjunto de alguma relação da entrada, ou que sejam compatíveis com o conjunto da entrada.

Operadores da álgebra do WIM estão divididos em duas classes: operadores de *manipulação de dados* e operadores de *mineração de dados*. Operações típicas de bancos de dados como seleção, agrupamento, interseção, união, diferença e junção, além de operações para cálculos matemáticos e estatísticos, são cobertas pelos operadores de manipulação de dados do WIM. A outra classe consiste de operações de mineração de dados, como agrupamento, classificação, regras de associação e análises de links [Liu, 2007], além de alguns métodos de recuperação de informação, como comparação de documentos e busca.

Um *programa* WIM é um conjunto de operações executadas em sequência. Tipicamente, a relação retornada pela última operação em um programa WIM contém os dados resultantes da mineração requerida pelo usuário.

Operações WIM possuem duas propriedades bem importantes. A primeira é que qualquer atributo de valor de uma relação em um conjunto pode ser usado como atributo de outras relações no mesmo conjunto. Isso significa que é possível escolher uma relação para se tomar as chaves, e outra para se tomar os valores a serem minerados. Logo, é possível fazer uma operação de seleção, por exemplo, e em seguida aplicar um método de mineração sobre valores que representam somente o subconjunto de chaves retornadas pela operação de seleção. Porém, os valores são tomados de uma outra relação.

A segunda propriedade importante é a possibilidade de se usar valores de uma relação

```
// Agrupando duplicatas para as coleções velha e nova:
relDupOld = Compare(relOld, sparse, exactmatch, text.V);
relClOld = Disconnect(relOld, connected, relDupOld.V);
relDupNew = Compare(relNew, sparse, exactmatch, text.V);
relClNew = Disconnect(relNew, connected, relDupNew.V);
// Comparando as coleções:
relSearch = Search(relOld, relNew, shingles, 20%, relClOld.V, relClNew.V);
// Eliminando filhos que têm a mesma URL dos pais:
relSearchUrl = CompGraph(relSearch.relSearch, exactMatch, relUrlOld.V,
       relUrlNew.V);
relSeDifUrl = Select(relSearch, value, relSearchUrl.V, ==, 0);
// Transformando documentos em instâncias:
relStart = Set(relOld, relSearch, intersection, relClOld.K, relSeDifUrl.S);
relStartInst = Aggregate(relOld, grouping, count, relStart.V);
relEnd = Set(relNew, relSearch, intersection, relClNew.K, relSeDifUrl.E);
relEndInst = Aggregate(relNew, grouping, count, relEnd.V);
// Juntando nodos instância com o grafo de similaridade:
relGenEnd = Set(relSearch, relNew, intersection, relSeDifUrl.E, relEndInst.K);
relGenSt = Set(relSearch, relOld, intersection, relGenEnd.S, relStartInst.K);
// Selecionando somente um pai por filho:
relGenFinal = Aggregate(relSearch, grouping, count, relGenSt.E);
```

Figure 1. Programa WIM para estudar a evolução do conteúdo textual da Web.

de nodos em operações aplicadas a uma relação de links que seja compatível com a relação de nodos. Dessa forma, atributos de relações de nodos são usados como se fossem atributos de relações de links, ou seja, rótulos de grafos.

A figura 1 apresenta um exemplo real de um programa WIM. Esse programa, que possui apenas algumas linhas de código, implementa todos os algoritmos usados para o estudo da evolução do conteúdo da Web (conforme já mencionado nesse resumo e apresentado em Baeza-Yates et al. [2008]), cuja implementação independente feita em C possui certa de 2.500 linhas de código. Quatro outros casos de uso podem ser encontrados na tese.

Não está no escopo desse artigo apresentar a função de cada operador, ou definir a sintaxe do WIM. Em termos gerais, a figura 1 mostra uma operação por linha. Em cada linha, antes do sinal de igualdade aparece o nome da relação de saída. Logo em seguida aparece o nome do operador da álgebra WIM que está sendo empregado. Entre parênteses aparecem os parâmetros, que são específicos para cada operador. Os parâmetros incluem o nome dos conjuntos de relações envolvidos, opções específicas do operador, e as relações

e atributos requisitados. Por exemplo, na primeira linha, o operador *Compare* é aplicado ao conjunto de relações *relOld*, com a opção *sparse*, sub-opção *exactmatch*. A relação solicitada chama-se *text*, e o atributo utilizado é o de valor (V).

A figura 2 apresenta uma ilustração das relações e operações existentes no programa WIM da figura 1. Os conceitos apresentados nessa seção serão exemplificados em termos gerais por meio dessa ilustração na figura 2, que é guiada pelo programa WIM da figura 1. A ilustração consiste de dois conjuntos de relações: *relOld* e *relNew*, representados em negrito. Em geral, as relações de saída não estão organizadas de forma que se possa identificar os seus conjuntos, pois o enfoque é na forma como operadores modificam relações e geram novas saídas.

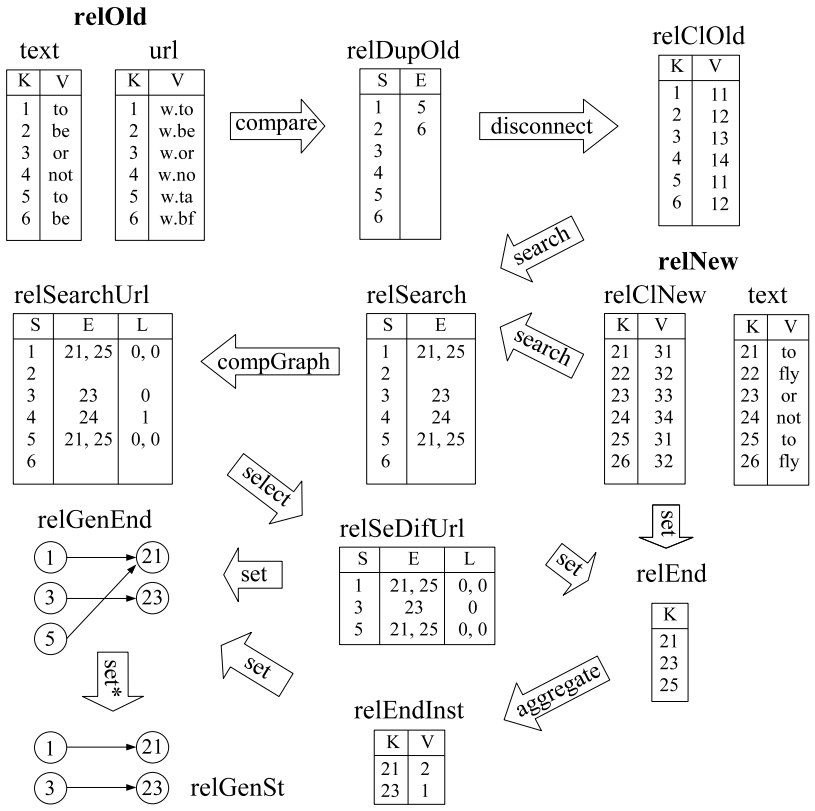


Figure 2. Ilustração de relações e operações de um programa WIM.

Inicialmente, a relação *text* do conjunto *relOld* é usada como entrada do operador *Compare*, retornando a relação *relDupOld*. Observe que *relOld* é uma relação de nodos, mas *relDupOld* é uma relação de links, que é compatível com *relOld*. Em seguida, o operador *Disconnect* é aplicado e retorna a relação *relClOld*. Veja que *relClOld* possui o mesmo conjunto de chaves de outras relações no conjunto *relOld*, o que é automaticamente

identificado pelo WIM, uma vez que a álgebra está previamente definida. Logo, *relClOld* fará parte do conjunto *relOld*.

Em seguida o operador *Search* é aplicado à dois conjuntos: *relOld* e *relNew*. Observe que a saída, *relSearch*, representa um grafo bipartido, onde o atributo de início (S) é compatível com o conjunto *relOld*, e o atributo de fim (E) é compatível com o conjunto *relNew*. Em seguida, o operador *CompGraph* é aplicado, gerando *relSearchUrl*, que vai para o mesmo conjunto de *relSearch*, e tem um novo atributo para representar um rótulo do grafo. Na sequência, uma operação de seleção elimina arestas cujo rótulo não seja igual a zero, resultando em *RelSeDifUrl*.

Uma operação de interseção de conjuntos (*Set*) é aplicada a duas relações de tipos diferentes, resultando em *relEnd*. Logo uma operação de agregação (*Aggregate*), e outras duas operações de conjuntos (*Set*), geram o resultado do programa, *relGenSt*, representado graficamente na figura. A relação *relGenSt* leva informação semântica importante como resultado da mineração. Nesse caso, essa saída representa documentos no atributo de início que geram conteúdo nos documentos do atributo de saída, ou seja, relações entre documentos pais (fontes de reuso de conteúdo) e filhos (gerados a partir de conteúdo antigo) [Baeza-Yates et al., 2008].

Conclusões

Encontrar informação na Web de forma automática é certamente uma tarefa desafiadora, que por natureza nunca terá uma solução ótima. A tese resumida nesse artigo representa um avanço importante para a solução parcial do problema de se encontrar informação na Web, e de automatizar tal processo. Portanto, a primeira contribuição importante da tese é teórica, ao mostrar ser possível prover um modelo de dados e uma álgebra que atendem de forma eficaz e eficiente as necessidades de mineração na Web.

A segunda contribuição importante tem caráter prático. Inicialmente, porque o WIM não é somente um modelo abstrato, mas um modelo que pode ser implementado em diferentes escalas. Nessa tese uma arquitetura foi implementada como um protótipo e testada, onde se demonstrou de forma prática que o modelo abstrato proposto é realmente aplicável aos problemas do mundo real. A outra contribuição prática da tese foi a experimentação feita com dados Web reais, e todas as conclusões que resultaram de trabalhos experimentais, sumarizadas nesse artigo na seção .

Finalmente, entende-se que a terceira contribuição importante da tese diz respeito ao impacto futuro que ela pode gerar. Por um lado, porque alguns dos trabalhos práticos

propostos e até já implementados pelo WIM mostrados na seção ainda estão em desenvolvimento, e podem emergir como contribuições experimentais importantes nas áreas de busca na Web e mineração de dados. Por outro lado, porque o projeto WIM continua. Uma implementação industrial está atualmente sendo projetada, podendo gerar novas publicações ou um produto no futuro.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Motivation

The World Wide Web is nowadays consolidated as the most important public source of information in the world, due to two main reasons. First, because in contrast to other media like radio and television, the Web is free for everybody to publish. Second, because of its accessibility, each user is free to choose where to navigate among a huge number of pages.

Although the Web seems to be properly shaped in order to allow humans to achieve their information need – for instance –, through search engines, the processing of a sample of the Web to discover patterns and return valid and useful information is not established yet. The reason is that corporations' information need, although probably published on the Web, is expensive to be processed and then discovered, as a consequence of the lack of special engines for Web data mining.

We define Web data mining, or simply Web mining, as the process of discovering useful information in Web data, by means of data mining techniques. The cost of finding out the right information is not associated to the lack of data mining techniques, but to the complexity of managing the required Web data, and to efficiently and effectively employing well known data mining techniques. With the goal of dropping the cost of Web mining, hence making the mining process more accessible, this thesis represents one of the first contributions towards the creation of a mining engine for the Web.

Web mining can be divided into three different types according to analysis targets [Liu, 2007; Chakrabarti, 2002], which are Web content mining, Web structure mining

and Web usage mining. *Web content mining* is the process of discovering useful information from the content of Web pages. *Web structure mining* is the process of using graph theory to analyze the node and the connection structure of a Web site or among Web sites. *Web usage mining* is the application that uses data mining to analyze and discover interesting patterns of user's usage data on the Web. Usage data is obtained when the user browses or makes transactions on the Web.

Web mining is a computation intensive task even after the mining tool itself has been developed. However, most mining software is developed in an ad-hoc manner and is usually not scalable nor reused for other mining tasks. Data mining, and in particular Web data mining, is an iterative process in which prototyping plays an essential role in order to easily experiment with different alternatives, as well as incorporating the knowledge acquired during previous iterations of the process itself. In order to facilitate prototyping, an appropriate level of abstraction must be provided to the user or programmer carrying out the Web data mining task.

Motivated by the lack of a framework to allow such abstraction, the central part of this thesis comprises the development of a model for intensive Web mining prototyping, which is referred to as the WIM – Web Information Mining – model. The WIM model includes a data model and an algebra. The data model is an abstract model that describes how data is represented and accessed. The algebra is composed by a set of operators for consistent data manipulation and mining. The data model and the algebra are specially designed to manage the three types of Web data in combination: documents, structures between documents, and usage data.

Considering the main computer science subfields, this thesis is located in the database subfield, and arises in a time when new types of data must be managed and new models need to be provided for new application demands. In May 2008, a representative group of database researchers met to discuss the state of database research and its impacts on practice [Agrawal et al., 2008]. It was the seventh meeting of this sort in twenty years. We present some of their conclusions and position WIM among the research lines they point out as challenging and needed for today's database applications. Basically they point out five main research topics that deserve special attention in coming years. Reasonably, WIM is closely related to three of them, as we present below.

The first research opportunity topic is referred to as "Revisiting Database Engines". They say "there are many popular data-intensive tasks from the last decade for which relational databases provide poor price/performance and have been rejected: critical scenarios

include text indexing, serving Web pages, and media delivery. New workloads are emerging in sciences and Web 2.0-style applications, among other environments, where database engine technology could prove useful, but not as bundled in current database systems".

Although WIM inherits the concepts around relational databases, it is not internally similar. For instance, the WIM storage architecture is column-oriented. On the other hand, WIM is not a complete database engine. For instance, it does not have a specific file system. Anyway, WIM is an alternative for a critical scenario to which relational databases cannot be employed, which is the Web mining world.

The second research opportunity mentioned in Agrawal et al. [2008] is called "Declarative Programming for Emerging Platforms". They say: "Programmer productivity is a key challenge in computing. [...] Today, the urgency of the problem is literally increasing exponentially as programmers target ever more complex environments [...]".

The WIM programming language, provided by its algebra, is a kind of declarative language that reduces code size by up to two orders of magnitude, which is one of the main contributions of our research. They say "There is a need for 'synthesis' work here to harvest useful techniques from the literature [...]". WIM operators embed a collection of data mining techniques. They also say "It is a unique opportunity for a fundamental 'reformation' of the notion of data management: not as a storage service, but as a broadly applicable programming paradigm". WIM seems to have achieved this goal for the Web mining field.

The third research opportunity pointed out, which is closely related to WIM, is referred to as "The Interplay of Structured and Unstructured Data". The title is clear, though they highlight: "A significant long-term goal for our community is to transition from managing traditional databases consisting of well-defined schemata for structured business data, to the much more challenging task of managing a rich collection of structured, semi-structured and unstructured data, spread over many repositories in the enterprise and on the Web". WIM also represents a contribution is this research field, with a proposal of integrating structured and unstructured data.

1.2 Context of the Thesis – Web Mining

According to Chakrabarti [Chakrabarti, 2002], Web mining is about finding significant statistical patterns relating hypertext documents, topics, hyperlinks, and queries, and using these patterns to connect users to the information they seek. The Web has become a vast

storehouse of knowledge, built on a decentralized yet collaborative manner. On the negative side, the heterogeneity and lack of structure makes it hard to frame queries and satisfy information needs. For many queries posed with the help of keywords and phrases, there are thousands of apparently relevant responses, but on closer inspection these turn out to be disappointing for all but the simplest queries.

The data to be mined is very rich, comprising texts, hypertext markups, hyperlinks, sites, and topic directories. This distinguishes Web mining as a new and exciting discipline, although it also borrows concepts from traditional data analysis.

Roughly speaking, Web Mining is the extraction of interesting and potentially useful patterns and implicit information from artifacts or activities related to the World-Wide Web. As already mentioned, there are three knowledge discovery domains that pertain to Web mining: Web content mining, Web structure mining, and Web usage mining.

Web content mining: it is an automatic process that goes beyond keyword extraction. Since the content of a text document presents no machine-understandable semantics, some approaches have suggested to restructure the document content in a representation that could be exploited by machines. There are two groups of Web content mining strategies: those that directly mine the content of documents and those that improve on the content search of other tools like search engines.

Web Structure Mining: the World-Wide Web can reveal more than just the information contained in documents. For example, links pointing to a document indicate its popularity, while links coming out of a document indicate its richness or perhaps the variety of topics covered by it. This can be compared to bibliographical citations. When a paper is cited often, it is probably important. The PageRank [Brin and Page, 1998] and the Kleinberg's HITS [Kleinberg, 1999] algorithms take advantage of this information conveyed by the links to find pertinent Web pages.

Web Usage Mining: Web servers record and accumulate data about user interactions whenever requests for resources are received. Analyzing access logs of different Web sites can help understand the user behavior and the Web structure, thereby improving the design of this colossal collection of resources. There are two main tendencies in Web Usage Mining driven by its applications: general access pattern tracking and customized usage tracking [Wang, 2003].

The general access pattern tracking analyzes the Web logs to understand access patterns and trends. These analyses can shed light on better structure and grouping of resource providers. Customized usage tracking analyzes individual trends. Its purpose is to cus-

tomize Web sites to users. The information displayed, the depth of the site structure and the format of the resources can all be dynamically customized for each user over time based on their access patterns.

1.3 Objectives of the Thesis

The main objective of this thesis is the development of WIM, a data model and algebra with an associated software prototype for Web mining. The goal of WIM is to facilitate the task of the Web miner when prototyping Web mining applications. In order to achieve this main objective, and in the future to have Web miners using the WIM software, we highlight three main research challenges, whose solution is a step towards the conception of WIM. They are: the WIM data model, the WIM algebra, and the WIM architecture and implementation.

The *WIM data model* refers to how data is represented and accessed. Notice that WIM deals with Web data, and we need to provide an abstraction to the user as a view of the data. Our choice was to create an adaptation from the well-known relational tables, in order to represent nodes of a graph, such as the documents on the Web, and the edges of this graph, such as hyperlinks connecting Web documents. The WIM language is based on dataflow programming, which means that operators are called in sequence and that parallel processing is allowed whenever independent tasks are requested.

The *WIM algebra* refers to the set of operators, including their syntactic and semantic aspects, which allow to manipulate and mine Web data. The challenge in designing the WIM algebra is not only concentrated on defining the set of operators, but also on how to guarantee the composition between operator's input and output, which is important to allow users to request any operator output to be input of any other existing operator, having as the only constraint the data types of the input and output.

The *WIM architecture and implementation* is highly important at this conceptual stage as a proof of concept of the WIM data model and algebra. Although we have implemented a prototype, it has properly worked for medium-sized datasets and helped test the WIM model by means of use cases. This practical challenge is related to the other three challenging aspects presented above, as the design of the WIM model needed to take into account the fact that the model was not abstract, rather, it had to be implemented and executed in the real world.

WIM was not conceived to solve all Web data mining problems. On one hand,

Web mining is an overloaded terminology used to different purposes. As we refer to Web mining as the application of data mining techniques to Web data, the WIM algebra was designed observing this definition. For instance, WIM was not conceived for intensive data processing only, like Hadoop [Hadoop, 2008] (with its associated language Pig Latin [Olston et al., 2008]), and MapReduce [Dean and Ghemawat, 2008] (with its language Sawzall [Pike et al., 2005]), which are frequently classified as data mining frameworks due to their capability of finding pieces of data among very large datasets, rather than using specific data mining techniques.

On the other hand, because there is a large number of data mining techniques that have been applied to Web data [Kosala and Blockeel, 2000; Liu, 2007], we have included in the first version of the WIM prototype some of the most common techniques, which, according to what we believe, are the most important ones for Web mining applications. The WIM algebra currently covers the following techniques: association rules and sequential patterns mining; unsupervised learning tasks such as k-means clustering and clustering based on graph structure and text; search and text comparison; and link analysis, with co-citation analysis and document relevance.

Through a complementary set of data manipulation operators, WIM allows different data mining techniques to be used sequentially for the same application, hence integrating the data. Furthermore, WIM is designed to easily allow the addition of new operations to implement new data mining techniques. Thus, as soon as the first version of the WIM tool is made available, we expect users with new needs to collaborate with the implementation of new techniques, to be integrated into the WIM tool, gradually improving its coverage of the Web mining world.

Another important objective of this thesis is the analysis of some Web mining applications. This objective was proposed in order to demonstrate that we retain enough knowledge regarding Web data mining, which is the application field of our research, and to guarantee that WIM is applicable to a set of real Web mining applications, allowing comparison between an ad-hoc and a WIM implementation. Having the analysis of some Web mining problems as one of the objectives of this thesis increases the chances of proposing a model that will actually be useful in the future.

The main application described in this thesis concerns a study on the evolution of Web content. The WIM program is implemented to identify parent documents, which are sources of copy, and child documents, which are more recent documents containing old content. Problems like duplication detection, URL comparison, association of a unique

parent to each child, are all addressed in this application. We have implemented the same solution both in an ad-hoc manner [Baeza-Yates et al., 2008] and using WIM, and we present a comparison between these solutions.

The second application concerns the manipulation of search engine usage logs in order to implement a document usage-based relevance weight, with the general objective of improving query ranking. For this application we used a Yahoo! search engine log with 22 million clicks and imported the regular pagerank from a Web dataset from the United Kingdom, containing 77 million entries[1].

The other applications are: i) a comparative study of linkage evolution between new pages with new content and new pages with old content, in which we have found that the linkage of new pages with new content evolves more rapidly than the linkage of new duplicated pages; ii) the manipulation of query logs, in order to identify a series of properties for each distinct query that appears in the log, for studying search engine user intent; and iii) the composition of a pool of documents for relevance assessment, aiming at creating a reference dataset for research on learning to rank in Web information retrieval. WIM was used to select the documents to be included in the pool, which used different weight functions such as TF-IDF and BM25, with AND filters (conjunctions) and including the Pagerank document relevance weight.

1.4 Main Contributions of the Thesis

The main contribution of this thesis is WIM – Web Information Mining, a model materialized as a software prototype for supporting Web miners when implementing their applications. We have designed the WIM data model specially for Web data and proposed an algebra with a set of preliminary operators for data manipulation and data mining. The set of operators for data mining is expected to be extended in the future, as users start using WIM and need to employ data mining techniques that are not yet available.

In Baeza-Yates et al. [2005a] and Baeza-Yates et al. [2005b] we presented our first view of WIM, in which its underlying data model was already based on the relational model. Preliminary applications of WIM were reported in Pereira-Jr and Baeza-Yates [2005]. Recently, a more general view of WIM was published in Pereira et al. [2009]. Although that work does not present the underlying formal concepts regarding the data

[1] This collection was obtained using a large set of .uk pages downloaded in May 2006 [Boldi and Vigna, 2004] by the Laboratory of Web Algorithmics, Università degli Studi di Milano (http://law.dsi.unimi.it/).

model and the operators' definitions, it reflects the current stage of our research, as detailed in this thesis.

WIM is not only an abstract model. An important contribution of this thesis is a prototype, which has demonstrated that WIM is feasible and works properly for our proposed applications, a strong evidence that WIM will also work well for applications in other scenarios. Despite the limitations of our prototype, the software developed so far can be seen as a beta version of a future tool.

An industrial scale version of WIM is already planned to be developed, which does not mean that WIM will not be provided as an open source tool. The goal is to have both projects, the public and the private, going on concurrently, since it is fundamental to have WIM well known among Web mining researchers.

WIM has been applied to a set of Web mining problems. For one of them, a study about the evolution of the Web content and the relationship of search engines with the content evolution behavior [Baeza-Yates et al., 2008], we have performed an extensive analysis on the results, which are presented in Chapter 6. As our main contributions with respect to this application, we show that: i) a large portion of the new Web documents has old content; ii) parents are clicked more often than other documents and are more likely to become parents again than other documents; and iii) search engines are biasing the content of the Web, because users usually select sources of content from the main results returned by their queries. In Baeza-Yates et al. [2006] we studied how documents that are used as sources for a given new document appear together in query results. It was another evidence that search engines bias the content of the Web.

Another important problem to which WIM was applied is a study on how and where duplicates and near-duplicates occur on the Web [Baeza-Yates et al., 2007b; Pereira-Jr et al., 2006]. We found out that the Web seems to have up to 50% more duplicates than previously reported in the literature. This may be explained by the fact that datasets used in previous works were created by following links. Since duplicates do not have many inlinks, they are often not crawled. We used, on the other hand, a dataset from the Chilean Web, in which all the domains provided by the domain name service in Chile were used to start the crawls, so that a less biased collection was created to better represent the Web.

We also performed a study on crawling techniques to decide which new URLs to fetch, based on the assumption that duplicates (which must be avoided) do not have their indegree (or other page quality measure) evolving with time. Thus, it is probably possible

to find a function based on the linkage structure evolution to decide whether to fetch or not a new URL. We have applied WIM to study the duplicate and new content evolution, and present partial results in Section 7.2. Further analyses are proposed as future work in Chapter 8.

WIM has also been used to compose a pool of documents for relevance assessment, in order to create a reference dataset for research on learning to rank, to become publicly available. We already have 100 queries assessed, and will soon put them available together with the collection of documents, the link structure between documents, and the real query logs from the TodoCL search engine[2]. The creation of this reference dataset is also a contribution of this thesis, given the applicability of WIM and the degree of interest of this dataset to the community.

This reference dataset will be used to solve another problem to which WIM has been applied. It is a proposal of evaluating a usage pagerank based on the order of clicks on search engine results. A similar research was published in Liu et al. [2008], though in parallel we had independently proposed this research topic, as reported in Pereira et al. [2008]. Finally, WIM has been applied to mine usage data with the goal of identifying user intent. The use cases mentioned here are presented in Chapter 7.

Finally, it is worth mentioning that two preliminary applications motivated the development of WIM. The first application was a study of evidence to find communities on the Web [Pereira Jr et al., 2004]. The second one was a proposal of different solutions to solve the problem of finding similar documents on the Web [Pereira-Jr and Ziviani, 2004]. More specifically, this problem may be stated as follows: for a given document A, find pieces of A that are published on the Web. From an application point of view, this problem is important to detect plagiarism, since document A may have been created after a "copy-and-paste" action, and to detect content overlap among search engines corpora.

1.5 Organization of the Thesis

The remainder of this thesis is organized as follows. Chapter 2 presents related work. Chapter 3 presents the WIM formal model, including its design goals. Chapter 4 presents the WIM algebra, including the seven data manipulation operators and the eight data mining operators, as well as a comparison between the WIM data manipulation operations and the relational algebra.

[2] TodoCL: www.todocl.cl

Towards more practical issues, Chapter 5 presents the WIM prototype architecture and some important details underlying our prototype implementation. A discussion involving efficiency and scalability issues, with alternative architectures, is also provided.

Two chapters present the use cases. In Chapter 6, as aforementioned, we present a complete study on one of the problems to which WIM was applied, which is how the Web content evolves with time, in terms of content reuse. The other four use cases to which WIM was applied to solve real Web mining problems are presented and discussed in Chapter 7. Finally, Chapter 8 presents the conclusions and future research challenges motivated by WIM and its applications.

Chapter 2

Related Work

There are two research lines that are related to WIM. The first research line is about data mining tools. Although they are not specially designed for Web data, they may be able to cover some Web mining applications. The second line is about query languages for Web data, which generally are not designed for mining tasks. Most of them are either based on SQL or are built on SQL. In this chapter we compare WIM with some of these tools. We also present the work related to one of the problems to which WIM was applied, which is a study on how Web content evolves through time, in terms of reuse. Then, we present some related data mining techniques that are important for this thesis.

2.1 Data Mining Tools

As for data mining frameworks, some commercial SQL-based database management systems have add-ons for data mining modules. The data mining extension of Microsoft SQL Server 2005 [2008] proposes to integrate traditional operations performed using SQL with the mining need. The data mining techniques that Microsoft SQL Server 2005 supports are: decision trees, association rules, sequence clustering, time series, neural networks, and text classification and categorization.

Another solution is provided by Oracle Data Mining [2008], a tool for application developers using SQL and Java APIs that automatically mines Oracle databases and deploys results in real-time throughout the enterprise. Oracle Data Mining models can be included in SQL queries and embedded in applications. The data mining techniques supported are related to classification, regression, anomaly detection, attribute importance assignment,

association rules, clustering, and feature extraction.

Similarly to Microsoft and Oracle, IBM offers DB2 Intelligent Miner [2008]. IBM's database mining capabilities integrate with existing systems to provide predictive analysis without moving data into proprietary data mining platforms. It is possible to use SQL, Web Services or Java to access DB2 Intelligent Miner from applications. The system, which is divided in modules for scoring, modeling and visualization, includes the following data mining approaches: association rules, classification (neural and tree classification), clustering, prediction mining, and sequential patterns mining.

There are several business intelligence tools with integrated data mining support. Examples are Angoss[1], Infor CRM Epiphany[2], Portrait Software[3], and SAS[4]. One difference of these solutions compared to WIM is that they are commercial tools driven to business applications. We note that all solutions have similar coverage in terms of mining capability, which is not focused on Web. Furthermore, WIM has been conceived considering more research than business applications, although it can easily be extended to business problems with the addition of new operators to its algebra.

Weka [Witten and Frank, 2005] is a well known data mining framework that implements a collection of machine learning algorithms for data mining tasks. It was developed for educational purposes and is an open source software. The algorithms may either be applied directly to a dataset or called from Java code. With respect to its differences to WIM, Weka is not designed for Web data and does not provide a specific language to perform data manipulation, as it is an isolated academic collection of algorithms.

2.2 Web Query Languages

From 1994 to 1998 several database research groups invested in designing and developing query languages for Web data, sometimes aiming at integrating semistructured and structured data. One of the first research initiatives was TSIMMIS [Chawathe et al., 1994; Garcia-Molina et al., 1997; Hammer et al., 1995], a project to develop tools that facilitate the rapid integration of heterogeneous information sources, including both structured and semistructured data. TSIMMIS has components to translate queries and data (source wrappers), to extract data from Web sites, to combine information from several sources

[1] www.angoss.com
[2] go.infor.com/inforcrm
[3] www.portraitsoftware.com
[4] www.sas.com

(mediator), and to allow browsing data sources over the Web. TSIMMIS uses a graph-based model that encapsulates data and structural information rather than a traditional one that requires a pre-defined schema. The type of the data is interpreted by the user from labels in the structure.

Another parallel initiative was W3QL [Konopnicki and Shmueli, 1995, 1998]. W3QL addresses the structure and content of Web nodes and their various types of data. A system called W3QS was implemented to execute W3QL queries. W3QL includes a crawler, considering the data dynamically published on the Web, rather than statically and locally stored. It also supports meta-search, by means of a process to automatically fill out forms. Its concept of view allows the system to keep query results up-to-date, by rerunning the queries with the frequency required by the user. The applications to which W3QL has been employed are quite basic, like searching for the figures in a given site or identifying broken links.

WebLog [Lakshmanan et al., 1996] is a Datalog-like language to query and manipulate the internal structure of HTML documents on the Web. WebLog is said to be capable of exploiting the partial knowledge users might have on the information being queried and dealing with the dynamic nature of information on the Web. Differently of most of the other languages, WebLog syntax is not based on SQL.

WebSQL [Arocena et al., 1997; Mendelzon et al., 1997] integrates textual retrieval with structure and topology-based queries. It proposes a theory of query cost based on the idea of query locality, that is, how much of the network must be visited to answer a particular query. The query locality proposal is a consequence of the fact that WebSQL allows queries on hyperlink paths among Web pages. WebSQL divides hyperlinks into three categories: internal links (within a page), local links (within a site), and global links. It is also possible to define new link types based on anchor texts, for example, links with anchor text "next".

The ARANEUS project [Atzeni et al., 1997b; Mecca et al., 1998] proposes a Web-base management system, with support to: i) queries, on structured and semi-structured data; ii) views, to reorganize and integrate data from heterogeneous sources; and iii) updates, to allow the maintenance of Web sites. It proposes a new data model for Web documents and hypertexts, and proposes languages for wrapping and creating Web sites, including techniques for publishing data on the Web. ARANEUS includes two different languages: ULIXES [Atzeni et al., 1997a] and PENELOPE. ULIXES is used to build database views of the Web, which can then be analyzed and integrated using database

techniques. PENELOPE allows the definition of derived Web hypertexts from relational views, which can be used to generate hypertextual views over the Web.

StruQL [Fernandez et al., 1997b] is the query language of the Strudel Web site management system [Fernandez et al., 1997a]. Even though StruQL was developed in the context of a specific Web application, it is a general purpose query language for semistructured data, based on a data model of labeled directed graphs. In Strudel, StruQL was used for two tasks: querying heterogeneous sources to integrate them into a site data graph, and for querying this data graph to produce a site graph.

WebOQL [Arocena and Mendelzon, 1998] is a language that supports a general class of data restructuring operations in the context of the Web. The main data structure provided by WebOQL is the hypertree, which are ordered arc-labeled trees with two types of arcs, internal and external. Internal arcs are used to represent structured objects and external arcs are used to represent references (typically hyperlinks) among objects. Arcs are labeled with records. Such a tree could be built, for example, from an HTML file, using a generic HTML wrapper. Sets of related hypertrees are collected into Web datasets. Both hypertrees and Webs can be manipulated using WebOQL and created as the result of a query.

In comparison to the other query languages mentioned so far, the ARANEUS' languages ULIXES and PENELOPE [Mecca et al., 1998], and the languages WebOQL [Arocena and Mendelzon, 1998] and StruQL [Fernandez et al., 1997b] represent the second generation of Web query languages, which have the additional ability to manipulate Web data and create new complex structures as the result of a query.

Lim and Ng [1997] present an integrated SQL query interface for Web search engines, bibliographic and relational databases. Web search engines are defined as virtual Web tables. They define a schema for each search engine, so that its results can be treated like relational tables, and then integrated to bibliographic and relational databases.

Acting as a meta-search engine as well, RAW [Fiebig et al., 1997] is a proposal for extending the relational algebra in order to process queries against the Web. Additionally to the known data types (int, bool, float, string), three new data types to deal with URLs, HTML-documents or fragments, and path expressions are proposed.

Whoweda [Bhowmick et al., 1998, 2000, 2002; Ng et al., 1998] is a metadata repository of useful Web information, available for querying and analysis. As relevant information becomes available on the Web, it is coupled from various sources, translated into a common Web data model (Web Information Coupling Model), and integrated with existing

data in Whoweda. At the warehouse, queries can be answered and Web data analysis can be performed quickly and efficiently since the information is directly available. Whoweda consists of two major components: a data manipulation module called Web Information Coupling System (WICS) and a data mining module called Web Information Mining System (WIMS). WICS focuses on the manipulation of information in the Whoweda system. It includes extraction and retrieval of information from the Web, its storage and organization, and manipulation via Web operators such as Web select, Web join and Web project. The information are fed into WIMS for mining by association rules.

