Universidade Federal do Amazonas
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Programa de Pós-Graduação em Informática

Estudo de Métricas e Técnicas de Indexação para Suporte a Junções por Similaridade

Nívea Michelle de Melo Carvalho

Manaus – Amazonas
Outubro de 2004

Nívea Michelle de Melo Carvalho

Estudo de Métricas e Técnicas de Indexação para Suporte a Junções por Similaridade

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Departamento de Ciência da Computação da Universidade Federal do Amazonas, como requisito parcial para obtenção do Título de Mestre em Informática. Áreas de concentração: Banco de Dados e Recuperação da Informação.

Orientador: Prof. Dr. Altigran Soares da Silva

Nívea Michelle de Melo Carvalho

Estudo de Métricas e Técnicas de Indexação para Suporte a Junções por Similaridade

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Departamento de Ciência da Computação da Universidade Federal do Amazonas, como requisito parcial para obtenção do Título de Mestre em Informática. Áreas de concentração: Banco de Dados e Recuperação da Informação.

Banca Examinadora

Prof. Dr. Altigran Soares da Silva – Orientador
Departamento de Ciência da Computação – UFAM/PPGI

Prof. Dr. Edleno Silva de Moura
Departamento de Ciência da Computação – UFAM/PPGI

Prof. Dr. João Marcos Bastos Cavalcanti
Departamento de Ciência da Computação – UFAM/PPGI

Manaus – Amazonas
Outubro de 2004

Para minha avó Diva. (in memoriam)

Agradecimentos

A Deus, acima de tudo.

À minha mãe, Vandamelina Carvalho, pelo exemplo, dedicação e apoio nos momentos mais difíceis.

Ao meu orientador, Altigran Soares da Silva, pela oportunidade, paciência, persistência, incentivo e por ter dedicado muitas horas na discussão e estudos necessários a realização deste trabalho.

Aos amigos Ketlen Teles, Keyla Ahnizeret, David Braga e Eduardo Abnader, pelo incentivo e apoio quando mais precisava.

Aos meus colegas da UFAM: Daniel Fernandes, Edson César, Jaques Moura, Moisés Carvalho, Germaine Martins, Júnio Freitas, Fabríco D'Morison e Célia Santos pela troca de conhecimento, experiências e pela amizade.

À todos os professores do Curso de Mestrado, pelo engajamento, competência e pelo conhecimento compartilhado.

Ao ISAE por proporcionar todas as condições necessárias para que eu pudesse participar do Curso de Mestrado; em especial à Dr^a Rosa Pontes, por ter sempre incentivado o meu crescimento profissional e pessoal.

Ao amigo Waldir Nonato pelo incentivo e apoio, sem os quais não teria sido possível concluir o curso.

Aos amigos Carlos Henrique, Yêdda Souza e Bruno Gadelha, por terem me estimulado a continuar quando as barreiras pareciam intransponíveis.

Ao Victor Otávio, pelo carinho e confiança fundamentais nos momentos em que seguir adiante parecia impossível.

À Elienai Nogueira e Mary Jani Fontenelle, pelo apoio administrativo.

A todos aqueles que ajudaram de alguma forma na realização deste trabalho, o meu mais profundo agradecimento.

Quando nada parece ajudar, eu vou e olho o cortador de pedras martelando sua rocha talvez cem vezes sem que nem uma só rachadura apareça. No entanto, na centésima primeira martelada, a pedra se abre em duas, e eu sei que não foi aquela a que conseguiu, mas todas as que vieram antes.

Jacob Riis

Resumo

As operações de junção em Bancos de Dados são feitas com base na comparação exata de valores dos atributos. Entretanto, é comum encontrarmos em bancos de dados o mesmo objeto do mundo real representado de formas distintas. Esse problema pode ser facilmente observado, por exemplo, em bibliotecas digitais, onde não é raro encontrar o nome de um mesmo autor escrito de diferentes formas, seja devido a erros de digitação, seja por falhas no processo de reconhecimento ótico de caracteres (OCR) ou abreviações. Nesse caso, como a comparação dos valores deve ser feita de forma aproximada, temos uma operação conhecida na literatura como junção por similaridade (similarity join). Nesta dissertação é apresentado um estudo de funções e estruturas de indexação para suporte à junção por similaridade, bem como a análise de eficiência e eficácia dos resultados obtidos através de experimentação. Como resultado deste estudo, propomos um modelo em dois níveis, denominado $2LM$ (*two-level matching*), que adota uma estrutura métrica de indexação para suporte a junções aproximadas e que contempla diversos tipos de erros comuns entre representações distintas de objetos, como erros de edição e supressão de termos. Por considerar para fins de junção objetos onde todos os termos apresentam erros de edição, experimentalmente verificou-se que o $2LM$ aplicado a coleções onde erros de grafia em todos os termos são freqüentes apresenta resultados muito superiores aos demais modelos, e em coleções com poucos erros, sua eficácia é muito próxima do melhor modelo disponível na literatura ($Soft - TFIDF$).

Palavras-chaves: Tecnologia da Informação, Recuperação de Informação, Banco de Dados, Junção por Similaridade, Integração de Bases de Dados, Espaços Métricos, Estruturas Métricas.

Abstract

Join operation in databases is based on the exact matching of attribute values. However, it is usual to find the same real word object represented on different ways. This problem can be easily observed, for example, in digital libraries, where it is not rare to find the name of a same author written in different forms, either due to typewriting mistakes, flaws in the process of optic recognition of characters (OCR) or abbreviations. In that case, where the values for comparison must be done in an approximate way, there is an operation known in the literature as *similarity join*. In this work a study of functions and indexing structures for *similarity join* is presented, as well as an efficiency and effectiveness analysis of the results obtained through experimentation. As a result of this study, we propose a two-level model, called $2LM$, that adopts a metric index structure for similarity join support and contemplates several types of common mistakes among different representation of objects, as misspelling and terms suppression. For considering objects where all terms present edition mistakes, experiments shown that $2LM$ applied to collections where misspelling in all terms are frequent presents better results than others solutions, and in collections with few mistakes, its effectiveness is close to the best model available in literature ($Soft - TFIDF$).

Sumário

1 Introdução **1**

1.1 Trabalhos relacionados 4

1.2 Organização da dissertação 7

2 Conceitos e Terminologia **9**

2.1 Similaridade e Distância 9

2.1.1 Funções de Distância 10

2.1.2 Consultas por similaridade 12

2.2 Estruturas de Indexação Métricas 13

2.2.1 Os efeitos da dimensionalidade (Dimensionality Curse) 17

2.3 Junção por Similaridade 18

2.3.1 Modelos para junção por similaridade 19

2.3.1.1 O uso de Distância de Palavra 19

2.3.1.2 O uso de Distância de Edição e Similares 21

2.3.1.3 Usando Modelos Híbridos 23

2.3.2 Limiares (thresholds) 24

2.4 Métodos de avaliação 24

2.4.1 Precisão e Revocação 24

2.4.2 Medida F_1 25

3 O Modelo 2LM - Two-level Matching **26**

3.1 Erros mais freqüentes 26

3.2 Métricas baseadas em Palavras 27

3.3 Métricas baseadas em Distância de Edição e Similares 29

3.4 Dinamicidade 30
3.5 O Modelo 2LM 31
3.5.1 Economizando Cálculos de Distância 33

4 Verificação Experimental 36
4.1 As Coleções de Teste 37
4.1.1 BDBComp 37
4.1.2 DBLP 39
4.2 Ferramentas Utilizadas 40
4.3 Descrição dos experimentos 40
4.3.1 Definição de Limiares (thresholds) 41
4.3.2 Experimentos em coleções previamente conhecidas 42
4.3.3 Experimentos em coleções dinâmicas 43
4.4 Resultados 44
4.4.1 BDBComp 44
4.4.2 DBLP 49

5 Conclusão e Trabalhos Futuros 53
5.1 Trabalhos futuros 54

Referências Bibliográficas 56

Lista de Figuras

2.1 Desigualdade Triangular 11
2.2 Range Query 12
2.3 3-Nearest Neighbor Query 13
2.4 R-Tree 14
2.5 vp-tree 14
2.6 M-Tree 15
2.7 D-Index 17
2.8 Exemplo de consulta usando o Modelo Vetorial 19
2.9 Função de similaridade de Jaro 22

3.1 Algoritmo de Range Query 34
3.2 Exemplo de 2LM 35

4.1 Exemplo da BDBComp 38
4.2 Lista Invertida utilizada em funções de similaridade de token 41
4.3 BDBComp - Resultado utilizando IDF 45
4.4 BDBComp - Resultados sem IDF 46
4.5 BDBComp - Resultados 47
4.6 DBLP - Resultado utilizando IDF 50
4.7 DBLP - Resultados sem IDF 51
4.8 DBLP - Resultados 52

Lista de Tabelas

1.1 Métricas de similaridade 7

1.2 Estruturas Métricas 8

4.1 Estatísticas da coleção BDBComp 39

4.2 Estatísticas da coleção DBLP 40

4.3 Escolha do Limiar para a função Jaro-Winkler 42

4.4 Escolha do Fator Multiplicador 43

4.5 Avaliação de Métricas com IDF sobre a BDBComp 45

4.6 Avaliação de Métricas sem IDF sobre a BDBComp 46

4.7 Comparação de Métricas de Token com suas variantes Híbridas - BDBComp 47

4.8 Avaliação de Métricas sobre a BDBComp 48

4.9 Tempo de Processamento e Cálculos de Distância - BDBComp 48

4.10 DBLP - Erros de Edição 50

4.11 Comparação de F_1 sobre Coleções Sintéticas da DBLP 51

4.12 Comparação de Métricas de Token com suas variantes Híbridas - DBLP 51

Capítulo 1

Introdução

O problema de identificar diferentes representações textuais que se referem a um mesmo objeto do mundo real é um grande desafio e tem estimulado pesquisas em diversas áreas, tais como Banco de Dados, Recuperação de Informação, Inteligência Artificial e Estatística [4].

Esse problema pode ser facilmente observado, por exemplo, em bibliotecas digitais, onde não é raro encontrar o nome de um mesmo autor escrito de formas distintas, seja devido a erros de digitação, seja por falhas no processo de reconhecimento ótico de caracteres (OCR) ou abreviações.