The WEBMINER tool [Cooley et al., 1997] provides a query language on top of external mining software for association rules and for sequential pattern mining. It is based on the adaptation of an existing miner to a particular problem. More precisely, a pre-processing algorithm groups consecutive page accesses by the same visitor into a transaction, according to some criterion. Then, a miner for association rules or sequential patterns is invoked to discover similar patterns among the transactions. The association rules' miner has been further customized to guarantee that no patterns are erroneously skipped.

In the same line as WEBMINER, WUM (Web Utilization Miner) [Spiliopoulou and Faulstich, 1998] is a system for discovering interesting navigation patterns. MINT is the WUM mining language, which supports the specification of statistical, structural, and textual criteria. To discover the navigation patterns satisfying the expert's criteria, WUM exploits an aggregated storage representation for the information in the Web server log.

We observe that Whoweda [Ng et al., 1998], WEBMINER [Cooley et al., 1997] and WUM [Spiliopoulou and Faulstich, 1998] are more advanced tools in comparison to the others, as they not only address the collection, access and simple manipulation of Web data, but also include features for mining, even though the mining tasks are simple and restrictive, limited to mining by association rules and sequential patterns.

Squeal [Spertus and Stein, 2000] is a language built on SQL to facilitate structure-based queries. Examples of the mentioned structure-based queries are: which pages are pointed to by two given pages, what are the titles of pages that point to my home page, what are the most linked to pages containing a given phrase, and which pages have the same text as my home page but appear on a different server. The authors do not present details of the Squeal data model or implementation issues.

WebBase [Cho et al., 2006; Raghavan and Garcia-Molina, 2003a] is an important

Web warehouse project that also incorporates a query language, whose main goals are to manage large collections of Web pages and to enable large-scale Web-related research [Raghavan and Garcia-Molina, 2003b]. They view a Web warehouse simultaneously as a collection of Web documents, as a navigable directed graph, and as a set of relational tables storing Web page properties. Thus, the Web warehouse is modeled as a collection of pages and links, with associated page and link attributes. The model incorporates the ranking of pages and links.

Wood and Ow [2005] developed an SQL extension that allows corporate databases to be joined to explicit information contained on corporate or external Web sites. Their implementation is called WEBVIEW and, as an example, they show how to mimic a search for a popular book using the Froogle[5] price comparison service, using its ISBN number, from which WEBVIEW covered 88 Web pages and retrieved 871 prices from various book sellers.

A survey of database techniques for the Web is presented by Florescu et al. [1998], covering the projects presented so far and a couple of other projects for unstructured data, rather than Web data only projects.

2.3 Comparison of WIM with Data Mining Tools and Web Query Languages

WIM does not fit any of the aforementioned classes of tools and languages. WIM is specific to Web mining, not general and not for business-driven data mining like the tools presented in Section 2.1. On the other hand, WIM is not simply a language for retrieval and manipulation of Web data. Actually, WIM does not address the problem of data acquisition. It is applied to static datasets, leaving the tasks of crawling and pre-processing to existing specific tools.

As we shall present in Chapter 3, the WIM data model is not directly derived from any Web query language data model presented earlier. First, because there is no standard; each language uses its own data model and there seems to exist no consensus among researchers with respect to which data model is the best or the most suitable for each scenario. Second, because our goals are different and, although we also handle Web data, our final objective is mining this kind of data, not simply representing, storing or manipulating it. However, the WIM data model merges concepts inherited from 90's Web data models and from search

[5] http://froogle.google.com or http://www.google.com/products

technology. The result is a data model able to deal with large scale datasets, for which more complex queries are allowed than simple search engine keyword queries. Furthermore, mining operations are available in WIM, by means of a consistent and extensible algebra.

Given the importance of WebSQL as a reference Web data model, we shall present the key differences with respect to WIM. Both WebSQL and WIM inherit concepts from the relational model. As we shall notice in Chapter 3, designing WIM based on concepts from the relational model has made it simpler and more easily comprehensible.

As aforementioned, one difference is how the two models view the Web. WebSQL views the Web as a collection of world-wide distributed documents, where querying these documents means accessing remote systems, even though indexes of the Web are allowed as source of content. WIM views the Web as datasets. Different Web datasets can represent documents in different domains, or even documents in different snapshots for a given domain.

In WebSQL, every element of a relation is identified by its URL. This means that objects that cannot be represented by a URL cannot be modeled. Some objects in HTML documents have an associated URL, like images, videos and, of course, hyperlinks, but others do not. For instance, the terms of a document, which are part of a vocabulary, cannot be directly represented. WIM supports the existence of objects that cannot be represented by URLs. As an illustration, the vocabulary of a Web dataset can be represented as a relation and used in conjunction with the Web relation for a given application. Although WIM has been applied so far only to Web datasets, other datasets, in which attributes of edges and nodes of a graph must be represented, may also be modeled using WIM.

An important consequence is that WIM can model other types of Web data that WebSQL cannot. For instance, in the early 90's Web usage data was not seen as a relevant Web data. Further, the Web was not interactive as today and user's tags, which are user-specific comments posted after visiting Web pages, did not exist. In Chapter 7 we present examples of WIM applications that handle usage data. By modeling types of data, WIM is also suitable to represent tags, although we have not used this kind of data in any WIM application so far. The ability of WIM in dealing with these new types of Web data is the most important advantage of WIM in comparison to WebSQL.

Figure 2.1 presents a comparison among Web query languages, data mining tools, and WIM, in terms of how structured and how clean (i.e., not raw) is the data that each class of tools proposes to manage. Web data is usually in its most raw and unstructured sort. Data mining tools are adequate for structured (corporate) data. Web query languages may

manipulate data as unstructured and raw as on the Web, providing a view of data that is more structured and not that raw as on the Web. WIM is developed for data previously crawled and pre-processed, that is, cleaner than data handled by Web query languages, but as structured as the data the query languages manipulate.

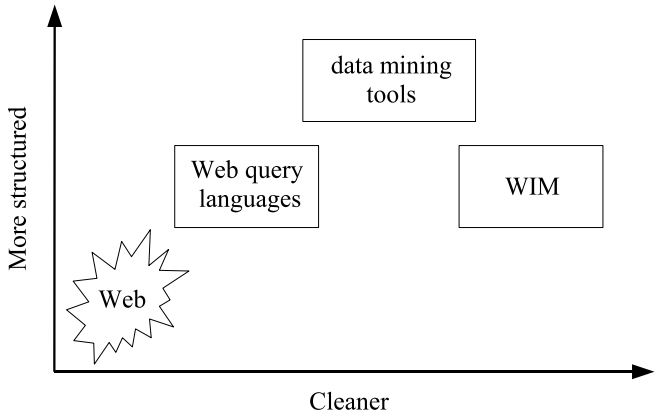


Figure 2.1. Comparison of Web languages, data mining tools, and WIM, in terms of how structured and how clean is the data that each class of systems manages.

2.4 Evolution of Web Content

In this section we present some works related with one of the applications to which WIM has been employed, which is presented in Chapter 6. We start with a couple of works that study the evolution of Web pages and finish with works about Web archiving. Despite the importance of these papers as related work, they have different approaches and different objectives, in comparison to our research on studying the evolution of the Web content.

Ntoulas et al. [2004] studied some aspects of the Web's evolution, such as birth, death, and replacement of documents. They crawled all pages from 154 sites on a weekly basis, for a period of one year. In a similar work using the same data set, Ntoulas et al. [2005] found that after a year about 60% of the documents and 80% of the links on the Web are replaced.

Cho and Roy [2004] studied the impact of search engines on the popularity evolution of Web documents. Given that search engines currently return popular documents at the top of search results, they showed that newly created documents are penalized because these documents are not well known yet. Pandey et al. [2005] proposed a simple solution

to this problem, based on the introduction of a controlled amount of randomness into search result. Baeza-Yates et al. [2004] showed that PageRank [Page et al., 1998] is biased against new documents.

On the other hand, Fortunato et al. [2006] showed that popular sites receive far less traffic than predicted, suggesting that the possible bias introduced by search engines does not lead to monopoly of information.

In a recent work, Toyoda and Kitsuregawa [2006] proposed the "novelty measure" to estimate if a newly linked URL is really new or if it is old but was not crawled in previous snapshots. The novelty measure is applied to an archive search engine, where new pages can be identified. Zhang and Suel [2007] proposed a general framework for indexing and query processing of archival collections. By storing the documents in parts, and considering that in archiving a great portion of the data is replicated, their approach results in significant reductions in the index size and query processing cost.

2.5 Related Data Mining Techniques

In this section we present some data mining techniques that WIM currently covers. Some of the concepts around these techniques shall be important when we present the WIM algebra in Chapter 4.

2.5.1 Association Rules

Association rule mining [Agrawal et al., 1993] searches for interesting relationships among items in a given data set. A typical example of association rule mining is market basket analysis. This process analyzes customer buying habits by finding associations between the different items that customers place in their shopping baskets. The discovery of such associations can help retailers develop marketing strategies by gaining insight into which items are frequently purchased together by customers. For instance, if customers are buying milk, how likely are they to also buy bread (and what kind of bread) in the same trip to the supermarket?

In the context of the Web, association rule mining is important for discovering patterns in usage data [Cooley et al., 1997; Ng et al., 1998; Spiliopoulou and Faulstich, 1998]. We now consider the definition of an association rule from Han and Kamber [2006]: Let $\mathcal{I} = \{i_i, i_2, ..., i_m\}$ be a set of items. Let D, the task-relevant data, be a set of database transactions where each transaction T is a set of items such data $T \subseteq \mathcal{I}$. Each transaction

is associated with an identifier, called TID. Let A be a set of items. A transaction T is said to contain A if and only if $A \subseteq T$. An association rule is an implication of the form $A \Rightarrow B$, where $A \subset \mathcal{I}$, $B \subset \mathcal{I}$, and $A \cap B = \emptyset$. The rule $A \Rightarrow B$ holds in the transaction set D with support s, where s is the percentage of transactions in D that contain $A \cup B$ (i.e., both A and B). This is taken to be the probability $P(A \cup B)$. The rule $A \Rightarrow B$ has confidence c in the transaction set D if c is the percentage of transactions in D containing A that also contain B. This is taken to be the conditional probability, $P(B|A)$. That is:
$support(A \Rightarrow B) = P(A \cup B)$, and
$confidence(A \Rightarrow B) = P(B|A)$.

Rules that satisfy both a minimum support threshold (min_sup) and a minimum confidence threshold (mim_conf) are called strong. By convention, support and confidence values are written so as to occur between 0% and 100%, rather than 0 to 1.0.

A set of items is referred to as an itemset. An itemset that contains k items is a k-itemset. The occurrence frequency of an itemset is the number of transactions that contain the itemset. An itemset satisfies minimum support if the occurrence frequency of the itemset is greater than or equal to the product of mim_sup and the total number of transactions in D. If an itemset satisfies minimum support, then it is a frequent itemset.

Association rule mining is a two-step process: find all frequent itemsets and generate strong association rules from the frequent itemsets, in which the rules must satisfy minimum support and minimum confidence.

2.5.2 Clustering

The process of grouping a set of physical or abstract objects into classes of similar objects is called clustering [Han and Kamber, 2006]. A cluster is a collection of data objects that are similar to one another within the same cluster and are dissimilar to the objects in other clusters. A cluster of data objects can be treated collectively as one group in many applications. In the context of the Web, clustering techniques are often used to classify documents for information discovery.

Suppose that a data set to be clustered contains n objects, which may represent persons, houses, documents, countries, and so on. Main memory-based clustering algorithms typically operate on either one of the following two data structures:

- Data matrix (or object-by-variable structure) – this represents n objects, such as persons, with p variables also called measurements or attributes), such as age, height,

weight, gender, and so on. The structure is in the form of a relational table, or *n-by-p* matrix (n objects $\times$ p variables).

- Dissimilarity matrix (or object-by-object structure) – this stores a collection of proximities that are available for all pairs of n objects. It is often represented by an *n-by-n* table.

WIM implements an important clustering algorithm called k-means [Berkhin, 2002]. The algorithm partitions a set of n objects into k clusters so that the resulting intracluster similarity is high but the intercluster similarity is low. Cluster similarity is measured in regard to the mean value of the objects in a cluster, which can be viewed as the cluster's centroid or center of gravity.

The k-means algorithm proceeds as follows. First, it randomly selects k objects, each of which initially represents a cluster mean or center. For each of the remaining objects, an object is assigned to the cluster to which it is the most similar, based on the distance between the object and the cluster mean. It then computes the new mean for each cluster. This process iterates until the criterion function converges. Typically, the square-error criterion is used. In other words, for each object in each cluster, the distance from the object to its cluster center is squared, and the distances are summed. This criterion tries to make the resulting k clusters as compact and as separate as possible. The method is relatively scalable and efficient in processing large data sets because the computational complexity of the algorithm is $O(nkt)$, where n is the total number of objects, k is the number of clusters, and t is the number of iterations.

2.5.3 Search and Text Comparison

Web search deals with the process of comparing a few keywords against a corpus of Web documents, responding to keyword queries with a ranked list of documents. The simplest kind of query one may pose involves relationships between terms and documents.

The vector space model [Salton et al., 1975] is a mean to relate terms and documents. It is based on the implicit assumption that the relevance of a document with respect to some query is correlated with the distance between the query and document. In the vector space model each document (and query) is represented in an n-dimensional Euclidean space with an orthogonal dimension for each term in the corpus. The degree of relevance between a query and document is measured using a distance function.

The most basic term vector representation simply flags term presence using bit vectors. This is known as the binary vector model. The document representation can be extended by including term and document statistics in the document and query vector representations. An empirically validated document statistic is the number of term occurrences within a document (term frequency or *tf*). The intuitive justification for this statistic is that a document that mentions a term more often is more likely to be relevant for, or about, that term.

Another important statistic is the potential for a term to discriminate between candidate documents. The potential of a term to discriminate between documents has been observed to be inversely proportional to the frequency of its occurrence in a corpus, with terms that are common in a corpus less likely to convey useful relevance information. A frequently used measure of term discrimination based on this observation is inverse document frequency (or *idf*). Using the *tf* and *idf* measures, the weight of a term present in a document can be defined as:

$$w_{t,D} = tf_{t,D} \times idf_t \tag{2.1}$$

where *idf* is:

$$idf_t = log(N/n_t) \tag{2.2}$$

where n_t is the number of documents in the corpus that contains term t, and N is the total number of documents in the corpus.

A distance function commonly used to score the distance between documents and query vectors is the cosine measure of similarity [Baeza-Yates and Ribeiro-Neto, 1999]:

$$S(D,Q) = \frac{\sum_{t \in Q} w_{t,D} \times w_{t,Q}}{\sqrt{\sum_{t \in Q} w_{t,D}^2 \times \sum_{t \in Q} w_{t,Q}^2}} \tag{2.3}$$

Okapi BM25 [Sparck-Jones et al., 2000] is another important proposal for calculating the similarity between documents and queries and is also addressed within this thesis. In Okapi BM25, documents are ordered by decreasing probability of their relevance to the query, $P(R|Q,D)$. The formulation takes into account the number of times a query term occurs in a document (tf), the proportion of other documents which contain the query term (*idf*), and the relative length of the document. A score for each document is calculated by summing the match weights for each query term. The document score indicates the

Bayesian inference weight that the document will be relevant to the user query. We omit details regarding the Okapi BM25 formula.

A different manner to identify similarity between texts is shingling [Broder et al., 1997; Pereira-Jr and Ziviani, 2004]. Differently of the two methods presented above, which are typically employed to compare queries and documents in a corpus, shingling has been proposed as a document representation to identify duplicate and near-duplicate documents within a given corpus. It is a simple technique that considers text windows of the document to represent it.

As an example, consider the use of shingles of size 3 and with granularity in the level of a sentence. Consider document D_1 containing seven sentences: $D_1 = s_1.\ s_2.\ s_3.\ s_4.\ s_5.\ s_6.\ s_7$, where s_i, $1 \leq i \leq 7$, is a sentence of the text, and i is the order of occurrence of the sentences in the text. The 3-shingling paragraphs for D_1 are: "$s_1.\ s_2.\ s_3.$", "$s_2.\ s_3.\ s_4.$", "$s_3.\ s_4.\ s_5.$", "$s_4.\ s_5.\ s_6.$", "$s_5.\ s_6.\ s_7.$".

2.5.4 Bibliographic Measures

The WIM algebra proposes operators to add links to a graph according to three bibliographic measures: co-citation [Small, 1973], bibliographic coupling [Kessler, 1963], and transitivity [Gonnet and Baeza-Yates, 1991].

Co-citation is used to measure subject similarity between two documents. If a document A cites documents B and C, documents B and C are co-cited by A. If many documents cite both documents B and C, this indicates that B and C may be related. The more documents that cite both B and C, the closer their relationship is.

Bibliographic coupling is the inverse of co-citation and considers that if two documents include the same references then they are likely to be related, i.e., if document A and B both cite document C this gives some indication that they are related. The more documents cited by both documents A and B, the stronger their relationship is.

Transitivity is used to walk through links of one document to other documents. If document A links to document B, and B links to document C, then from A it is possible to reach C with an operation of transitivity.

2.5.5 Document Ranking

In information retrieval, language models, such as the space vector model presented in this section, are used to retrieve documents that are similar to a query. With the abundance

of documents on the Web and the characteristic of Web queries of being very short, a query independent document ranking (weight) is important for retrieval and mining. We use two measures, Brin and Page's PageRank [Brin and Page, 1998; Page et al., 1998] and Kleinberg's HITS [Kleinberg, 1999].

PageRank is a sophisticated query-independent link citation measure which uses global link information and is stated to be the primary link recommendation scheme employed in the Google search engine. PageRank is designed to simulate the behavior of a "random Web surfer" who navigates the Web by randomly following links. If a page with no outgoing links is reached, the surfer jumps to a randomly chosen bookmark. In addition to this normal surfing behavior, the surfer occasionally jumps spontaneously to a bookmark instead of following a link. The PageRank of a page is the probability that the Web surfer will be visiting that page at any given moment.

To calculate the PageRank for a page, all of its inbound links are taken into account. These are links from within the site and links from outside the site. It is given by:

$$PR(A) = (1-d) + d(PR(T_1)/C(T_1) + ... + PR(T_n)/C(T_n)) \tag{2.4}$$

where $PR(A)$ is the PageRank of page A, $PR(T_i)$ is the PageRank of pages T_i which link to page A, $C(T_i)$ is the number of outbound links on page T_i, and d is a damping factor which can be set between 0 and 1.

Hyperlink Induced Topic Search (HITS) is a method used to identify two sets of pages that may be important: Hub pages and Authority pages [Kleinberg, 1999]. Hub and Authority pages have a mutually reinforcing relationship – a good Hub page links to many Authority pages (thereby indicating high Authority co-citation), and a good Authority page is linked to by many Hubs (thereby indicating high Hub bibliographic coupling). Each page in the Web graph is assigned two measures of quality: an Authority score $Au[u]$ and a Hub score $H[u]$.

HITS-based scores may be computed either using local or global link information. Local HITS has two major steps: collection sampling and weight propagation. Global HITS is computed for the entire Web graph at once so there is no collection sampling step (which is the case in this thesis). The Hub and Authority score computation is a recursive process where Au and H are updated until convergence (initialized with all pages having the same value).

Chapter 3

A Data Model for Web Mining

Most Web mining algorithms and softwares currently available have been developed in an ad-hoc manner, with specific data structures in mind. The data structures are used to represent the Web data to be mined. Three types of Web data are usually combined in different degrees for mining purposes, namely Web content, Web structure, and Web usage. A large variety of different data structures and formats are used by existing solutions to store and access all required sources of data, typically graphs, text documents, and relational tables, together with associated indexing structures.

While this approach has certainly led to very significant advances in Web data mining, it also poses severe limitations to current research. On one hand, this diversity limits the ready reuse of existing tools and algorithms by other researchers, unless an open source approach is taken, not currently a very widespread choice. On the other hand, even more importantly, it also hampers the direct comparison of existing research contributions. A more abstract model is required to overcome these limitations.

When we started our research in designing the WIM data model, we did not have in mind any model, traditional or not, to follow or base on. We started studying the properties of Web data and Web mining applications, and our design converged to a model that borrows some concepts from the relational model. Although the WIM data model has expressive differences, it inherits general abstract concepts and terminology, such as relation and attribute, which make it easier for an external person to abstract based upon the relational model than from scratch.

In this chapter we present the WIM design goals and the formal WIM data model, with illustrations of the notation and definitions.

3.1 Design Goals

In order to achieve the objectives of the thesis, we established the main WIM design goals, presented in this section.

Feasibility. The model might be implementable and the resulting tool/prototype should be useful for different applications. This means that the WIM data model is not just an abstract model.

Simplicity. Simplicity is important to allow the design and implementation of a prototype to be applied to a set of use cases as a proof of concept of WIM. Furthermore, simplicity will help users implement applications based on the WIM data model and algebra, and extend WIM according to their specific demands.

Extensibility. WIM must be extensible. This means not only extension of the software architecture but also extension of the algebra. The first is important to allow scalability and the implementation of a more sophisticated software in industrial scale, extending the prototype we have implemented so far. The latter is important for a similar reason: once the implemented prototype does not cover a large number of data mining techniques, new operations will be required when extending the prototype to a tool for wide use.

Representativeness. WIM must provide a layer of data independence from the raw data that is to be mined. It means that we needed to propose a standard to represent the Web data to be recognized by WIM. Generally, Web data has been widely represented as a graph, but as WIM is a model for mining, associations between links and nodes must be provided, so that operations can be applied to different data types. In this sense, an algebra might also be designed based on most common Web data types, in which operators need to benefit from special types of data present on the Web: documents, relationships between documents, and usage data. The WIM algebra is presented in Chapter 4.

Compositionality. WIM is intended to be more than a simple library. It is rather a declarative programming language. It is intended to deal with specific data types, and then an algebra to manipulate and mine data is required. Therefore, the compositionality [Abiteboul et al., 1995] property is a requisite, in order to allow sequences of operations

to manipulate input data. Without such property, the output of a given operation would be useless for other operations and the algebra would not achieve its goal.

Applicability to other scenarios. Web documents and links are not the only entities to be represented in WIM. In spite of the focus on Web data, other data to be mined can be modeled according to the WIM model. This does not mean that we need to propose a general model, but it will be relevant if in the future the model can be extended to other scenarios beyond the Web. On the other hand, considering the Web as a scenario, there are other entities to be represented besides Web documents. The terms of documents, queries sent to search engines, search engine user sessions, among others, might be represented by the WIM model. Similarly, relationships between those entities are not limited to physical HTML hyperlinks. Logical links may exist and must be represented, such as a graph of users' navigation.

3.2 Formal Data Model

In this section we present the notation and definitions for formalizing the WIM data model. In some cases they are adapted from the standard relational database literature [Abiteboul et al., 1995; Codd, 1970]. In the WIM data model, two types of relations are defined: *node relations*, which are defined in Section 3.2.2, and *link relations*, which are defined in Section 3.2.3. Node relations exist to represent nodes of graphs, such as documents of a Web dataset, terms of a document, or queries or sessions of a query log; and link relations exist to model edges of graphs, such as links of a Web graph, word distance among terms of a document, similarity among queries, or clicks of a query log.

3.2.1 Object Set

An *Object Set* O can be any dataset that users want to model using WIM. In most application scenarios, it is a collection of Web documents, in which each object is a different document in this collection. However, several other datasets may have interesting objects to be modeled in WIM. For instance, usage data is not a typical collection of documents, but can still be represented. In this thesis we present some usage data applications to which WIM has been applied.

Figure 3.1 presents an example of an object set, with six objects, which are Web documents. Each document has a URL, a textual content (which is just a couple of words

in this simplified example), and an identifier, which is defined by the user who manages the collection.

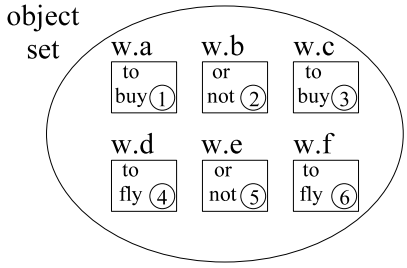


Figure 3.1. Example of an object set.

3.2.2 Node Relation

Given two domains D_1 and D_2, R_N is a node relation on these two domains if it is a set of 2-tuples, each of which has its first element from D_1 and its second element from D_2. Exceptionally permitted as output of WIM algebra operators (presented in Section 4), a node relation can be unary rather than binary. In this case, a node relation on domain D_1 is a set of 1-tuple.

The first attribute is referred to as the *Primary Key Attribute* K, and the second attribute (when it exists), as the *Value Attribute* V. Then a node relation R_N in the WIM model is defined as $R_N = (K, V) \mid (K)$. The domain (or type[1]) of K is integer. The possible types of V are integer, floating point, or string; i.e., $t_V \in \{int, float, string\}$[2].

A node relation represents objects from an object set. A single node relation cannot represent more than one object set, although several node relations can represent the same object set, and a node relation can represent a subset of an object set. Figure 3.2 presents two examples of node relations, which represent the same object set, illustrated in Figure 3.1. Relation *url* represents the URLs of a set of six Web objects, whereas relation *text* represents the text content of the Web objects.

Observe that the main difference so far between the WIM and relational data models is the number of attributes in each relation. As we shall further explain below in this chapter, this is a requirement for WIM, because subsets of tuples in relations might be represented as new relations which are part of the same dataset.

[1] In this text we also use the term type to express the domain of attributes.

[2] For the sake of simplicity, boolean and character types are not directly allowed, though they can be represented by integer and string types, respectively.

url

K	V
1	w.a
2	w.b
3	w.c
4	w.d
5	w.e
6	w.f

text

K	V
1	to buy
2	or not
3	to buy
4	to fly
5	or not
6	to fly

Figure 3.2. Two node relations: *url* and *text*.

3.2.3 Link Relation

Given four domains D_1, D_2, D_3 and D_4, R_L is a link relation on these four domains if it is a set of 4-tuples with its first element from D_1, its second element from D_2, its third element from D_3, and its fourth element from D_4. A link relation can also be ternary rather than quaternary. In this case, given three domains D_1, D_2 and D_3, R_L is a link relation on these three domains if it is a set of 3-tuples with its first element from D_1, its second element from D_2, and its third element from D_3.

The first attribute is referred to as the *Primary Key Attribute* K of the link relation, the second as the *Start Attribute* S, the third as the *End Attribute* E, and the fourth (when it exists) as the *Value Attribute* which represents a label for an edge. Then a link relation R_L in the WIM model is define as $R_L = (K, S, E, L) \mid (K, S, E)$. The latter means that there is no label value associated with edges of the graph.

Individually, S and E represent the nodes of a graph and, for this reason, elements in S and in E are typically referred to as nodes in this thesis. S must represent objects from a single object set, as well as E must represent objects from a single object set. Often, S and E represent objects from the same object set, although S and E may represent objects from two different object sets. Together, S and E represent edges of a graph. If S and E represent objects from different object sets, then the link relation represents a bipartite graph, which is useful to make associations between different collections. The type of K, S and E is integer, and the possible types for L are integer, floating point, or string.

Figure 3.3-*(a)* presents an example of a link relation called *frequency*, whereas Figure 3.3-*(b)* presents the graph represented by this relation. Observe that, for key value 5 (start node 3), the associated values in attributes E and L are null (represented by a hyphen), which means there is no link from node 3, though it is a node of the graph, as represented in Figure 3.3-*(b)*.

A difference between node and link relations is that in node relations the attribute

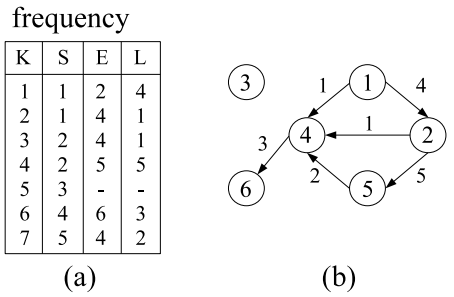


Figure 3.3. Relational and graphical representation of link relation *frequency*.

K stores the identifier of each object represented, whereas in link relations the attribute K stores the identifier of each edge between objects, which are typically represented by a node relation. Observe that the attribute set $\{S, E\}$ cannot be used as superkey for link relations, because typically there may exist two edges that point from a node d_1 to a node d_2, then the values in attribute set $\{S, E\}$ may not be unique in a link relation. Furthermore, there may exist nodes represented without any edge, as illustrated in Figure 3.3.

3.2.4 Relation Set

In spite of the restriction with respect to the number of attributes in the relations, the WIM data model has a mechanism to associate value attributes in different relations according to their respective key attributes. The concept of *Relation Set* allows to associate value attributes from different relations, according to the attribute key K of node relations, or attributes S and E of link relations.

A set of relations $R_1, R_2, \ldots, R_n$ belong to the same *Relation Set*, if and only if:

- Every relation in the relation set belongs to the same type, i.e., either $t_{R_i} = node$ or $t_{R_i} = link$, for $1 \leq i \leq n$. The type of a relation set is represented as t_S.

- If $t_S = node$, the key attribute in each relation identifies objects from the same object set O.

- If $t_S = link$, the start nodes (i.e., attribute S) of each relation identify objects in the same object set O_1 and the end nodes (i.e., attribute E) of each relation identify objects from the same object set O_2. Typically, object sets O_1 and O_2 are the same set, i.e., $O_1 = O_2$.

A relation set representing node relations is referred to as a *Node Relation Set* and is represented by symbol S_N, whereas a relation set representing link relations is referred to as a *Link Relation Set*, and is represented by symbol S_L (we remind that a single S is the representation of attribute start, and not a representation for relation set).

According to the above definition, relations *url* and *text* in Figure 3.2 can be defined within the same node relation set, because they are both node relations and they represent the same object set (shown in Figure 3.1). Relation *frequency* in Figure 3.3 could be part of a link relation set that represents graphs composed by the documents in Figure 3.1, but it does not belong to a relation set like the one that includes the node relations *url* and *text*, because *frequency* is a link relation. Then, by definition, *frequency* cannot belong to a node relation set.

Observe that the definition of relation set does not impose that every relation in a set must have the same K, for node relation sets, and the same values in attributes S and E, for link relation sets. This means that relations within the same relation set can have different number of tuples. Actually, this property is the main reason for defining relations with only one value attribute, and grouping them as a relation set.

Although relations in the same relation set need not represent the same objects in an object set, the union of nodes of every relation in the set must correspond to objects from the same object set. This condition is relaxed for link relations, to which start and end nodes may represent different objects, corresponding to a bipartite graph. For link relation sets, the union of start nodes of every relation in the set must correspond to objects from the same object set, and the same condition holds for end nodes.

For a node relation set S_N, we refer to *key set* as the union of all values of each attribute K of relations in S_N . Note that the key set must represent a single object set, as stated before. For a link relation set S_L, we refer to *start set* as the union of all elements in each attribute S of relations in S_L, and as *end set* as the union of all elements in each attribute E of relations in S_L.

3.2.5 Compatibility

Relation sets exist to associate relations of the same type that represent the same object set. The concept of *compatibility* is important to establish an association between relation sets of different types, but which represent the same object set. If two relation sets of the same type represented the same object, they would be defined as a unique relation set, which means that compatibility needs not be defined for relation sets of the same type. The

concept of compatibility is specially important for the WIM algebra, as we shall present in the next chapter.

There are two situations to define compatibility. The first is when both the start and end sets of a link relation set represent the same object set, and the second is when they represent different object sets (for which a bipartite graph is represented).

A node relation set S_N and a link relation set S_L are *compatible* to each other if they represent the same object set O. More formally, S_N and S_L are compatible if both the start and end sets of S_L consists of foreign keys in the key set of S_N. We use the symbol '$\Leftrightarrow$' to express compatibility between two relation sets ($S_N \Leftrightarrow S_L$).

If the start and end sets of S_L represent different objects, then S_L does not have a single compatible S_N for both start and end sets of S_L. This means that start and end sets of a link relation set can have different compatible node relations. In this case, if S_{N_1} represents the object set O_1, which is the object set represented by the start set of S_L, then the start set of S_L is compatible with S_{N_1}. More formally, the start set of S_L is compatible with S_{N_1} if the start set of S_L consists of foreign keys in the key set of S_{N_1}. The same definition is valid for the end set: the end set of S_L is compatible with S_{N_2} if it consists of foreign keys in the key set of S_{N_2}. We use the symbol '$\Rightarrow$' to express compatibility between a start or end set of link relation sets and node relations ($S(S_L) \Rightarrow S_{N_1}$ and $E(S_L) \Rightarrow S_{N_2}$).

For illustration, observe that the node relation set presented in Figure 3.2 is compatible with the set to which the link relation presented in Figure 3.3-*(a)* belongs. Link relation *frequency* represents edges between nodes, which are objects in Figure 3.1. Observe in Figure 3.3-*(b)* that this link relation does not represent a bipartite graph.

To illustrate the situation in which a link relation represents a bipartite graph, and then has different compatible relations for start and end sets, consider the example illustrated in Figure 3.4. Suppose *query* is the only relation in a node relation set *querySet*, *text* is the only relation in a node relation set *textSet*, and *result* is the only relation in a link relation set with the same name[3]. Relation *result* represents results of searches for elements of *query* in document texts represented by relation *text*, using a given comparison technique.

Observe in relation *result* that elements in attribute E are documents returned by queries represented in attribute S. For instance, document number 4 (to fly) was returned

[3] Two relations within the same relation set cannot have the same name, but a given relation set can have the same name of its relation. We shall see in the next section that this situation is fairly common in the WIM data model.

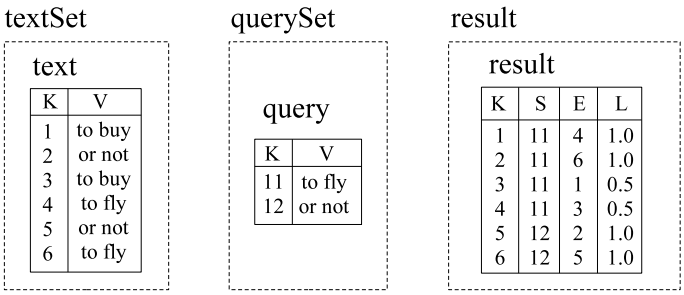


Figure 3.4. An example of a link relation with different compatible node relations for the start and end sets.

by query number 11 (to fly), with a similarity measure represented as value 1.0 in attribute L.

3.3 Concluding Remarks

In this chapter we have presented the WIM data model. The WIM data model has been designed to deal with Web data, allowing two different types of data: link and node relations. In addition, it aims to save space when storing attribute values and to take advantage of the association between node and link relation sets, in cases that the edges of a link relation set refer to documents of a node relation set, i.e., when the sets are compatible.

The remaining of this thesis shall demonstrate some important features of the WIM data model, like: how the WIM algebra is appropriately built upon the WIM data model presented in this chapter; how the model can be effectively and efficiently implemented; and how a WIM prototype has been successfully used to manipulate and mine diverse kinds of Web data.

Chapter 4

An Algebra for Web Mining

In this chapter we present details of the WIM algebra, including the syntax of the WIM operators and the underlying WIM language. The definition of each WIM operator is presented and didactically illustrated. Before introducing the WIM algebra, Section 4.1 presents the concepts of WIM operation, program and database.

Section 4.2 presents a high level view of the WIM language as well of the patterns used to describe the operators' syntax. The WIM operators are divided into two categories. Section 4.3 presents the data manipulation operators, which permit traditional operations on data, like selection and join. Then, Section 4.4 presents the data mining operators, which are used to certain mining tasks on the data. Finally, Section 4.5 presents a comparison between the WIM data manipulation operators and the relational algebra.

4.1 Operations, Programs and Databases

An *operation* is the application of an operator to relations of one or two relation sets. An *operator* is a previously defined function included in the WIM *algebra*, which shall be presented in this chapter. The output of any operation must be a new relation, which can be used afterwards in the same program. This means that this output relation will belong to some relation set. We shall see in Section 4.1.1 that output relations can belong to a previously existing relation set, or a new relation set might be created. The decision of to which relation set an output relation belongs is automatically taken by WIM, based on the algebra.

A WIM *Program* is a sequence of operations, defined according to the WIM language

built upon the WIM algebra. The WIM language is a *dataflow* programming language, which is derived from the functional programming paradigm. A program is modeled like a directed graph, in which nodes represent operations and edges represent the data flow. This approach allows parallelism of tasks when the manipulated data is independent in different parts of the program.

It is intrinsic to any operator definition the type of the input relation set(s), and the type and properties of the output relation. The input of an operator typically includes other operator-specific parameters that are not relation sets or relations. Some parameters are previously named *options*, or user-defined *values*.

Figure 4.1 presents an example of a WIM program with just three operations. The output relation is preceded by the sign "=", for any operation. The operator comes after the sign and is followed by a list of parameters surrounded by parentheses. The first parameters are the input relation sets, followed by possible textual options, and then possible values. Finally, the list of relations and their attributes appear. This list of relations does not appear in the beginning, just after the associated relation set, because the number of possible relations vary for each operator. Observe as an example the operator *Search* in Figure 4.1. Set *querySet* is the first input relation set, *textSet* is the second input relation set, *OR* is an option for operator *Search*, and finally *query.V* and *text.V* are input relations followed by the corresponding attribute to which the operator is applied (V in both cases). In Section 4.2 we shall present the WIM language syntax, in which the WIM operators are detailed, including the operators that appear in Figure 4.1. We shall present real examples of WIM programs when discussing the use cases, in Chapters 6 and 7.

```
result = Search(querySet, textSet, OR, query.V, text.V);
cluster = Select(result, value, ==, 1.0, result.L);
singleCluster = Set(textSet, result, intersection, text.K, cluster.E);
```

Figure 4.1. An illustration of a WIM program with three operations.

A *WIM Database* consists of any set of relations, divided accordingly into relation sets, which are registered to be used in programs. Relations in different databases cannot be used in the same WIM program. Relations registered in a WIM database are referred to as *Permanent Relations*, and relations returned by a WIM program are referred to as *Temporary Relations*. Temporary relations can only be used within the program they are

created[1]. Observe that a relation set may contain permanent and temporary relations together, during the execution of programs.

4.1.1 Relation Set of an Output

The WIM data model would be useless if operators' output could not be reused in a program, because, as operators manipulate and mine data, very often several data mining and manipulation operations are needed in sequence in a program, in order to return specific pieces of data to users.

Each WIM operation specification must define the operator's output characteristics (apart from the input characteristics), like the output relation type, the output relation set, whether the output relation has the value attribute or not and, if true, the type of the value attribute. Further, the relation set of which the operator's output belongs must be defined, otherwise users do not know what can be done with each operator's output. There are three possibilities: i) the output can belong to an input relation set; ii) the output relation can belong to a new relation set, which is created containing just that relation; iii) the output belongs to an existing relation set which is not an operation's input.

To illustrate the first possibility, observe operators Select and Set in Figure 4.1, which return output relations that belong to the input relation set. Figure 4.2 shows relation *cluster* as a new relation in relation set *result*. Relation *cluster* only contains edges whose value is equal to 1.0, which is the condition for the selection (edges $11 \rightarrow 1$ and $11 \rightarrow 3$ are excluded for relation *cluster*). Relation *singleCluster* does not have attribute V. It is the result of the intersection of attribute K from relation *text* and attribute E from relation *cluster*. Observe that the new relation *singleCluster* belongs to relation set *textSet*, which is the first input of the operation.

Notice that, for the sake of simplicity, link relations *result* and *cluster* are illustrated using a list of end nodes (and their respective labels) associated to each single start node, and that attribute K is omitted in these link relations. This is just a representation we use to illustrate link relations in this thesis (which does not mean that WIM allows multi-valued attributes). For example, for start node 12, actually there are two tuples: $(12, 2, 1.0)$ and $(12, 5, 1.0)$ (omitting attribute K).

The second possibility for an operator's output is to belong to a new relation set. In this case there are two other cases: either the output relation set is compatible with

[1] In spite of the absence of a WIM algebra operation for materialization, any WIM implementation might provide such feature, so that temporary relations can become permanent.

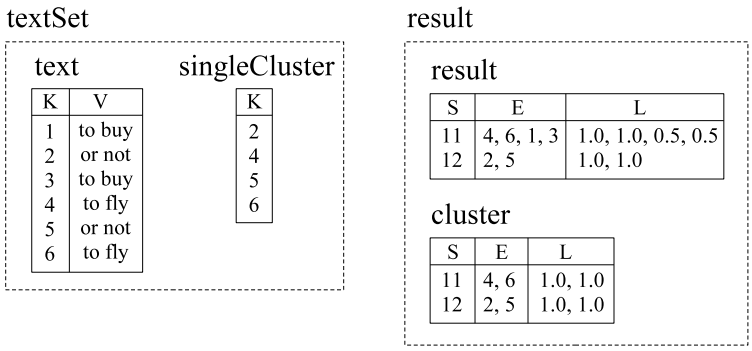


Figure 4.2. Two compatible relation sets, with new relations (*cluster* and *singleCluster*) as result of operations.

an input relation set, or it is not compatible with anything. The latter case might be avoided when designing WIM operations, because the WIM algebra takes advantage of the concepts of relation set and compatibility.

The Search operator in Figure 4.1 is an example of an operator that returns an output relation within an new relation set which is compatible with the input one. Relation *result* is already represented in Figure 3.4. Observe that relation set *result* is created because of the Search operator. The start set of relation set *result* is the foreign key of (and then compatible with) the node relation set *querySet*, and the end set is foreign key of node relation set *textSet*.

The third possibility for an operator's output is to belong to an existing relation set which is not an operation's input. For illustration, suppose that the fourth operation of the WIM program shown in Figure 4.1 is another operation over relations *text* and *query*, returning other link relation with attributes S and E that represent different foreign keys, respectively in relation sets *textSet* and *querySet*. This output would be a relation in the existing relation set *result*. Note that WIM can automatically associate new output relations to existing or new relation sets.

4.1.2 Properties of the Operations

The most important property of WIM operations is that any value attribute of a relation in a set can be used as value attributes of other relations in the same set, indistinctively. This property means that users can choose a relation from which the key attribute is taken, and other relation in the same set from which the value attribute is taken. This is important because the need of accessing value attributes from other relations after performing a

sequence of operations in a WIM program is quite frequent. This property is called *inheritance*, because it means that relations returned by operations actually inherit attributes from other relations in the set.

For example, observe relation *singleCluster* in Figure 4.2, which is the output of the WIM program shown in Figure 4.1. This relation has no value attribute, but its key attribute could be used to filter values from other relations in the same set, such as relation *text*. Thus, it is possible to apply an operator only to elements identified by the list $(2, 4, 5, 6)$, which is the list of elements in the key attribute from relation *singleCluster*, but whose values appear in relation *text*, which are in the list (*or not*, *to fly*, *or not*, *to fly*).

To understand how to use inherited attributes in WIM, observe again Figure 4.1. Note that in every case there is no differentiation from which relation is the value attribute and from which relation is the key attribute. This is because both attributes come from the same relation. For instance, the first operation uses the value and key attributes from relation *query* (see Figure 3.4) for the first input set, and the value and key attributes from relation *text*, for the second input set. If a user wants to use only the values in relation *text* that exist in relation *singleCluster*, they might replace the first operation to this one:

result = Search(querySet, textSet.singleCluster, OR, query.V, text.V);

Observe that the relation which the key attribute is taken from is specified with a dot and then its name, just after specifying the relation set (*textSet*).

The second most important property of the operations in the WIM model is the inheritance of attributes from a compatible node relation set, by a relation of a link relation set. Some operators are specially designed taking into consideration this property. For example, observe relation *cluster* in Figure 4.2. A given operation could be applied to node pairs for edges of the graph represented by relation *cluster*, whose start nodes are represented in relation set *querySet* (see Figure 3.4), and whose end nodes are represented in relation set *textSet* (see Figure 4.2). For that, values from relation sets *querySet* and *textSet* would be inherited and used. For instance, the first edge is $11 \rightarrow 4$, then tuples identified as 11 and 3, respectively in relations *query* and *text*, could have their attribute values compared.

4.2 The WIM Language

As already mentioned, the WIM language is a dataflow language and so a WIM program consists of a sequence of operations. The high level syntax of a WIM program is defined

in Figure 4.3. We use a BNF-like[2] notation to specify the WIM syntax. The actual syntax specification of each operator is independently presented in the respective section in this chapter. Appendix 1 presents the complete specification of the WIM language syntax.

```
WIMProgram := Operator ';' { Operator ';' }
Operator := Select | Calculate | CalcGraph | Aggregate | Set | Join |
            Convert | Search | Compare | CompGraph | Cluster |
            Disconnect | Associate | Analyze | Relink
```

Figure 4.3. High level syntax of a WIM program.

Our specification language is a set of derivation rules, written as:

Symbol := Expression

where *Symbol* is a nonterminal and *Expression* consists of one or more sequences of symbols. Alternative sequences are separated by the vertical bar '|', indicating a choice, the whole being a possible substitution for the symbol on the left. Symbols that never appear on a left-hand side are terminals and are indicated between single quotes. On the other hand, symbols that appear on a left-hand side are non-terminals. Note that every non-terminal must appear once and only once on the left-hand side. Optional items are enclosed in square brackets (for example, [item]), and items that can repeat zero or more times are enclosed in curly brackets (for example, { item }). Some additional symbols used in the specification of the operators are presented in Figure 4.4.

We are not strict with the BNF-like notation we have used. For instance, we use abbreviations, such as to represent the alphabet in Figure 4.4 ("a, b, ..., z", rather than listing every letter), and we do not explicitly differentiate where blank spaces are needed to separate terms and where they are used only for improving presentation. For example, the symbol "NumValue" in Figure 4.4 is defined with blank spaces before and after the terminal '.' just for presentation purpose.

4.3 Data Manipulation Operators

Data manipulation operators are those operators required to retrieve and prepare data to be used by another operator or transformed to an adequate format. These operators are

[2]BNF (Backus-Naur Form) is a standard for formal language syntax specification.

```
Digit := '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
IntValue := Digit { Digit }
NumValue := IntValue [ '.' IntValue ]
Letter := 'A' | 'B' | ... | 'Z' | 'a' | 'b' | ... | 'z'
Name := Letter | Digit { Letter | Digit }
OutputRel := Name [ '.' Name ]
OutputLinkRel := Name [ '.' Name ]
OutputNodeRel := Name [ '.' Name ]
InputSet := Name
InputNodeSet := Name
InputLinkSet := Name
NumRel := Name '.' Attr
IntRel := Name '.' Attr
TxtRel := Name '.' Attr
AnyRel := Name '.' Attr
Attr := 'K' | 'V' | 'S' | 'E' | 'L'| 'SE'
```

Figure 4.4. General rules with symbols used in the specification of operators.

used for selecting tuples, merging relations, and executing calculation.

When defining operators, we consider that the output relation is referred by an apostrophe (for example, R'). Then, if the output is a node relation, it is referred to as R'_N, whereas if the output is a link relation, it is referred to as R'_L. Attributes in $\{K', V', S', E', L'\}$ are attributes of the output relation. Input node relations are referred to as $R_{N_i}, 0 \leq i \leq 3$, whereas input link relations are referred to as $R_{L_i}, 1 \leq i \leq 3$. When the input is just one relation, then its index is not represented (for example, R_N is used rather than R_{N_1}). Attributes in $\{K_i, V_i\}$ are attributes of node relations in the input, whose index i represents the i^{th} input node relation. Similarly, attributes in $\{K_i, S_i, E_i, L_i\}$ are attributes of link relations in the input, whose index i represents the i^{th} input link relation. Input relations surrounded by square brackets ([]) are optional.

The definition of each operator always inform: i) if the input and output relations are node (R_N) or link relations (R_L), or if it does not matter (simply R is used); ii) if the output relation goes to the same relation set as the input (for example, $\{R_N, R'_N\} \in S_N$), or if the output relation set is compatible to the input relation set (for example, consider $R'_N \in S_N$ and $R_L \in S_L$, then $S_N \Leftrightarrow S_L$); iii) if the output represents a subset of tuples from the input ($K' \subseteq K$) or if the output represents the same set of tuples from the input ($K' = K$); iv) if the output value attribute (V' for node relations and L' for link relations) exists in the output (for instance, for node relations, the output $R'_N(K', V')$ means that V'

exists, whereas $R'_N(K')$ means that V' does not exist); v) other constraints, like if S from an input link relation must have a compatible node relation set, i.e., $S \Rightarrow S_N$.

4.3.1 Select

This operator selects tuples from the input relation (which can be a node or a link relation), according to a condition, such as equal, different, greater than, etc., which is applied to a numeric attribute. The Select operator is defined as:

$$\begin{array}{ll} R'_N(K') & \leftarrow Select(R_N), \text{ where } \{R_N, R'_N\} \in S_N \wedge K' \subseteq K \text{ or} \\ R'_L(K', S', E') & \leftarrow Select(R_L), \text{ where } \{R_L, R'_L\} \in S_L \wedge K' \subseteq K \end{array}$$

Observing the definition of the Select operator, we note that it accepts both node and link relations as the input relation. The first part defines how it works for node relations and the second one for link relations. Observe that the resulting output relation has the same type of the input relation set.

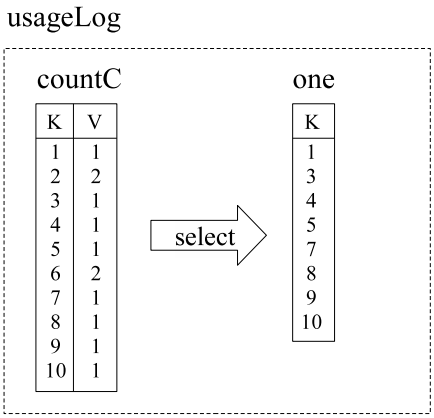
According to the definition of the Select operator, its output may be a subset from the input ($K' \subseteq K$), and the value attribute does not exist in the output relation (output defined as $R'_N(K')$ for node relations and $R'_L(K', S', E')$ for link relations). Further, the output relation belongs to the same set of the input relation set ($\{R_N, R'_N\} \in S_N$ and $\{R_L, R'_L\} \in S_L$).

Figure 4.5 presents the syntax of the Select operator in the WIM language. It has tree possible options: *Value*, *Attribute*, and *Top*. For the Value option, the values of V in the input relation are compared against a given value passed by the user. For the option Attribute, the comparison is performed between two value attributes of different relations in the same relation set. For the option Top, only a given number of elements with the highest or lowest values are returned. In this case, the conditions for comparison presented above are not used.

```
Select := OutputRel '=' 'Select' '(' InputSet ',' BodySelect ')'
BodySelect := 'value' ',' NumRel ',' SelOperation ',' NumValue |
              'attribute' ',' NumRel ',' NumRel ',' SelOperation |
              'top' [ ',' Order ] ',' NumRel ',' IntValue
SelOperation := '==' | '! =' | '<' | '<=' | '>' | '>='
Order := 'increasing' | 'decreasing'
```

Figure 4.5. Syntax of the Select operator.

Figure 4.6 presents an example of the Select operator, with the Value option. Only tuples whose value of attribute V in the relation *countC* from the relation set *usageLog* is equal to 1 are returned in relation *one*. We notice that most of the examples in this section are taken from the WIM use case programs which shall be presented in Chapters 6 and 7. In order to let the reader associate parts of the use cases discussed in those chapters and the examples in this chapter, we preserve the original name of the relations and relation sets (like *usageLog* and *one*).



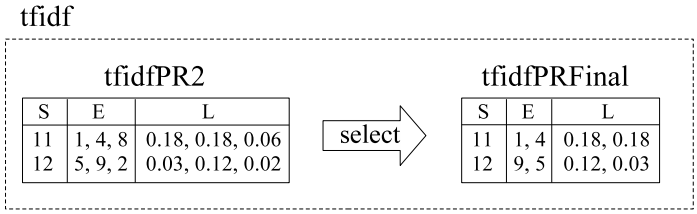

```
one = Select(usageLog, value, countC.V, ==, 1);
```

Figure 4.6. Example of the Select operator with the Value option.

As defined above, Select can also be applied to link relations. In Figure 4.7 it is applied to the relation set *tfidf* with the option Top, to return the two highest values of the attribute L in relation *tfidfPR2*, which represents the labels of links from start to end nodes. As we can see, for the start node 11, the link to end node 8 is not included in the output relation *tfidfPRFinal*, because it has the lowest value in relation *tfidfPR2*. For start node 12, the link to end node 2 is also not included. Observe that the end nodes in *tfidfPRFinal* are sorted according to the L values in relation tfidfPR2, as a result of applying the Select operator with the option Top.

4.3.2 Calculate

The Calculate operator is used for mathematical and statistical calculations using numeric attributes. Examples of possible calculations are sum, multiplication, and average. The




```
tfidfPRFinal = Select(tfidf, top, decreasing, tfidfPR2.L, 2);
```

Figure 4.7. Example of the Select operator with the option Top, applied to a link relation.

Calculate operator is defined as:

$$R'_N(K', V') \leftarrow Calculate(R_{N_1}, [R_{N_2}]), \text{ where } \{R_{N_1}, R_{N_2}, R'_N\} \in S_N \wedge K' = K_1 \text{ or}$$
$$R'_L(K', S', E', L') \leftarrow Calculate(R_{L_1}, [R_{L_2}]), \text{ where } \{R_{L_1}, R_{L_2}, R'_L\} \in S_L \wedge K' = K_1$$

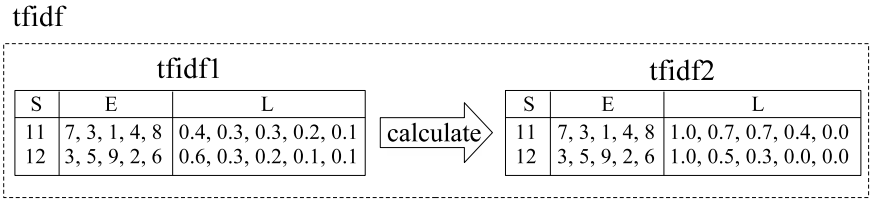
Both node and link relations are accepted. Its output relation belongs to the same set of the input relation and the value attribute receives the result of the calculation performed. Since $K' = K_1$, the output relation does not represent a subset of the input relation.

Figure 4.8 presents the syntax of the Calculate operator. Two options are possible: *Constant* and *Pair*. For the option Constant, the calculation is performed between a constant value passed as a parameter and each value of an attribute. For the option Pair, the calculation is performed between the corresponding values of two attributes of the two relations.

```
Calculate := OutputRel '=' 'Calculate' '(' InputSet ',' BodyCalc ')'
BodyCalc := 'constant' ',' ConstOper ',' NumRel ',' NumValue |
            'pair' ',' PairOper ',' NumRel ',' NumRel
ConstOper := 'sum' | 'difference' | 'multiplication' | 'division' | 'average' |
             'deviation' | 'mod' | 'normalize' | 'absolute' | 'max' | 'min'
PairOper := 'sum' | 'difference' | 'multiplication' | 'division' | 'average' |
            'mod' | 'percentage' | 'max' | 'min'
```

Figure 4.8. Syntax of the Calculate operator.

Figure 4.9 presents an example of the Calculate operator with the option Constant. It is applied to relation *tfidf1* to normalize values between 0 and 1, returning relation *tfidf2* which belongs to the same relation set *tfidf*.




```
tfidf2 = Calculate(tfidf, constant, normalize, tfidf1, 1);
```

Figure 4.9. Example of the Calculate operator with the option Constant, for normalization of values between 0 and 1.

4.3.3 CalcGraph

The CalcGraph operator applies a calculation operation for each link of a graph, so that the calculation is applied to pairs of start and end nodes. The CalcGraph operator is defined as:

$$R'_L(K', S', E', L') \leftarrow CalcGraph(R_L), \text{ where } \{R_L, R'_L\} \in S_L \wedge K' = K \wedge S \Rightarrow S_{N_1} \wedge E \Rightarrow S_{N_2}$$

According to the definition, the input and the output are link relations that belong to the same relation set. The value attribute in the output is used to store the result of the calculation.

Clauses $S \Rightarrow S_{N_1}$ and $E \Rightarrow S_{N_2}$ mean that there must exist a compatible node relation associated to the start set of the input relation (as discussed in Section 3.2) and a compatible node relation for the end set, respectively. Compatible node relations must exist because the given numerical attributes from the node relations S_{N_1} and S_{N_2} are used to perform the calculation involving each start-end pair of a given relation of the input link relation set S_L. As there is no restriction with respect to sets S_{N_1} and S_{N_2}, then it is possible that they represent the same set, i.e., $S_{N_1} = S_{N_2}$.

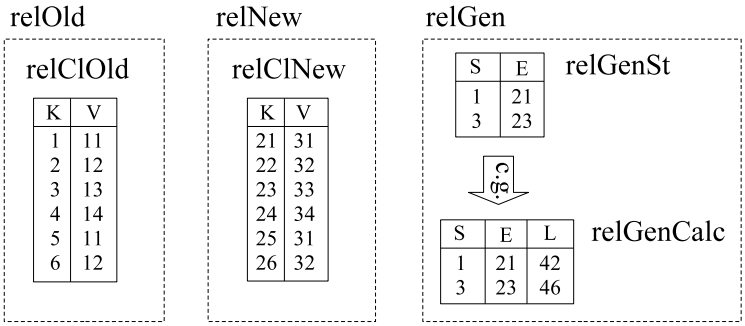
The syntax of the CalcGraph operator is presented in Figure 4.10. Observe that the two numerical input relations are taken from the compatible node relation sets for attributes start and end of S_L. Since the relation from which the start and end attributes are taken is different from the relation whose value attributes are taken, the former needs to be explicitly indicated (in passage "InputLinkSet '.' AnyRel").

As an example of this operator, consider the relation *relGenSt* (from the relation set *relGen*) represented in Figure 4.11, whose compatible node relation set for the start set is *relOld* and for the end set is *relNew*. Applying the sum operation to the value attributes

```
CalcGraph := OutputLinkRel '=' 'CalcGraph' '(' InputLinkSet '.' AnyRel ','
        CalcGraOperator ',' NumRel ',' NumRel ')'
CalcGraOperator := 'sum' | 'difference' | 'multiplication' | 'division' |
        'average' | 'mod' | 'percentage' | 'max' | 'min'
```

Figure 4.10. Syntax of the CalcGraph operator.

of both compatible relations results in 42 (11 + 31) and 46 (13 + 33), which are values that compose the value attribute L of the output relation *relGenCalc*.




```
relGenCalc = CalcGraph(relGen.relGenSt, sum, relClOld.V, relClNew.V);
```

Figure 4.11. Example of the CalcGraph operator for the sum of values in compatible relations.

4.3.4 Aggregate

The Aggregate operator is used to combine and group values of one or more relations of a node or link relation set. The Aggregate operator is defined as:

$$
\begin{aligned}
R'_N(K', V') \quad & \leftarrow Aggregate(R_{N_1}, [R_{N_2}], [R_{N_3}]), \text{ where } \{R_{N_1}, R_{N_2}, R_{N_3}, R'_N\} \in S_N \wedge \\
& \qquad K' \subseteq K \text{ or} \\
R'_L(K', S', E', L') \quad & \leftarrow Aggregate(R_{L_1}, [R_{L_2}], [R_{L_3}]), \text{ where } \{R_{L_1}, R_{L_2}, R_{L_3}, R'_L\} \in S_L \wedge \\
& \qquad K' \subseteq K
\end{aligned}
$$

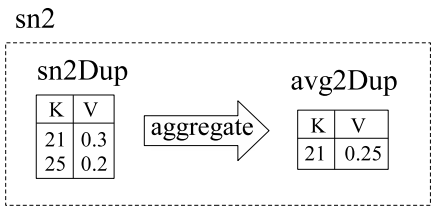
Figure 4.12 presents the syntax of the Aggregate operator. It accepts two options: *Single* and *Grouping*. In both options an aggregation operation (like sum, average, maximum, counting, etc.) is applied to value attributes from relations in a relation set, which

means that V' (and L', for link relations) is required to store the result of the aggregation operation.

```
Aggregate := OutputRel '=' 'Aggregate' '(' InputSet ',' BodyAgg ')'
BodyAgg := 'single' ',' AggOperator ',' NumRel |
           'grouping' ',' AggOperator ',' NumRel [ ',' NumRel ]
                      [ ',' NumRel ]
AggOperator := 'sum' | 'average' | 'count' | 'max' | 'min' | 'deviation' |
           'geometric' | 'mode' | 'median'
```

Figure 4.12. Syntax of the Aggregate operator.

The option Single means that an operation is applied to all values of a single attribute. In this case the result is a relation with a single tuple. In Figure 4.13, the Aggregate operator with the option Single is applied to relation *sn2Dup* from relation set *sn2*. The output relation *avg2Dup* belongs to set *sn2* and stores the average of the values in the input relation.

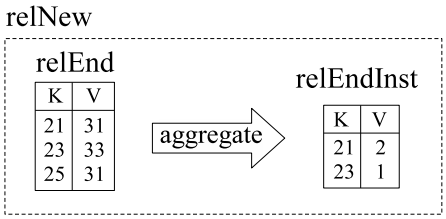



```
avg2Dup = Aggregate(sn2, single, average, sn2Dup.V);
```

Figure 4.13. Example of the Aggregate operator with the option Single, for calculating the average.

For the option Grouping, sets of tuples are removed, if their values in one or two chosen relations are equal, i.e., replicated. Then, only tuples with different values in the chosen relation(s) are kept in the output. A third attribute may be used to have one of the possible operations applied to its values. Also for the option Grouping, the output value attribute stores the result of the requested operation.

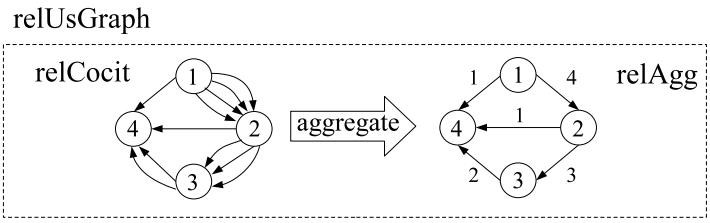
In Figure 4.14, the Aggregate operator with the option Grouping is applied to relation *relEnd* from relation set *relNew*. The operation results in relation *relEndInst*, where only one of the two tuples with value 31 in relation *relEnd* is kept and the new value attribute in the output relation *relEndInst* counts the number of tuples with each value in the input.




```
relEndInst = Aggregate(relNew, grouping, count, relEnd.V);
```

Figure 4.14. Example of the Aggregate operator with the option Grouping, for counting the number of tuples with the same value.

Aggregate can also be applied to link relations. Observe the graphs in Figure 4.15, where the Aggregate operator is applied to relation *relCocit*. The number of links for each start-end pair is counted and returned as the value of the attribute L in the output link relation *relAgg*.



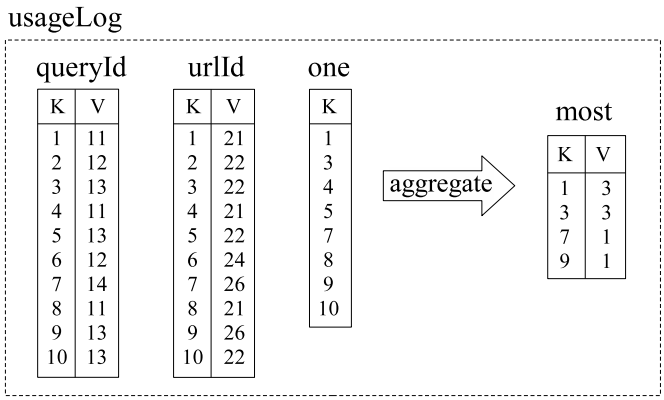

```
relAgg = Aggregate(relUsGraph, grouping, count, relCocit.SE);
```

Figure 4.15. Example of the Aggregate operator with the option Grouping, applied to a link relation.

The last example, more complex, is shown in Figure 4.16, where Aggregate is applied to the relation set *usageLog*, with the option for counting the number of elements. The keys are taken from Relation *one* and the other two relations in the set are used for aggregation, rather than only one relation. They are relations *queryId* and *urlId*. In this case, any different pair of values for these relations must appear in the output relation, which is relation *most*, where the attribute V stores the number of tuples with the same pair of values.

As the keys are taken from relation *one*, tuples in relations *queryId* and *urlId* that are not in *one* are not considered, which are tuples 2 and 6. It works like if these tuples did not exist in relations *queryId* and *urlId*. Note that this is not a feature of the Aggregate operator. Rather, it is the WIM ability that key and value attributes of different relations

in the same set can be used together, as if they were part of the same relation, or even as if relations allowed multiple value attributes.




```
most = Aggregate(usageLog.one, grouping, count, queryId.V, urlId.V);
```

Figure 4.16. Example of the Aggregate operator with the option Grouping, applied to two attributes of a relation.

4.3.5 Set

This operator is used for *Union*, *Intersection* or *Difference* of tuples from two different relations, which need not be of the same type. The user must choose the attribute to be compared for each input relation, which is not necessarily the identifier attribute K (or S and E for link relations). The Set operator can be defined as:

$$
\begin{aligned}
R'_1(K') &\leftarrow Set(R_1, R_2), \text{ where } \{R_1, R'_1\} \in S_1 \wedge R_2 \in S_2 \wedge K' \subseteq K_1 \text{ or} \\
R'_2(K') &\leftarrow Set(R_1, R_2), \text{ where } \{R_1, R_2, R'_2\} \in S \wedge K' = \{K_1 \cup K_2\} \text{ or} \\
R'_3(K') &\leftarrow Set(R_1, R_2), \text{ where } R_1 \in S_1 \wedge R_2 \in S_2 \wedge R'_3 \notin \{S_1, S_2\} \wedge \\
&\qquad\qquad\qquad\qquad K' = \{K_1 \cup K_2\}
\end{aligned}
$$

The Set operator accepts inputs of different types. This means that, for instance, the first input can be a node relation and the second input can be a link relation. In any case the output has the same type as the first input set. As there are four different combinations of input types for each of the options, the Set operator is not defined above with differentiation of node and link inputs. Consequently, when link relations are used as input, either attribute S or E might replace attribute K in the definition above. Further, the output is not represented with all the attributes it should have. Just K' is showed, to

generally inform that the value attribute V' (or L' for link relations) does not exist in the node relation output.

The first definition, R'_1, is for the options Intersection and Difference. In these cases, the output relation belongs to the same relation set as the first input, and $K' \in R'_1$ may be a subset of $K_1 \in R_1$.

For the option Union there are two possibilities. If the two inputs are relations of the same relation set, i.e., $\{R_1, R_2\} \in S$, then the output, represented by R'_2, belongs to the set of the inputs ($\{R_1, R_2, R'_2\} \in S$). But if the input sets are different, i.e., $R_1 \in S_1$ and $R_2 \in S_2$, then the output relation, represented by R'_3, belongs to a new relation set that is not an input set. Figure 4.17 presents the syntax of the Set operator.

```
Set := OutputRel '=' 'Set' '(' FirstInputSet ',' SecInputSet ','
          SetOperation ',' AnyRel ',' AnyRel ')'
FirstInputSet := Name
SecInputSet := Name
SetOperation := 'union' | 'intersection' | 'difference'
```

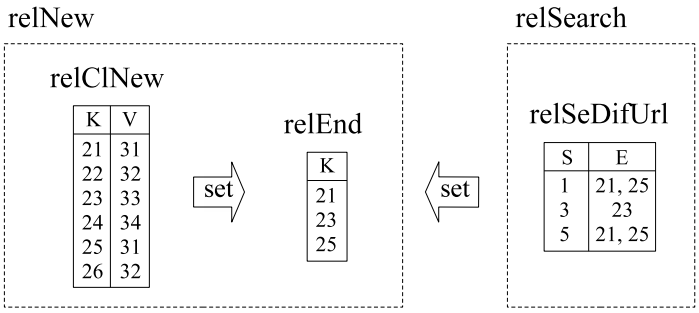
Figure 4.17. Syntax of the Set operator.

As aforementioned, the Set operator allows inputs of different types, i.e., a node relation and a link relation. For example, observe the Set operator applied to relations *relClNew* and *relSeDifUrl* in Figure 4.18, which belong to different sets. Consider that the first input passed is *relClNew*, and the operation is intersection. The key attribute is used from *relClNew* (the value attribute is not required) and the end attribute is used from *relSeDifUrl*. The result is relation *relEnd*, which represents a subset of tuples from *relClNew*, identified by 21, 23 and 25. Note that *relEnd* belongs to the same set as *relClNew*, which is the first input.