Um exemplo prático pode ser observado ao executarmos a consulta "Casanova" na *Biblioteca Digital Brasileira de Computação* (BDBComp)[1]. Como resultado, são obtidas três representações distintas para o mesmo autor: "Marco A. Casanova", "Marco Antônio Casanova" e "Marco Antonio Casanova". Para resolver este problema, adota-se em geral um processo manual, onde uma pessoa pesquisa os nomes mais próximos e atribui o mesmo identificador a eles. Essa tarefa, além de suscetível a erros, torna-se inviável considerando-se o crescimento das coleções e o tempo consumido para a retificação. Dessa forma, soluções que possibilitem a identificação automática de objetos similares representam uma necessidade real. Na literatura corrente esta tarefa é geralmente conhecida como *data cleaning* [12].

Ao longo deste texto nós utilizaremos o termo *nome* para nos referirmos a uma *string* que representa uma entidade do mundo real. Assim, por exemplo, as *strings* "Marco Antonio Casanova", "Marco A. Casanova" e "Marco Antônio Casanova" são *nomes* pois, na *BDBComp*

[1] http://www.lbd.dcc.ufmg.br/bdbcomp

são utilizadas para identificar o autor Marco Antônio Casanova. Da mesma forma, na *CDDB*[2], banco de dados com informação sobre mais de 38 milhões de músicas, o título de CDs e músicas são exemplos de *nomes*. Esses *nomes* são comparados aos existentes na coleção toda vez que algum usuário tenta cadastrar um CD ou música.

Outra situação em que é necessário identificar *nomes* distintos que se referem ao mesmo objeto é quando se necessita associar informações disponíveis em bancos de dados heterogêneos, onde, em geral, os identificadores dos objetos não são idênticos. A importância de resolver esse problema cresce a medida que aumenta o número de coleções disponíveis na *World Wide Web* (WWW). Diversos estudos voltados à padronização dessas coleções foram desenvolvidos, gerando como produto a tecnologia *Extensible Markup Language* (XML) [3] que facilita a identificação do conteúdo através de tags. Entretanto, instâncias XML de diferentes bancos de dados podem apresentar o mesmo objeto do mundo real de forma distinta, como pode ser visto no exemplo a seguir.

Considerando ainda a Biblioteca Digital BDBComp, suponha que se deseja adicionar uma conferência recém-realizada. Para integrar as informações de maneira adequada, necessita-se identificar quais os autores com artigos nesta nova conferência já estão representados na biblioteca digital. No entanto, não há garantia de que o *nome* dado a um certo autor na conferência seja exatamente igual ao *nome* que o representa na BDBComp.

Em bancos de dados, a operação de associar dados de dois conteúdos distintos com base na comparação dos valores de um dado atributo que ocorre nos dois conjuntos é chamada de junção [32]. No nosso caso, como a comparação dos valores deve ser feita de forma aproximada, temos uma operação conhecida na literatura como *junção por similaridade*, ou *similarity join* [11].

Diversas abordagens têm sido propostas recentemente para o problema de junção por similaridade, todas elas estão baseadas no uso de métricas de similaridades. Por exemplo, Cohen [11] utiliza a similaridade do *cosseno* (Modelo Vetorial) para implementar uma solução de integração baseada em valores de atributos de um banco de dados relacional; Jaro e Winkler [7, 5], assim como Smith e Waterman [17], utilizam funções que medem a dissimilaridade de edição, ou distância de edição, para comparar *strings* e assim identificar objetos similares.

Um algoritmo simples para junção por similaridade computa o *Produto Cartesiano* entre

[2] http://www.cddb.com
[3] http://www.w3.org

dois conjuntos e decide que pares de objetos podem ser considerados similares segundo um valor mínimo de similaridade. Esse algoritmo tem o custo esperado O(n^k), onde k é o número de elementos de junção e n o número médio de objetos nas coleções envolvidas. Ou seja, tendo-se uma coleção A (autor,título) e uma coleção B(autor, título), para realizar a junção entre A.[autor] e B.[autor], seria necessário realizar n^2 computações da similaridade entre as coleções. Considerando a BDBComp, k seria, pelo menos, igual a 91, o que tornaria o emprego de modelos convencionais de integração muito dispendioso.

Outro fator importante a ser considerado é a freqüência da manutenção da coleção e seu impacto sobre as similaridades calculadas. Coleções como a BDBComp sofrem atualizações constantes uma vez que a cada novo evento, novos objetos são inseridos. Neste caso, a aplicação de modelos que façam uso da freqüência inversa dos termos (IDF) são prejudicados pois baseiam-se no conhecimento prévio da coleção.

Buscando aumentar a escalabilidade e manter a efetividade dos modelos, foram desenvolvidos métodos para otimização das estruturas de indexação, abordando inclusive a utilização de estruturas métricas.

Neste trabalho é apresentado um estudo de funções e estruturas de indexação para suporte à junção por similaridade, bem como a análise de eficiência e eficácia dos resultados obtidos através de experimentação. Como resultado deste estudo, propomos um modelo em dois níveis, denominado $2LM$ (*two-level matching*), que adota uma estrutura métrica de indexação para suporte a junções aproximadas e que contempla diversos tipos de erros comuns entre representações distintas de objetos, como erros de edição e supressão de termos. Por considerar para fins de junção objetos onde todos os termos apresentam erros de edição, experimentalmente verificou-se que o $2LM$ aplicado a coleções onde erros de grafia em todos os termos são freqüentes apresenta resultados muito superiores aos demais modelos, e em coleções com poucos erros, sua eficácia é muito próxima do melhor modelo disponível na literatura ($Soft - TFIDF$)

Para os experimentos realizados foram utilizadas duas coleções de publicações: *Biblioteca Digital Brasileira de Computação* (BDBComp) e *DataBase systems and Logic Programming* (DBLP) [4].

Sobre essas coleções, foram avaliados tanto modelos cuja similaridade é calculada segundo distância de edição e distância de conjunto de palavras quanto modelos híbridos que combinam

[4] http://www.dblp.uni-trier.de

essas duas medidas. Os experimentos mostram que, sobre coleções que apresentam muitos erros de grafia, o modelo proposto é mais eficaz que aqueles encontrados na literatura, ainda que esteja associado a um custo maior de computação.

1.1 Trabalhos relacionados

Os trabalhos de Cohen [6, 8] definem um sistema de gerência de dados chamado WHIRL *Word-based Heterogeneous Information Representation Language* que permite a junção baseada na similaridade entre bancos de dados distintos. WHIRL baseia-se no Modelo Vetorial [2, 3] para calcular a similaridade entre os valores de atributos de um banco de dados relacional e assim viabilizar a junção. Para isso, assume que estes valores são *strings* longas representando identificadores de objetos. No caso de uma biblioteca digital, um provável identificador seria a *string* do nome do autor. Para conferir maior escalabilidade à solução proposta, WHIRL adota uma política de poda que consiste em considerar primeiramente os objetos cujos identificadores contenham as palavras mais raras e considerar os demais objetos apenas se necessário. Por exemplo, na junção de "Joao Marcelo Azevedo Arcoverde", WHIRL primeiramente consideraria os objetos cujos identificadores tivessem as palavras mais raras, provavelmente, "Arcoverde", e os demais apenas se necessário, e ordenaria os resultados com base na maior similaridade. Entretanto, WHIRL restringe os resultados da consulta aos identificadores que possuam pelo menos uma palavra exatamente igual às da consulta; dessa forma, não são considerados erros na grafia o que deixa fora do conjunto de respostas ocorrências de "Arcoverdde", por exemplo.

Uma das abordagens possíveis para tratar esse problema é relaxar a restrição de igualdade utilizada normalmente ao comparar-se os argumentos de uma consulta aos atributos dos objetos do banco de dados. Dessa forma, seriam considerados não apenas os *nomes* iguais, mas também aqueles que forem similares segundo alguma métrica de similaridade.

No estudo desenvolvido por Cohen *et al* [4], os autores analisam diversas métricas de similaridade para comparação de duas *strings*, dividindo-as em três grupos:

1. Métricas baseadas em Distância de Edição e similares. Aqui foram classificadas as funções desenvolvidas por Levenshtein, Smith e Waterman [17], Jaro [7] e Winkler [5], por exemplo. Os resultados dos experimentos apresentam Jaro-Winkler e Monge-Elkan como as funções que obtiveram melhores resultados na junção aproximada de bancos de dados heterogêneos;

2. Métricas baseadas em palavras. Nesta categoria são classificadas as soluções baseadas em conjuntos de palavras, como a *similaridade de Jaccard*, que divide o número de palavras comuns entre as *strings* pelo número total de palavras distintas. Entre os modelos baseados em palavras propostos, a similaridade do *cosseno* (Modelo Vetorial), também chamada TF-IDF, é a que apresenta melhores resultados.

3. Modelos Híbridos. São descritas as funções que combinam métricas do primeiro grupo e métricas do segundo grupo, e também é proposto um modelo chamado Soft-TFIDF, o qual obteve os melhores resultados dentre todos os demais.

Embora os resultados com Soft-TFIDF tenham se destacado, esse modelo apenas utiliza as comparações de distância de edição sobre os documentos que possuem pelo menos uma palavra em comum com a consulta; dessa forma, não haveria resultados ao tentar-se comparar "Arcoverdde" com "Arcoverde" e "Joao Marcelo Azevedo Arcoverde", por exemplo.

O modelo proposto nesta dissertação difere do apresentado por Cohen [4] por considerar como resposta não só os documentos que possuam termos em comum com o da consulta, mas também aqueles que possuam apenas termos similares.

Outra situação em que junções por similaridade têm se mostrado interessantes e mobilizado esforços é na integração de documentos XML de diferentes fontes. Nesse contexto, encontra-se o trabalho de Tejada [9], que apresenta um sistema de identificação de objetos, denominado *Active Atlas*, onde é adotada uma abordagem para integração de coleções XML que utiliza o modelo vetorial para determinar a similaridade entre objetos. Enquanto o trabalho de Tejada é limitado à aplicação em objetos de estrutura plana, a abordagem de Dorneles [13] pode ser aplicada a objetos complexos. Dorneles [13] propõe um método para pesquisa em coleções XML que calcula a similaridade considerando os múltiplos níveis da estrutura e os diferentes domínios dos elementos XML. As métricas de similaridade apresentadas foram divididas em dois tipos: MAV *Metrics for Atomic Values* utilizadas em elementos XML atômicos; e MCV (*Metrics for Complex Values*) utilizadas em elementos complexos, estes divididos em duas classes, *tupla* e *coleção*. A proposta é adotar métricas diferentes conforme o domínio dos atributos (*string*, data, número, etc) no caso de objetos atômicos e, no caso de MCV, utilizar uma das métricas propostas para combinar os resultados gerando um valor de similaridade final.