4.3.6 Join

The Join operator is used to add an external attribute to a given relation, so that it works like a left outer join [Garcia-Molina et al., 2001]. It is defined as:

$$
\begin{aligned}
R'_N(K', V') \quad &\leftarrow Join(R_{N_1}, R_{N_2}, R_{N_3}), \text{ where } \{R_{N_1}, R'_N\} \in S_{N_1} \wedge \\
&\qquad \{R_{N_2}, R_{N_3}\} \in S_{N_2} \wedge K' = K_1 \text{ or} \\
R'_L(K', S', E', L') \quad &\leftarrow Join(R_{L_1}, R_{L_2}, R_{L_3}), \text{ where } \{R_{L_1}, R'_L\} \in S_{L_1} \wedge \\
&\qquad \{R_{L_2}, R_{L_3}\} \in S_{L_2} \wedge K' = K_1
\end{aligned}
$$




```
relEnd = Set(relNew, relSearch, intersection, relClNew.K, relSeDifUrl.E);
```

Figure 4.18. Example of the Set operator with the option Intersection, applied to two relations of different types.

The operator is used to join values of relation R_{N_3} (or R_{L_3}) from S_{N_2} (or S_{L_2}) into relation set S_{N_1} (or S_{L_1}). Three relations are input. The first relation comes from S_{N_1} and the second relation comes from S_{N_2}. they are then used for comparison before associating the value attribute of relation R_{N_3} in S_{N_2} to set S_{N_1}, as part of the output relation. Figure 4.19 presents the syntax of the Join operator.

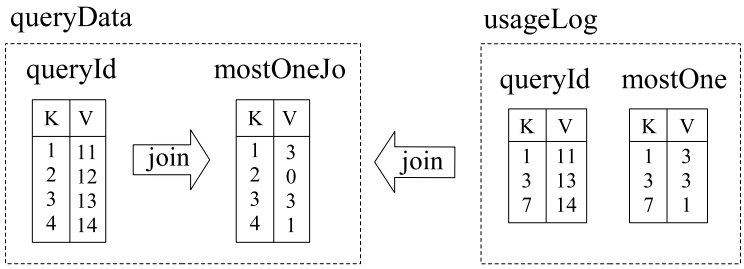
A default value must be passed to be used when a value in the first relation is not found in the second relation. When a value in the first relation is found more than once in the second relation, only one value from the second relation is joined, which is the first value found. This decision is because the design of operators might avoid operations that add new tuples, so that the output would not belong to any input's set (as discussed in Section 4.1.1).

```
Join := OutputRel '=' 'Join' '(' FirstInputSet ',' SecInputSet ',' AnyRel ','
        AnyRel ',' AnyRel ',' DefaultValue ')'
FirstInputSet := Name
SecInputSet := Name
DefaultValue : StringValue | NumValue
StringValue := "'" Name "'"
```

Figure 4.19. Syntax of the Join operator.

Figure 4.20 presents an example of the Join operator. Relation *mostOne* from relation set *usageLog* has its value used to generate the output *mostOneJo*, which belongs to relation set *queryData*. The two attributes used for comparison are referred to as *queryId*, but belong to different relation sets (otherwise the Join operator would not be required).

Observe that tuple 2 in relation *queryId* of set *queryData* does not have its associated value (12) found in *mostOne*, so that a default value 0 passed by the user is included for relation *mostOneJo*.




```
mostOneJo = Join(queryData, usageLog.mostOne, queryId.V, queryId.V, mostOne.V,
                                                                        0);
```

Figure 4.20. Example of the Join operator.

4.3.7 Convert

The Convert operator is used to convert relations of a type into relations of an other type, so that users can dynamically create relations based on other relations. The operator is defined as:

$$
\begin{aligned}
R'_N(K', V') \quad & \leftarrow Convert(R_L), \text{ where } K' = \{S|E|L\} \wedge V' = \{S|E|L\} \text{ if } V' \text{ exists or} \\
R'_L(K', S', E', L') \quad & \leftarrow Convert(R_{N_1}, R_{N_2}, [R_{N_3}]), \text{ where } S' = \{K_1|V_1\} \wedge E' = \{K_2|V_2\} \wedge \\
& \qquad\qquad L' = \{K_3|V_3\} \text{ if } L' \text{ exists}
\end{aligned}
$$

The first possibility is the conversion of a link relation into a node relation. Either the start, the end, or the value attribute of the link relation can be chosen to be the key and the value attributes of the returned node relation. The value attribute in the output relation may exist or not.

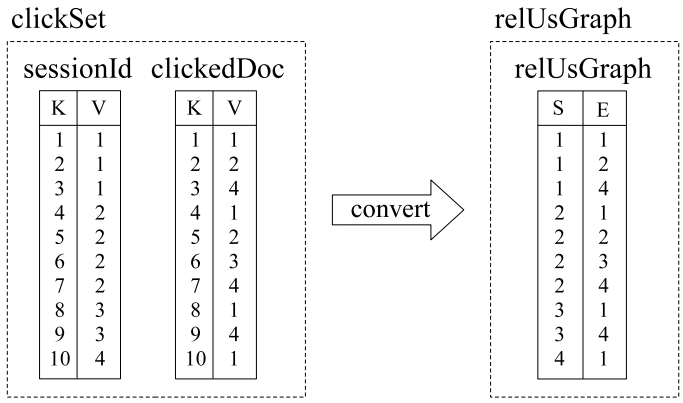
The second possibility is the conversion of relations of a node relation set into a link relation. In this case up to three node relations with associated attributes may be chosen: to represent the start, the end, and the value attributes of the output link relation, respectively. Figure 4.21 presents the syntax of the Convert operator.

Figure 4.22 presents an example of the Convert operator. The value attribute from relation *sessionId* and the value attribute from relation *clickedDoc* are converted into the start and end attributes of the output relation *relUsGraph*, respectively. Note that the

```
Convert := OutputNodeRel = 'Convert' '(' InputLinkSet ',' IntRel
                [ ',' AnyRel ] [ ',' AnyRel ] ')' |
           OutputLinkRel = 'Convert' '(' InputNodeSet ',' IntRel ','
                IntRel [ ',' AnyRel ] ')'
```

Figure 4.21. Syntax of the Convert operator.

output belongs to a new relation set, whose default name is the same as the output relation, *relUsGraph*.




```
relUsGraph = Convert(clickSet, sessionId.V, clickedDoc.V);
```

Figure 4.22. Example of the Convert operator.

4.4 Data Mining Operators

The data mining operators are used to process complete typical Web mining tasks, such as clustering, searching, textual comparison, link analysis, and co-citation analysis.

4.4.1 Search

This operator allows a textual attribute of a node relation to be searched in a textual attribute of another node relation. The typical application for this operator is querying, although it is also important for comparison of texts in general. Several comparison methods are included, like conjunctive and disjunctive comparisons, TF-IDF [Salton et al., 1975], Okapi BM25 [Sparck-Jones et al., 2000], exact match, and shingles (text win-

dows) [Broder et al., 1997]. The Search operator is defined as:

$$R'_L(K', S', E', L') \leftarrow Search(R_{N_1}, R_{N_2}), \text{ where } R_{N_1} \in S_{N_1} \wedge R_{N_2} \in S_{N_2} \wedge S' \Rightarrow S_{N_1} \wedge E' \Rightarrow S_{N_2}$$

The output is a new link relation R'_L, whose start attribute is compatible with input set S_{N_1}, and end attribute is compatible with input set S_{N_2}. Then, start nodes represent tuples in the first input relation set, where queries are typically represented, whereas end nodes represent tuples in the second input relation set, where the texts for searching are typically represented. Thus, the output is a bipartite graph, and users may take advantage of this property by using other operators in the sequence of their programs, for example to manipulate only the list of documents returned, discarding the queries.

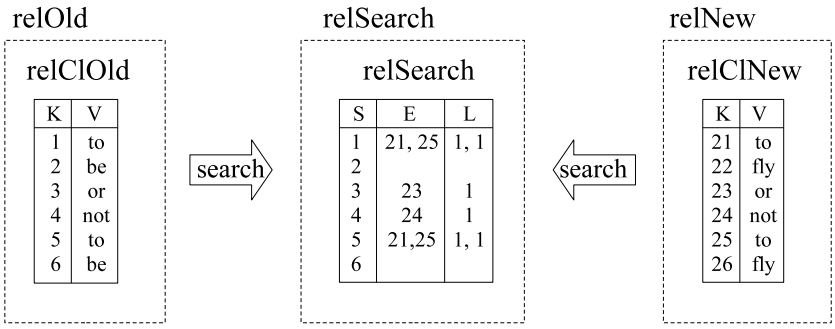
The order or ranking of the results of a query are implicitly represented by the order in which the end nodes appear. The end node at position one for a given start node represents the top ranking document for that query. The value attribute in the output link relation stores the similarity between the query and the text. Figure 4.23 presents the syntax of the Search operator.

```
Search := OutputLinkRel '=' 'Search' '(' FirstInputNodeSet ','
          SecInputNodeSet ',' TxtRel ',' TxtRel ',' CompMethod ')'
FirstInputNodeSet := Name
SecInputNodeSet := Name
CompMethod := SimpleMatch | Shingles | Tfidf | Bm25
SimpleMatch := 'ExactMatch' | 'AND' | 'OR'
Shingles := 'Shingles' ',' NumRes [ ',' MinSim ',' Overlap ',' ShinSize ]
NumRes := IntValue
MinSim := NumValue
Overlap := 'yes' | 'no'
ShinSize := IntValue
Tfidf := 'TFIDF' ',' NumRes
Bm25 := 'BM25' ',' NumRes [ ',' k1 ',' k3 ',' b ]
k1 := NumValue
k3 := NumValue
b := NumValue
```

Figure 4.23. Syntax of the Search operator.

Figure 4.24 presents an example of the Search operator. The value attribute from relation *relClOld* is searched in the value attribute from relation *relClNew*. The output is

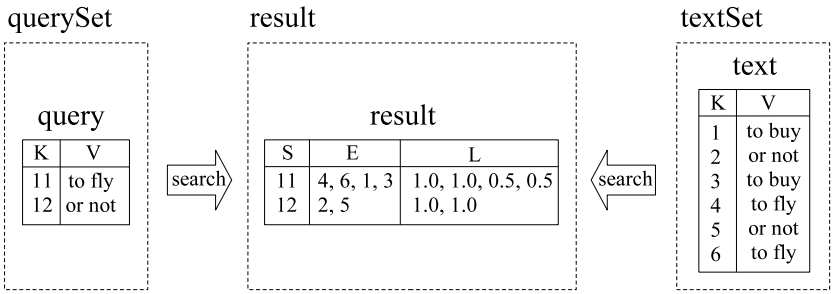
relation *relSearch*. For instance, the text *to*, existent in tuples 1 and 5 in *relClOld*, is found in tuples 21 and 25 in *relClNew*.




```
relSearch = Search(relOld, relNew, shingles, 100%, relClOld.V, relClNew.V);
```

Figure 4.24. Example of the Search operator used for text comparison.

Figure 4.25 presents another illustration, which contains the same relations presented in Figure 3.4. Relation *query* has two queries that are searched in relation *text*. The result is relation *result*.




```
result = Search(querySet, textSet, OR, query.V, text.V);
```

Figure 4.25. Example of the Search operator, as typically used for querying.

4.4.2 Compare

The Compare operator permits the comparison among all elements of a textual attribute of a single input node relation. The comparison methods are the same as for the Search operator, presented in Section 4.4.1. Compare is defined as:

$$R'_L(K', S', E', L') \leftarrow Compare(R_N), \text{ where } R_N \in S_N \wedge R'_L \in S_L \wedge S_L \Leftrightarrow S_N$$

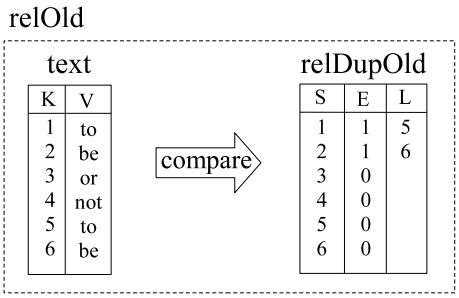
The output is a new link relation R'_L compatible with input set S_N. Thus, each link represents a pair of tuples in S_N, for associating nodes that share a minimal similarity percentage informed by the user. The comparison is based on a textual attribute of some relation in S_N (typically the text of documents).

Figure 4.26 presents the syntax of the Compare operator, which has two options: *Sparse* and *Dense*. For the option Sparse, the output does not have a new attribute, apart from the start and end nodes to represent the graph. A link is created between a pair of similar documents, in only one direction. For example, if documents A, B and C are similar, links are created, for example, from A to B and from A to C, not between every pair of similar documents. Note that the similarity value cannot be represented, because not all the links are represented (the representation resembles a cluster of documents). For the option Dense, every direction is represented in the output, to which the presence of the value attribute in the output makes sense, to store the similarity between each pair of nodes with similar content.

```
Compare := OutputLinkRel '=' 'Compare' '(' InputNodeSet ',' BodyComp ')'
BodyComp := 'Sparse' ',' TxtRel ',' CompMethod |
            'Dense' ',' TxtRel ',' CompMethod
CompMethod := SimpleMatch | Shingles | Tfidf | Bm25
SimpleMatch := 'ExactMatch' | 'AND' | 'OR'
Shingles := 'Shingles' ',' NumRes [ ',' MinSim ',' Overlap ',' ShinSize ]
NumRes := IntValue
MinSim := NumValue
Overlap := 'yes' | 'no'
ShinSize := IntValue
Tfidf := 'TFIDF' ',' NumRes
Bm25 := 'BM25' ',' NumRes [ ',' k1 ',' k3 ',' b ]
k1 := NumValue
k3 := NumValue
b := NumValue
```

Figure 4.26. Syntax of the Compare operator.

Figure 4.27 presents an example of the Compare operator, which is applied to relation *text* from relation set *relOld*, with the option Sparse and using the comparison method Exact Match. Relation *relDupOld* shows that the text of tuples 1 and 5 (*to*) and tuples 2 and 6 (*be*) are the same.




```
relDupOld = Compare(relOld, sparse, exactmatch, text.V);
```

Figure 4.27. Example of the Compare operator with the option Sparse.

4.4.3 CompGraph

Allowing the same operations as operators Search and Compare, the CompGraph operator receives a link relation and compares pairs of textual attributes of the start and end nodes. The CompGraph operator is defined as:

$$R'_L(K', S', E', L') \leftarrow CompGraph(R_L, R_{N_1}, R_{N_2}), \text{ where } \{R_L, R'_L\} \in S_L \wedge R_{N_1} \in S_{N_1} \wedge R_{N_2} \in S_{N_2} \wedge K' = K_1 \wedge \{S_1, S'\} \Rightarrow S_{N_1} \wedge \{E_1, E'\} \Rightarrow S_{N_2}$$

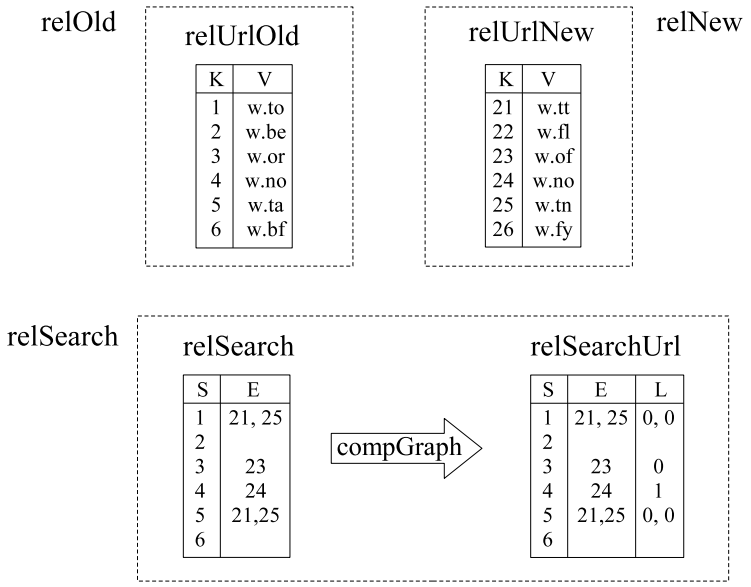
The output is a link relation that belongs to the same set of the input. The value attribute in the output is used to store the similarity of the compared documents. Similarly to the CalcGraph operator presented in Section 4.3.3, clauses $S \Rightarrow S_{N_1}$ and $E \Rightarrow S_{N_2}$ mean that there must exist a compatible node relation associated to the start set of the input relation, and a compatible node relation for the end set. Compatible node relations must exist because textual values attributes of them are used for the comparison operation, which involves each start-end pair of the input link relation set S_L.

Figure 4.28 presents the syntax of the CompGraph operator. We notice that the two input relations are taken from the compatible node relation sets.

For illustration, the CompGraph operator is applied to relation *relSearch* in Figure 4.29, for the comparison of URLs of the associated node relation sets *relOld* and *relNew*, also presented in the figure. The output is relation *relSearchUrl*. The link from 4 to 24 has similarity 1 because the URLs of tuple 4 in *relUrlOld* and tuple 24 in *relUrlNew* are the same.

```
CompGraph := OutputLinkRel '=' 'CompGraph' '(' InputLinkSet '.'
        AnyRel ',' TxtRel ',' TxtRel ',' CompMethod ')'
CompMethod := SimpleMatch | Shingles | Tfidf | Bm25
SimpleMatch := 'ExactMatch' | 'AND' | 'OR'
Shingles := 'Shingles' ',' NumRes [ ',' MinSim ',' Overlap ',' ShinSize ]
NumRes := IntValue
MinSim := NumValue
Overlap := 'yes' | 'no'
ShinSize := IntValue
Tfidf := 'TFIDF' ',' NumRes
Bm25 := 'BM25' ',' NumRes [ ',' k1 ',' k3 ',' b ]
k1 := NumValue
k3 := NumValue
b := NumValue
```

Figure 4.28. Syntax of the CompGraph operator.




```
relSearchUrl = CompGraph(relSearch.relSearch, exactmatch, relUrlOld.V,
                         relUrlNew.V);
```

Figure 4.29. Example of the CompGraph operator.

4.4.4 Cluster

The Cluster operator is used to create clusters of tuples in node relations. It is defined by:

$$R'_N(K', V') \leftarrow Cluster(R_{N_1}, [R_{N_2}], [R_{N_3}]), \text{ where } \{R_{N_1}, R_{N_2}, R_{N_3}, R'_N\} \in S_N \wedge K' = \{K_1 \cup K_2 \cup K_3\}$$

According to the definition, the output belongs to the same set of the input, and every element of the input relation is represented in the output, as $K' = \{K_1 \cup K_2 \cup K_3\}$. The output value attribute stores the cluster number associated to each input element.

Figure 4.30 presents the syntax of the Cluster operator. The operator receives a node relation set and a few relations. The value attributes are used to produce clusters by using the k-means algorithm [MacQueen, 1967]. The number of clusters is also a parameter passed by the user. Both Euclidean and Manhattan distances are available.

```
Cluster := OutputNodeRel '=' 'Cluster' '(' InputNodeSet ',' ClusMethod ','
           NumRel [ ',' NumRel ] [ ',' NumRel ] ',' NumValue ')'
ClusMethod := 'Euclidean' | 'Manhattan'
```

Figure 4.30. Syntax of the Cluster operator.

4.4.5 Disconnect

The Disconnect operator allows the identification of clusters in link relations, rather than in node relations like the Cluster operator. It is defined as:

$$R'_N(K', V') \leftarrow Disconnect(R_L), \text{ where } K' = \{S \cup E\} \wedge R_L \in S_L \wedge \text{if } S_L \Leftrightarrow S_N, R'_N \in S_N$$

The input is a link relation R_L, in which each node of the graph is classified into a cluster, according to its connectivity in the graph. Thus, the output is a node relation that contains all the nodes of the input graph, and a value attribute to associate a cluster number to each node. Observe that if both start and end attributes in R_L are compatible with the same node relation set, then the output of the Disconnect operator belongs to this node set. Other subsequent operators in a program may take advantage of this property.

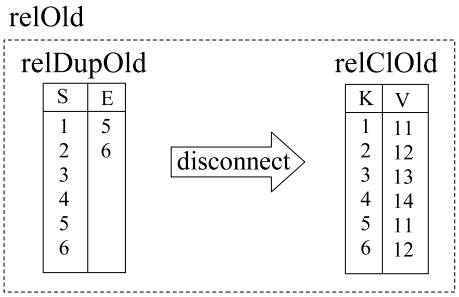
Figure 4.31 presents the syntax of the Disconnect operator. There are two possibilities for this operator: to create clusters of connected components, in which the direction is not

important, and to create clusters of strongly connected components [Cormen et al., 1990], which takes into account the direction of links.

```
Disconnect := OutputNodeRel '=' 'Disconnect' '(' InputLinkSet ','
              DistMethod ')'
DistMethod := 'Connected' | 'Strongly'
```

Figure 4.31. Syntax of the Disconnect operator.

For illustration, Disconnect is applied to link relation *relDupOld* in Figure 4.32, to identify clusters of connected components. As nodes 1 and 5, and nodes 2 and 6, are connected in relation *relDupOld*, the same cluster numbers are associated to these connected nodes in relation *relClOld*, which are respectively numbers 11 and 12. Nodes without links in *relDupOld* have a unique cluster number in *relClOld* (nodes 3 and 4).




```
relClOld = Disconnect(relOld, connected, relDupOld.V);
```

Figure 4.32. Example of the Disconnect operator with the option Connected.

4.4.6 Associate

The Associate operator is used for mining by association rules. It can be applied to data in both textual and numeric formats. For any data format, the user can choose to return either frequent itemsets, association rules, or sequential rules [Liu, 2007]. Although sequential rules is not the most common technique in association rules mining, it is quite useful for usage log mining, in which the action of the user can be observed in an evolutionary way. The Associate operator is defined as:

$$R'_L(K', S', E', L') \leftarrow Associate(R_{N_1}, R_{N_2}), \text{ where } \{R_{N_1}, R_{N_2}\} \in S_N \wedge R'_L \text{ is a new relation which } S' \text{ and } E' \text{ have no relationship with } K_1 \text{ or } K_2$$

Output R'_L must represent rule implications. As WIM is not specially designed to deal with implications, we use link relations to represent them, in which a start node represents an item in the left-hand side of the rule implication, and an end node represents an item in the right side, whereas the label is an identifier for the rule. Then, nodes of the graph represent items (whose possible representations are presented below) rather than tuples from input or other compatible relation.

Figure 4.33 presents the syntax of the Associate operator.

```
Associate := OutputLinkRel '=' 'Associate' '(' InputNodeSet ','
        BodyAssoc ',' Support ',' Confidence ')'
BodyAssoc := 'Value' ',' RuleMethod ',' IntRel ',' IntRel |
        'Term' ',' RuleMethod ',' IntRel ',' TxtRel
RuleMethod := 'Itemset' | 'Association' | 'Sequencial'
Support := NumValue
Confidence := NumValue
```

Figure 4.33. Syntax of the Associate operator.

As an example of the graphical representation of rule implications, observe again Figure 4.15. If relation *relAgg* was the result of the Associate operator, this graph would mean that, for instance, for rule number 1 (label 1), items 1 and 2 would imply in item 4; and for rule 2, item 3 would imply in item 4. Other two rules would exist for labels 3 and 4 respectively.

From this representation, users can observe the output graph in their analysis or use other WIM operators to manage data before analyzing. For example, by using the Aggregate operator, users can recover rules with several items involved, or find out the number of occurrences of each item in the left-hand or right-hand side of an implication.

Apart from the rule method (one of the three referred above) and the values of support and confidence (necessary for association rules [Agrawal et al., 1993]), the Associate operator has two options regarding the representation of items and transactions: *Value* and *Term*. Users must choose one of these options according to the attributes they want to use in order to represent transactions and items.

The Value option is used when the input relation set has two relations having integer values attributes, one to represent transactions and other to represent items. For example, consider Figure 4.22. Relations *sessionId* and *clickedDoc* could be used to represent, respectively, transactions and items, in order to associate clicked documents according to sessions in which they are clicked.

For the Value option, observe that nodes of the output graph are represented by integers of the attribute that represent the items in the input relation. Then, the output does not go to the same set as the input. However, users can apply other operations to associate results of the Associate operator to other available relations. Following the example above, as there might be a relation to represent documents, results of the Associate operator could be merged with that relation, by applying the Set operator or the Join operator.

The second option is Term, which is used when the input relation has an integer to represent transactions and terms of a textual attribute are used as items. Consider, for example, a relation set with a relation for identifying user sessions, which is an integer, and a relation for the query text. It is possible to associate terms of the query according to sessions in which queries are requested.

For the option Term, the nodes of the output graph are represented by identifiers of the terms existent in the chosen input textual attribute. As part of WIM pre-processing tasks, the vocabulary of each textual attribute must be processed, as a requirement for the proper indexing of texts. This vocabulary is represented as another relation, for which an identifier is associated to each different term. We do not intend to present further details, but WIM is able to recover the same identifiers of the textual attribute vocabulary, and make the output of the Associate operator with the option Term automatically compatible with the vocabulary relation. Then, the output is not compatible with the input, though, for the option Term, it is compatible with another relation.

For both options, relations that do not represent usage data can be employed as well. For instance, with the option Term, the text of documents can be used to associate terms in documents.

4.4.7 Analyze

Analyze is an operator for link analysis. It is used for estimating the relevance of a node in a graph, according to its connectivity. The following methods are available: *Pagerank* [Brin and Page, 1998], *in-degree*, *authority*, and *hub* [Kleinberg, 1999]. Analyze is defined as:

$$R'_N(K', V') \leftarrow \mathit{Analyze}(R_L), \text{ where } K' = \{S \cup E\} \wedge R_L \in S_L \wedge \text{if } S_L \Leftrightarrow S_N, R'_N \in S_N$$

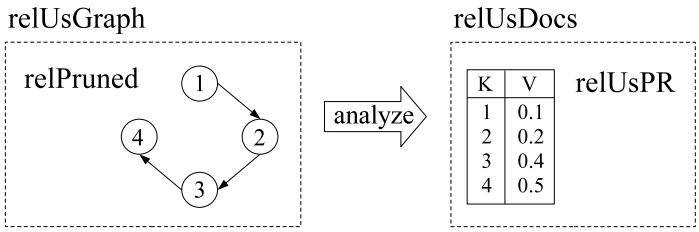
The input is a link relation R_L, in which each node of the graph is associated to a

relevance weight, according to the method informed by the user. Thus, the output is a node relation that contains all the nodes of the input graph, whose value attribute stores the relevance weight for each entry. Similarly to the Disconnect operator, if both start and end attributes in R_L are compatible with the same node relation set, then the output of the Analyze operator belongs to this node set.

Figure 4.34 presents the syntax of the Analyze operator, and Figure 4.35 presents an example of this operator. It is applied to relation *relPruned* to calculate Pagerank, returning relation relUsPR (whose values are estimated just for illustration).

```
Analyze := OutputNodeRel '=' 'Analyze' '(' InputLinkSet ',' Algorithm ')'
Algorithm := 'Pagerank' | 'Hub' | 'Authority' | 'Indegree'
```

Figure 4.34. Syntax of the Analyze operator.




```
relUsPR = Analyze(relUsGraph, pagerank, relPruned.SE);
```

Figure 4.35. Example of the Analyze operator.

4.4.8 Relink

The Relink operator adds new links to a link relation, according to one of the following methods: *co-citation* (bibliographic co-citation [Small, 1973]), *coupling* (bibliographic coupling [Kessler, 1963]) or *transitivity* [Gonnet and Baeza-Yates, 1991]. Relink is defined by:

$$R'_L(K', S', E', L') \leftarrow Relink(R_L), \text{ where } K \subseteq K' \wedge \{S' \cup E'\} = \{S \cup E\} \wedge R_L \in S_L \wedge \text{if } S_L \Leftrightarrow S_N, R'_L \in S_L$$

The output value attribute stores the distance between node pairs of the relation. Existing links have label 0, and new links have label 1. Thus, R'_L contains the edges of

the input graph plus new edges added according to the chosen method. If both attributes start and end are compatible with the same given node relation set, then the output R'_L belongs to the input set. Figure 4.36 presents the syntax of the Relink operator.

```
Relink := OutputLinkRel '=' 'Relink' '(' InputLinkSet ',' BodyRelink ')'
BodyRelink := CitMethod ',' Direction ',' Density ',' |
              'Transitivity'
CitMethod := 'CoCitation' | 'Coupling'
Direction := 'Both' | 'Single'
Density := 'All' | 'Adjacent'
```

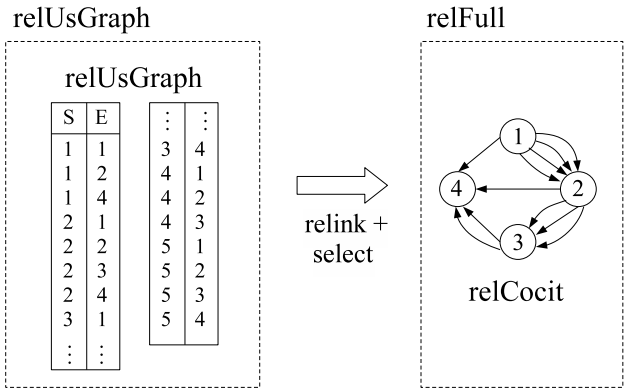
Figure 4.36. Syntax of the Relink operator.

For the options Co-citation and Coupling, it is not obvious in which direction to include a link and between which pair of nodes sharing a co-citation or coupling unit a link may be inserted. Regarding the direction, the user can choose both directions or a single direction. Regarding the number of links, the user can choose to add a link between all the pairs which share a co-citation or coupling unit, or to add links only between adjacent nodes.

For example, consider the link relation *relUsGraph* in Figure 4.37, where start and end nodes are represented in two columns (unlike the other examples in this chapter). For instance, start node 1 links to end nodes 1, 2, and 4. The application of the Relink operator with the option Co-citation, single direction, and adding links only between adjacent nodes, results in the same links existent in the input graph with label 0, plus the links represented in graph *relCocit*, with label 1. For instance, for start node 1, links are inserted from end node 1 to 2 and from end node 2 to 4, which are the adjacent co-cited nodes. The example of Figure 4.37 includes only the selection of nodes with label 1, represented in relation *relCocit* with the label omitted.

4.5 Comparison with the Relational Algebra

The WIM algebra was not designed after the relational algebra, but after analyzing the most important Web mining tasks, and observing the kind of data manipulation needed to deal with Web mining applications. In this section we discuss the latter in order to compare the WIM data manipulation operators with the relational algebra, despite their different goals.




```
relFull = Relink(relUsGraph, cocitation, single, adjacent, relUsGraph.SE);
          relCocit = Select(relFull, value, !=, 0, relFull.L);
```

Figure 4.37. Example of the Relink operator. The Select operator is applied afterwards, for eliminating the links existent in *relUsGraph*, facilitating visualization.

A fundamental operation in the relational algebra is projection. WIM does not require an operator for projection, as WIM relations have limited number of attributes, and relations are organized in sets. Another fundamental operation in the relational algebra is selection, which is used to select tuples according to user-specified criteria on any type of attribute. WIM operator Select is applied only to numerical attributes, though the selection of textual attributes can be done with the data mining operator Search. Selection of multiple documents with overlapped content within a relation set, which is not covered by the relational algebra, can be done with the Compare operator.

The selection of specific values in Web mining applications is not as common as in relational databases. For example, searching for the name "Catherine Smith" does not mean anything in most Web mining applications, but is important in relational databases, because the tuple for this and other related entries might eventually need to be modified. The implication is that the Search operator allows a textual attribute of a relation set to be searched in a textual attribute of another relation set. If the name "Catherine Smith" is the query, then the user will need to create a relation with only that entry to be used as the input of the Search operator, which is not as simple as in the relational algebra select. For the same reason, WIM does not allow search using regular expressions (for example, character '%' substitutes any character).

In order to keep the WIM language and implementation simple, the WIM algebra does not provide several conjunctive or disjunctive clauses in a single query. However, users

can have the same result by applying a few selection or querying operations in sequence, and using the option Union of the Set operator for disjunctive selection.

Regarding set operations, the relational algebra allows the union, intersection and difference of query results. A requirement is that the two input relations must have a similar set of attributes. WIM operator Set does not have this requirement, as a consequence of the WIM data model design. The output relation is compatible with the first input (with exception of the option Union), then the other attributes that the input relation sets have do not matter. This flexibility is very important to the power of the WIM algebra. Further, WIM Set allows the merging of relations of different types, i.e., link and node relations.

Relational algebra joining operations allow multiple attributes of two relations to be joined. As so far WIM allows only one attribute to be joined, multiple calls to the Join operator are required to join multiple attributes. The relational algebra has different types of join. The main semantic difference among the implementations is how to represent tuples that are not found in both relations. Options in the relational algebra include: not representing tuples for missing values, using the value of one of the relations when the value exists, and using a null value. Since for the WIM model it is important to keep compatibility of output relations with respect to input relations, rather than merging all the attributes of two relations, WIM's Join operator allows only the inclusion of one attribute of the second input relation into the first input relation (left outer join), so that the output is compatible with the first input.

A limitation of the WIM join is that only one attribute of each input relation set can be used for comparison when joining. Although it is a limitation, it allows simple and efficient implementation of the operator, and does not seem to expressively limit the set of problems to which the operator can be applied, as shown in the use cases presented in this thesis. Relational databases tend to have much more sparse data, requiring to join multiple attributes, than Web databases, as intrinsic property of Web objects like documents, link structure and usage data. Web data attributes, when available to one entry, are often available to all other entries. Note that WIM can be extended in the future to implement other types of joins.

Aggregation operations are important in the relational model and even more important in WIM. For this purpose the Aggregate operator is implemented in WIM.

Regarding database modifications, the WIM data model does not allow the modification of existing attributes of relations. Instead, new attributes can be added or new relations and relation set created. The relational model allows the update of both numerical

and textual attributes, whereas WIM allows operations on numerical attributes, by using operators Calculate and CalcGraph, returning a new attribute as result of a mathematical or statistical operation, but only for numerical attributes. We do not see any advantage in allowing operations on strings, such as adding a prefix, for Web data.

WIM does not allow the insertion of tuples to existing relations without the use of an operator. This action is not required for the designed WIM data model, in which insertion of new Web data is modeled as a new relation or relation set in the database.

4.6 Concluding Remarks

In this chapter we have presented the WIM algebra, divided into data manipulation operators and data mining operators. The operators obey the compositionality property, which means that the output of any operator can be used as input of any other operator. Constraints are limited to the type of the input relation and attributes, which are characteristics intrinsic to each operator defined in this chapter.

We have included in this chapter a comparison of the WIM algebra data manipulation operators and the relational algebra. We have shown the similarities between the two algebras, and the differences that are important taking into consideration our application scenario, mining Web data. Although WIM was not designed after the relational algebra, WIM performs the most important and general relational tasks.

Chapter 5

Prototype Architecture and Implementation

The WIM model presented in Chapter 3 and its algebra presented in Chapter 4 were materialized as a software prototype, which allowed to run a set of use case applications over the prototype as a proof of concept. In this chapter we present the architecture of the WIM prototype, including the main issues involved in its implementation, and discuss efficiency and scalability issues, along with proposals of alternative architectures.

5.1 Software Prototype Architecture

In this section we present the software architecture of the WIM software prototype. Figure 5.1 presents the general architecture of the WIM prototype, with the data organization, the processing modules, and how they interact. The WIM architecture is composed of six main modules: Compiler, Executor, Indexer, Visualizer, Preprocessor and Web crawler.

The data flow is initiated at the Web crawler and at the Preprocessor. The *Web crawler* is responsible for collecting Web data, whereas the *Preprocessor* translates the raw Web data into a format that WIM can recognize. Users can upload datasets by means other than using the crawler module, a feature needed for the manipulation of private datasets. After processing a WIM program, module *Visualizer* presents the output to the user by means of a friendly interface.

The Web crawler and the Visualizer modules are not native of WIM. There are several open source crawlers, and specific tools for visualization of graphs and analysis of statistical

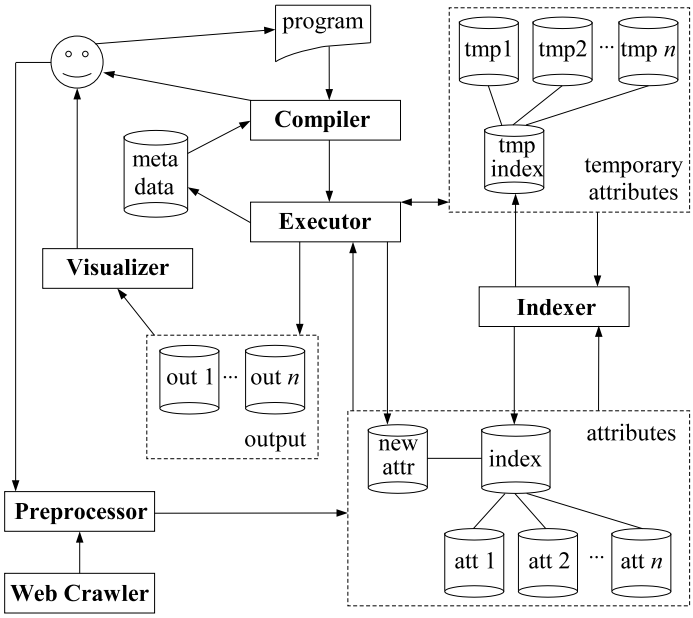


Figure 5.1. WIM general software prototype architecture.

results, which may be added to WIM.

The *Indexer* is a module that checks for the need of indexing attributes of relations, receiving an attribute as input and returning the index for that attribute, which is represented in Figure 5.1 as a single database (named 'index'). The Indexer is very important to ensure efficiency and scalability up to some extent.

The WIM storage architecture is column-oriented [Baeza-Yates et al., 2000; Stonebraker et al., 2005], which means that each attribute in a relation set is stored independently of each other, following the data model presented in Chapter 3. We shall discuss the underlying implementation design decision in Section 5.2. The WIM prototype differentiates between *stored* relations and *temporary* relations. A temporary relation is created during a program execution, and is used only within that program. Attributes of temporary relations are volatile to the program and most are not stored on disk (an exception is made for large textual attributes). Regular attributes of relations are stored on disk and associated to registered relation sets, and can be opened by different WIM programs.

All metadata are stored in a database. The administrator needs to set up this database, declaring the relation sets that will compose the WIM database, and providing further information about each relation relation, such as the path of the real data file and the format in which it is stored. The meta database also stores the grammar and the

semantic rules of the WIM language, which is required in case the user needs to modify the language, in order to include a new operator or a new option for an existing operator.

The main modules of WIM are the Compiler and the Executor. The *Compiler* has two important tasks. First, it has to parse the WIM program code, according to the WIM operators that drive the WIM language, verifying whether it is free of syntax errors. Then, the compiler recognizes the tokens, such as operators, input and output relations, attribute names, and specific options for each operator, so that the semantic analyzer can run. An example of semantic test to be performed is whether a given requested relation actually exists for the requested relation set, and whether the type of the relation's attribute is the same as required by the operator.

The second important task of the compiler is to generate a main function in C source code. For that, the compiler uses the tokens from the input WIM program to select a previously defined C function that must be called for each existing operator. The selection is mainly based on the type of the input relation, on the type of the attribute involved, and on the options requested. Furthermore, in order to generate the main function, variables must be created and memory must be correctly allocated. Control functions, for instance to translate data types, are often needed. Finally, the compiler has the task of compiling the C code, using an external C compiler.

Module *Executor* receives the executable program from the WIM compiler, after compiling the C code. While the program is running, attributes are loaded on demand, and temporary attributes may be created to store attributes of temporary relations. At the end, the output is presented to the user by means of files, or delivered to the Visualizer module. If required by the user, temporary relations can become part of the WIM database, i.e., stored on disk for future use, either as part of an existing relation set or as a new relation set. This action required module Executor to update the metadata.

5.2 Implementation Issues

This section presents the most relevant issues addressed when materializing the WIM model into the WIM tool.

Web mining applications have two important differences in comparison with regular database applications:

1. Web content is highly dynamic, but Web data sets are **static**, as a result of crawling a Web snapshot. Therefore, deletion and insertion of tuples need not be considered,

since it is more efficient to create a new database for new crawls than to insert tuples with different properties into an existing relations[1].

2. Most Web mining applications do not deal with several attributes simultaneously: only the text or the graph structure is used in many cases.

These differences justify the storage of each attribute of a relation independently of each other, hence the decision was to make the WIM storage architecture column-oriented.

WIM uses only two data types to represent elements: floating point and string. An exception is for attributes that represent identifiers, which are integers. Characters are treated as strings, and integers as floating points. This solution is an alternative to substantially simplify the implementation of the WIM software prototype, without significant loss of time and space performance and without any loss of effectiveness. We refer to the WIM software data types as numeric and textual, rather than floating point and string, respectively.

The values of an attribute are sorted according to the order in which the identifier (attribute K) of the relation are represented. Link relations are identified by the start node S, which means that a reference to a given start node of the graph appears only once in any physical representation of a link relation. In fact, links are physically represented as a list of end nodes for each distinct start node of the graph (adjacency lists).

In Section 5.2.1 we present our decisions regarding the design of the first level functions, which are C functions declared to make the interface between modules Compiler and Executor. In Section 5.2.2 we present and justify the choice of the data structures to represent attributes, used in the first level functions.

5.2.1 First Level Functions

As introduced in the previous section, the WIM compiler maps each line of a WIM program into C function calls, as part of module Executor, generating a C file with the main function for that WIM program. It means that there is a set of functions for each operator, referred to as *first level* functions, that are implemented as part of module Executor, but must be

[1] An information retrieval problem is the design of incremental textual databases, avoiding the creation of a new database when a new crawl takes place. Our WIM design does not provide such functionality, because WIM is not designed towards very specific applications, such as crawling several snapshots of the Web under a given domain. However, WIM can deal with such applications by storing each crawl as an independent relation set.

known by module Compiler, as part of the interface between these modules. The first level functions are called from the main C function of each WIM program.

Roughly, the strategy for designing the first level functions depends on if the operator accepts link and/or node relations, accepts the manipulation of numerical and/or textual attributes, and sometimes it depends also on the options, when different options imply using a different number of attributes. This organization aims to pass to each first level function only the attributes and parameters that it needs, saving time and space.

Thus, if an operator accepts only one type of relation and one type of attribute, and its options do not imply in using different sets of attributes, only one first level function will be available. This is the case for operators Analyze, CalcGraph, CompGraph, Cluster, Disconnect, Relink, and Search.

Operators Aggregate, Associate and Calculate require more than one function, due to the need for different sets of attributes according to the different options used. For instance, option Single of the Aggregate operator requires only one extra attribute to be informed, whereas option Grouping requires two or three attributes, depending on sub-options.

Some operators accept both node and link relations as input: Aggregate, Calculate, Join, Set, and Select. In this case a different function is called for each option. Operators Join and Set also accept the manipulation of different types of attributes: integers (for manipulation of identifiers), numerical and textual attributes. Then, different functions are also required to treat each case separately.

Specific options are managed beyond the interface between the compiler and the executor. Therefore, the implementation of each operator can have as many functions as wanted, because inner functions are not important for the interface between the compiler and the executor. This means that each operator may have its own independent internal implementation, which facilitates the creation of new operations to be aggregated to WIM by independent collaborators in the future, and also allows scalability of the mining operation.

We notice that first level functions include functions not only to implement the operators of the WIM algebra, but also to convert data in different types available in WIM, to open attributes, to manage metadata, and to output data in order to be properly shown to the user.

Regarding the manipulation of attributes, numerical attributes of relations are entirely loaded in the main function and passed to other functions as parameters. Memory is released as soon as the attribute is no longer needed. This pre-loading does not happen

with textual attributes, due to memory requirements, hence textual attributes are loaded on demand.

5.2.2 Data Structures

In order to ensure uniformity among operators, WIM is designed with only a few different data structures, to be used by main functions automatically generated by the compiler. There are three data structures to represent node attributes (i.e., attributes of node relations) and two data structures to represent link attributes (i.e., attributes of link relations).

For node attributes, the first data structure is an array of integers. Some operators like Select return a subset of tuples from the input, without any new attribute, thus a simple array of integers represents the output relation. There is no representation for textual attributes, because texts are not opened and loaded in the main function and passed to first level functions.

The second data structure for node relations is an array of floating points. For the cases in which all the tuples of a relation are represented, the representation is *implicit*, and a floating point array is enough, since the order of tuples is preserved. This distinction between explicit and implicit identifiers is important to save space when representing attributes.

The third data structure for node relations is an array of pairs of an integer and a floating point. It is required whenever the identifiers must be *explicit*, i.e., whenever the output relation is a subset from the input, and the output includes a new attribute.

Link relations do not differentiate between implicit and explicit identifiers, as most of the times explicit representations would be required. The first data structure is an array in which each element has an integer to store the start node, an integer to store the number of end nodes for the corresponding start node, and an array of integers to store the end nodes. This representation is used whenever no other attribute of the relation must be represented, apart from the start and end nodes.

The second data structure is used whenever additional attributes are needed. Instead of having a list of integers to represent the end nodes, a list of integer–floating point pairs is used, so that the floating point value represents the label of the graph for numerical labels.

Having just a few data structures that can be recognized by the Compiler and Executor modules simplifies the interaction between these modules. For instance, in order to ensure compositionality, WIM must convert data structures, which would be much more

difficult if there were several possible data types to convert. For instance, a given operator may return a labeled graph, but the function that implements the next operator may demand an unlabeled graph as input. Then the conversion must be done before calling that function.

5.3 Efficiency and Scalability

The WIM data model allows efficiency and scalability that depend only on the software architecture and implementation. The design of the WIM as a dataflow modeling language is a great advantage regarding efficiency. First, dataflow languages allow non-dependent parts of a program to run in parallel. For example, consider the following program containing three operations:

$\text{outputRelation}_1 = \text{operator}_1(\text{inputRelation}_1, \{\text{options}\});$
$\text{outputRelation}_2 = \text{operator}_2(\text{inputRelation}_2, \{\text{options}\});$
$\text{outputRelation}_3 = \text{operator}_3(\text{outputRelation}_1, \text{outputRelation}_2, \{\text{options}\});$

Notice that the first and second operations are independent, as they are applied to different input relations, and the second operation does not use the output relation of the first operation. Then, they can run in parallel, whereas the third operation waits for both processes to finish, once it needs the output relations of the first and second operations.

The other important advantage of the WIM design as a dataflow modeling language is that the computational complexity of a WIM program is always equals to the computational complexity of the operation with highest complexity. Considering again the example program above, suppose operations 1, 2 and 3 have computational time complexity $O(n)$, $O(n^2)$ and $O(n\log n)$, respectively. As operations 1 and 2 run in parallel, the time complexity of the whole program is given by:
$T = \text{Max}(O(n), O(n^2)) + O(n\log n) = O(n^2) + O(n\log n) = O(n^2)$.

The conclusion is that the complexity of a WIM program depends on the complexity of the algorithms that implement the operators. In general, the operations with highest complexity are the mining operations. Taking into account that an ad-hoc implementation of the mining algorithm has the same complexity as the WIM mining operation, for a given Web mining problem, then a WIM program has the same complexity as the ad-hoc implementation. This ensures that the WIM model is designed to be efficient, and state-of-the-art implementations of the WIM operations ensure that a WIM program is as efficient as an ad-hoc implementation of the same task.

WIM does not include the development of a specific file system. The first reason for not extending is simplicity, aiming the fast implementation of an initial version of the tool prototype. The second reason is to allow scalability, since a prototype can use the operating file system, and an industrial-scaled implementation can use a distributed file system like HDFS (Hadoop Distributed File System) [Borthakur, 2007].

Regarding scalability, we suggest three main levels of scales for both data storage and processing architectures, as shown in Figure 5.2. The first level consists of data and processing in one single server. The second level consists of distributed processing, with data replicated in different servers, and each server running parallel tasks when possible. The third and most scalable architecture is distributed for each operation, rather than for the WIM program, and also supposes data storage replication and partitioning. In this case, each mining task is implemented to be processed distributively.

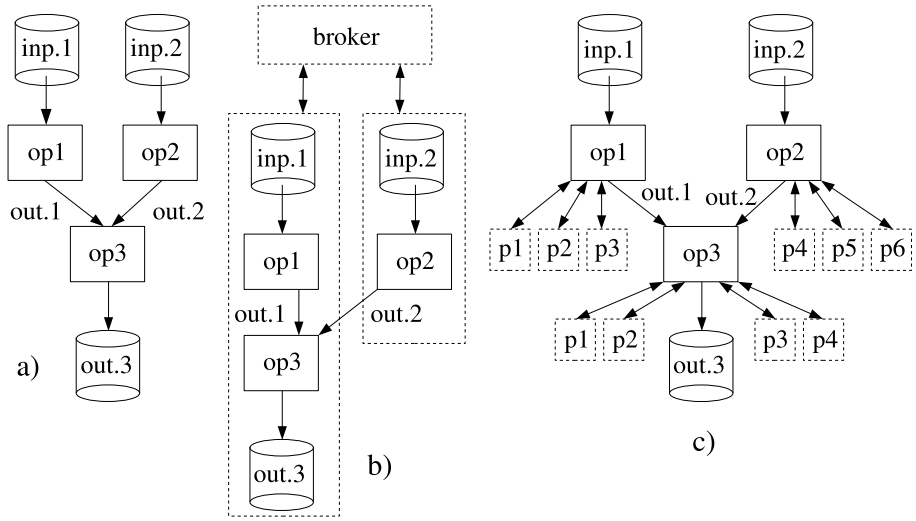


Figure 5.2. Three architecture levels to support different scales of data storage and processing.

Figure 5.2(a) refers to the current implementation of our prototype. It is not distributed, and data is stored locally. This architecture is adequate for small (around 3 million Web pages) to medium (around 100 million Web pages) datasets, depending on the operation to be processed. For instance, the comparison of a full collection using shingle paragraphs would not work properly for medium size datasets. On the other hand, operations like link analysis are suitable. Apart from its simplicity, another positive aspect of this architecture is that some operations (e.g., operations 1 and 2) can run in parallel, simultaneously making use of different processors in a multi-processing server.

Figure 5.2(b) refers to a parallel implementation. It has the advantage of parallel processing, though no gain in storage capacity. It is interesting in case of small- to medium-sized datasets with high processing. In the figure, dashed boxes mean servers. Observe that one machine is the broker, which has the task of managing metadata, distributing the data, and associating the execution of different operations in different machines.

Figure 5.2(c) refers to distributed processing of each operation. This architecture is the most efficient for large datasets and massive processing, as intensive processing mining tasks can be distributed into as many servers as available or needed. In the figure, the broker is not represented explicitly, but it is the server in which the operations are managed and sent to other servers (processors $p1$ to $p5$). This is our alternative for a forthcoming industrial scale implementation of WIM, which can run on Hadoop [Hadoop, 2008].

Observe that the parallelization of operations does not imply changing the design of the WIM data model depending only on the implementation, which shows that the WIM model is scalable.

5.4 Concluding Remarks

The WIM model presented in Chapters 3 and 4 is not only an abstract model. Rather, it has been implemented and actually can support Web miners. In this chapter we have presented some issues underlying our first implementation of a WIM software prototype, including its architecture.

As our current implementation is still a prototype, future modifications will be required to support industrial scale applications. For instance, the architecture of the Executor module will need to follow a distributed architecture, as suggested in Section 5.3, which does not imply changing the WIM data model. We are also planning to integrate modules Compiler and Executor, resulting in a simpler implementation using only one programming language. Notice that the advantage of having two separate modules is to allow the implementation of the parts in different languages, as we currently have the compiler in Java and the Executor in C, since the latter demands efficiency.

Chapter 6

Use Case: Genealogical Trees on the Web

The Web allows everybody the opportunity to become a publisher. Entities like companies, products, services, and people can be represented on the Web. One supposes that many of these potential publishers either have insufficient content or do not know how to represent their interests. Hence, some of the publishers refer to the Web itself to find good representations for their entities.

Little is known about the evolution of the textual content on the Web. We know how Web components (such as URLs and figures) evolve [Ntoulas et al., 2005] and how the structure evolves [Baeza-Yates and Poblete, 2006], but not how the content evolves. Our work provides the first step towards understanding how old content is used to create new content. That is, we want to find the original sources, if any, of the content or part of the content of a new page. We regard each source as a *parent* of a new page, in order to define a *genealogical tree* for the Web. The study of the genealogical tree allows us to understand what portion of the pages are either totally new parents or parents that are children of other parents.

Our experiments consider several representative snapshots of the Chilean Web and one snapshot of the Spanish Web. We estimate that 23.7% of new Web documents that appeared within a span of a year have content from previously published documents (see Section 6.5.5 for estimations). Most of them represent inter-site copies (approximately 75%), in which the publishers use content from a parent document from another site, and they need to find this document.

Web search engines are widely used to provide users with content that approximates what they are looking for. Web publishers are also Web users, and frequently are advanced search engine users. It is natural that if they need to find content on the Web, they will perform a query on a search engine.

In this direction, in addition to the genealogical tree study, we analyze whether there is any association between the sources of reused content (the parents) and the results of real queries from a search engine log. We see that parents are more connected to the Web graph and have a much higher Pagerank than other pages. Probably as a consequence, parents appear more often as results of queries and are much more clicked, which is shown in our analysis.

Our results are evidence that some Web publishers actually performed queries in order to find some content and republish. Thus, the conclusion is that part of the Web content is biased by the ranking function of search engines. Exploring our results beyond the scope of our research would explain the impact of the user's copy behavior on the quality of the search engine results, and how search engine designers can profit from that behavior, for example by associating a better page quality value for a previously low-quality page that is used as source of copy. In this case a child page would inherit properties of its parent (in case they are not duplicates or near-duplicates, that is, only part of the content is copied).

The main contributions of this chapter are: (i) to demonstrate how WIM can be applied to a real Web mining problem; (ii) to propose a methodology to study the genealogy of the Web content; (iii) to study the evolution of textual content on the Web, *i.e.*, how pieces of documents are reused; (iv) to generalize the content reuse results to the whole Web (or other subsets of the Web), providing an estimation of how much content is reused on the Web; and (v) to study how search engine ranking algorithms may influence the evolution of Web content. To the best of our knowledge, these contributions are not covered in previous work.

6.1 Conceptual Framework

6.1.1 Document Representation

We use the concept of shingles [Broder, 1998] to represent a document (the document fingerprint). A **shingle paragraph** (also referred here as just "shingle" or "paragraph") is a sequence of three sentences of the document. A **sentence** is a sequence of words ended by a period. If a period is not found until the 150th character, then the sentence is finished

at that point and a new sentence begins at the 151th character. This limitation is due to the fact that some documents have no period (for example, some program codes).

Each document is represented by the list of its shingles paragraphs, with overlap of sentences. As an example (also presented in Section 2.5 in this thesis), suppose we have a document D_1 containing seven sentences: $D_1 = s_1.\ s_2.\ s_3.\ s_4.\ s_5.\ s_6.\ s_7$, where s_i, $1 \leq i \leq 7$, is a sentence of the text, and i is the order of occurrence of the sentences in the text. The shingle paragraphs for D_1 are: "$s_1.\ s_2.\ s_3.$", "$s_2.\ s_3.\ s_4.$", "$s_3.\ s_4.\ s_5.$", "$s_4.\ s_5.\ s_6.$", "$s_5.\ s_6.\ s_7.$".

With the exception of the first two and last two sentences, every other sentence of the text is used three times in the document representation, in order to give a high degree in the comparison step of our method. The number of shingle paragraphs for a document is $|S| - 2$, where S is the set of sentences.

In our experiments we considered only documents with more than 450 characters and at least three shingle paragraphs, or equivalently five sentences. Preliminary experiments demonstrated that for considering similar two documents, it is necessary to have a minimal similarity between them, trying to avoid false matches (occurring in cases that only one or two popular shingle paragraphs are identical). We did not consider short documents because they cannot be represented by a minimal number of shingle paragraphs, and thus cannot reliably be compared with others. Around 25% of the documents in each collection were removed for these reasons.

6.1.2 Document Instance

We define a **cluster** as a set of documents with exactly the same textual content for a given collection. Each document in a collection is either (i) **duplicate**, if it belongs to a cluster, or (ii) **unique**, otherwise.

Each different content in a given collection is represented as a different **instance**. If a set of documents are duplicates among them, their contents are the same and they are represented by a unique instance. If a document is unique, its content is represented by an instance. The number of instances in a collection is the number of unique documents plus the number of clusters. By definition, an instance has no duplicate, although it represents either multiple documents or a unique document.

Most of the studies and conclusions presented in this chapter are concerned with the instance rather than with the document. The collections have a large number of duplicates, and thus it is incorrect to say that every duplicate in the same cluster is a parent when

part of the duplicate's content is found in a more recent collection. The concept of instance represents an important solution for the duplication problem in this work, since it compares content over different data sets.

6.1.3 Inter-Collection Relations

Consider a collection as being a snapshot of a given Web subset. Two documents, in two distinct collections, are **coexistent** (or they coexist), if their URLs are exactly the same. In this case, the same document URL exists in both collections (the content may differ). Two instances I_1 and I_2 in two distinct Web collections coexist, if at least one of the documents that I_1 represents has the same URL as one of the documents that I_2 represents.

An instance in a more recent collection has a **parent** instance in an older collection if it shares a minimal percentage of shingle paragraphs with the parent and the instance in the new collection is not represented in the old collection (it does not have a coexistent instance). The instance in the new collection is referred to as a **child**. The minimal percentage of shingle paragraphs used in this work is 20% (parent and child instances must share at least 20% of their content). After a manual analysis in a sample, we did not find false matches for this minimal similarity percentage.

A new instance is **orphan** if it does not have a coexistent instance or a parent in the old collection. An old instance is **sterile** if it does not have a coexistent instance or a child in the new collection.

In this research we study two kinds of relations: **inter-site** and **intra-site**. Excluding the `http://` prefix from the URL, the remaining of the string before finding a slash gives the site to which the document belongs. Inter-site relations require that the parent and the child belong to different sites, whereas for intra-site relations the parent and the child belong to the same site. Our study treats these relations separately because intra-site relations tend to occur when publishers reuse the content of their own site. For inter-site relations the way in which the publishers find the parent is much more difficult to guess.

The site to which a document belongs to is taken from its URL. Excluding the `http://` prefix, the remaining of the URL string before finding a slash gives the site which the document belongs to.

Mirrors were detected for inter-site relations (detection is not needed for intra-site relations). Two sites are considered mirrors of one another if at least 75% of their documents are clustered together (are duplicates in the same cluster) [Bharat and Broder, 1999], and

each site has at least 10 documents. This threshold guarantees that a minimal number of documents are clustered together.

6.1.4 Genealogical Trees on the Web

A genealogical tree on the Web is a representation for parents and children in different snapshots of a given Web subset. Each instance is classified into a different genealogical tree component. For the description of the components, let P_t be the set of parents in a snapshot t whose children belong to a snapshot $t+1$. Let C_t be the set of children in a snapshot t whose parents belong to a snapshot $t-1$. *Each document of each collection* is labeled as one of the following components:

1. Without Relation: represents instances that are not parent or child instances in a collection. They are sterile and/or orphan instances.

2. Original Parents (OrP): represents parents that are not children neither were parents in the previous collection (generating some child in the current collection). This component represents parents that have no relation with the older collection. The original parents set in a collection is the difference between the parents set and the union of the children set and the parents set in the previous snapshot, as shown in Equation 6.1.

$$\mathrm{OrP}_t = P_t \setminus (C_t \cup P_{t-1}) \tag{6.1}$$

For example, the original parents set of collection $t = 2004$ is the difference between the parents set of collection pair 2004-2005 (where $t = 2004$) and the union of the children set and the parents set in the previous snapshot (for collection pair 2003-2004, where $t - 1 = 2003$).

Notice that as we are looking for the original parent instances in snapshot t, P_{t-1} represents the parents in snapshot $t-1$ that still exist in snapshot t. We do not include coexistent instances in Equation 6.1 because it is obvious that for a parent in snapshot $t-1$ being a parent again in snapshot t, it has to exist.

3. Old Parents (OlP): represents instances that were parents in the previous collection, and are parents again in this collection. It means that they have some child in the current

collection. The set operation shown in Equation 6.2 indicates how old parents are found.

$$\mathrm{OlP}_t = P_t \cap P_{t-1} \tag{6.2}$$

As an example, the old parents for collection $t = 2004$ is the intersection between the parents set of collection pair 2004-2005 and the parents set for collection pair 2003-2004 (where $t - 1 = 2003$).

4. Children and Parents (CnP): represents instances that are children (with respect to the older collection) and parents (with respect to the more recent collection), as shown in Equation 6.3.

$$\mathrm{CnP}_t = P_t \cap C_t \tag{6.3}$$

As an example, the children and parents set of collection $t = 2004$ is the intersection between the parents set of collection pair 2004-2005 (where $t = 2004$) and the children set of collection pair 2003-2004 (where $t = 2004$).

5. Sterile Children (StC): represents children that are not parents, as shown in Equation 6.4. This component represents children that have no descendants in the more recent collection.

$$\mathrm{StC}_t = C_t \setminus P_t \tag{6.4}$$

For a given collection, each parent is classified as either original parent, old parent, or child and parent. It is easy to verify that $P_t = \mathrm{OrP}_t \cup \mathrm{OlP}_t \cup \mathrm{CnP}_t$. Equivalently, each children is classified as either child and parent or sterile child ($C_t = \mathrm{CnP}_t \cup \mathrm{StC}_t$). By definition, children and parents instances belong to both, the parents set and the children set.

Figure 6.1 illustrates a genealogical tree and its components. Every collection represented in this example has 10 instances. Continuous arrows represent parent/child relations and dashed arrows represent coexistent instances.

Notice that for the oldest collection of the data set, represented as $col.t_1$, it is not possible to classify a parent because there is no data about parents of instances in this collection. In this case we represent all the parents as original parents, but we know that a portion of them must be in a different class. Equivalently, for collection t_5, it is not possible to know which instances are children and parents or which ones are sterile children. These documents are represented in the figure with a question mark.

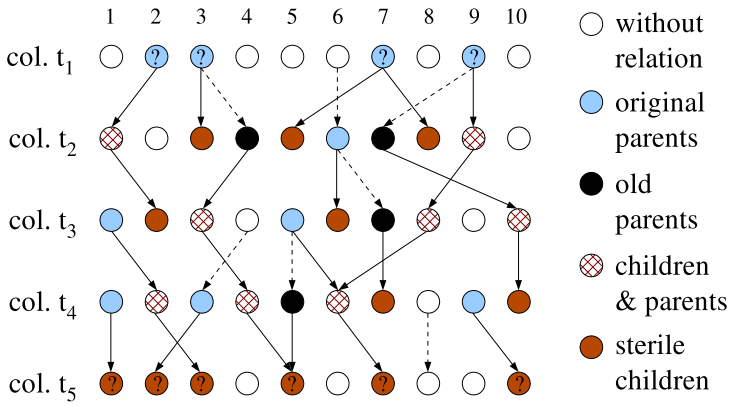


Figure 6.1. Example of the genealogical tree and its components.

6.2 Summary of the Algorithms

In this section we summarize the main algorithms designed and implemented for this work. Basically we present the algorithm to detect duplicates, the algorithm to find parent and child document candidates, the algorithms to filter the candidates in order to return parent and child instances, and the algorithm to select the parents, in order to associate only one parent for each child. Although separately the algorithms are not new and have no innovative aspects, their combination for the purpose of analyzing the Web content evolution is new and has successfully been employed.

6.2.1 Duplicate Detection

The algorithm to find duplicates works by clustering documents with the same content [Cho et al., 2000]. Initially, collection C (with n documents) is divided into m sub-collections S_i, $0 \leq i < m$. The algorithm runs in m steps. For each sub-collection S_i, $0 \leq i < m$, the text of the documents in S_i is first inserted into a hash table.

Next, the documents of C are searched in the hash table. For each new duplicate pair found, a new cluster is created and the duplicate pair is inserted into the new cluster. If one of the documents of the pair was previously inserted into a given cluster, then the other document of the pair is inserted into this cluster. At the end of each iteration i, the sub-collection S_i is excluded from C ($C = C - S_i$). At the end, the algorithm returns a set of clusters, with each cluster containing a list of duplicate documents.

6.2.2 Detecting Candidates

The algorithm to detect candidate parents and children is similar to the algorithm to detect duplicates, summarized in the previous section. The main differences are the number of collections involved and the representation of the document (now the shingles are used to represent the document, as described in Section 6.1.1). Instead of searching for documents of the same collection, the algorithm to find parents and children is applied to a pair of old–new Web collections.

The shingle paragraphs of the old collection are inserted into the hash table (in parts) and the shingle paragraphs of the new collection are searched. If a new document shares three or more shingle paragraphs with some document of the old collection, the old-new document pair is stored as candidate. At the end, for each old document, a list of child candidates is stored.

6.2.3 Finding Parent and Child Instances

After finding parent and child document candidates, two steps are now required: obtaining the list of parent and child instance candidates, and filtering the parent and child instances from the candidates.

Figure 6.2 summarizes the algorithm to obtain parent and child instance candidates. Along the first loop the old documents are instantiated, and along the second loop new documents found as child instance candidates, are instantiated. With this second loop, the list of child candidate documents for each old instance is used to generate the list of child instance candidates for each old instance.

Figure 6.3 summarizes the algorithm to filter candidate instances and find parents and children for a collection pair. The algorithm works by labeling old and new found instances as parent-child instances or as coexistent instances. If both documents of a parent-child candidate pair are labeled as coexistent, this pair cannot be a parent-child, although other child candidate in the list of the parent candidate can become a real child. In this case, the old instance is labeled as parent and coexistent, meaning that the parent exists in the new collection but a new document was generated with its content in the mean time.

The algorithm presented in Figure 6.3 works for unrestricted relations. For inter-site relations, preliminary algorithms are executed to identify the site to which a document belongs and mirrors of each site (see Section 6.1.3 for further details).

```
For each old document ODi
    If ODi is unique
        Create an old instance OIk;
        Keep the list of child candidates Cj of ODi to OIk;
    Else
        If it is the first occurrence of the ODi cluster
            Create an old instance OIk associated to the ODi cluster;
            Keep the list of child candidates Cj of ODi to OIk;
For each old instance OIk
    For each child candidate Cj in the list of OIk
        If Cj is unique
            Make Cj be a child candidate instance CIn;
            Include CIn in the list of child instance candidates for OIk;
        Else
            If it is the first occurrence of Cj cluster as candidate for OIk
                Make the Cj cluster be a child candidate instance CIn;
                Include CIn in the list of child instance candidate for OIk;
```

Figure 6.2. Algorithm to obtain parent and child instance candidates in a collection pair.

```
For each old instance OIk
    For each child candidate instance CIn
        For each ODi ∈ OIk
            For each Cj ∈ CIn
                If URL(ODi) = URL(Cj)
                    Label OIk and CIn as coexistent;
        If CIn is not a coexistent
            If OIk and CIn share at least 20% (threshold) paragraph
                Label OIk as parent and CIn as child, associating them;
For each old instance OIk
    Classify it as either coexistent, parent, parent and coexistent, or sterile;
For each new instance
    Classify it as either coexistent, child or orphan;
```

Figure 6.3. Algorithm to filter candidates to find instances of parents and children.

6.2.4 Selecting Parents

The output of the algorithm presented in Figure 6.3 can be used to classify each document into a different component of the Web genealogical tree. For our specific study, we follow processing the data in order to associate only one parent for each child. This association is

required because every near-duplicate instance in the old collection is considered a parent when one of the near-duplicates has a child. We ran preliminary experiments and detected a high number of parents. They expressively introduced noise to the results, impeding the correct classification of parents.

If we detected near-duplicates instead of duplicates (see Section 6.2.1), we could have inaccurate results. First, because, clusters of near-duplicates are intrinsically not accurate. Suppose page A shares 70% of its content with page B, which in turn shares 70% with C. It is possible that A and C share only 40% of their content, making the decision of which documents to cluster together a hard task. Second, because we study the evolution of content reuse in small parts of documents. The minimal similarity allowed is 20%, which is too low to perform clustering of near-duplicates.

For each child instance, we select a parent instance from its list of parents. The parent that shares the highest number of paragraphs with the child is selected. When the number of paragraphs is the same for more than one parent (this situation is not frequent), it does not matter which parent is chosen. In this case we select the parent with the lowest identifier. This heuristic is used in order to select the same document in case this situation occurs again for another child.

After associating a parent for each child, we separate intra-site and inter-site relations, and apply the mirror filter for inter-site relations.

6.3 Implementation Using WIM

We have also implemented a WIM program in parallel to the ad-hoc programs that implement the algorithms previously presented in this section. The WIM implementation is slightly different from the ad-hoc implementation. Nevertheless, the WIM solution addresses the same sub-tasks the ad-hoc solution presented in Section 6.2 does, which are: to identify duplicates within a snapshot (Section 6.2.1), in order to avoid the association of multiple parents to a child; to compare parts of every document of the old snapshot against every document of the new snapshot (Section 6.2.2); to filter old-new pairs with the same URL (Section 6.2.3), which means that the documents are the same in both snapshots; and to select one parent when a child has various parents (Section 6.2.4), which may occur for near-duplicate parents.