As métricas de similaridade e modelos estudados nesta dissertação, podem ser adotados para

mensurar a similaridade de elementos atômicos XML, contribuindo assim para a apuração da similaridade final em modelos como o proposto por Dorneles [13].

Em todos os casos expostos, é evidente o benefício da adoção de técnicas que permitam relaxar a restrição de igualdade imposta às consultas convencionais, ou seja, funções que permitam, por exemplo, identificar que "Arcoverde" e "Arcoverdde" são similares. O cálculo da diferença entre os caracteres de duas *strings* é chamado de cálculo de distância de edição. Diversas funções foram desenvolvidas para calcular a distância de edição entre *strings*, sendo algumas apresentadas no Capítulo 2.

Entretanto, o custo de cálculo das distâncias de edição é muito alto, e para minimizar o número de cálculos necessários para operações de junção, foram estudadas estruturas métricas de indexação. Uma solução foi proposta por Ciaccia e Patella [14], a M-Tree, uma árvore balanceada dinâmica que indexa objetos usando funções métricas de distância. Traina [18] propõe uma árvore métrica chamada Slim-Tree, também balanceada e dinâmica, cujo algoritmo de divisão de nodos (*split*) proposto contribui para minimizar a intersecção de valores entre os nodos, apresentando experimentalmente melhores resultados que a M-Tree, tanto em termos de número de acessos a disco quanto em termos de número de distâncias calculadas.

Dohnal [19] apresenta um índice dinâmico chamado D-Index, específico para o uso com funções métricas de distância. Ao contrário da organização em árvores, o D-Index apresentou bons resultados em operações de ambientes onde há uma alta taxa de deleção e inclusão de objetos. Outra abordagem de estruturas métricas são as estruturas da família OMNI, propostas por Traina [22] e os algoritmos OMNI Seq+ e OMNI Seq* propostos por Digout e Nascimento [25].

Como resumo dos trabalhos relacionados são apresentadas na Tabela 1.1 as métricas de similaridade e na Tabela 1.2 as estruturas métricas.

Nesta dissertação apresentamos um modelo em dois níveis utilizando uma função métrica de edição associada à M-Tree no primeiro nível e aplicaremos a similaridade do cosseno, com e sem uso do IDF, no segundo nível.

Trabalho	*Características*
WHIRL	Baseia-se no modelo Vetorial para promover junção de bancos de dados distintos
Smith-Waterman	Métrica baseada em distância de edição usada para similaridade entre DNA
Levenshtein	Métrica baseada em distância de edição que considera o número de operações de inserção, deleção e alteração de caracter
Jaro	Métrica baseada em distância de edição que considera os caracteres em comum e a ordem dos mesmos entre duas strings
Jaro-Winkler	Métrica baseada em distância de edição variante de Jaro que considera também o tamanho do maior prefixo em comum entre duas strings
Monge-Elkan	Métrica baseada em distância de edição que aplica uma função secundária de similaridade sobre substrings
Bag of Words	Métrica baseada em conjunto de palavras que considera os termos em comum entre duas strings (intersecção)
Similaridade de Jaccard	Métrica baseada em conjunto de palavras que divide o número de termos em comum entre duas strings (intersecção) pelo número de termos distintos (união)
Modelo Vetorial ou TF-IDF	Métrica baseada em conjunto de palavras que utiliza a similaridade do cosseno ponderada pela importância dos termos na coleção
Soft-TFIDF	Modelo híbrido que associa uma função de distância de edição ao Modelo Vetorial (TF-IDF), considerando para cálculo de similaridade tanto os termos em comum quanto os similares
Active Atlas	Junção de objetos planos XML utilizando o Modelo Vetorial
MAV/MVC	Junção de objetos complexos XML considerando os múltiplos níveis da estrutura e utilizando métricas distintas dependendo do tipo do objeto

Tabela 1.1: Métricas de similaridade

1.2 Organização da dissertação

Esta dissertação está organizada em cinco capítulos. No Capítulo 2 serão apresentados conceitos necessário à compreensão do trabalho desenvolvido. O Capítulo 3 detalha o modelo em dois níveis proposto. O Capítulo 4 descreve os experimentos realizados bem como detalhes sobre as coleções utilizadas nos mesmos. Também serão apresentadas informações sobre o ambiente de desenvolvimento e ferramentas adotadas. Os resultados dos experimentos assim como a análise dos mesmos serão relatados neste capítulo. No Capítulo 5 é apresentado um resumo dos resultados e as conclusões obtidas, além das contribuições do trabalho e sugestões de trabalhos futuros.

Trabalho	*Características*
R-Tree	Primeiro SAM (Spacial Access Method). Árvore métrica estática baseada em retângulos (MBRs). Apresenta problemas de sobreposição (overlap) das regiões de cobertura.
$R^+ - Tree eR^* - Tree$	Variantes R-Tree que minimizam o overlap
vp-tree	Árvore binária estática que utiliza um pivô para guiar a divisão da árvore em bolas métricas (região determinada por centróide e raio de cobertura).
mvp-tree	Árvore binária estática baseada na vp-tree que utiliza vários pivôs.
M-Tree	Árvore métrica, dinâmica e balanceada. Primeira estrutura métrica dinâmica.
Slim-Tree	Árvore métrica, dinâmica e balanceada baseada na M-Tree. Otimiza cálculos de distância e acessos à disco através de novo algoritmo de split e processo denominado *slim down*.
OMNI family	Família de métodos de acesso aplicáveis sobre outras estruturas como a Seqüencial (OSeq), Slim-Tree (OSlimTree) e a R-Tree (ORTree). Reduz a dimensionalidade organizando os índices através de arquivo de coordenadas OMNI. Até 10 vezes mais rápido e até 10 vezes menos cálculos de distância que os métodos isolados.
$OSeq^+ eOSeq^*$	Algoritmos sobre a estrutura OMNI-Sequencial que reorganizam os arquivos de coordenadas OMNI. Até 3 vezes mais rápido que OSeq.
D-Index	Estrutura métrica multi-nível que combina técnica de clustering a estratégia baseada em pivô. Apresentou-se 6 vezes mais rápido que a M-Tree e utiliza metade do espaço em disco.

Tabela 1.2: Estruturas Métricas

Capítulo 2

Conceitos e Terminologia

Neste capítulo serão apresentados os conceitos e terminologia necessários à compreensão do modelo proposto. Inicialmente discutiremos noções de similaridade e distância no contexto de busca, em seguida apresentaremos algumas estruturas de indexação métricas e modelos de junção por similaridade disponíveis na literatura e, por fim, os métodos utilizados para avaliar a eficácia dos modelos de junção por similaridade.

2.1 Similaridade e Distância

As operações de pesquisa em bancos de dados tradicionais são baseadas na igualdade dos valores dos atributos. A partir de um objeto de consulta, é retornado um conjunto de objetos do banco de dados cujos atributos casam com aqueles definidos na consulta. Um exemplo de consulta seria "Quais as movimentações bancárias da conta conta 11613-0, agência 2905-x, Banco do Brasil, no mês de agosto de 2004?".

Entretanto, há casos onde não se pode garantir que os valores de atributos de diferentes objetos que representam o mesmo objeto do mundo real sejam exatamente iguais, como é o caso de bancos de dados multimídia e também de bancos de dados heterogêneos.

Por exemplo, se a consulta "Quais as publicações de Alberto Laender?" fosse submetida à BDBComp, utilizando como critério a igualdade do nome do autor, seriam recuperados apenas 5 dos 16 artigos contidos na coleção, porque os demais estão associados a "Alberto Henrique Frade Laender" e "Alberto H. F. Laender".

Para esses casos, o critério mais adequado é o de similaridade entre os valores dos atribu-

tos. A similaridade entre *nomes* pode ser definida a partir de *funções de dissimilaridade*, ou distância, ou por meio de *funções de similaridade*. Em *funções de similaridade*, quando dois *nomes* comparados forem exatamente iguais, o valor retornado é o valor máximo, normalmente 1.00, e quanto mais distintos forem os *nomes* menor é o valor retornado pela função. O contrário ocorre em *funções de distância*, onde o valor zero representa o casamento exato, ou seja, a igualdade entre os valores comparados, e o valor aumenta a medida que cresce a diferença entre os *nomes*.

2.1.1 Funções de Distância

Antes de definir funções de distâncias é necessário compreender a definição de Espaço Métrico. Um Espaço Métrico é um par $\mathbb{M} = (\mathcal{D}, d)$, onde $\mathcal{D}$ é o domínio do vetor de características (chaves de indexação) e $d()$ é uma função de distância.

Sendo $\mathcal{D}$ o domínio de características do objeto a serem utilizadas para fins de comparação (por exemplo: palavras, coordenadas, cores), uma função de distância é uma função d : $\mathcal{D}^2 \rightarrow \mathbb{R}_0^+$, tal que, para cada objeto O_x, O_y, $O_z \in \mathcal{D}$, sejam satisfeitos os seguintes axiomas:

$$0 < d(O_x, O_y) < \infty, \quad O_x \neq O_y \quad e \quad d(O_x, O_x) = 0 \quad (não \quad negatividade) \tag{2.1}$$

$$d(O_x, O_y) \quad = \quad d(O_y, O_x) \quad (simetria) \tag{2.2}$$

A função d será considerada uma *função métrica de distância* se, além dos axiomas apresentados, também obedecer à desigualdade triangular, dada por:

$$d(O_x, O_z) \quad + \quad d(O_z, O_y) \quad \geq \quad d(O_x, O_y) \tag{2.3}$$

A desigualdade triangular é fundamental em buscas em espaços métricos pois possibilita a economia de cálculos de distância, como pode ser visto no exemplo a seguir.

Suponha uma busca pelo ponto mais próximo do ponto Q em uma coleção composta por 3 pontos, conforme Figura 2.1.