Figure 6.4 presents the WIM program for the Web evolution study application, which is explained below. An illustration of relations and operations used in this program is

```
// Clustering duplicates for both old and new collections:
relDupOld = Compare(relOld, sparse, exactmatch, text.V);
relClOld = Disconnect(relOld, connected, relDupOld.V);
relDupNew = Compare(relNew, sparse, exactmatch, text.V);
relClNew = Disconnect(relNew, connected, relDupNew.V);
// Comparing the collections:
relSearch = Search(relOld, relNew, shingles, 20%, relClOld.V, relClNew.V);
// Eliminating children with the same URL of parents:
relSearchUrl = CompGraph(relSearch.relSearch, exactMatch,
            relUrlOld.V, relUrlNew.V);
relSeDifUrl = Select(relSearch, value, relSearchUrl.V, ==, 0);
// Translating start and end nodes into instance nodes:
relStart = Set(relOld, relSearch, intersection, relClOld.K, relSeDifUrl.S);
relStartInst = Aggregate(relOld, grouping, count, relStart.V);
relEnd = Set(relNew, relSearch, intersection, relClNew.K, relSeDifUrl.E);
relEndInst = Aggregate(relNew, grouping, count, relEnd.V);
// Merging instance nodes with the similarity graph:
relGenEnd = Set(relSearch, relNew, intersection, relSeDifUrl.E,
            relEndInst.K);
relGenSt = Set(relSearch, relOld, intersection, relGenEnd.S, relStartInst.K);
// Selecting only one parent per child:
relGenFinal = Aggregate(relSearch, grouping, count, relGenSt.E);
```

Figure 6.4. WIM program to study the textual evolution of the Web.

presented in Figure 6.5. Observe that most of the operations and relations used in this and next WIM program illustrations were explained in Chapter 4 when exemplifying the use of the operators of the WIM algebra. For this reason we explain below the WIM program with focus on semantic aspects.

The program considers two relation sets as input: an older Web collection, *relOld*, and a more recent collection, *relNew*. Relation sets have their names in bold in the figure. The illustrations do not show the relation sets to which the output relations belong. Initially the Compare operator is used to identify duplicates in documents within each collection. As a graph is returned, the Disconnect operator is used to associate a cluster identifier to each node of the graph. The output, *relClOld*, goes to relation set *relOld* for the older dataset. Note that the illustration projects relations *text* and *url* besides relation *relClOld*, just because in underlying operations, these relations from *relOld* will be required, but only for the keys present in *relClOld*. Figure 6.5 does not show the steps to generate relation *relClNew*, because they are the same as to generate *relClOld*.

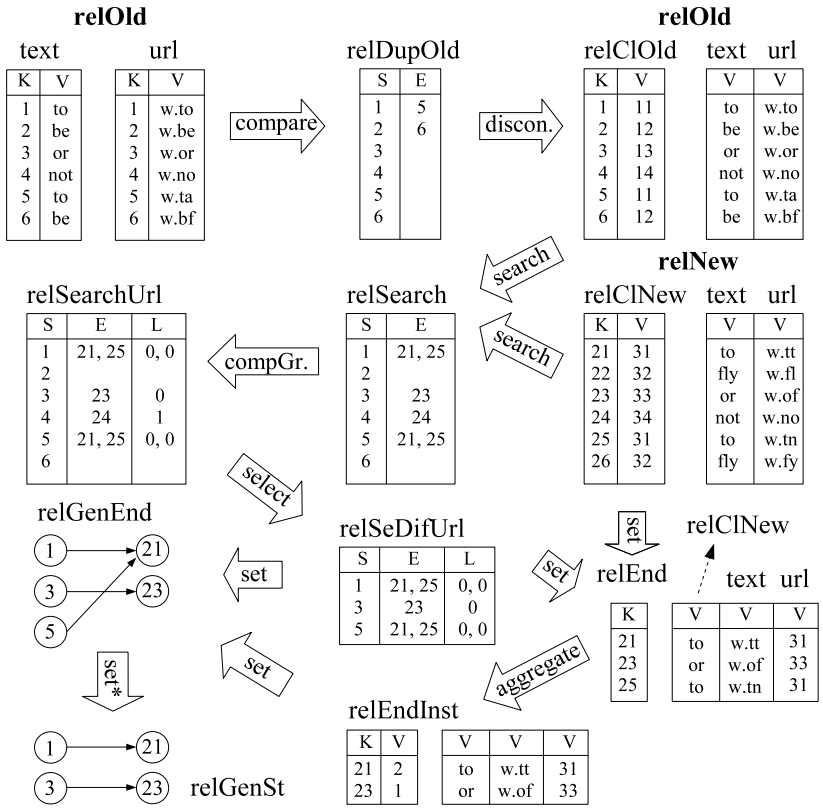


Figure 6.5. Illustration of relations and operations for the WIM program to study the Web content reuse and evolution.

Next, the Search operator is used to compare documents between collections *relClOld* and *relClNew*. The result is *relSearch*, a link relation whose start nodes are compatible to relation *relClOld* and end nodes are compatible to relation *relClNew*. The next step is to filter out pairs with the same URL, which are not parent-child pairs. The CompGraph operator is used to compare the URL of each linked pair. For the example in Figure 6.5, only pair 4–24 has the same URL. Links whose nodes have different URLs are then selected.

At the right side of Figure 6.5, the Set operator is used to return the intersection of elements in *relClNew* and end nodes in *relSeDifUrl*. Together with the Aggregate operator, which is applied to relation *relClNew* of relation set *relEnd*, the two operations are used to identify instances of documents, i.e., to identify duplicates within a dataset. The same steps exist to identify duplicates in the start node of relation *relSeDifUrl*, returning relation *relStartInst*, which is omitted in Figure 6.5.

Then the filtered nodes that remained in relation *relEndInst* are used to effectively filter the graph *relSeDifUrl*, first for the end nodes and later for the start nodes, which are

represented in Figure 6.5 with an asterisk (*) for the set operation, because the other input relation of the Set operator is *relStartInst*, not represented in the figure. The last operation, which is also not represented, is the aggregation of end nodes, in order to associate only one parent to each child, eliminating near-duplicates in the parents set.

6.3.1 Comparison with an Ad-hoc Solution

Before implementing a solution using WIM for this problem, we had implemented an ad-hoc solution in C, which was the natural way since WIM did not exist. The ad-hoc algorithms are presented earlier in this chapter. For our implementation, all the programs have together approximately 2,500 lines of code, for the same functionality, and took 1.5 month to be implemented by an advanced programmer.

In contrast, the WIM program presented in Figure 6.4, which took one day to be written, is shorter than the pseudo-code of the ad-hoc solution presented in this section. The program ran for the same datasets that the ad-hoc solution had run, and very similar results were found (the slight difference is due to minor differences in internal functions of the two solutions).

Regarding efficiency, we cannot compare the WIM program with the ad-hoc program because the code of the ad-hoc program is not optimized, and in fact takes much longer than the WIM program. Our comparison supposes that the ad-hoc program must have at least all the comparison steps of the WIM program in Figure 6.4, which are operators Compare (applied to both old and new datasets) and Search.

Note that WIM programs are multi-threaded, as the operators follow a dataflow approach. Hence we can readily process the output as it is generated, which exploits the parallelism of multi-core processors. This functionality allowed the three comparison operators to run in parallel. Then we also supposed that the ad-hoc implementation would run the comparison steps in parallel. The result was that the whole WIM program took 9.02 hours to run[1], from which only 6.7 minutes were spent with the other operators that are not comparison operations.

[1] The datasets used in the experiments are presented in the next section. The values are the average of runs for different datasets.

6.4 Data Set

For the experiments we have used five collections of pages of the Chilean Web, crawled in five distinct periods of time, from July 2002 to February 2006. Table 6.1 presents the main characteristics of the collections. The HTML tags were excluded from the documents, thus the metadata in the table represents data on the text found in the documents in each collection.

Table 6.1. Characteristic of the collections.

Col. name	Crawling date	total number of docs. (mi)	Size (Gbytes)
2002	Jul 2002	1.04	2.3
2003	Aug 2003	3.11	9.4
2004	Jan 2004	3.13	11.2
2005	Feb 2005	3.14	11.3
2006	Feb 2006	3.72	14.5

Each collection was crawled by the Chilean search engine TodoCL[2]. In order to compose the collections, the complete list of the Chilean Web primary domains were used to start the crawling, guaranteeing that a set of pages under every Chilean domain (`.cl`) was crawled, once the crawls were pruned by depth. Domains outside the Chilean primary domain were only crawled if their IP address was from a Chilean IP provider. The collections have successfully been used for other researches in characterizing the Web [Baeza-Yates and Poblete, 2006].

Any Web collection is a biased and partial image of the Web [Bennouas and Montgolfier, 2007]. We decided to use the Chilean collections because the way in which they were crawled indicates that they are the least biased data set for the kind of study we have done. As far as we know, Chile is the only country where a series of annual snapshots have been collected, using as seed the complete list of Chilean domains.

6.5 Genealogical Tree for the Chilean Web

In this section we present our study of the Chilean Web genealogical tree. Most of the results are presented as percentages in relation to the number of instances. The number of

[2] TodoCL: www.todocl.cl

instances for collections 2002, 2003, 2004, 2005 and 2006 is 416,300, 1,262,900, 1,157,100, 1,396,200, and 1,808,500, respectively.

6.5.1 Coexistent Instances

Figure 6.6 presents the percentage of **coexistent** instances among each collection pair, in relation to the old collection (first bar of each pair) and in relation to the new collection (second bar of each pair).

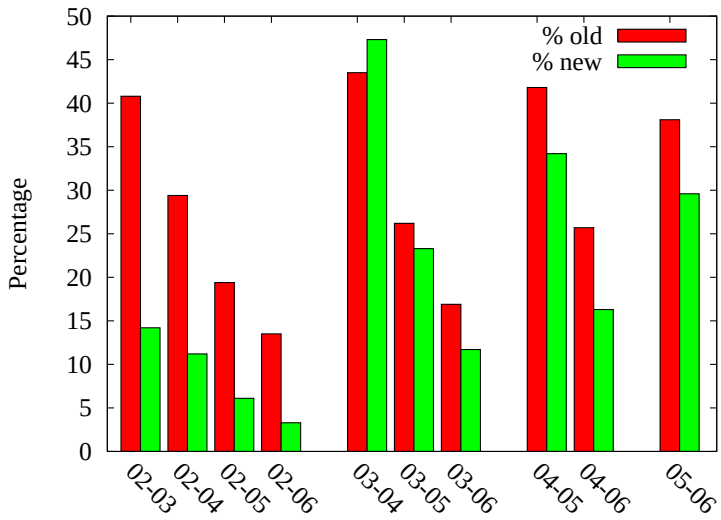


Figure 6.6. Percentage of coexistent instances among collection pairs.

For instance, around 41% of the instances in the old collection 2002 continue to exist in collection 2003. These instances represent around 14% of the new collection, 2003. The percentage changes due to the difference of the number of instances in each collection. Notice that the percentage of coexistent instances among the years (fixing the old collection) decreases linearly according to the time. The exception is collection pair 2003-2004, in which the difference of time from collection 2003 to 2004 is short, resulting in a large number of coexistent instances.

6.5.2 Parents and Children

In this section we study the number of parents and the number of children for collection pairs. Table 6.2 presents the number of parent instances, child instances and child documents, for both intra-site and inter-site relations. The number of child documents is

calculated by counting the number of documents represented by a child instance, that is, the number of documents in a cluster of an instance.

Table 6.2. Number of parent instances, child instances, and child documents, for intra-site and inter-site relations, for each collection pair (in thousands).

	Intra-site			Inter-site		
col.	par.	child	ch. doc.	par.	child	ch. doc.
02-03	13.7	23.7	41.5	12.7	27.0	63.8
02-04	10.1	12.1	18.6	11.4	39.2	67.8
02-05	10.3	12.4	21.3	10.1	32.1	44.5
02-06	8.6	10.5	14.2	9.5	20.0	38.6
03-04	21.3	41.3	69.5	19.0	70.9	115.1
03-05	29.9	39.9	54.8	21.7	65.1	99.5
03-06	26.5	31.7	40.8	22.8	60.8	126.8
04-05	34.6	43.6	62.4	20.1	51.9	83.5
04-06	29.8	34.0	46.5	28.2	64.5	132.6
05-06	27.7	40.3	58.2	23.7	64.0	144.2

For instance, 12,700 inter-site parent instances in collection 2002 generated 27,000 instances is collection 2003. These 27,000 instances represent a total of 63,800 documents in collection 2003. Data show that, on average, the number of intra-site parents is higher than the number of inter-site parents. On the other hand, the number of intra-site children is much lower than the number of inter-site children. On average, an intra-site parent generates 1.37 child instances and 2.07 child documents, whereas an inter-site parent generates 2.78 child instances and 5.05 child documents.

Comparing the coexistent data presented in Section 6.5.1 and relation data presented in this section, we see that the percentage of coexistent instances decreases according to the time more than the number of parents and children decreases. For example, from collection pair 2002-2003 to collection pair 2002-2006, the number of coexistent instances in relation to the old collection decreases 70%, whereas the number of parents decreases 25% and the number of children decreases 26%, for inter-site relations. Furthermore, the number of inter-site children increases in some cases. For instance, collection pair 2002-2003 has 27,000 children, whereas collection pair 2002-2004 has 39,200 children.

The values presented above indicates that from 2002 to 2006 (and also from 2003 to 2005), many pages died and could not generate a child, but the children of part of those dead pages became parents, generating new children and propagating the content. In this

case, for example in collection 2002, part of the parents of documents in collection 2004 or 2005 are in fact grandparents. In next sections we present further discussions about this behavior.

Figure 6.7 plots the percentage of parent instances, relative to the number of instances of the old collection; the percentage of child instances, relative to the number of new instances for the new collection; and the percentage of child documents, relative to the number of new documents for the new collection. Both intra-site and inter-site relations are presented. We focus our study on the adjacent collection pairs, that is, 2002-2003, 2003-2004, 2004-2005 and 2005-2006. The percentage of children is lower than the percentage of parents only for collection pair 2002-2003, because collection 2002 is considerably smaller than collection 2003.

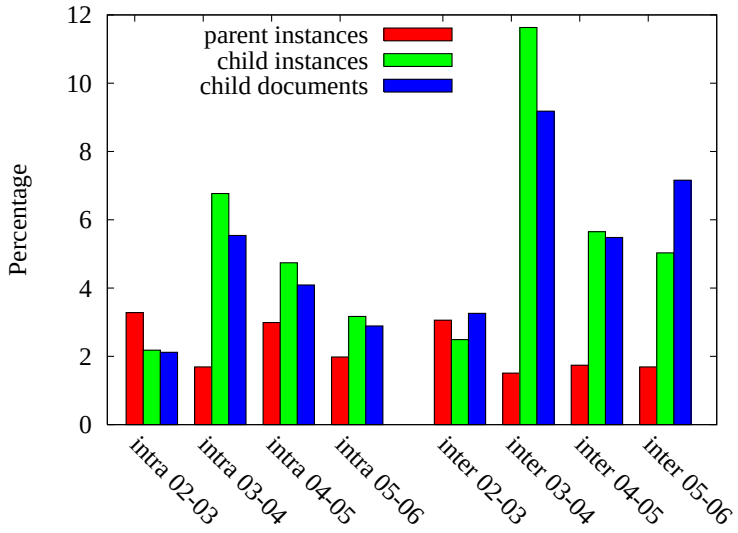


Figure 6.7. Percentage of parent instances, child instances, and child documents, for intra-site and inter-site relations.

Collection pair 2003-2004 presents the highest number of children for both intra-site and inter-site relations. This collection pair represents the shortest elapsed time between the crawling periods, as shown in Table 6.1, suggesting that relations are easier to be identified in shorter intervals (the difference of one year among the collections crawls may be enough for part of the children to die).

It is important to notice that the percentage of inter-site children is considerably higher than the percentage of intra-site children, and that the sum of both percentages represent the total percentage of children. On average, 4.5% of old instances are parents, 10.4% of new instances are children, and 9.9% of new documents are children.

6.5.3 Linkage Among Relatives

In this section we study how children acknowledge their parents with links to them. The collections have no external links, so that we are not able to study links to parents that no longer exist in a collection.

Table 6.3 presents the number of parents divided by the number of parents acknowledged by a child, for both intra-site and inter-site parents. For instance, one in each 8.4 intra-site parents in collection pair 2002-2003 are acknowledged by a child. The lower the value, the higher the total number of links from children to parents for a collection pair.

Table 6.3. Number of parents divided by the number of parents acknowledged by a child.

Col. pair	intra-site	inter-site
02-03	8.4	35.3
02-04	8.8	39.8
02-05	7.1	66.6
02-06	14.1	257.3
03-04	9.5	147.8
03-05	8.0	142.4
03-06	27.8	366.2
04-05	17.0	119.2
04-06	50.5	479.7
05-06	15.1	209.3
average	16.6	186.4

Intra-site parents are much more acknowledged than inter-site parents. This is simple to understand, as internal links in a site are usually common. Furthermore, the number of acknowledged inter-site parents is also relatively high: one link for every 35.3 inter-site parents, as observed for collection pair 2002-2003, is a significant value, given that the probability of a parent being acknowledged by a random document is extremely low.

6.5.4 Chilean Web Genealogical Tree

In this section we present the components of the Web genealogical tree as defined in Section 6.1.4, for the Chilean Web data. Table 6.4 presents the percentage of parents for both intra-site and inter-site relations, considering only the intermediate collections, in which the genealogical tree components can be studied. The second column presents again, for comparison purposes, the number of parents. The following columns present,

respectively, the percentage of original parents, old parents, and children and parents (see the definitions in Section 6.1.4).

Table 6.4. Percentage of parents for each component.

Collection	# parents (thousands)	original parents	old parents	child. & par.
intra 2003	21.3	96.5%	0.7%	2.8%
intra 2004	34.6	87.1%	2.1%	10.8%
intra 2005	27.7	87.5%	2.3%	10.3%
inter 2003	19.0	90.0%	2.1%	7.9%
inter 2004	20.1	81.1%	7.9%	11.0%
inter 2005	23.7	88.1%	3.6%	8.3%

According to the genealogical tree definition, if a given document exists in a time t_0 and generates a child in a time t_1, if the document is not crawled in t_1 but is crawled in a time t_2 (snapshot t_1 is skipped), it would wrongly be associated as a child of its own child (which would be considered a child and parent). For our data set, only 1.5% of the documents are skipped in a crawling, on average. In any case, we verified that they had negligible influence in our results, less than 5 documents wrongly appeared as children and parents.

Observing the 2003 collection in Table 6.4, we see that the percentage of original parents for this collection is the highest one among the three collections. This scenario is probably due to the small size of collection 2002. In this case many documents in collection 2003 should probably be children of documents in 2002, but their parents are not represented in collection 2002.

Data presented in the table demonstrate mainly two important issues. First, the percentage of children and parents and the percentage of old parents are higher for inter-site relations. Second, the percentage of children and parents is higher than the percentage of old parents. In order to understand these issues, Table 6.5 presents the probability of an instance becoming a parent in each component. The second column of the table presents the number of coexistent previous parents (referred as EPP), *i.e.*, the intersection between the parents set in snapshot $t-1$ and the coexistent instances set in snapshot t ($\text{EPP}_t = P_{t-1} \cap E_t$, where E_t represents the coexistent instances between snapshots $t-1$ and t)

The number of coexistent previous parents, which is presented in thousands, is used to calculate the probability of a coexistent previous parent becoming a parent, presented in

Table 6.5. Probability of an instance becoming a parent, for each parent component.

Collection	EPP (k)	P(OlP)	P(CnP)	P(OrP)
intra 2003	5.4	0.027	0.024	0.016
intra 2004	15.4	0.046	0.088	0.027
intra 2005	28.3	0.022	0.064	0.018
inter 2003	5.2	0.078	0.056	0.014
inter 2004	16.4	0.097	0.031	0.015
inter 2005	23.2	0.036	0.037	0.016

the third column of Table 6.5, where $P(\text{OlP}_t) = |\text{OlP}_t|/|\text{EPP}_t|$ (see Section 6.1.4 for details about the variables). Given that an instance is coexistent and was a parent in the previous collection, the values in this column represent the probability of this instance becoming a parent.

The fourth column presents the probability of a child becoming a parent, given by $P(\text{CnP}_t) = |\text{CnP}_t|/|C_t|$. Note that data in this column represents the percentage of parent and child instances in relation to the number of sterile children. For instance, for inter-site relations in collection 2003, 5.6% of the children are classified as children and parents, whereas 94.4% are sterile children.

The fifth column presents the probability of an orphan instance becoming a parent (see Section 6.1.3 for the definitions), given by $P(\text{OrP}_t) = |\text{OrP}_t|/(|\text{INS}_t| - (|\text{EPP}_t| + |C_t|))$, where INS_t is the set of instances for snapshot t.

Table 6.5 shows that the probability of a child or a coexistent previous parent becoming a parent is higher than the probability of an orphan instance becoming a parent. This conclusion is true for both intra-site and inter-site parents, although for inter-site parents this probability is, on average, more than twice higher than for intra-site parents. In summary, an important conclusion is that instances with a previous relation (as either parent or child) are more likely to be parents than documents without relations. Thus, instances inside the genealogical tree are more fertile than other instances.

6.5.5 Beyond the Chilean Web

In this section we discuss how part of the results found for the Chilean Web can be generalized to the whole Web (or to other Web data sets). We are interested in estimating the number of children in a Chilean Web snapshot generated from parents outside Chile.

We have used a Web collection from Spain with 16.2 million pages, crawled in September 2004 [Baeza-Yates et al., 2007a]. We used the Spanish collection as the old collection and the Chilean 2005 collection as the new collection, and the same algorithms used for studying the Chilean Web.

We have found 11,800 new instances that are children from Spanish pages and from pages in the Chilean 2004 collection. These pages in the Spanish collection are either parents or children from the Chilean collection 2004. We have found 25,300 new instances in collection 2005 that are children only from Spanish pages. Thus, the total number of relations from Spanish pages is 37,100 instances. Collection 2005 has a total of 95,400 children from the Chilean collection 2004, considering intra-site plus inter-site relations. Comparing to the number of children from Spain, there are around two or three times more children from Chile than from Spain, in the Chilean 2005 collection.

In order to estimate the total number of children that may exist in the 2005 Chilean collection, we first estimate how big the Spanish and Chilean Webs are, in comparison to all the Webs in Spanish speaking countries. We use the number of unique host names, which is measured by the Internet Systems Consortium[3]. We see that the Spanish speaking countries have a total 15.6 million host names, whereas Spain and Chile have 3.0 million and 745,000, respectively, representing 19.6% and 4.6% of host names in Spanish speaking countries.

A simple estimation is to consider that the other Webs from Spanish speaking countries (the other 75.6%, according to the number of host names) tend to generate the same number of children as the Chilean Web. In this case, there would exist 143,000 more children in the 2005 Chilean collection, considering the overlap with the 2004 Chilean collection, or 97,200 more children, excluding children from the 2004 Chilean collection.

This simple estimation does not take into account that there are other sites in Spanish language outside Spanish speaking countries, and that the Chilean Web also has pages in other languages. For these reasons we guess our estimation is a lower bound. Thus, the 2005 collection would have at least 217,900 children (95,400 from the 2004 Chilean collection, 25,300 from the Spanish collection, and 97,200 estimated for other Webs), which represent 23.7% of *new* instances in the 2005 collection. This percentage may also be valid for other Web data sets, as the Chilean Web has similar characteristics in comparison to other Webs [Baeza-Yates and Poblete, 2006].

[3]Internet systems consortium's domain survey, October 2007, http://www.isc.org/ds/

6.6 Genealogy and Search Engines

In this section we start associating relations, specially parents, with metrics used by search engines to rank the results, and with the search engine results and click-through data. Our objective is to characterize the parents, which reflects the characterization of the user behavior when reusing content.

We carried out a series of experiments, presented in the following sections. In every case we compare data considering all the instances of a collection, considering only the intra-site parent instances, and considering only the inter-site parent instances (and sometimes the child instances too). Taking into account that intra-site relations are characterized by local reuse of the user's own Web site content, the metrics might present different results for intra-site and inter-site parents.

6.6.1 Genealogy and Pagerank

In this section we study the Pagerank [Page et al., 1998] relevance measure for parents, children and instances in general. For clustered instances we chose the document of that cluster with the highest Pagerank, due to the fact that this document is probably the parent of the other duplicates in its cluster and it would be chosen by the search engine to be returned if its content matches a query, eliminating duplicates in the answers. This heuristic for choosing the document to represent the cluster is also used for other experiments in the following sections.

Table 6.6 presents the average Pagerank for all the instances of the old collection, for parent and child intra-site and inter-site instances, for the adjacent collection pairs. The average for the collection pairs is also presented.

Table 6.6. Average Pagerank for old instances, parent instances and child instances. Values are multiplied by 10^5 for better visualization.

Collection pair	all	parents		children	
		intra	inter	intra	inter
02-03	0.082	0.070	0.080	0.022	0.034
03-04	0.029	0.027	0.052	0.022	0.048
04-05	0.032	0.027	0.081	0.021	0.020
05-06	0.033	0.038	0.042	0.021	0.019
average	0.044	0.040	0.064	0.021	0.030

The average Pagerank for child instances is very low, probably a consequence of the recent creation of the new instance. The average Pagerank for intra-site parents is very close to the average Pagerank for old instances (all instances). The average Pagerank for inter-site parents is quite high, on average 60% higher than for intra-site parents. This high difference indicates that parents are better connected on the Web graph than other documents, thus they are easier to be found than many other documents. In Section 6.6.3 we directly study the relationship between the search engine results and the parents.

Figure 6.8 presents the average Pagerank for the different components of the Web genealogical tree, that is, for original parents, old parents, children and parents, and sterile children. The first set of bars represents intra-site relations and the second set represents inter-site relations.

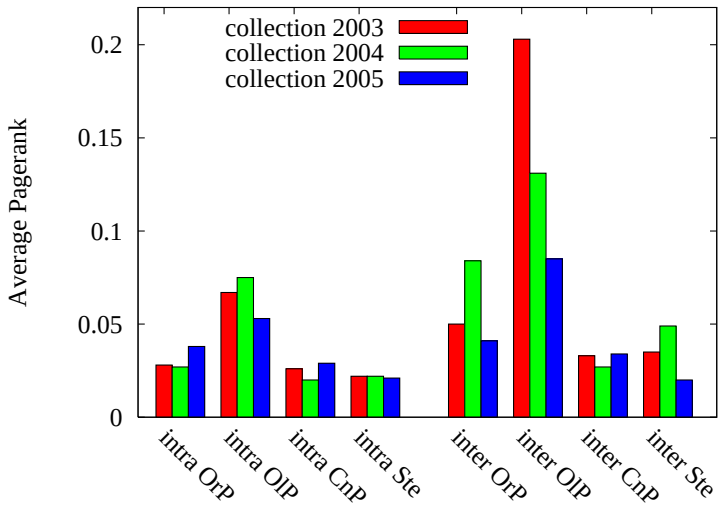


Figure 6.8. Average Pagerank for components of the Web genealogical tree. Values are also multiplied by 10^5.

The figure shows evidence that old parents have a higher Pagerank. On the other hand, sterile children have low Pagerank.

6.6.2 Genealogy and the Web Macro Structure

In this section we study how parents and children are connected in the Web graph macro structure [Broder et al., 2000]. We previously identified the Web macro structure component to which each document of a collection belongs, according to the Web macro structure

component to which its site belongs. This heuristic is reasonable, given that by reaching a site, the user can also reach every document in that site.

Considering the average for all five Chilean collections, tunnels, islands, the in, the out and the main macro structure components [Broder et al., 2000] have 3.3%, 8.8%, 9.8%, 18.8% and 59.4% of the documents, respectively. Due to the large volume of data to be presented, we present together the out and main components, which are characterized by their connectivity, given that they are reachable from more pages.

Table 6.7 presents the percentage of connected components (main and out) for the whole old collection, and for intra-site and inter-site parents and children. Intra-site children have high connectivity because the child belongs to the same site of the parent, so that if the parent has high connectivity the child will have too. Intra-site parents have also high connectivity. This behavior may be due to the high volume of modifications in sites with more resources and more pages. As expected, the percentage of connected components for inter-site parents is higher than for inter-site children.

Table 6.7. Percentage of the Web macro structure connected components (main and out) for relations.

Collection pair	all	parents		children	
		intra	inter	intra	inter
02-03	74.9	90.3	85.2	89.8	83.3
03-04	72.9	88.2	90.5	95.3	93.3
04-05	82.6	93.5	87.1	91.1	73.3
05-06	81.9	89.0	91.3	87.4	67.1
average	78.1	90.3	88.5	90.9	79.2

Figure 6.9 presents the percentage of the Web macro structure connected components for elements of the Web genealogical tree. We see that inter-site child and parent instances are as weakly connected as sterile child instances (with an outlier for the 2004 children and parents).

6.6.3 Genealogy and Query Results

In this part of the experiments we simulate a user performing queries in the past, and analyze click-through data. Initially, we observe whether the queries return the parents and how they are returned. The simulation is real because we used query logs and the same Web collection and query processor (we had access to the query processor as a black

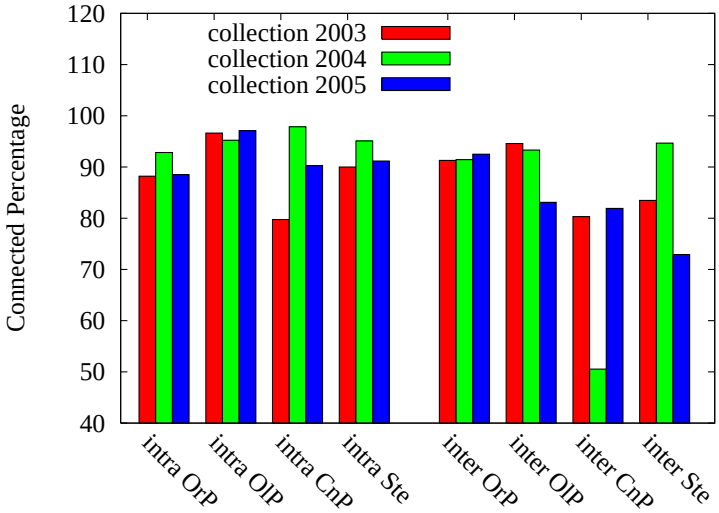


Figure 6.9. Percentage of the Web macro structure connected components (main and out) elements of the Web genealogical tree.

box) used by the search engine TodoCL. Given that this search engine was popular in Chile at that period, we try to associate queries (and clicks) to the parents, given that possibly part of the publishers of children used the TodoCL search engine (or other search engine whose ranking may not differ that much) in order to find content.

The query log we have available contains queries over a period of 10 months, from February to November 2004, and was applied to the collection 2004. The period of the logs starts one month after the collection 2004 was crawled and finishes two months before the collection 2005 was crawled (see Table 6.1). The one million most frequent queries were used and the top 5 results were considered. In this set, the most frequent of queries has 750,200 requests, whereas the least frequent query has 33 requests.

With the queries processed we used their results to compare how documents in general are returned, how intra-site parents and children are returned and how inter-site parents and children are returned. The children considered are for collection pair 2003-2004, that is, children in collection 2004. We perform the same study considering all the click-through data of the query log.

Figure 6.10 presents the average number of top documents returned per **occurrence** of queries (frequency of query is not considered), for documents in general (the first bar) and for components of the genealogical tree. On average a document is returned in 0.38 different queries. If the document is an inter-site original parent, on average the parent is returned in 0.87 different queries, which represents an increase of more than 120% in

relation to the average number of documents.

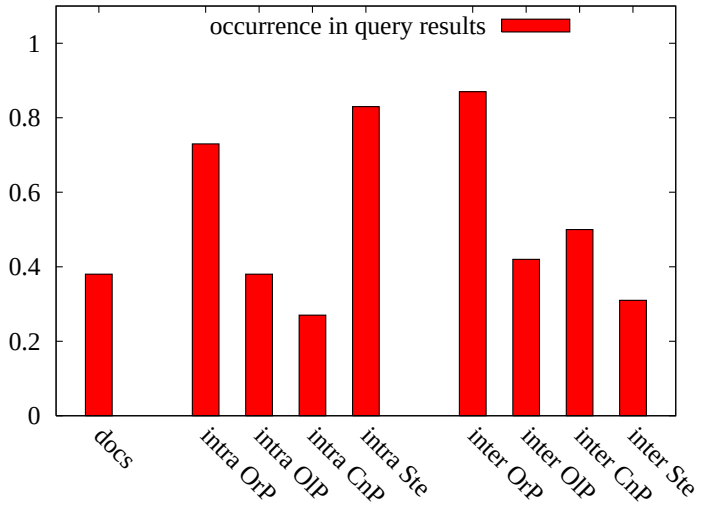


Figure 6.10. Average number of documents returned per occurrence of queries.

Figure 6.11 presents the average number of top documents returned for **all queries** (frequency is now considered), for documents in general and for components of the genealogical tree. For example, if a document d occurred in two queries A and B, submitted respectively 6 and 4 times, document d occurred for 10 requests in total. The intuition is that documents that appear more in results of queries are more likely to be copied.

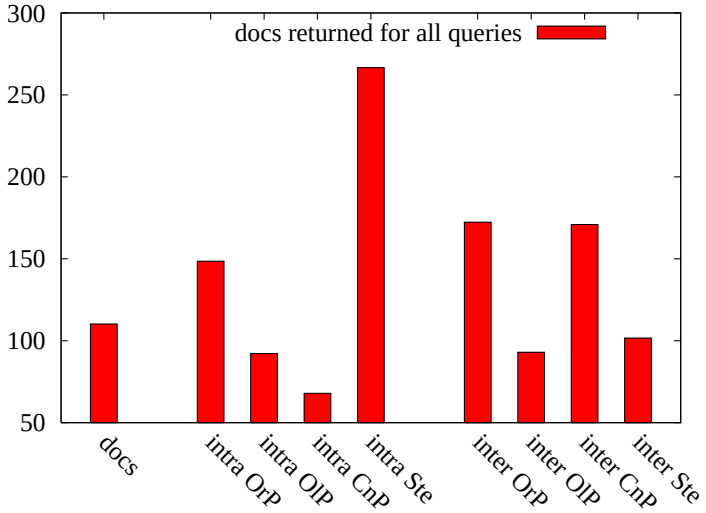


Figure 6.11. Average number of documents returned for all queries.