Considere que já temos pré-calculadas as distâncias $d(x, y) = 5.66$, $d(x, z) = 7.14$ e $d(y, z) = 3.60$. Suponha ainda que encontramos x e que sua distância de Q é $d(Q, x) = 2.0$

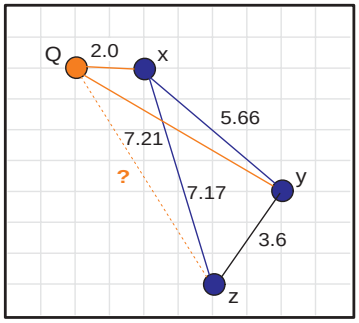

Figura 2.1: Desigualdade Triangular

e depois encontramos y cuja distância de Q é $d(Q, y) = 7.21$. Através do uso da propriedade da desigualdade triangular, não é necessário calcular a distância entre Q e z, como pode ser observado a seguir:

$$\begin{aligned} d(Q, y) &\leq d(Q, z) + d(y, z) \\ d(Q, y) - d(y, z) &\leq d(Q, z) \\ 7.21 - 3.60 &\leq d(Q, z) \\ 3.61 &\leq d(Q, z) \end{aligned} \tag{2.4}$$

Como se deseja obter o ponto mais próximo e já é sabido que $d(Q, x) = 2.0$, não há necessidade de calcular a distância entre Q e z pois, através da desigualdade triangular, já constatou-se que $d(Q, z)$ será maior ou igual a 3.61. Dessa forma, foi possível evitar um cálculo de distância.

Apresentaremos a seguir alguns exemplos de funções métricas de distância :

Exemplo 1 *Distância de* Levenshtein*, também chamada de* LEdit*, calcula o menor número de operações de edição (inserção, deleção, substituição) necessárias para transformar uma string s em outra string t.*

Por exemplo, seja s = "Nascimento" e t = "Nasimento", a $d_E(s,t)$ é igual a 1, pois basta uma operação de inclusão do caractere 'c' para que t seja igual a s.

Exemplo 2 *Métricas de* Minkowski *(L_p), são métricas sobre espaços vetoriais, definidas como $L_p(v_x, v_y) = (\sum_{j=1}^{n} (|v_x[j] - v_y[j]|^p)^{\frac{1}{p}}$ $(p \geq 1)$. A Distância Euclideana (L_2) é um caso específico da métrica de Minkowski, onde $p = 2$; outro caso é a distância de Manhattan - ou "city block" - (L_1), com $p = 1$.*

Exemplo 3 *A similaridade de Jaccard é uma função onde cada string é vista como um conjunto de palavras e a distância entre duas strings s e t é dada por :*

$$Jaccard(s,t) = \frac{|\ s \cap t\ |}{|\ s \cup t\ |} \tag{2.5}$$

2.1.2 Consultas por similaridade

Há dois tipos básicos de consultas por similaridade: consulta por abrangência (*Range Query*) e consulta aos vizinhos mais próximos (*Nearest Neighbor Query*) [15].

Range Query. Em uma *Range Query*, a partir de uma coleção de objetos S, de um objeto de consulta Q e uma distância de busca máxima, também chamada de raio de abrangência, rQ, a pesquisa $RangeQuery(Q,rQ,S)$ selecionará todos os objetos O em S tal que $d(O,Q) \leq rQ$, ou seja, todos os objetos cuja distância de Q não exceda rQ. Por exemplo: "Quais os autores da BDBComp com distância de edição (*Levenshtein*) até 2 em relação à "Ademir Alvarenga de Oliveira"?" Neste caso, o objeto de consulta é "Ademir Alvarenga de Oliveira", o domínio S é o conjunto de autores da BDBComp e o raio de busca (distância máxima) são 2 operações de edição. Uma resposta para essa consulta seria "Ademir Alvarega de Oliveira".

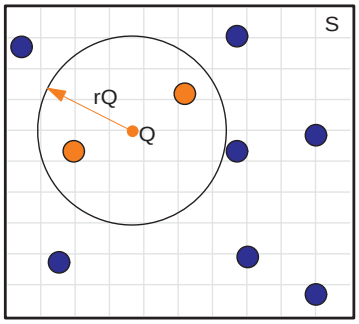

Figura 2.2: Range Query

Nearest Neighbor Query. Dada uma coleção S, um valor de consulta $Q \in \mathcal{D}$, e um valor inteiro $k \geq 1$, uma pesquisa de vizinhos mais próximos $\mathrm{NN}(Q,k,S)$ seleciona os k objetos em S que estão mais próximos de Q. Havendo empate de distâncias, pode-se criar uma lista de empates ou apenas selecionar o número suficiente de objetos. Um exemplo seria "Selecione os 2 autores da BDBComp mais próximos de "Agma Juci Traina" segundo a *similaridade de Jaccard*. Essa consulta é uma busca 2-NN tendo como referencial o objeto "Agma Juci Traina". Como resposta teríamos "Agma Juci Machado Traina", cuja *similaridade de Jaccard* é $\frac{3}{4}$, e "Agma Traina", com *similaridade de Jaccard* igual a $\frac{2}{4}$.

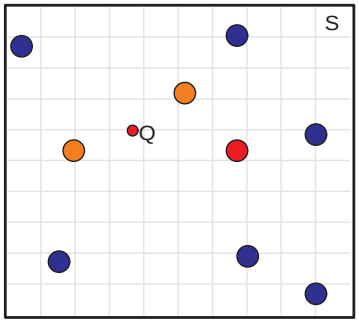

Figura 2.3: 3-Nearest Neighbor Query

2.2 Estruturas de Indexação Métricas

As estruturas de indexação são ferramentas essenciais para SGBDs, pois tornam eficientes o armazenamento e recuperação de grandes volumes de dados. Isto é particularmente verdadeiro para o caso de consultas aproximadas onde o número de objetos a serem considerados como candidatos a respostas é geralmente grande.

Apresentaremos a seguir algumas estruturas de indexação métricas disponíveis na literatura.

Em 1984, Guttman [29] apresentou o conceito de R-Trees, as primeiras estruturas denominadas *Métodos de Acesso Espaciais - SAM* (Spatial Access Methods).

Para organizar e particionar as buscas no espaço, estes tipos de árvore consideram a distância relativa entre objetos, em lugar de sua posição absoluta no espaço multidimensional; e apenas requerem que a função usada para medir a distância entre objetos seja uma função métrica. Assim, a desigualdade triangular pode ser aplicada para restringir o espaço de pesquisa. Entretanto, como pode ser visto na Figura 2.4, a R-Tree apresenta sobreposição ou *overlap* das regiões de cobertura dos nodos; dessa forma, para recuperar o ponto X dois diferentes caminhos precisam ser seguidos $A \rightarrow G$ e $C \rightarrow O$.

Embora tenham sido implementadas diversas variações de estruturas baseadas nas R-Trees visando minimizar os problemas de *overlap*, por exemplo, $R^+ - Tree$ e $R^* - Tree$, todas mostraram deficiência em trabalhar com espaços de alta dimensionalidade. [30, 31]

Suprindo essa deficiência, surgiram os *Métodos de Acesso Métricos - MAM* (Metric Access Methods) , que são estruturas de dados para espaços métricos que abrangem dados espaciais de dimensões mais altas.

Burkhard e Keller, em [24], apresentaram três maneiras de realizar busca recursiva em estru-

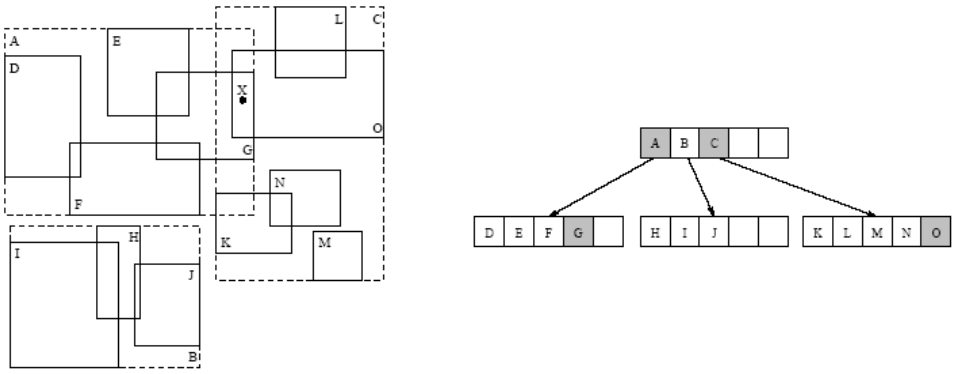

Figura 2.4: R-Tree

turas em forma de árvore usando pivôs para direcionar a busca através dos nodos. Yianilos [21] propõe uma árvore métrica onde um elemento por nodo é escolhido como pivô e guia a divisão da árvore em *bolas métricas*, onde uma *bola métrica* é uma região do domínio métrico determinada por um centróide e um raio de cobertura. Esses pivôs foram denominado *vantage-point*, e a árvore recebeu o nome de *vp-tree*, ou *vantage point tree*. Na Figura 2.4, é apresentado um nodo raiz de uma vp-tree e o particionamento das regiões do domínio. A vp-tree apresentada tem 3 vias, ou seja, um nodo interno possui apontadores para 3 sub-árvores.

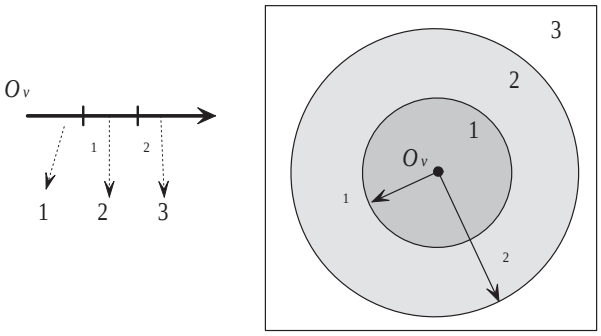

Figura 2.5: vp-tree

Baseando-se na *vp-tree*, Bozkaya e Ozsoyoglu , propuseram a *mvp-Tree* (*multi-vantage point tree*) que escolhe n pontos de referências. Essas estruturas métricas, contudo, eram estáticas, ou seja, não permitiam inserções eu exclusões posteriores à sua construção.

Apesar da eficácia das *vp-trees* e das *mvp-trees* ter sido claramente demonstrada, o fato de serem intrinsicamente estáticas limitava sua aplicabilidade a ambientes de banco de dados dinâmicos. Além disso, contrariamente aos SAMs, algumas árvores métricas apenas tentaram reduzir o número de distâncias computacionais requeridas para responder a uma pesquisa, não

se importando com o custo de I/O. [14]

A primeira estrutura métrica dinâmica, a M-Tree, apresentada por Ciaccia [14] em 1997, é uma árvore métrica paginada, dinâmica e balanceada com nodos de tamanho fixo que são mapeados para páginas em disco e que armazena os objetos em nodos folhas. Cada nodo corresponde a uma região no espaço métrico. A M-Tree é composta por dois tipos de nodos:

- O *nodo interno* (*route object*) armazena o objeto pivô do nodo, a distância entre o pivô e o nodo pai, o raio de cobertura da região indexada e um vetor de ponteiros para suas sub-árvores;
- O *nodo folha* (*data objects*) armazena os identificadores dos objetos e o vetor de características que estão sendo utilizadas para a indexação do conjunto de dados.

Um exemplo de M-Tree utilizando como função métrica a função de distância de edição de *Levenshtein*, pode ser vista na Figura 2.6.

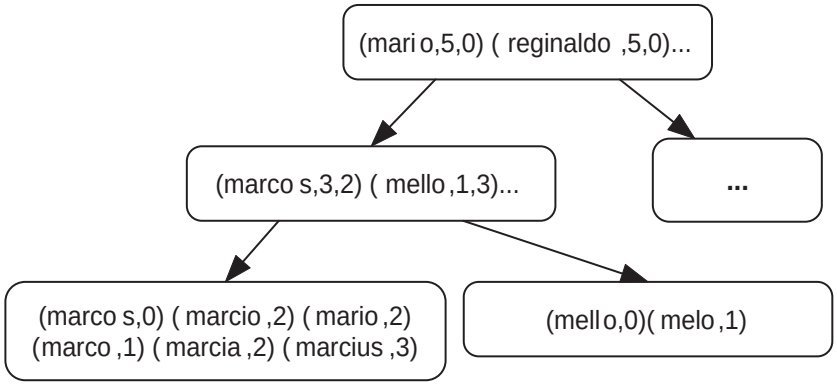

Figura 2.6: M-Tree

Seguindo o conceito da *M-Tree*, foi proposta por Traina em 2000 [18], a *Slim-Tree*, uma árvore métrica dinâmica que reduz o *overlap* entre os nodos e possui um processo chamado de *slim-down*, realizado após a inserção de um novo nodo, que torna a árvore mais estreita e rápida.

Os resultados obtidos experimentalmente por Traina demonstraram que a Slim-Tree apresenta vantagens quando comparada à M-Tree, tanto em relação à acessos à disco quanto a cálculos de distância. O diferencial da Slim-Tree é um algoritmo de *split* baseado no utilizado pela *Árvore Espalhada Mínima* (Minimal Spanning Tree) além do processo de *slim-down*.

Uma família de métodos de acesso denominada *OMNI Family* foi apresentada por Traina [22]. A idéia principal é eleger um conjunto de objetos que sirvam de pivô para guiar a busca,

chamados de *focos globais* (*Omni-foci base*), e promover a redução da dimensionalidade $\mathcal{D}$ do espaço mapeando os objetos em uma dimensão mais baixa d, determinada pelo número de focos. As distâncias entre os focos e os demais objetos do conjunto, também chamadas *Coordenadas OMNI* são pré-computadas, agilizando a pesquisa. Quando um objeto é inserido no conjunto de dados, as coordenadas Omni são avaliadas e armazenadas.

Esse método de pode ser aplicado sobre uma segunda estrutura de indexação, por exemplo, no trabalho realizado em [22], os autores desenvolveram os algoritmos Omni-Sequencial, OmniB-Tree e OmniR-Tree sobre as estruturas *sequencial*, B-Tree e R-Tree respectivamente. Resultados dos experimentos realizados comparando os métodos de indexação Omni-Sequencial e OmniR-Tree com *sequencial scan*, R-Tree e Slim-Tree, mostraram que a família OMNI chega a ser até 10 vezes mais rápida e utiliza até 10 vezes menos cálculos de distância e acessos a disco que os demais métodos.

Uma proposta para reestruturação da estrutura OMNI foi apresentada por Digout *et al* [25], que introduz dois métodos de acesso denominados *Omni Sequential*$^+$ (OSeq$^+$) e *Omni Sequential** (OSeq*). O método OSeq$^+$ reduz o tempo necessário à realização de consultas por similaridade através da ordenação do conjunto de coordenadas Omni segundo o objeto foco e a partir da reestruturação do arquivo.

O algoritmo OSeq*, construído a partir do OSeq$^+$, escolhe mais focos que os anteriores e relaxa a redução da dimensionalidade obtendo D' $(d < D' < D)$ dimensões. Entretanto, durante a consulta, são escolhidos apenas os focos que melhor promovam a poda. Os experimentos com OSeq$^+$ e OSeq* realizados com conjuntos até 256 dimensões, apresentaram um processamento até 3 vezes mais rápido quando comparado ao OSeq proposto em [22].

Em 2003, Dohnal [19] propôs uma estrutura métrica multi-nível denominada *D-Index*, que combina uma nova técnica de *clustering* à estratégia baseada em pivô e tem os objetos armazenados em buckets com acesso direto para evitar dependências de buckets hierárquicos. Por exemplo, para *range queries* até ρ, onde ρ é um número real restrito à $0 \leq \rho \leq d^+$, sendo d^+ é a distância máxima entre dois objetos, no máximo um bucket precisa ser acessado em cada nível além do bucket de exclusão.

O D-Index particiona os dados em 2^m subconjuntos através de uma função ρ split, onde m é a ordem da função, ou seja, em uma função de primeira ordem, são criados três subconjuntos, onde os dois primeiros são chamados *separable sets*, por serem separados por uma distância 2ρ,

e o último *exclusion set*.

No primeiro nível a função de split separa os objetos de toda a coleção e para qualquer outro nível, objetos mapeados para o *exclusion bucket* do nível precedente são candidatos para armazenamento em *separable buckets* deste nível. O *exclusion bucket* do último nível é o bucket de exclusão de todo o conjunto.

Uma função $\rho - split$ de primeira ordem é uma função $bps^{\rho}(x, x_v)$ que usa um objeto $x_v \in \mathcal{D}$ e uma distância média d_m para particionar o arquivo de dados em três subconjuntos $BPS^{1,\rho}_{[0]}$, $BPS^{1,\rho}_{[1]}$ e $BPS^{1,\rho}_{[2]}$ - o resultado é a função seguinte que retorna o índice para o conjunto ao qual o objeto x pertence.

$$bps^{1,\rho} = \begin{cases} 0 & \text{se} \quad d(x, x_v) \leq d_m - \rho \\ 1 & \text{se} \quad d(x, x_v) > d_m + \rho \\ - & \text{para para o restante} \end{cases} \tag{2.6}$$

Na Figura 2.7 é apresentada uma função ρ-split de primeira ordem.

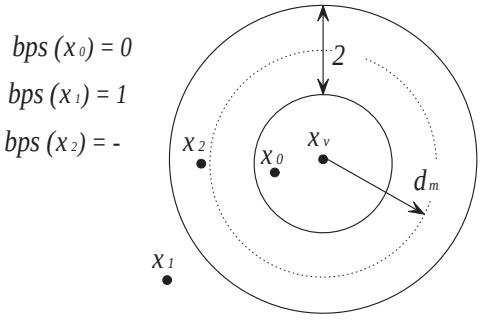

Figura 2.7: D-Index

A função ρ-split pode ser generalizada através da concatenação de n funções ρ-split de primeira ordem, com o propósito de se obter uma função ρ-split de ordem n.

O D-Index se mostrou muito eficiente para consultas realizadas para objetos muito próximos, ou seja, quando utiliza-se um baixo raio de abrangência. Através de experimentações, Dohnal observou que o D-Index é 6 vezes mais rápido que M-Tree e usa metade do espaço, o que representa apenas 20% mais espaço que organizações seqüenciais.

2.2.1 Os efeitos da dimensionalidade (Dimensionality Curse)

Quando a dimensionalidade do espaço cresce, ou seja, a quantidade de características consideradas do objeto aumenta, na maioria das vezes, a diferença das distâncias entre o objeto mais

próximo e o objeto mais distante se torna desprezível. Essa situação, também chamada *Dimensionality Curse*, dificulta o processo de indexação de dados com alta dimensionalidade, pois inviabiliza a poda por desigualdade triangular. O problema da *Dimensionality Curse* é definido pelo teorema de Beyer *et al*, [10], apresentado seguir

Seja d a dimensionalidade de um dado espaço de dados e X_d um ponto nesse espaço, se

$$lim_{d\to\infty} \quad var(\frac{dist_d(X_d)}{E[dist_d(X_d)]}) = 0, \quad então: \quad \frac{D_{max_d} - D_{min_d}}{D_{min_d}} \overrightarrow{p} 0. \tag{2.7}$$

onde N é o número de pontos de dados, D_{min_d} e D_{max_d} as distância mais próxima e mais distante de N pontos em relação à origem, $E[x]$ e $var[X]$ o valor e a variância esperados da variável randômica X.

Para melhor ilustrar o problema de dimensionalidade em índices, considere o uso de árvores métricas, que indexam hierarquicamente os dados segundo o raio de cobertura, por exemplo a *M-Tree*, quando uma página em memória secundária contém objetos próximos e também objetos distantes, o raio de cobertura dos nodos internos aumenta, diminuindo consideravelmente a poda por desigualdade triangular. Nesse caso, onde a redução do número de acesso a disco promovida pela poda é muitas vezes inferior a 20%, uma busca seqüencial se tornaria mais rápida .

2.3 Junção por Similaridade

Uma junção por similaridade é uma pesquisa que combina objetos de dois subconjuntos de D em um conjunto tal que as condições de similaridade sejam satisfeitas.

As condições de similaridade entre dois objetos são definidas de acordo com uma função métrica de distância d. Formalmente, a junção de similaridade $X \bowtie^{sim} Y$ entre dois conjuntos finitos $X = \{x_1, ..., x_N\}$ e $Y = \{y_1, ..., y_M\}$ ($X \subseteq \mathcal{D}$ e $Y \subseteq \mathcal{D}$) é definida como um conjunto de pares $X \bowtie^{sim} Y = \{(x_i, y_j) \| d(x_i, y_j) \leq \mu \}$ onde μ é um número real tal que $0 \leq \mu^+$. Se os conjuntos X e Y forem os mesmos, trata-se de uma *self-join* ou auto-junção.