We see that intra-site sterile documents are returned in a high number of requests, which means that new documents with some old content from the same site may be relevant for a large set of requests. Comparing Figures 6.10 and 6.11, we see a similar behavior for each component. For instance, inter-site original parents and children and parents appear more frequently than old parents and sterile children.

Figure 6.12 presents the average number of **clicks** per document, for documents and for components of the genealogical tree. We see that intra-site original parents are frequently clicked, and that inter-site original and old parents are much more clicked than documents in general. Inter-site sterile documents have very low number of clicks.

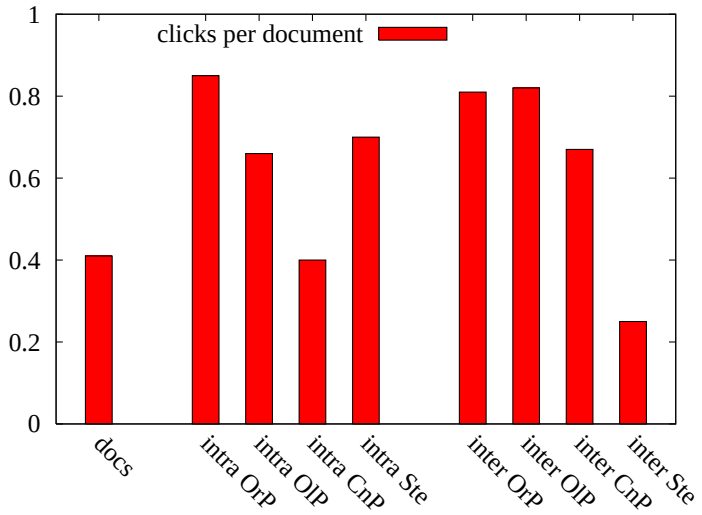


Figure 6.12. Average number of clicks per document.

For the three measures, in general we see that the values for inter-site parents are considerably higher than for documents in general, and also higher than for intra-site parents. These results represent evidence that part of the parents are associated to queries.

Characterizing the parents Considering this relation between inter-site parents and queries, we study the distribution of the frequency of parents in query results, with the goal of understanding whether the parents are the most often returned set of documents or not. Our intuition is that the most returned documents are not the most copied documents. The most returned documents have normally a high Pagerank and not too much text. Maybe they are a good source of links for copied documents, and we guess that documents

returned by queries and copied are returned by more specific queries rather than generic queries.

Figure 6.13 presents the distribution of documents according to the frequency with which they are requested in queries (the same measure used in Table 6.11), in a logarithmic scale. Figure 6.14 present an equivalent distribution, but only for inter-site parents.

Note that the axis range differs between Figure 6.13 and Figure 6.14. The frequency can be modeled as a power law. Note that every point plotted for parents is also represented as a point in the plot for parents, given that a parent is a subset of the document set, and the frequency of that document is obviously the same.

For instance, the last point at the bottom of Figure 6.13 means that one document was returned around 7 million times. The first point at the top means that a large number of documents were returned in only one query, which has the minimal frequency found in the log (33 requests).

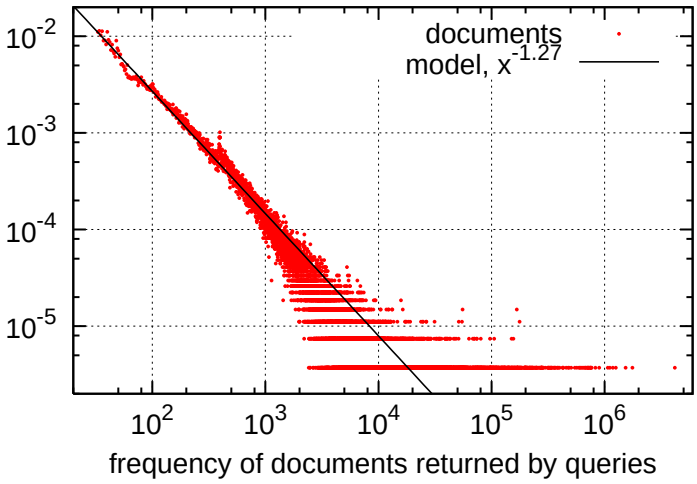


Figure 6.13. Distribution of documents in general returned by queries, according to their frequencies.

Comparing the general plot with the parents plot, we see that most of the parents in Figure 6.14 are represented with a low frequency in Figure 6.13. For example, the points in Figure 6.13 are concentrated between frequencies 1,000 and 10,000, while the points in Figure 6.14 are more concentrated between frequencies 100 and 1,000. Figure 6.13 has many points after frequency 100,000. This is not the case in the distribution in Figure 6.14. Not only the range is smaller for parents, but also the power law has absolute exponent smaller than in the general case, showing that they are less spread.

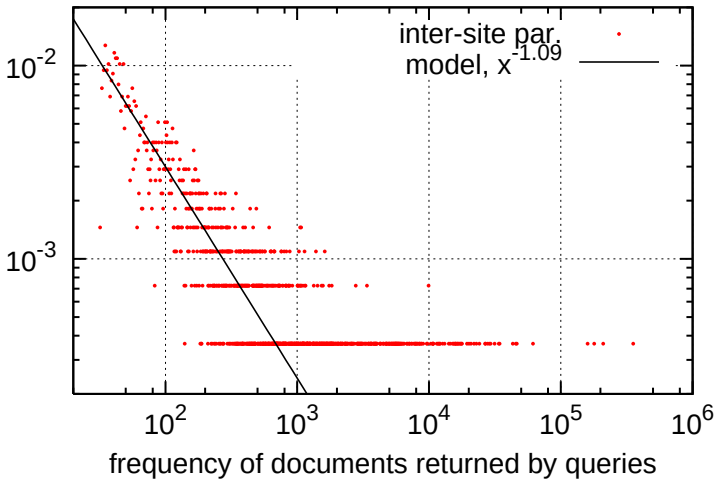


Figure 6.14. Distribution of inter-site parents returned by queries, according to their frequencies.

Figure 6.15 presents together, the distribution of clicks on documents in general, and the distribution of clicks on inter-site parents. The same conclusions stated for the distribution of documents returned by queries are valid for the distribution of clicks. Observe that parents are not the most clicked documents.

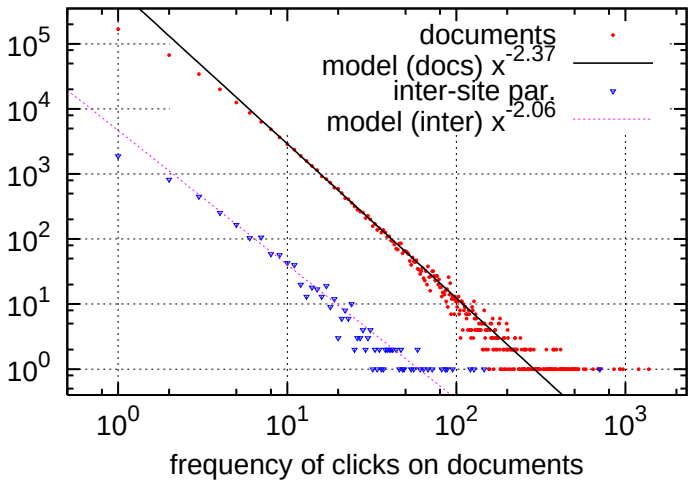


Figure 6.15. Distribution of clicks on documents in general and on inter-site parents.

The values presented in Figures 6.11 and 6.12 show that parents are returned in queries and they are clicked much more frequently than documents in general. At the same time, the plots show that the parents are not the most frequent documents returned

by queries or the most clicked documents. These facts reinforce our intuition that part of the queries are used with the intention to copy a content, and in these cases the documents are not so frequent, probably because the query is more specific.

6.7 Concluding Remarks

We have investigated the evolution of textual content on the Web. We have shown that a significant portion of the Web content has been evolving from old content. We have presented estimations to generalize our finds to other Web data sets. We estimated that 23.7% of the new Web documents that appear within the span of a year have content from previously published documents, which is a high percentage. We also verified that previously copied pages are more likely to become parents again.

We have introduced the concept of genealogical tree on the Web, and studied its components. Basically, we have observed that inter-site parents have high pagerank relative to other documents, are well connected to the Web graph, appear frequently as result of real queries and are clicked frequently after a search. These results indicate that search engine ranking algorithms bias part of the Web content.

This use case is a successful application of WIM to a real Web mining problem. We have compared the WIM solution with a possible ad-hoc solution which has only the mining tasks of the WIM solution, and demonstrated that the running time overhead introduced by WIM is insignificant, whereas the number of lines of code using WIM can be reduced by orders of magnitude.

Chapter 7

Other Use Cases

In this chapter we present four other use cases to which WIM has been applied. The first use case is the implementation of a "user browsing" pagerank, based on usage data. The second use case is a comparison of the Web linkage evolution, in terms of duplicated and new content pages. For this use case we present preliminary results, which indicate that duplicated-content pages pagerank evolves less than new-content pages pagerank. The third use case is an implementation for manipulating usage data, aiming user querying intent prediction. The fourth use case is an implementation to select documents from a Web dataset for composing a pool of documents for relevance assessment.

7.1 Studying a Usage Pagerank for Ranking Improvement

The second use case is a proposal of usage pagerank, i.e., a document relevance weight based on a click graph. According to the assumption that the clicks flow within a user session most of the time indicates that the recently clicked documents are more relevant for that query, we propose to compose a graph with the order of clicks within a section and study the pagerank for this usage graph. As an example of such graph, suppose that for a session the user clicked in page A, then in B and then in C. The graph is composed by a link from A to B and another link from B to C. A single graph is produced as a result of processing all sessions in this way.

Figure 7.1 presents the WIM program for the usage pagerank application. The objective of this program is to return new relations to represent properties of documents from

```
// Converting a node relation with session and click data into a link relation:
relUsGraph = Convert(clickSet, sessionId.V, clickedDoc.V);
// Associating urls of the usage log:
relUsDocs = Aggregate(clickSet, grouping, count, url.V);
// Associating the pagerank to clicked documents:
relDocsPr = Join(clickSet.relUsDocs, relLargeCol, url.V, url.V, pr.V, 0.0);
// Manipulating the usage graph to calculate the usage pagerank:
relFull = Relink(relUsGraph, cocitation, single, adjacent, relUsGraph.SE);
relCocit = Select(relFull, value, relFull.L, ! =, 0);
relAgg = Aggregate(relUsGraph, grouping, count, relCocit.SE);
relPruned = Select(relUsGraph, value, relAgg.V, >, 20);
relUsPR = Analyze(relUsGraph.relPruned, pagerank);
```

Figure 7.1. WIM program to study a usage Pagerank.

the query log. The new relation set must have the following value attributes: the URL of the document, the real pagerank (which is not calculated but joined in from another dataset), and the new usage pagerank. With this output dataset we can propose a re-ranking function to move documents in results of queries, based on the comparison of the real and the usage pagerank.

Figure 7.2 illustrates the relations and their attributes, and the operations used in the WIM program of Figure 7.1. Relation sets have their names in bold in the figure and in other illustration figures in this chapter. The illustrations do not show the relation sets to which the output relations belong. Next we explain the WIM program for the use case to study a usage Pagerank.

The program uses two datasets as input: a Yahoo! query log containing 22 million clicks, represented by relation set *clickSet* in Figures 7.1 and 7.2, and a Web dataset from United Kingdom, with 77 million entries [Boldi and Vigna, 2004], from where the pagerank data is taken, represented by relation set *relLargeCol*.

The program starts by converting click data to a graph, so that each session identifier in relation set *clickSet* becomes a start node in relation *relUsGraph*, and every clicked document identifier becomes an end node. The Aggregate operator is applied to relation *url* in clickSet, so that relation *relUsDocs* ('Us' stands for 'Usage') contains only distinct URLs, without replication. Note that the illustration projects relations *clickedDoc* and *url* besides relation *relUsDocs*, just because in underlying operations, these relations from *clickSet* will be required, but only for the keys present in *relUsDocs*. Other figures in this chapter also make use of this mechanism for enriching the illustration. In the next steps the

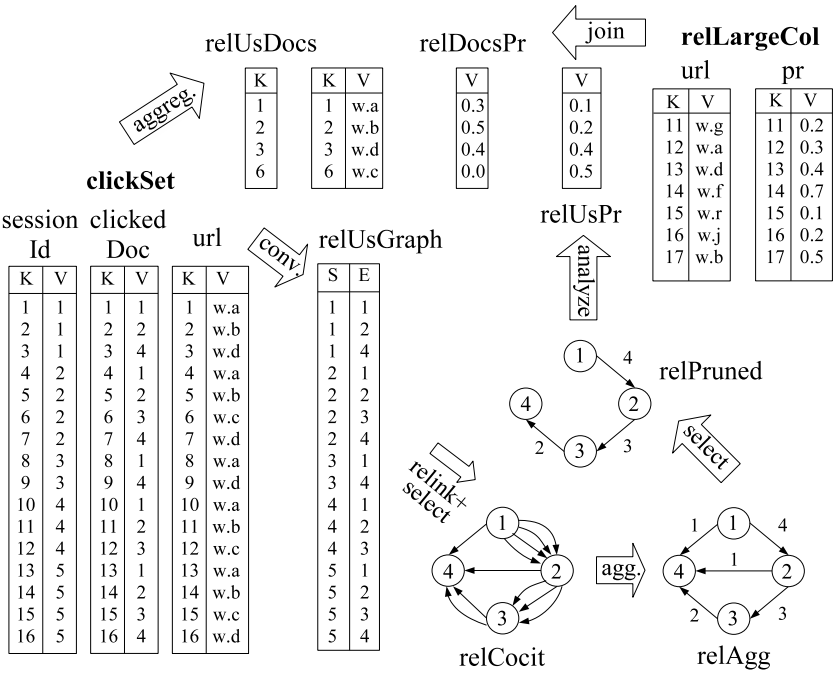


Figure 7.2. Illustration of relations and operations for the WIM program to study a usage pagerank.

pagerank and the usage pagerank will be associated with elements in relation *relUsDocs*.

The pagerank values for a subset of documents in *relUsDocs* are taken from *relLargeCol*. The Join operator is used to compare attributes representing URLs of both relation sets and associate a pagerank for the found URLs. In the example of Figure 7.2, URL 'w.c' is not found in *relLargeCol*, then a default value is associated to the corresponding entry in *relDocsPr*.

The graph *relUsGraph*, converted from relations *sessionId* and *clickedDoc* in set *clickSet*, is finally used to calculate the usage pagerank. The Relink operator is used to insert new links to *relUsGraph*, which is returned in relation *relFull*. Observe that the Relink options used allow the introduction of new links only between adjacent end nodes for each start node. By selecting only links with label 1, relation *relCocit*, which contains only the new links introduced by the Relink operator, is the usage graph to which we want to calculate the new usage pagerank.

Then the links are aggregated, so that links in relation *relAgg* are labeled with the number of clicks performed by every user from a given document to other. In this example we select links according to a given threshold, and the returned graph is used to calculate the pagerank for its nodes, returning *relUsPr*.

7.2 Comparing the Web Linkage Evolution of Duplicated and New-content Pages

The third use case is a comparative study of linkage evolution between new pages with new content and new pages with old content (duplicated content). The hypothesis is that the pagerank, or other link analysis measure, of duplicate or near-duplicate Web pages evolves less than the in-degree of Web pages with new content. The intuition behind this hypothesis is that duplicated pages are, most of the times, not returned by search engines, and consequently they are not found and not linked.

The result of this research may be important for Web crawler designers. The duplication problem is solved in order to avoid indexing duplicate content [Manku et al., 2007], but not to avoid fetching a page and discovering that it is a duplicate. These tasks consume time and space of search engines. On the other hand, the Web is very dynamic, and a large number of new pages are published every day. Due to resource limitations, search engines are not able to index all the new pages found every day, and many URLs are placed in the frontier, where the URL is known but the content has not been fetched yet [Eiron et al., 2004]. The frontier problem is to select the frontier pages which should be crawled and indexed. What search engines do in order to deal with this problem is choosing the frontier pages with a good connectivity, which can be represented by their pagerank or other page importance measure [Dasgupta et al., 2007; Bennouas and Montgolfier, 2007].

If the hypothesis above is proved, then the page importance is a good heuristic not only for the frontier problem, but also for avoiding crawling duplicates. Furthermore, it will be possible to study different page importance measures, in order to decrease the number of crawled duplicates and improve the availability of resources.

Figure 7.3 presents an algorithm to start studying the problem presented above. The algorithm has two stages: first the pagerank is calculated for time t_1, and second it is calculated for time t_2, comparing both duplicated and new-content pages.

Figure 7.4 presents the WIM program that implements the algorithm presented in Figure 7.3. Figure 7.5 presents an illustration of the relations and operations, and is used as an example during the explanation of the program below.

Initially the pagerank of pages in *sn1* is calculated, using the link relation set *sn1Link*, which is the compatible link relation set with respect to *sn1*. This operation is not illustrated in Figure 7.5. The Search operator is used to identify pages in *sn1PR* that previously existed in *sn0*. These pages are represented as end nodes of the *sn1URL* output link re-

Consider three datasets sn0, sn1 and sn2, which represent snapshots of a domain of the Web at three different times, t_0, t_1 and t_2;
First study the pagerank at a time t_1:
1 Recover the new pages sn1N from sn1, which are the pages whose URL did not exist in t_0 for sn0
2 Compare page content of sn1N against sn0, so as to obtain sn1NDup, that is the list of near-duplicates in sn1N regarding sn0;
3 Recover the list of documents in sn1N with new content, that is sn1NUni = sn1N − sn1NDup;
4 Calculate the pagerank (or other page connectivity measure) for pages in sn1NDup and pages in sn1NUni;
Now study the pagerank at a time t_2:
5 Recover sn2Dup, which are pages in sn2 whose URL was represented in sn1NDup;
6 Recover sn2Uni, which are pages in sn2 whose URL was represented in sn1NUni;
7 Calculate the pagerank of pages in sn2Dup and pages in sn2Uni;
8 Compare the pagerank for the different sets.

Figure 7.3. Algorithm to study the linkage evolution for duplicated and new-content pages.

lation. The Set operator returns the difference between the snapshot at time t_1 (*sn1PR*, containing Pagerank), and *sn1URL*. Relation *sn1New* represents the new documents, with new URLs, in collection *sn1*.

The next step is to identify pages in *sn1New* that are duplicates from some page in *sn0*, using shingles [Broder et al., 1997] as comparison method. The result is *sn1New2*, a link relation whose end node represents pages in *sn1New* that are duplicates of pages from *sn0*. The Aggregate operator is used to return *sn1New3*, from which duplicate entries at the end attribute are excluded. The intersection between *sn1New* and the end node of *sn1New3* is *avg1NDup*, a relation in set *sn1New* which represents only duplicated documents. The difference between these relations is *avg1NUni*, which represents new-content documents in *sn1*. The Aggregate operator is used to return a relation containing the average pagerank for both sets of pages, for time t_1.

Notice that in Figure 7.5 some operations are prefixed by a character '*', which means that another attribute is input but could not be shown in the figure, but are explicit in the WIM program in Figure 7.4. Actually some operators could not be represented in the figure (For instance the operator that generates relation *avg1NUni* is not represented).

```
// First study the pagerank at a time t1:
sn1PR = Analyze(sn1Link, pagerank);
sn1URL = Search(sn0, sn1, total, sn0.url, sn1PR.url);
sn1New = Set(sn1, sn1, difference, sn1PR.K, sn1URL.E);
sn1New2 = Search(sn0, sn1.sn1New, shingle, 100, text.V, text.V);
sn1New3 = Aggregate(sn1New2, grouping, count, sn1New2.E);
sn1NDup = Set(sn1, sn1New3, intersection, sn1New.K, sn1New3.E);
sn1NUni = Set(sn1, sn1New3, difference, sn1New.K, sn1New3.E);
avg1NDup = Aggregate(sn1, single, average, sn1NDup.V);
avg1NUni = Aggregate(sn1, single, average, sn1NUni.V);
// Now study the pagerank at a time t2:
sn2PR = Analyze(sn2Link, pagerank);
sn2DupURL = Search(sn1.sn1NDup, sn2.sn2PR, total, url.V, url.V);
sn2UniURL = Search(sn1.sn1NUni, sn2.sn2PR, total, url.V, url.V);
sn2Dup = Set(sn2, sn2DupURL, intersection, sn2PR.K, sn2DupURL.E);
sn2Uni = Set(sn2, sn2UniURL, intersection, sn2PR.K, sn2UniURL.E);
avg2Dup = Aggregate(sn2, single, average, sn2Dup.V);
avg2Uni = Aggregate(sn2, single, average, sn2Uni.V);
```

Figure 7.4. WIM Program to study the average pagerank evolution for duplicated and new-content pages.

At the second part, the pagerank of pages in collection *sn2* is calculated, generating *sn2PR*. This operation is not shown in Figure 7.5. Pages in *avg1NDup* and *avg1NUni*, from collection *sn1*, whose URLs remain existing in collection *sn2*, are returned, respectively in relations *sn2DupURL* and *sn2UniURL*. They are link relations whose end nodes represent pages in *sn2*. The Set operator is used to return documents from *sn2PR* that are, respectively, duplicated and new-content pages in collection *sn1*, that remain existing in *sn2*. Finally the average pagerank is calculated for duplicated and new-content pages, for time t_2.

We ran the WIM program presented in Figure 7.4 for a Chilean Web collection set, which is the same dataset used for the genealogical tree study presented in Chapter 6. The collections are very representative, given that the complete list of the Chilean Web primary domains were made available and used to start the crawling. Collection *sn0* was crawled in August 2003, collection *sn1* was crawled in January 2004, and collection *sn2* was crawled in February 2005.

Table 7.1 presents the average pagerank for duplicated and new-content pages, for the

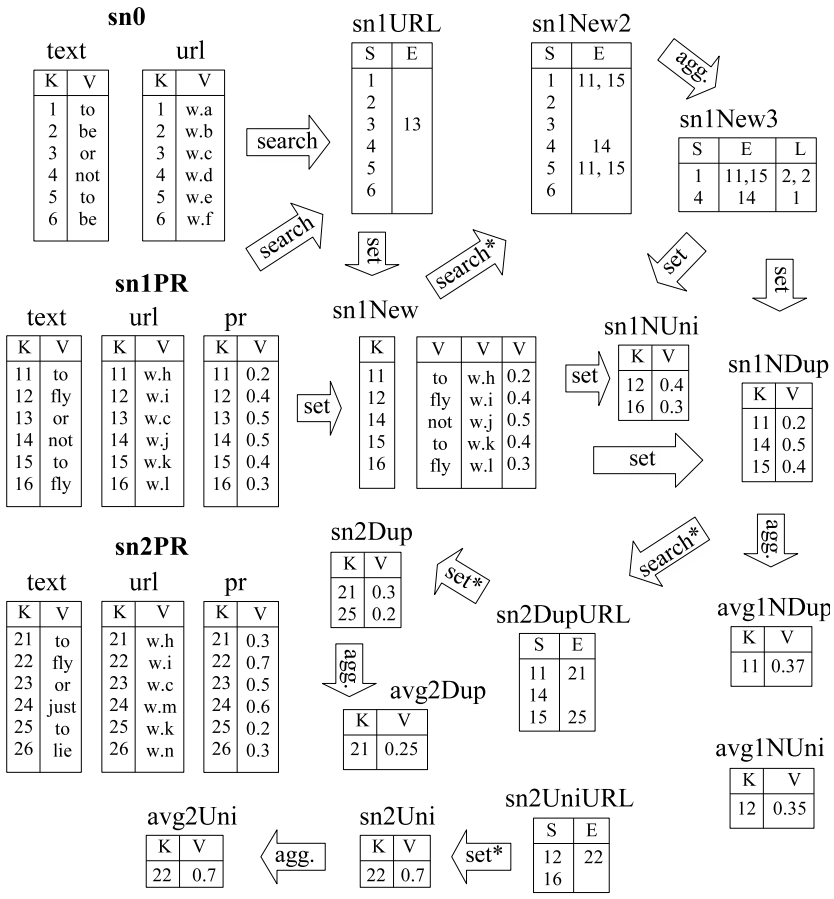


Figure 7.5. Illustration of relations and operations for the WIM program to study the average pagerank evolution of duplicated and new-content pages.

Chilean collections presented above. Note that the average pagerank for duplicated pages decreases as time goes on, whereas the average pagerank for new-content pages evolves positively. The results are evidence that the hypothesis presented at the beginning of this section is relevant.

7.3 Manipulating Search Engine Usage Data for Mining User Intent

Queries submitted to search engines define three main user intent categories, according to the taxonomy of Web search proposed by Broder in [Broder, 2002]. They are: i) navigational, in which users search for a specific reference, possibly already known to

Table 7.1. Average pagerank for duplicated and new-content pages, for the Chilean collections 2004 and 2005.

	75% similarity		100% similarity	
collections	dup.	new-cont.	dup.	new-cont.
sn1 (2004)	0.697	0.580	0.714	0.579
sn2 (2005)	0.641	0.740	0.646	0.733
difference (%)	-8.0	27.6	-9.5	26.6

him, and once he finds it, he goes to that place and abandons the query session; ii) informational, which users are looking for information about a certain topic and before finding a satisfactory result, they often visit a number of Web pages; and iii) transactional, which users want to do some interaction, such as downloading, purchasing, or making a bank transfer.

In general, research on user query profiling try to use statistics extracted from query logs to classify queries in one of the categories presented above [Lee et al., 2005; Rose and Levinson, 2004]. However, if we look carefully to some queries we note that it is not possible to put those queries into only one class. For example, if two different users search for "Michael Jackson", it is possible that one person looks for the Michael Jackson official home page, and the other person looks for information related to Michael Jackson, like news, blogs, images or fan club.

By classifying a query into only one category, the search engines are modeling the need of the majority of the users and those ones with intent deviating from the majority, likely, may not be satisfied. So, the point is that different users have different intents, even for the same query, and the existing models of query intent classification are not considering that.

We have implemented a WIM program to manipulate data from query logs, in order to identify a series of properties for each distinct query that occurs in the log. Taking into account these properties, we have defined functions to identify a large number of queries with both navigational and informational characteristics. Furthermore, we also propose to study queries with commercial intent, rather than queries with transactional intent, given that search systems cannot take advantage of identifying a transactional query in order to improve the ranking. On the other hand, studying commercial queries can lead to increase the profit of commercial search engines.

Figure 7.6 presents the WIM program to manipulate the query log, in order to return

```
// Materialize new relation with distinct queries and number of clicks:
queries = Aggregate(usageLog, grouping, count, query.V);
// Count the number of clicks on sponsored links:
spon = Select(usageLog, value, sponsor.V, ==, 1);
countSpon = Aggregate(usageLog.spon, grouping, count, sessionId.V);
numSpons = Aggregate(usageLog.countSpon, grouping, sum, queryId.V,
           countSpon.V);
numSponsJo = Join(queryData, usageLog.numSpons, queryId.V,
           queryId.V, numSpons.V, 0);
// Count the number of sessions:
click = Select(usageLog, value, sponsor.V, ==, 0);
ses = Aggregate(usageLog.click, grouping, count, queryId.V, sessionId.V);
numSes = Aggregate(usageLog.ses, grouping, count, queryId.V);
numSesJo = Join(queryData, usageLog.numSes, queryId.V, queryId.V,
           numSes.V, 0);
// Count the number of sessions with only one click:
countClick = Aggregate(usageLog.click, grouping, count, click.V);
one = Select(usageLog, value, countClick.V, ==, 1);
oneClick = Aggregate(usageLog.one, grouping, count, queryId.V);
oneClickJo = Join(queryData, usageLog.oneClick, queryId.V, queryId.V,
           oneClick.V, 0);
// Count the number of sessions with only one click on the most clicked:
most = Aggregate(usageLog.one, grouping, count, queryId.V, urlId.V);
mostOne = Aggregate(usageLog.most, grouping, max, queryId.V,
           most.V);
mostOneJo = Join(queryData, usageLog.mostOne, queryId.V,
           queryId.V, mostOne.V, 0);
```

Figure 7.6. WIM Program to manipulate usage data.

the properties to study the search engine user intent. For each distinct query, we need to calculate properties like: the number of sessions in which the query appears, the total number of clicks, the number of clicks on sponsored links, the number of sessions with only one click, the number of sessions with only one click on the most clicked document, the number of sessions with more than one click, and the standard deviation of the number of clicks on documents. We ran the WIM program of Figure 7.6 for a Yahoo! query log containing 22 million queries. Further evaluation of the results were not performed for this application.

We omitted the passage of the program that manipulates queries with sessions with

multiple clicks, just to avoid repetition of similar operations. Examples of relations and operations for this application are shown in Figure 7.7. Unlike the other illustrations presented in this chapter, Figure 7.7 shows relation sets like single relations with multiple attributes. Actually, the shown attributes are value attributes of relations in a relation set. This representation is used here to make the illustration compact, hence in a proper shape to be semantically explained.

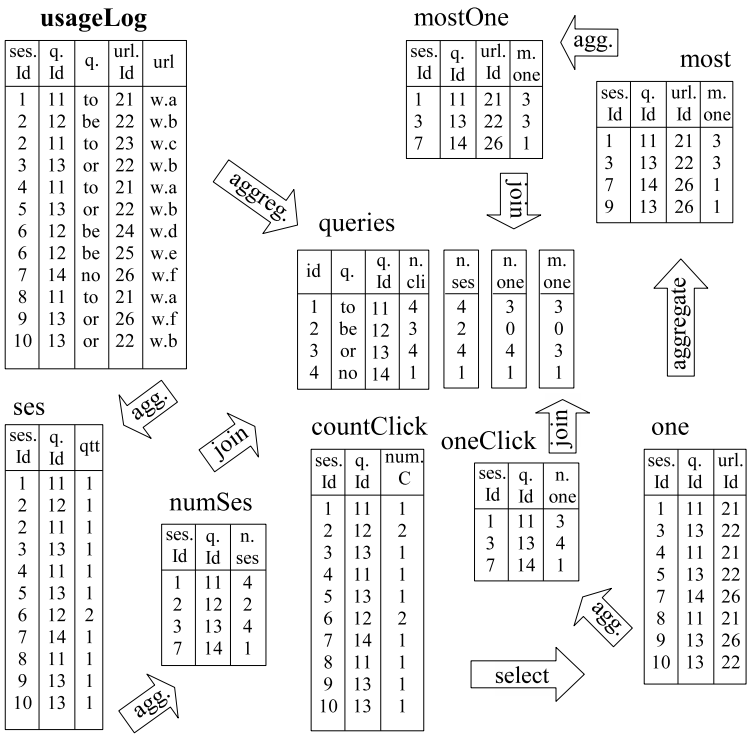


Figure 7.7. Illustration of relations and operations for the WIM program to manipulate usage data.

The program starts with the Aggregate operator, used to return the list of queries (relation *queries*) and the total number of clicks on each query. The next part is the sponsorship data, that is not represented in Figure 7.7. Relation *spon* contains only clicks on sponsored links. The Aggregate operator is first used to count the number of clicks on sponsors in each session, and later to sum the number of clicks in each session, for each query. The Join operator is required to associate each entry in relation *numSpons* to the correct entry in the new relation *queries*.

The next part counts the number of sessions for each query. In the beginning, only clicks on non-sponsored links are selected, returning relation *click*, which is used in other parts of this program. The Aggregate operator is used to count the number of clicks within

the same value in *sessionId* ('*ses.Id*' in Figure 7.7), for each query that occurs within a session. Observing relation *click* in Figure 7.7, we see that only for session id 6 the query is the same within the session. Then, in relation *ses*, only one entry is kept for session id 6. The Aggregate operator is used again to group the queries, and attribute *numSes* ('*n.ses*' in Figure 7.7), which contains the number of sessions in which each query appears, is joined to relation *queries* (relation set *usageLog*).

In the next part, the Aggregate operator is used to count the number of clicks for each session. Relation *countClick* would be used to process data for multiple clicks, in order to obtain data for calculating the informational characteristic of a query. In this example, *countClick* is used to count the number of clicks on sessions with only one click. Tuples representing sessions with only one click are selected and returned in relation *one*, and relation *oneClick* is returned with the number of sessions with only one click, for each query, which is later joined to relation *queryData* (relation set *usageLog*).

Relations with the same query id and URL id are grouped in relation *most*, which contains the number of clicks for each different pair of query id and URL id. Only the highest values for each query id are returned in relation *mostOne*, and are later joined into relation *queryData*.

7.4 Composing a Pool of Documents for Relevance Assessment

Our fifth WIM use case consists of simply composing a pool of documents returned by different retrieval methods, for a set of queries. As we intend to put our Chilean collections presented in Section 6.4 and usage log available for research, we also want to provide a relevance assessment, so that this data set may be useful in researches on learning to rank. The first part is to select the documents to be considered relevant, for each selected query, and this task can be done using WIM.

Figure 7.8 presents the WIM program for the task of selecting the documents. For illustration, in this example we present the TF-IDF method combined with pagerank and with filter "AND" in the terms of the queries, although the real program includes other combinations and also method BM-25. An illustration of the relations and operations is found in Figure 7.9 and is used in the explanation of the program below.

Initially the Search operator is used to retrieve documents for each query, according to the TF-IDF similarity measure, comparing the text of the query and the text of the

```
// Search using TFIDF, include a filter AND, and make the intersection:
tfidf = Search(queryList, dataSet, tfidf, 10,000, text.V, text.V);
relAnd = Search(queryList, dataSet, and, 10,000, text.V, text.V);
tfidf2 = Set(tfidf, tfidf, intersection, tfidf.SE, relAnd.SE);
// Now include pagerank as feature:
tfidfPR = Calculate(tfidf.tfidf2, multiplication, tfidf.L, relAnd.L);
// Eliminate duplicates in the results and select the top results
tfidfPR2 = Aggregate(tfidf.tfidfPR, grouping, max, clusId.V, tfidfPR.L);
tfidfPRFinal = Select(tfidf, top, decreasing, tfidfPR2.L, 2);
```

Figure 7.8. WIM Program to compose a pool of documents for relevance assessment.