2.3.1 Modelos para junção por similaridade

2.3.1.1 O uso de Distância de Palavra

Uma vez que duas *strings* podem ser consideradas conjuntos de palavras, métricas baseadas na comparação entre as palavras das duas *strings* também podem ser consideradas para fins de junção por similaridade.

Golgher *et al* apresentam uma proposta de mensurar a similaridade entre duas *strings* s e t a partir do número de palavras em comum entre elas, ou seja, $\mid s \bigcap t \mid$ [27]. Essa medida será referenciada aqui como *Bag of Words*.

Outra métrica bastante encontrada na literatura é a *similaridade de Jaccard*, que é uma função baseada na comparação de conjuntos de palavras. A distância entre as *strings* s e t segundo *Jaccard* é dada por :

$$\text{Jaccard(s,t)} = \frac{\mid s \cap t \mid}{\mid s \cup t \mid} \tag{2.8}$$

O modelo vetorial, também chamado de similaridade do Cosseno ou TF-IDF, foi desenvolvido por Gerard Salton, para utilização em um sistema de recuperação de informação chamado SMART [3]. Embora a proposta do modelo vetorial tenha surgido em 1968, tendo como objetivo a solução de problemas de busca em documentos, esse modelo é utilizado até hoje por apresentar bons resultados em vários problemas de RI pois leva em consideração o casamento parcial e a proximidade dos documentos em relação aos termos da consulta.

No modelo TF-IDF, cada termo da *string* é considerado uma coordenada dimensional, podendo ser disposto em um espaço euclideano de n dimensões (onde n é o número de termos) e a posição do objeto em cada dimensão é dada pelo seu peso. Na Figura 2.8 é representada a consulta Q = "Agma Juci Traina" e o objeto O_1 = "Agma Traina".

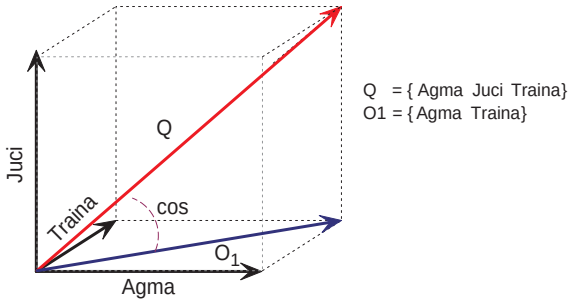

Figura 2.8: Exemplo de consulta usando o Modelo Vetorial

Existem várias formas de cálculo do peso do termo na coleção. A mais freqüentemente utilizada é baseada no modelo de Salton e Buckey [2], onde tenta-se balancear características em comum nos documentos (intra-document : TF term frequency) e características de distinção entre eles (inter-document: IDF inverse document frequency).

$$IDF(t) = log(\frac{N}{n_t}) \tag{2.9}$$

Onde N é número total de documentos de uma coleção e n_t é o número de documentos onde a palavra t ocorreu.

Considerando uma coleção composta por 100 *nomes* de autores ($N = 100$), e sabendo-se que o termo "Agma" ocorre em 3 *nomes* de autores distintos ($n_t = 3$), o IDF("Agma") = $\log(\frac{100}{3})$, ou seja, é aproximadamente 1.5228. Suponha também que o termo "Traina" esteja presente em 10 *nomes* da mesma coleção; nesse caso, o IDF("Traina") seria igual a 1.0000, sendo menor que o IDF de "Agma" que é um termo menos freqüente.

Para determinar as coordenadas do *nome* d no eixo t, utiliza-se a seguinte equação:

$$w(d,t) = TF(d,t) \cdot IDF(t) \tag{2.10}$$

onde TF é a freqüência do termo no documento.

Considerando o *nome* de autor d = "Agma Juci Traina", e os IDF("Agma")=1.5228, IDF("Juci")=2.0000 e o IDF("Traina")=1.0000, pode-se calcular a importância (w(d,t)) dos termos para o documento da seguinte forma: $w(d, "Agma'') = 1.5228$, $w(d, "Juci'') = 2.0000$ e $w(d, "Traina'') = 1.0000$.

As distâncias entre um *nome* e outro indicam seu grau de similaridade, ou seja, *nomes* que possuem os mesmos termos acabam sendo colocados em uma mesma região do espaço e, em teoria, tratam de autores similares.

O objeto de consulta também é representado por um vetor. Dessa forma, os vetores dos *nomes* podem ser comparados com o vetor da consulta e o grau de similaridade entre cada um deles pode ser identificado. Dessa forma, o *nome* "Agma Traina" estaria mais próximo de "Agma Juci Traina" que de "Caetano Traina".

Os *nomes* de autores mais similares (mais próximos no espaço) à consulta são considerados relevantes para o usuário e retornados como resposta para ela. A similaridade dos *nomes* é dada

pela equação abaixo:

$$sim(\overrightarrow{d}, \overrightarrow{q}) = \frac{\overrightarrow{d} \cdot \overrightarrow{q}}{|\overrightarrow{d}| \cdot |\overrightarrow{q}|} = \frac{\sum_{i=1}^{t} w_{d,i} \cdot w_{q,i}}{\sqrt{\sum_{i=1}^{t} w_{d,i}^2} \cdot \sqrt{\sum_{i=1}^{t} w_{q,i}^2}} \tag{2.11}$$

Embora o modelo TF-IDF seja largamente utilizado, ele não trata erros de grafia. Por exemplo, uma consulta à autora "Agma Traina" a partir de "Agna Traina" não consideraria para cálculo de similaridade o termo "Agna", pois o mesmo apresenta um erro de escrita. Portanto, nada garante que a autora "Agma Traina" apresente maior similaridade em relação à consulta que o autor "Caetano Traina". Para tratar esse problema, são adotadas métricas de distância de edição, apresentadas a seguir.

2.3.1.2 O uso de Distância de Edição e Similares

Ao comparar duas *strings*, por exemplo "Daiane" e "Diane", é possível calcular a similaridade entre elas através de uma função de distância de edição. Funções de distância de edição mapeiam um par de *strings* para um número real r, onde, quanto menor o valor de r, maior a similaridade entre as duas *strings*.

Atualmente é possível encontrar na literatura várias funções de distância utilizadas para a tarefa de junção por similaridade de *nomes* de objetos. Serão apresentadas a seguir, as funções de distância de *strings* mais comumente utilizadas.

A *Distância de Levenshtein*, também chamada de *LEdit*, é a distância mais conhecida [4]. Ela é calculada atribuindo-se o custo 1 para cada operação de edição necessária para transformar uma *string* em outra; onde uma operação de edição pode ser: inclusão, exclusão ou alteração de um caracter. Por exemplo, a distância de *Levenshtein* entre as *strings* s = "Daiane dos Santos" e t = "Diani dos Santos" é igual a 2; pois, para transformar a *string* s em t, é realizada uma inclusão do caractere 'a' na posição 2 de t e uma alteração do segundo caractere 'i' de t pelo caractere 'e'.

Em 1981, Smith e Waterman publicaram um trabalho no qual utilizaram uma função de distância para alinhar seqüencias de DNA [17]. A função de distância Smith-Waterman, possui dois parâmetros principais: custos associados a cada letra do alfabeto e um valor relacionado ao custo de gap (inserção ou deleção de caractere). Posteriormente, essa função foi utilizada por Cohen em experimentos de junção por similaridade [4] .

Jaro, em 1985 [7], apresentou uma função métrica de distância, cujas variantes apresentam bons resultados na tarefa de junção por similaridade de bancos de dados [4]. Esta função é baseada no número e na ordem de caracteres comuns entre duas *strings*. Dadas as *strings* s = $a_1...a_k$ e t = $b_1...b_L$, um caracter a_i em s é definido como *em comum com* t se houver um $b_j = a_i$ em t tal que $i - H \leq j \leq i + H$, onde $H = \frac{min(|s|,|t|)}{2}$. Seja $s' = a'_1....a'_k$ os caracteres em s em comum com t (na mesma ordem em que aparecem em s) e seja t'=b'...b'$_L$ análogo em relação a s; a transposição para $s' \cdot t'$ é a posição i tal que $a'_i \neq b'_i$. Seja $T_{s',t'}$ metade do número de transposições para s' e t'. A função de similaridade métrica de Jaro para s e t é dada por:

$$Jaro(s,t) = \frac{1}{3} \cdot (\frac{|s'|}{|s|} + \frac{|t'|}{|t|} + (\frac{|s'| - T_{s',t'}}{|s'|})) \tag{2.12}$$

Na Figura 2.9, é apresentada a comparação entre duas *strings* utilizando a função métrica de Jaro. As entradas com fundo cinza representam a diagonal principal e cada caracter 1 apresentado em negrito é considerado *em comum* entre as duas *strings*.

	A	G	N	A
A	1	0	0	0
G	0	1	0	0
M	0	0	0	0
A	0	0	0	1

Figura 2.9: Função de similaridade de Jaro

Dessa forma, tem-se: $Jaro(\text{“}Agma'', \text{“}Agna'') = \frac{1}{3} \cdot (\frac{3}{4} + \frac{3}{4} + \frac{(3-0)}{3}) = 0.8333$.

Uma variante da função de Jaro foi apresentada por Winkler [5], que propôs o uso do tamanho P do maior prefixo em comum entre as *strings* s e t. Seja P'=max(P,3), a função Jaro-Winkler é definida como:

$$Jaro - Winkler(s,t) = Jaro(s,t) + \frac{P'}{10} \cdot (1 - Jaro(s,t)) \tag{2.13}$$

Aplicando-se Jaro-Winkler, sobre o exemplo anterior, tem-se $Jaro - Winkler(\text{“}Agma'', \text{“}Agna'') = 0.8333 + 0,0556 = 0.8889$. De acordo com Cohen [4], os melhores resultados de junções de similaridade obtidos em seu trabalho foram alcançados através da associação da função de Jaro-Winkler com a similaridade do *cosseno*. Portanto, os experimentos aqui apresentados utilizarão a função de *Levenshtein*, por ser a mais comumente

utilizada, e de Jaro-Winkler por ter obtido melhores resultados em junções de *nomes*.

Modelos que consideram tanto as palavras quanto a distância de edição entre os termos são chamados de modelos híbridos e serão apresentadas a seguir.