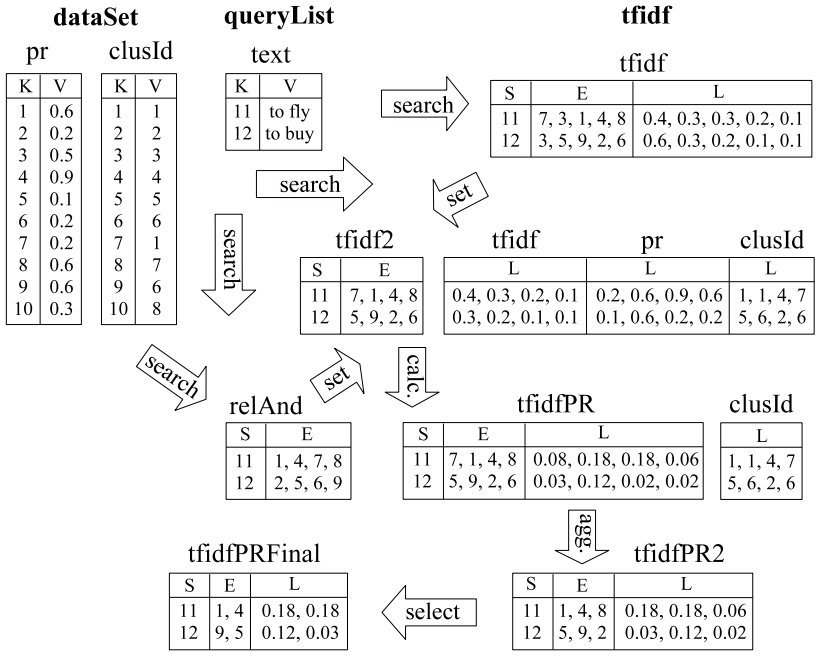


Figure 7.9. Partial illustration of relations and operations for the WIM program to compose a pool of documents returned by different search methods.

document. To be compact, Figure 7.9 does not present the relation *text* for relation set *dataSet*, and only five documents are returned for each query. Value 10,000 is the maximum number of documents to be returned per query. Relation *tfidf* is a link relation, where its label attribute stores the similarity between the query and the returned documents.

In parallel with the search using method TF-IDF, the search using method AND is returned as relation *relAnd*, which is then merged with relation *tfidf* through the Set operator and option Intersection, returning relation *tfidf2*. The intersection is important

to ensure that all the terms of the query appear in the result of the query using TF-IDF.

The Calculate operator is used to multiply the TF-IDF similarity in relation *tfidf* and the pagerank value in relation *relAnd*, for each document returned per query. The result is relation *tfidfPR*, to which the Aggregate operator is used to eliminate duplicates, based on the duplicate cluster number represented by relation *ClusId* in relation set *dataSet*. Note that end nodes 7 and 1 belong to the same cluster number 1. The one with lowest value is eliminated from the output relation *tfidfPR2*. Finally, the Select operator is used to return only the 30 (or the 2 in the illustration of Figure 7.9) top ranked documents.

7.5 Concluding Remarks

In this chapter we have presented other four use cases to which WIM has been applied. With the exception of the application for generating a pool for relevance assessment, which is applied to Web data but is not a mining task, the use cases are real examples of Web mining research problems solved using WIM.

WIM has demonstrated to be effective for our use cases, since just a few lines of code are needed to implement the solutions through WIM. We observe that the data manipulation operators are well designed to Web data, once they are able to shape the data according to the required task employed in each use case. The data mining operators designed so far have demonstrated to be useful for our Web mining prototyping examples, and to properly interoperate with the data manipulation operators.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

The central topic of this thesis is WIM, a model and tool for fast Web mining prototyping. We have presented the WIM formal data model, the WIM algebra with its preliminary set of operators, and five use cases (for one of them we have performed an extensive analysis of the problem) as a proof of concept to WIM.

WIM has demonstrated to have some important properties. The current implementation of the WIM tool is *efficient* to manage several tens of million of tuples in relations, which is the case of the shingles-based comparison used for the Web evolution study, and the datasets used for the usage pagerank study.

Still regarding efficiency, as WIM is based on dataflow programming, the computational complexity of a WIM program is given by the sum of the computational complexity to run each operator independently, which, according to the computational complexity theory, is given by the computational complexity of the operator with highest complexity. Clearly, the mining operators have higher complexity than the manipulation operators, and any ad-hoc implementation of a given solution previously implemented through WIM would need to have an implementation of the mining task, making the computational complexity of the two solutions equivalent.

The WIM model is designed to be *scalable*. Industrial scale implementations of the tool will be able to run distributively, as we indicated in Section 5.3. The current version of the WIM tool, which is not parallel, is scalable under the limits of using one WIM server (optimized when the server is multi-processed), as different levels of indexes may be

provided for different amounts of data to be processed.

The operators are designed obeying the compositionality property, which means that any output relation of a given type can be input of any operator that is defined to input a relation of that type. Thanks to this property and to a consistent conceptual model, WIM is easily *extensible*. For instance, for the usage pagerank application presented in Section 7.1, we have effectively implemented an extension for the Analyze operator. Instead of selecting links that represent just a few clicks, before the call to the Analyze operator, we created a pagerank-like function that is able to include the number of clicks, which is the label of the graph, as a variable of the pagerank calculation function. Thus, we have simulated an external Web miner contributing to the extension of the WIM model. If respecting some implementation rules, changes of one user can be made available for every WIM user.

WIM has shown to be *effective* for a set of real Web mining problems. The operators are specially designed to take advantage of the association between link and node relations, and also to take advantage of the concepts of view and compatibility, allowing inheritance of attributes and properties after an operation. We have compared the WIM data manipulation operators with the relational algebra in Chapter 4, showing the similarities and explaining the situations in which they are not similar, as they have different purposes.

With respect to the Web content evolution study, we have introduced the concept of genealogical tree on the Web, and studied its components. We have estimated that almost one fourth of the new Web documents that appear within the span of a year have content from previously published documents. Further, we have studied the relationship between search engines and the copy behavior, and demonstrated that parents are more clicked and appear more frequently in the results of queries. This conclusion is an important evidence that search engines bias the content of the Web, as people often click on the top results for choosing the documents as source of content to reuse.

WIM was applied not only to a Web evolution study, but also to four other Web mining problems. This thesis does not extend to the analysis of all the problems, though some of them seem to be relevant research topics that will be further analyzed, given that the solution prototyping is already done using WIM.

8.2 Future Work

We have invested most of our time in delivering a version of the WIM prototype for public use. Our intention is to set up a site for the WIM project, containing a description of the

tool, examples of use, and a Web interface for programming in WIM, so that users can log on and submit their query programs.

The other future steps are planned in this order: to implement the WIM operations that have not been implemented so far, and incorporate an existent Web crawler, so that users will be able to create their own Web datasets, ready to be exported and used as WIM relation sets.

We think that the WIM textual programming language syntax is not so intuitive, though it has been important as a proof of concept before implementing a graphical interface, with which the user will be able to graphically choose operators, input sets, relations, and options, according to what is previously recorded in the meta database. This interface upgrade is important not only to the input, but also to visualize output results. The tendency is to integrate WIM with a workflow management system for data exploration and visualization, like VisTrails [Bavoil et al., 2005].

Regarding extensibility, we propose as future work the creation of a set of operators to deal with machine learning approaches, so that supervised learning tasks can be performed through WIM. With the WIM kernel developed, the task of including a new operator is concentrated in the implementation, as WIM is already designed to allow the creation of new operators.

Aiming to exploit parallel computer architectures, we intend to design and develop a map-reduce [Dean and Ghemawat, 2008] version of WIM based in Hadoop [Hadoop, 2008]. This task includes the implementation of a more scalable version of the WIM tool, as we propose in Section 5.3, and a modified software architecture to include the map-reduce paradigm.

Probably the most challenging goal of this thesis is to provide a framework for Web miners, which will be actually useful and will drop the time spent when prototyping Web mining applications. After making the prototype available for Web miners, we expect to watch their interactions and create a collection of many use cases supported by WIM. With an extended list of use cases we will be able to analyze WIM with higher precision, which will be important as a future expected result of this thesis.

So far we have motivated some research topics on Web mining, by proposing applications to WIM. We discuss the future directions of the main research lines we have initiated. With respect to the Web evolution study, we have several proposes for future research. We will soon be given a new dataset with recent snapshots of the Chilean Web. The new dataset was also seeded with the set of all `.cl` domains, but this time exhaustive crawls

were done, ensuring a large coverage of the Chilean Web. They represent a more accurate dataset for researches dealing with changes from one snapshot to another.

Thus, the first task will be to perform the same study for the new dataset. Then we will change the method for comparison. The idea is to extend a method of near-duplicates detection, so that the method can also detect copies of short passages of the Web document. We will study an evolution model for the Web content. Still depending on external data, we want to use Yahoo! usage log to observe how new parents are returned to users and how they are clicked, in order to demonstrate that parents are really more clicked than other documents. More importantly, we want to propose a new ranking method, based on the assumption that most parents are documents chosen by expert users, then parents tend to be more important than other documents. We intend to proceed with a relevance evaluation experiment to demonstrate this hypothesis.

WIM has been used to compose a pool of documents for relevance assessment, aiming the creation of a reference dataset for learning to rank using click-through data. We have already assessed 100 queries, and the next step before publishing the data is to write a report with a description of the dataset, presenting some statistics. We will use this dataset to experiment on our proposal of usage pagerank presented in Chapter 7, which has been another application to WIM. Depending on the relevance of our results, we will experiment with other datasets from Yahoo!.

Although we believe the research directions presented so far are the most promising, we also want to continue developing the other use cases presented in this thesis. Preliminary results presented in Section 7.2 demonstrated that the pagerank of documents duplicated from other existent documents does not evolve well with time, and this can be exploited to avoid fetching URLs with a high probability of being duplicates. We also want to analyze the results of the WIM program to manipulate Web data presented in Section 7.3, according to our proposal of automatically classifying queries regarding user intent according to usage data (as presented in Chapter 7), rather than based on manual classification. Then, we will hypothesize that transactional queries [Broder, 2002] are not important from the perspective of search engines, whereas queries with commercial intention play an important role in the era of sponsored links in query results.

Bibliography

Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of Databases*. Addison-Wesley, Boston, MA.

Agrawal, R., Ailamaki, A., Bernstein, P. A., Brewer, E. A., Carey, M. J., Chaudhuri, S., Doan, A., D. Florescu, M. J. F., Garcia-Molina, H., Gehrke, J., Gruenwald, L., Haas, L. M., Halevy, A. Y., Hellerstein, J. M., Ioannidis, Y. E., Korth, H. F., Kossmann, D., Madden, S., Magoulas, R., Ooi, B. C., O'Reilly, T., Ramakrishnan, R., Sarawagi, S., Stonebraker, M., Szalay, A. S., and Weikum, G. (2008). The Claremont Report on Database Research. http://db.cs.berkeley.edu/claremont/.

Agrawal, R., Imieliński, T., and Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the ACM International Conference on Management of Data*, pages 207–216, Washington, DC, USA. ACM.

Arocena, G. O. and Mendelzon, A. O. (1998). WebOQL: Restructuring documents, databases, and Webs. In *Proceedings of the 14th International Conference on Data Engineering*, pages 24–33, Washington, DC, USA. IEEE Computer Society.

Arocena, G. O., Mendelzon, A. O., and Mihaila, G. A. (1997). Applications of a Web query language. *Comput. Netw. ISDN Syst.*, 29(8-13):1305–1316.

Atzeni, P., Masci, A., Mecca, G., Merialdo, P., and Tabet, E. (1997a). ULIXES: Building relational views over the Web. In *Proceedings of the 13th International Conference on Data Engineering (short paper)*, page 576, Washington, DC, USA. IEEE Computer Society.

Atzeni, P., Mecca, G., and Merialdo, P. (1997b). To weave the web. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 206–215, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Baeza-Yates, R., Castillo, C., and Efthimiadis, E. N. (2007a). Characterization of national Web domains. *ACM Trans. Inter. Tech.*, 7(2):9.

Baeza-Yates, R., Castillo, C., and Saint-Jean, F. (2004). *Web Dynamics*, chapter Web Dynamics, Structure and Page Quality, pages 93–109. Springer.

Baeza-Yates, R., Jones, T., and Rawlins, G. (2000). New approaches to information management: Attribute-centric data systems. In *Proceedings of the 7th International Symposium on String Processing and Information Retrieval*, pages 18–27, A Coruña, Spain. IEEE Computer Science Press. (Invited paper).

Baeza-Yates, R., Pereira, A., and Ziviani, N. (2007b). Understanding content reuse on the web: Static and dynamic analyses. In Nasraoui, O., Spiliopoulou, M., Srivastava, J., Mobasher, B., and Masand, B., editors, *Advances in Web Mining and Web Usage Analysis*, volume 4811 of *LNCS*, pages 227–246. Springer.

Baeza-Yates, R., Pereira, A., and Ziviani, N. (2008). Genealogical trees on the Web: a search engine user perspective. In *Proceedings of the 17th International World Wide Web Conference*, pages 367–376, Beijing, China.

Baeza-Yates, R., Pereira-Jr, A., and Ziviani, N. (2006). The evolution of web content and search engines. In *Proceedings of the 8th ACM Workshop on Web Mining and Web Usage Analysis*, pages 58–67, Philadelphia, PA, USA.

Baeza-Yates, R., Pereira-Jr, A. R., and Ziviani, N. (2005a). WIM: an information mining model for the web. In *Proceedings of the Third Latin American Web Congress*, pages 233–241, Buenos Aires, Argentina. IEEE Computer Society Press.

Baeza-Yates, R., Pereira-Jr, A. R., and Ziviani, N. (2005b). WIM: an information mining model for the web (extended abstract). In *Proceedings of the First DEXA International Workshop on Data Management in Global Data Repositories*, pages 1155–1159, Copenhagen, Denmark. IEEE Computer Society Press.

Baeza-Yates, R. and Poblete, B. (2006). Dynamics of the chilean Web structure. *Computer Networks*, 50(10):1464–1473.

Baeza-Yates, R. and Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. ACM Press/ Addison-Wesley.

Bavoil, L., Callahan, S. P., Scheidegger, C. E., Vo, H. T., Crossno, P., Silva, C. T., and Freire, J. (2005). Vistrails: Enabling interactive multiple-view visualizations. In *Proceedings of the IEEE Visualization*, pages 135–142.

Bennouas, T. and Montgolfier, F. (2007). Random web crawls. In *Proceedings of the 16th International World Wide Web Conference*, pages 451–460. ACM Press.

Berkhin, P. (2002). Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA.

Bharat, K. and Broder, A. (1999). Mirror, mirror on the Web: a study of host pairs with replicated content. In *Proceedings of the 8th International Conference on World Wide Web*, pages 1579 – 1590, Toronto, Canada.

Bhowmick, S., Madria, S., Ng, W.-K., and Lim, E.-P. (1998). Web bags: are they useful in a Web warehouse? In *Proceedings of the 5th International Conference of Foundation of Data Organization*, pages 210–220, Kobe, Japan.

Bhowmick, S. S., Madria, S. K., Ng, W. K., and Lim, E.-P. (2000). Data visualization operators for WHOWEDA. *Computer Journal*, 43(5):364–385.

Bhowmick, S. S., Ng, W.-K., Madria, S. K., and Mohania, M. K. (2002). Constraint-free join processing on hyperlinked Web data. In *Proceedings of the 4th International Conference on Data Warehousing and Knowledge Discovery*, pages 255–264. Springer-Verlag.

Boldi, P. and Vigna, S. (2004). The WebGraph framework I: Compression techniques. In *Proceedings of the 13th International World Wide Web Conference*, pages 595–601, New York, NY, USA. ACM Press.

Borthakur, D. (2007). The hadoop distributed file system: Architecture and design. http://hadoop.apache.org/core/docs/current/hdfs_design.pdf.

Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of the 7th Internation World Wide Web Conference*, pages 107–117.

Broder, A. (1998). On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences*, pages 21–29. IEEE Computer Society.

Broder, A., Glassman, S., Manasse, M., and Zweig, G. (1997). Syntactic clustering of the Web. In *Proceedings of the 6th International World Wide Web Conference*, pages 391–404.

Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., and Wiener, J. (2000). Graph structure in the web. In *Proceedings of the 9th International World Wide Web Conference*, pages 309–320, Amsterdam, Netherlands.

Broder, A. Z. (2002). A taxonomy of Web search. *SIGIR Forum*, 36(2):3–10.

Chakrabarti, S. (2002). *Mining the Web: Discovering Knowledge from Hypertext Data.* Morgan Kauffman, San Francisco, USA.

Chawathe, S., Garcia-molina, H., Hammer, J., Irel, K., Papakonstantinou, Y., Ullman, J., and Widom, J. (1994). The TSIMMIS project: Integration of heterogeneous information sources. In *Proceedings of the IPSJ Conference*, pages 7–18.

Cho, J., Garcia-Molina, H., Haveliwala, T., Lam, W., Paepcke, A., Raghavan, S., and Wesley, G. (2006). Stanford webbase components and applications. *ACM Transactions on Internet Technology*, 6(2):153–186.

Cho, J. and Roy, S. (2004). Impact of search engine on page popularity. In *Proceedings of the 13th International World Wide Web Conference*, pages 20–29, New York, USA.

Cho, J., Shivakumar, N., and Garcia-Molina, H. (2000). Finding replicated Web collections. In *Proceedings of the 19th ACM International Conference on Management of Data*, pages 355–366.

Codd, E. T. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387.

Cooley, R., Mobasher, B., and Srivastava, J. (1997). Web mining: Information and pattern discovery on the World Wide Web. In *Proceedings of the 9th International Conference on Tools with Artificial Intelligence*, pages 558–567, Washington, DC, USA. IEEE Computer Society.

Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1990). *Introduction to algorithms.* MIT Press/McGraw-Hill, San Francisco, CA.

Dasgupta, A., Ghosh, A., Kumar, R., Olston, C., Pandey, S., and Tomkins, A. (2007). The discoverability of the web. In *Proceedings of the 16th International World Wide Web Conference*, pages 421–430. ACM Press.

DB2 Intelligent Miner (2008). http://www-306.ibm.com/software/data/iminer/.

Dean, J. and Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

Eiron, N., McCurley, K., and Tomlin, J. (2004). Ranking the web frontier. In *Proceedings of the 13th International World Wide Web Conference*, pages 309–318, New York, USA. ACM Press.

Fernandez, M., Florescu, D., Kang, J., Levy, A., and Suciu, D. (1997a). STRUDEL: a Web site management system. *SIGMOD Record*, 26(2):549–552.

Fernandez, M., Florescu, D., Levy, A., and Suciu, D. (1997b). A query language for a Web-site management system. *SIGMOD Record*, 26(3):4–11.

Fetterly, D., Manasse, M., and Najork, M. (2003). On the evolution of clusters of near-duplicate Web pages. In *Proceedings of the First Latin American Web Congress*, pages 37–45, Santiago, Chile.

Fiebig, T., Weiss, J., and Moerkotte, G. (1997). Raw: a relational algebra for the web. In *Proceedings of the Workshop on Management of Semistructured Data*, Tuscon, Arizona.

Florescu, D., , Levy, A., and Mendelzon, A. (1998). Database techniques for the World-Wide Web: a survey. *SIGMOD Record*, 27(3):59–74.

Fortunato, S., Flammini, A., Menczer, F., and Vespignani, A. (2006). Topical interests and the mitigation of search engine bias. In *Proc. Natl. Acad. Sci. USA 103(34): 12684-12689*. National Academy of Science.

Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J. D., Vassalos, V., and Widom, J. (1997). The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8(2):117–132.

Garcia-Molina, H., Ullman, J. D., and Widom, J. (2001). *Database Systems: The Complete Book*. Prentice Hall.

Gonnet, G. H. and Baeza-Yates, R. (1991). *Handbook of algorithms and data structures: in Pascal and C*. Addison-Wesley Longman Publishing Co., Boston, MA, second edition.

Hadoop (2008). http://hadoop.apache.org/.

Hammer, J., García-Molina, H., Ireland, K., Papakonstantinou, Y., Ullman, J., and Widom, J. (1995). Information translation, mediation, and mosaic-based browsing in the TSIMMIS system. In *Proceedings of the International Conference on Management of Data*, page 483, San Jose, California, United States.

Han, J. and Kamber, M. (2006). *Data mining: comcepts and techniques*. Morgan Kaufmann, San Francisco, USA, second edition.

Kessler, M. M. (1963). Bibliographic coupling between scientific papers. American Documentation.

Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632.

Konopnicki, D. and Shmueli, O. (1995). W3QS: A query system for the World-Wide Web. In *Proceedings of the International Conference on Very Large Data Bases*, pages 54–65.

Konopnicki, D. and Shmueli, O. (1998). Information gathering in the World-Wide Web: the W3QL query language and the W3QS system. *ACM Trans. Database Syst.*, 23(4):369–410.

Kosala, R. and Blockeel, H. (2000). Web mining research: A survey. *SIGKDD Explorations: Newsletter of the Special Interest Group (SIG) on Knowledge Discovery & Data Mining*, 2.

Lakshmanan, L. V. S., Subramanian, I. N., and Sadri, F. (1996). A declarative language for querying and restructuring the web. In *Proceedings of the 6th International Workshop on Research Issues in Data Engineering*, pages 12–21, Washington, DC, USA. IEEE Computer Society.

Lee, U., Liu, Z., and Cho, J. (2005). Automatic identification of user goals in web search. In *Proceedings of the 14th International Conference on World Wide Web*, pages 391–400, Chiba, Japan.

Lim, E.-P. and Ng, W. K. (1997). A relational interface for heterogeneous information sources. In *Proceedings of the IEEE International Forum on Research and Technology Advances in Digital Libraries*, pages 128–139, Washington, DC, USA. IEEE Computer Society.

Liu, B. (2007). *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. Springer.

Liu, Y., Gao, B., Liu, T., Zhang, Y., Ma, Z., He, S., and Li, H. (2008). BrowseRank: letting web users vote for page importance. In *Proceedings of the 31st International ACM Conference on Research and Development in Information Retrieval*, pages 451–458.

MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297. University of California Press.

Manku, G. S., Jain, A., and Sarma, A. D. (2007). Detecting near-duplicates for web crawling. In *Proceedings of the 16th International World Wide Web Conference*, pages 141–150. ACM Press.

Mecca, G., Atzeni, P., Masci, A., Sindoni, G., and Merialdo, P. (1998). The araneus web-based management system. In *Proceedings of the International Conference on Management of Data*, pages 544–546, Seattle, Washington, United States.

Mendelzon, A. O., Mihaila, G. A., and Milo, T. (1997). Querying the World Wide Web. *Journal of Digital Libraries*, 1(1):68–88.

Microsoft SQL Server 2005 (2008). http://www.microsoft.com/sql/technologies/dm.

Ng, W.-K., Lim, E.-P., Huang, C.-T., Bhowmick, S., and Qin, F.-Q. (1998). Web warehousing: An algebra for Web information. In *Proceedings of the Advances in Digital Libraries Conference*, pages 228–237. IEEE Computer Society.

Ntoulas, A., Cho, J., Cho, H. K., Cho, H., and Cho, Y.-J. (2005). A study on the evolution of the Web. In *US - Korea Conference on Science, Technology, and Entrepreneurship*, pages 1–6, Irvine, USA.

Ntoulas, A., Cho, J., and Olston, C. (2004). What's new on the Web? the evolution of the Web from a search engine perspective. In *Proceedings of the 13th International World Wide Web Conference*, pages 1–12, New York, USA.

Olston, C., Reed, B., Srivastava, U., Kumar, R., and Tomkins, A. (2008). Pig Latin: A not-so-foreign language for data processing. In *Proceedings of the 28th ACM International Conference on Management of Data*, pages 1099–1110.

Oracle Data Mining (2008). http://www.oracle.com/technology/products/bi/odm.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1998). The pagerank citation ranking: Bringing order to the Web. Technical Report CA 93106, Stanford Digital Library Technologies Project, Stanford, Santa Barbara.

Pandey, S., Roy, S., Olston, C., Cho, J., and Chakrabarti, S. (2005). Shuffling a stacked deck: the case for partially randomized ranking of search engine results. In *Proceedings of the 31st International Conference on Very large Data Bases*, pages 781–792.

Pereira, A., Baeza-Yates, R., and Ziviani, N. (2008). A model for web mining applications - conceptual model, architecture, implementation and use cases. Technical Report RT.DCC. 001/2008, Federal University of Minas Gerais.

Pereira, A., Baeza-Yates, R., Ziviani, N., and Bisbal, J. (2009). A model for fast Web mining prototyping. In *Proceedings of the 2nd ACM International Conference on Web Search and Data Mining*, pages 114–123, Barcelona, Spain.

Pereira-Jr, A. and Baeza-Yates, R. (2005). Applications of an information mining model to data mining and information retrieval tasks. In *Proceedings of the First DEXA International Workshop on Integrating Data Mining, Databases and Information Retrieval*, pages 1031–1035, Copenhagen, Denmark. IEEE Computer Society Press.

Pereira-Jr, A. R., Baeza-Yates, R., and Ziviani, N. (2006). Where and how duplicates occur in the web. In *Proceedings of the Fourth Latin American Web Congress*, pages 127–134, Cholula, Mexico. IEEE Computer Society Press.

Pereira-Jr, A. R. and Ziviani, N. (2004). Retrieving similar documents from the web. *Journal of Web Engineering (JWE)*, 2(4):247–261.

Pereira Jr, A. R., Ziviani, N., and Baeza-Yates, R. (2004). Combinação de evidências para identificação de comunidades na web. In *Proceedings of the PhD and MSc Workshop, WebMedia and LA-Web Joint Conference*, pages 245–248, Ribeirão Preto, Brazil. PhD and MSc Track.

Pike, R., Dorward, S., Griesemer, R., and Quinlan, S. (2005). Interpreting the data: Parallel analysis with Sawzall. *Scientific Programming*, 13(4):277–298.

Raghavan, S. and Garcia-Molina, H. (2003a). Complex queries over Web repositories. In *Proceedings of the 29th International Conference on Very Large Data Bases*, pages 33–44, Berlin, Germany.

Raghavan, S. and Garcia-Molina, H. (2003b). Representing web graphs. In *Proceedings of the International Conference on Data Engineering*, pages 405–416.

Rose, D. E. and Levinson, D. (2004). Understanding user goals in Web search. In *Proceedings of the 13th International Conference on World Wide Web*, pages 13–19, New York, NY, USA. ACM Press.

Salton, G., Wong, A., and Yang, C. S. (1975). A vector space model for automatic indexing. *Communications of ACM*, 18(11):613–620.

Small, H. G. (1973). Co-citation in the scientific literature: A new measure of relationship between two documents. *Journal of the American Society for Information Science*, 24:265–269.

Sparck-Jones, K., Walker, S., and Robertson, S. E. (2000). A probabilistic model of information retrieval: development and comparative experiments. *Inf. Process. Manage.*, 36(6):779–808.

Spertus, E. and Stein, L. A. (2000). Squeal: A structured query language for the Web. In *Proceedings of the 9th International World Wide Web Conference*, pages 95–103, Amsterdam, The Netherlands.

Spiliopoulou, M. and Faulstich, L. C. (1998). WUM: A Web utilization miner. In *Proceedings of the Workshop on the Web and Databases*, pages 109–115. Springer Verlag.

Stonebraker, M., Abadi, D. J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E., O'Neil, P., Rasin, A., Tran, N., and Zdonik, S. (2005).

C-store: a column-oriented DBMS. In *Proceedings of the 31st International Conference on Very Large Data Bases*, pages 553–564, Trondheim, Norway. VLDB Endowment.

Toyoda, M. and Kitsuregawa, M. (2006). What's really new on the web? identifying new pages from a series of unstable web snapshots. In *Proceedings of the 15th International Conference on World Wide Web*, pages 233–241, Edinburgh, Scotland.

Wang, J. (2003). *Data mining: opportunities and challenges*. IGI Publishing, Hershey, PA, USA.

Witten, I. H. and Frank, E. (2005). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, USA, second edition.

Wood, C. A. and Ow, T. T. (2005). WEBVIEW: an SQL extension for joining corporate data to data derived from the web. *Communications of ACM*, 48(9):99–104.

Zhang, J. and Suel, T. (2007). Efficient search in large textual collections with redundancy. In *Proceedings of the 16th International Conference on World Wide Web*, pages 411–420, Banff, Alberta, Canada.

Appendix 1: WIM Language Syntax

Below we present the syntax of the WIM language, according to the BNF-like language specification described in Chapter 4, in which the operators are specified separately. Note that comment lines start with character '#', and that each symbol is defined only once.

Specification of a WIM program:
WIMProgram := Operator ';' { Operator ';' }
Operator := Select | Calculate | CalcGraph | Aggregate | Set | Join |
Convert | Search | Compare | CompGraph | Cluster |
Disconnect | Associate | Analyze | Relink

Frequently used symbols:
Digit := '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
IntValue := Digit { Digit }
NumValue := IntValue ['.' IntValue]
Letter := 'A' | 'B' | ... | 'Z' | 'a' | 'b' | ... | 'z'
Name := Letter | Digit { Letter | Digit }
OutputRel := Name ['.' Name]
OutputLinkRel := Name ['.' Name]
OutputNodeRel := Name ['.' Name]
InputSet := Name
InputNodeSet := Name
InputLinkSet := Name
NumRel := Name '.' Attr
IntRel := Name '.' Attr
TxtRel := Name '.' Attr
AnyRel := Name '.' Attr
Attr := 'K' | 'V' | 'S' | 'E' | 'L' | 'SE'

```
# Operator Select:
Select := OutputRel '=' 'Select' '(' InputSet ',' BodySelect ')'
BodySelect := 'value' ',' NumRel ',' SelOperation ',' NumValue |
        'attribute' ',' NumRel ',' NumRel ',' SelOperation |
        'top' [ ',' Order ] ',' NumRel ',' IntValue
SelOperation := '==' | '! =' | '<' | '<=' | '>' | '>='
Order := 'increasing' | 'decreasing'

# Operator Calculate:
Calculate := OutputRel '=' 'Calculate' '(' InputSet ',' BodyCalc ')'
BodyCalc := 'constant' ',' ConstOper ',' NumRel ',' NumValue |
        'pair' ',' PairOper ',' NumRel ',' NumRel
ConstOper := 'sum' | 'difference' | 'multiplication' | 'division' | 'average' |
        'deviation' | 'mod' | 'normalize' | 'absolute' | 'max' | 'min'
PairOper := 'sum' | 'difference' | 'multiplication' | 'division' | 'average' |
        'mod' | 'percentage' | 'max' | 'min'

# Operator CalcGraph:
CalcGraph := OutputLinkRel '=' 'CalcGraph' '(' InputLinkSet ','
        CalcGraOperator ',' NumRel ',' NumRel ')'
CalcGraOperator := 'sum' | 'difference' | 'multiplication' | 'division' |
        'average' | 'mod' | 'percentage' | 'max' | 'min'

# Operator Aggregate:
Aggregate := OutputRel '=' 'Aggregate' '(' InputSet ',' BodyAgg ')'
BodyAgg := 'single' ',' AggOperator ',' NumRel |
        'grouping' ',' AggOperator ',' NumRel [ ',' NumRel ] [ ',' NumRel ]
AggOperator := 'sum' | 'average' | 'count' | 'max' | 'min' | 'deviation' |
        'geometric' | 'mode' | 'median'

# Operator Set:
Set := OutputRel '=' 'Set' '(' FirstInputSet ',' SecInputSet ','
        SetOperation ',' AnyRel ',' AnyRel ')'
FirstInputSet := Name
SecInputSet := Name
SetOperation := 'union' | 'intersection' | 'difference'
```

```
# Operator Join:
Join := OutputRel '=' 'Join' '(' FirstInputSet ',' SecInputSet ',' AnyRel ','
        AnyRel ',' AnyRel ',' DefaultValue ')'
DefaultValue : StringValue | NumValue
StringValue := "'" Name "'"

# Operator Convert:
Convert := OutputNodeRel = 'Convert' '(' InputLinkSet ',' IntRel
                [ ',' AnyRel ] [ ',' AnyRel ] ')' |
        OutputLinkRel = 'Convert' '(' InputNodeSet ',' IntRel ','
                IntRel [ ',' AnyRel ] ')'
# Operator Search:
Search := OutputLinkRel '=' 'Search' '(' FirstInputNodeSet ',' SecInputNodeSet ','
        TxtRel ',' TxtRel ',' CompMethod ')'
FirstInputNodeSet := Name
SecInputNodeSet := Name
CompMethod := SimpleMatch | Shingles | Tfidf | Bm25
SimpleMatch := 'ExactMatch' | 'AND' | 'OR'
Shingles := 'Shingles' ',' NumRes [ ',' MinSim ',' Overlap ',' ShinSize ]
NumRes := IntValue
MinSim := NumValue
Overlap := 'yes' | 'no'
ShinSize := IntValue
Tfidf := 'TFIDF' ',' NumRes
Bm25 := 'BM25' ',' NumRes [ ',' k1 ',' k3 ',' b ]
k1 := NumValue
k3 := NumValue
b := NumValue

# Operator Compare:
Compare := OutputLinkRel '=' 'Compare' '(' InputNodeSet ',' BodyComp ')'
BodyComp := 'Sparse' ',' TxtRel ',' CompMethod |
        'Dense' ',' TxtRel ',' CompMethod

# Operator CompGraph:
CompGraph := OutputLinkRel '=' 'CompGraph' '(' InputLinkSet ','
        TxtRel ',' TxtRel ',' CompMethod ')'
```

```
# Operator Cluster:
Cluster := OutputNodeRel '=' 'Cluster' '(' InputNodeSet ',' ClusMethod ','
        NumRel [ ',' NumRel ] [ ',' NumRel ] ',' NumValue ')'
ClusMethod := 'Euclidean' | 'Manhattan'

# Operator Disconnect:
Disconnect := OutputNodeRel '=' 'Disconnect' '(' InputLinkSet ','
        DistMethod ')'
DistMethod := 'Connected' | 'Strongly'

# Operator Associate:
Associate := OutputLinkRel '=' 'Associate' '(' InputNodeSet ','
        BodyAssoc ',' Support ',' Confidence ')'
BodyAssoc := 'Value' ',' RuleMethod ',' IntRel ',' IntRel |
        'Term' ',' RuleMethod ',' IntRel ',' TxtRel
RuleMethod := 'Itemset' | 'Association' | 'Sequencial'
Support := NumValue
Confidence := NumValue

# Operator Analyze:
Analyze := OutputNodeRel '=' 'Analyze' '(' InputLinkSet ',' Algorithm ')'
Algorithm := 'Pagerank' | 'Hub' | 'Authority' | 'Indegree'

# Operator Relink:
Relink := OutputLinkRel '=' 'Relink' '(' InputLinkSet ',' BodyRelink ')'
BodyRelink := CitMethod ',' Direction ',' Density ',' | 'Transitivity'
CitMethod := 'CoCitation' | 'Coupling'
Direction := 'Both' | 'Single'
Density := 'All' | 'Adjacent'
```