2.3.1.3 Usando Modelos Híbridos

Para conjugar as vantagens dos métodos baseados em distância de palavras com os métodos baseados em distância de edição, foram propostos modelos para junção aproximada que levam em conta os dois tipos de métricas.

Monge e Elkan [16] propuseram uma abordagem para comparação de duas *strings* longas s e t. As *strings* são divididas em substrings $s = a_1...a_K$ e $t = b_1...b_L$ e é definida uma função secundária de distância sim'.

$$Monge - Elkan(S,T) = \frac{1}{K}\sum_{i=1}^{K} max_{j=1}^{L} sim'(a_i, b_j). \tag{2.14}$$

Buscando combinar distâncias de edição com distâncias de palavra, Cohen baseou-se no modelo de Monge e Elkan para propor uma função de distância chamada SoftTFIDF [4]. A SoftTFIDF é uma implementação do TF-IDF, ou Modelo Vetorial, que considera tanto os termos que façam parte da intersecção entre as duas *strings* S e T quanto os termos similares. Para isso, é definida uma função secundária chamada $CLOSE(\Theta, S, T)$ que retorna o conjunto de palavras $w \in S$ tal que exista alguma palavra $v \in T$ onde a $dist'(w, v) > \Theta$. Como função secundária foram utilizadas funções de distância, tais como Jaro e Jaro-Winkler.

Dessa forma, SoftTFIDF pode ser definida como:

$$SoftTFIDF(S,T) = \sum_{w \in CLOSE(\Theta,S,T)} V(w,S) \cdot V(w,T) \cdot D(w,T) \tag{2.15}$$

onde $V(w,S) = V'(w,S)/\sqrt{\sum_{w'} V'(w,S)^2}$ e $V'(w,S) = log(TF_{w,S}+1) \cdot log(IDF_w)$ e D(w,T) = $\max_{v \in T} dist(w,v)$.

Assim, ao comparar duas *strings*, por exemplo, S = "Agna Traina" e T = "Agma Juci Traina", utilizando como função secundária de similaridade a distância de Jaro-Winkler com $\Theta = 0.9$, além da palavra "Traina", comum às duas *strings*, a palavra "Agna" em S cuja distância de "Agma" em T, segundo Jaro-Winkler, é 0.8889, também seria considerada. O fator

$D(w, T)$ reduz a similaridade retornada conforme a distância de edição do termo; dessa forma, ao comparar-se as *strings* "Agma Traina" com "Agna Traina" e "Agma Traina" é garantido que "Agma Traina" tenha uma similaridade maior.

2.3.2 Limiares (thresholds)

Para definir os objetos que são similares e assim promover a junção entre duas coleções, é necessário primeiramente definir os limiares ou *thresholds* que serão adotados, ou seja, determinar a tolerância a erros admitida.

No caso de métricas baseadas apenas em caracteres ou métricas baseadas apenas em conjunto de palavras, basta a definição de um limiar. Quando dois *nomes* são comparados com base em uma função de similaridade, quanto maior esse limiar, menor a tolerância a erros. Um limiar muito alto pode deixar fora da junção representações distintas do mesmo objeto do mundo real e um limiar muito baixo pode realizar a junção de objetos distintos. A escolha do limiar é fator fundamental para a eficácia do processo de junção por similaridade; essa escolha depende da qualidade da coleção, da necessidade de recuperação de todos os relevantes e do nível de erro admitido. Quanto menor o limiar de similaridade, mais objetos serão recuperados e a tendência é recuperar também mais objetos distintos.

Em métricas híbridas, há necessidade de definição de dois limiares, um a ser utilizado sobre a métrica baseada em caracteres, e outro sobre a métrica baseada em conjunto de palavras.

2.4 Métodos de avaliação

Assim como em sistemas de RI, os modelos propostos para avaliação de junções por similaridade utilizam as métricas de Precisão e Revocação (Precision Recall) e Média Harmônica (F1). Revisamos estas métricas a seguir.

2.4.1 Precisão e Revocação

Precisão e Revocação são estimativas de qualidade bastante utilizadas para avaliação de respostas em modelos que utilizam técnicas de Recuperação de Informação. Tendo como referência uma base de documentos relevantes previamente preparada R, analisou-se o conjunto de resultados retornados pelos modelos segundo a precisão e revocação, cujos conceitos seguem:

- *Precisão* é a fração das respostas relevantes retornadas como resultado N, comparando-se com a base de relevantes R [1].

  A *Precisão* pode ser definida pela Equação 2.16:

$$\mathrm{p} = \frac{\mid N \cap R \mid}{\mid R \mid} \tag{2.16}$$

Revocação é a fração de todas as respostas relevantes que foram retornadas como resultado; ou seja, representa a aproximação do total de relevantes definido em R. Pode ser definida como a fração de documentos relevantes que foi recuperada pelo sistema [1]. A Equação 2.17 a seguir, define a revocação:

$$\mathrm{r} = \frac{\mid N \cap R \mid}{\mid N \mid} \tag{2.17}$$

2.4.2 Medida F_1

Para representar resultados de Precisão e Revocação através de um único valor de medida de qualidade [1], pode-se utilizar a *média harmônica* ou *medida* F_1. A medida F_1 pode ser definida como:

$$F_1 = \frac{(\alpha^2 + 1)pr}{\alpha^2 p + r} \tag{2.18}$$

onde α indica o peso dos valores de precisão e revocação. Quando considerado $\alpha = 1$, a medida F_1 pode ser reescrita como:

$$F_1 = \frac{2pr}{p + r} \tag{2.19}$$

A média harmônica retorna valores no intervalo $[0, 1]$, onde, 0 significa que nenhum documento foi recuperado e 1 que o sistema recuperou todos os relevantes com precisão máxima. Dessa forma, $F_1 = 1$ representa o melhor resultado da combinação de precisão e revocação.

Capítulo 3

O Modelo 2LM - Two-level Matching

Neste Capítulo apresentaremos o método de junção denominado $2LM$, *Two-level Matching*. Na Seção 3.1 serão apresentados os erros mais comumente encontrados em representações distintas de identificadores de objetos. Na Seção 3.2 e 3.3 serão apresentadas as métricas baseadas em palavras e em distância de edição, respectivamente; sendo ressaltadas as limitações de ambas no contexto de junção por similaridade. Na Seção 3.4 será abordada a questão da dinamicidade das coleções de dados e, finalmente na Seção 3.5, mostraremos que através da combinação dos dois tipos de métricas é possível obter uma solução adequada para o problema de junção por similaridade de *nomes* de entidades.

3.1 Erros mais freqüentes

Ao tentar associar *nomes* distintos que representam a mesma entidade do mundo real, com base na comparação dos seus valores, diversos tipos de diferenças sintáticas são encontrados. A literatura atual e a análise de diversas coleções nos apresentam os erros mais comuns, que são descritos a seguir.

Um dos principais problemas observados é a ocorrência de supressão ou abreviação de palavras, como é a situação da autora "Agma Juci Machado Traina" que pode ser encontrada na BDBComp tanto como "Agma Traina", onde houve supressão, quanto como "Agma J. M. Traina", onde ocorreram abreviações. No trabalho desenvolvido por Tejada [9] são definidas *transformações unárias e n-árias* de *strings* dentre as quais a *Abreviation* que visa converter termos com abreviações conhecidas em sua forma completa, tais como "Av." em "Avenue"

ou “3rd” em “third”. Ressalta-se que a transformação unária *Abreviation* depende do prévio conhecimento do idioma do conteúdo que se deseja expandir. Também é definida a *Computed Abbreviation* que verifica se uma palavra é um subconjunto de outra, como “Blvd” e “Boulevard”. Entretanto, quando trata-se de um domínio mais extenso e imprevisível, como o de nomes de pessoas, realizar tais transformações não parece ser aplicável.

Outra situação observada é a existência de erros de grafia nos *nomes*, normalmente devido a falhas no processo de digitação ou de OCR. Um exemplo desse tipo de erro pode ser observado na BDBComp onde encontramos o mesmo autor com os *nomes* “Adenilso Simão” e “Adenilson Simão”.

Além dos erros de grafia, supressões e abreviações de *nomes*, quando o domínio envolve nomes de pessoas, pode haver a inversão da ordem das palavras, como no caso de apresentar o sobrenome primeiro. Por exemplo, “Alberto Laender” e “Laender Alberto” representam a mesma entidade do mundo real.

Estes tipos de erro são os mais comumente considerados na literatura [4, 9], e impedem que a comparação entre *nomes* de bases distintas seja feita diretamente, através da igualdade.

3.2 Métricas baseadas em Palavras

Uma abordagem para junção por aproximação seria a utilização de funções baseadas na similaridade de conjunto de palavras, aqui denominadas *funções de token*, como a *similaridade de Jaccard* ou o *Modelo Vetorial* [3]. Essas funções tratam os *nomes* como conjuntos de palavras, ignorando a ordem em que as palavras se apresentam na *string*. Embora algumas dessas funções apresentem bons resultados em diversos problemas de RI, todas apresentam limitações que comprometem a eficácia do processo de junção por similaridade, pois realizam comparações exatas, não admitindo erros de edição.

Para ilustrar as deficiências encontradas no uso de métricas baseadas somente em palavras, considere a string str_q = “Antônio Casanova” e uma coleção C contendo diversos autores, dentre os quais: str_1 = “Antônio Silva”, str_2 = “Antônnio Casanova”, str_3 =“A Cassanova” e str_4 =“Pedro Casanova”. Ao utilizar-se uma função métrica de similaridade baseada em palavra, como, por exemplo, a *similaridade de Jaccard*, as strings str_1 e str_2 apresentariam o mesmo valor de distância em relação à str_q, pois :

$$Jacc(str_q, str_1) = \frac{|\{Antônio\}|}{|\{Antônio, Casanova, Silva\}|} = \frac{1}{3} \tag{3.1}$$

e

$$Jacc(str_q, str_2) = \frac{|\{Casanova\}|}{\{Antônio, Casanova, Antônnio\}} = \frac{1}{3} \tag{3.2}$$

Embora str_2 ("Antônnio Casanova") apresente claramente maior semelhança com o autor da consulta que str_1 ("Antônio Silva"), devido às duas strings possuírem 1 termo em comum com str_q ("Antônio Casanova") a similaridade de ambas, segundo Jaccard, é a mesma. Isso ocorre porque a *similaridade de Jaccard* não considera o IDF das palavras mas apenas o número de palavras em comum com a consulta, dessa forma, tanto "Antônio" quanto "Casanova" tem o peso 1 para fins de cálculo da similaridade.

No entanto, aplicando-se sobre o mesmo exemplo o *Modelo Vetorial*, onde os termos mais comuns possuem IDF menor que os mais raros, é muito provável que "Antônio", por tratar-se de um nome muito freqüente, possua um IDF inferior ao de "Casanova" e, dessa forma, o resultado com maior similaridade seria str_2 ("Antônnio Casanova").

Apesar de ser largamente aplicado a problemas de RI, o *Modelo Vetorial* tradicional não garante, por exemplo, que a string str_2 ("Antônnio Casanova") seja considerada mais similar a str_q que str_4 ("Pedro Casanova"), uma vez que no cálculo da similaridade do cosseno a palavra "Antônnio", por não ser exatamente igual a "Antônio", não seria considerada.

Além disso, tanto usando a *similaridade de Jaccard* quanto o *Modelo Vetorial* tradicional ou qualquer outra função de similaridade baseada somente em palavra, a string str_3 = "A Cassanova", não faria parte do conjunto de respostas, uma vez que não possui palavras em comum com str_q.

Por não relaxar a igualdade na comparação das palavras, as funções de similaridade baseadas em palavras apresentam deficiências para a identificação de *nomes* de objetos onde ocorram erros de grafia. Funções que se preocupam com esse aspecto serão apresentadas na Seção 3.3.

Para avaliar experimentalmente o modelo proposto nessa dissertação, foram implementadas variações de algumas métricas baseadas em palavras, chegando-se aos seguintes modelos híbridos:

- *Soft Bag* : é uma variação de *Bag of Words* onde são considerados para fins de cálculo

da intersecção entre os dois conjuntos de palavras, não só as palavras exatamente iguais mais também aquelas consideradas similares segundo uma função de distância de edição e um valor ϵ pré-definido. Para diferenciar o resultado da intersecção exata, onde a cada coincidência soma-se um ao valor da intersecção, optou-se por somar o valor da similaridade entre os termos comparados. Dessa forma, garante-se que a *string* "Caetano Traina" seja mais similar ao *nome* "Caetano Traina" que a *string* "Gaetano Traina".

Nos experimentos realizados, utilizou-se como função de similaridade de edição $Jaro - Winkler$ e como limiar de similaridade $\epsilon \geq 0.95$;

- *Soft Jaccard* : assim como *Soft Bag*, considera-se como intersecção os termos comuns ou cuja similaridade seja maior ou igual a ϵ, segundo uma função de similaridade de edição, ponderando-se o resultado segundo a o valor obtido de similaridade entre os *nomes*. Nos experimentos realizados, também utilizou-se $Jaro - Winkler$ e limiar de similaridade $\epsilon \geq 0.95$;

- *Soft Cosseno* : buscando evitar o uso de IDF, implementou-se uma versão da similaridade do cosseno não ponderada, ou seja, onde os IDFs de todos os termos eram iguais a 1.00. Assim como o *Soft TFIDF*, em *Soft Cosseno*, considera-se, além dos termos exatamente iguais, os termos similares segundo uma função de similaridade de edição. Foi utilizada a função de *Jaro-Winkler* e limiar de similaridade $\epsilon \geq 0.95$.

3.3 Métricas baseadas em Distância de Edição e Similares

Quando ocorrem erros de grafia ou OCR, como entre "Adenilso Simão" e "Adenilson Simão", uma função de distância de edição, poderia ser empregada para medir a similaridade entre os *nomes*. Entretanto, distâncias de edição crescem significativamente quando tratamos de *strings* longas, principalmente quando há supressão, abreviação ou inversão de palavras. Por exemplo, a distância de *Levenshtein* entre "Marco Antônio Casanova" e "Casanova Marco Antônio" é 16, enquanto que entre "Marco Antônio Casanova" e "Alberto Laender" é 15. Dessa forma, o uso de funções de distância sobre toda a *string* torna-se pouco útil no processo de junção por aproximação. Além disso, o custo de calcular-se a distância de edição é alto, por exemplo, o custo de *Levenshtein* é $O(n^2)$, onde n é o tamanho das *strings*.

Embora, em geral, o número de distâncias calculadas possa ser reduzido com o uso de estruturas métricas [14, 18, 19, 22, 25], é sabido que quando a dimensionalidade do espaço aumenta, ou seja, quando a quantidade de características do objeto consideradas para o cálculo da similaridade cresce, na maioria das vezes, a diferença das distâncias entre o objeto mais próximo e o mais distante se torna desprezível, inviabilizando a poda por desigualdade triangular. Essa situação, chamada *Dimensionality Curse* [22], é notada quando, por exemplo, aplicamos uma função métrica de similaridade baseada em distância de edição sobre *strings* longas; nesse caso, cada caractere passa a representar uma característica ou dimensão no espaço, caracterizando o problema como de *alta dimensionalidade*.

A diferença de dimensionalidade utilizando-se métricas baseadas em palavras e métricas baseadas em distância de edição é bastante significativa. Por exemplo, o *nome* "Marco Antônio Casanova" tem 3 palavras, portanto seria representado no Modelo Vetorial em um Espaço Euclideano de 3 dimensões, onde cada dimensão representa uma palavra do *nome*. Entretanto, se fosse utilizada uma função de similaridade baseada na distância de edição, como *Levenshtein*, as características consideradas seriam caracteres e não mais palavras, dessa forma, "Marco Antônio Casanova" seria representado em um espaço de 22 dimensões.

Nos casos em que os *nomes* das entidades são *strings* longas, uma árvore métrica como a M-Tree apresentaria *nomes* muito distantes armazenados no mesmo nodo folha, o que, conseqüentemente, aumentaria o raio de cobertura dos nodos internos. O aumento dos raios de cobertura traria redução da poda por desigualdade triangular levando à necessidade de calcular a distância de edição entre grande parte dos *nomes* da coleção.

Dessa forma, considerando a ineficácia diante de supressão, abreviação ou inversão de palavras e o alto custo associado ao uso de funções métricas de distâncias de edição, aplicá-las sobre todo o *nome* do objeto não é uma alternativa recomendável.

3.4 Dinamicidade

Considerando, por exemplo, que a DBLP possui mais de 500 mil artigos e que seu banco de dados cresce a medida que novos trabalhos são publicados, é interessante utilizar um modelo de integração que possibilite que novas publicações sejam inseridas sem que toda a coleção tenha que ser reprocessada.

Dessa forma, um fator importante a ser considerado ao adotarmos uma estrutura de armazenamento para suporte à junções aproximadas é a freqüência em que ocorrem inclusões e exclusões de objetos na coleção. Nesse aspecto, estruturas métricas dinâmicas, como a M-Tree e Slim-Tree seriam mais apropriadas que árvores estáticas, como, por exemplo, a R-Tree.

Outro aspecto a ser avaliado é a necessidade de pré-processamento da coleção para cálculos da freqüência inversa do termo (IDF). Modelos como o Vetorial e o Soft-TFIDF utilizam a freqüência relativa dos termos para fins de cálculo da similaridade entre conjuntos de palavras. Esses modelos baseiam-se no conhecimento prévio de toda coleção, o que não ocorre em coleções dinâmicas como da DBLP e da BDBCOMP.

Em coleções dinâmicas, ao inserir um novo objeto na coleção, os IDFs de toda a coleção sofreriam alteração o que poderia levar a uma avaliação indevida da importância dos termos. Por exemplo, considere que uma nova biblioteca digital está sendo formada pela UFAM. No início, serão inseridos na biblioteca trabalhos publicados por professores locais, por exemplo, “Altigran Soares da Silva”. Embora o termo “Altigran” seja raro no mundo real, no início da coleção ele teria menor IDF que, por exemplo, “Maria”. Sendo assim, nada garantiria, que “Altigran Soares” fosse mais similar a “Altigran Silva” que a “Maria Soares”. Além disso, a medida que novas coleções fossem acrescentadas à biblioteca, os valores de similaridade anteriormente calculados não seriam mais válidos, uma vez que os IDFs sofreriam alteração. Dessa forma, não seria possível garantir que um objeto s que obteve similaridade em relação a t igual a 0.98 quando a coleção possuía 2 mil objetos, é mais similar a t que um objeto u que obteve valor de similaridade igual a 0.92, inserido quando a coleção já apresentava 10 mil objetos .

Portanto, quando tratamos coleções dinâmicas, ou usamos IDF e recalculamos a similaridade entre todos os objetos da coleção quando novos objetos são inseridos, ou partirmos para uma solução onde o IDF não é utilizado.

A seguir será apresentada a solução proposta para junção por similaridades.

3.5 O Modelo 2LM

Devido às deficiências apresentadas ao utilizar-se isoladamente funções de distância de edição e funções de similaridade de palavra, propõe-se um modelo híbrido que combine os dois tipos de métricas para solucionar as situações expostas nas Seções 3.1 a 3.2.

Embora o modelo híbrido SoftTFIDF apresentado por Cohen *et al* tenha mostrado melhores resultados que os demais nos experimentos relatados em [4], este modelo apresenta limitações que deixariam de fora do resultado *nomes* que não possuam pelo menos um termo em comum com a consulta, que é o caso de "A Cassanova" em relação à "Alberto Casanova", por exemplo.

Isso ocorre porque, assim como o *Modelo Vetorial* tradicional ou TFIDF, o SoftTFIDF primeiramente identifica quais os *nomes* que possuem palavras em comum com a consulta e, apenas sobre as palavras que compõem esses *nomes*, aplica a função de distância de edição. Adotando essa política de poda, o SoftTFIDF reduz a dimensionalidade do espaço da mesma forma que o TFIDF, ou seja, cada palavra da consulta equivale a uma dimensão do espaço euclideano, reduzindo consideravelmente as computações de distância de edição, contudo, mostra-se falho para casos em que erros de edição ocorram em todos os termos, como o de "A Cassanova".

Buscando também abranger os casos em que nenhuma palavra do *nome* tem casamento exato com palavras da coleção, no modelo denominado $2LM$, propõem-se primeiramente identificar as palavras similares às da consulta, segundo alguma função métrica de distância de edição e calcular a similaridade do cosseno com e sem uso do IDF, tanto para os *nomes* com termos exatamente iguais quanto para aqueles que contém apenas termos similares aos da consulta.

Dessa forma, sendo um *nome* um conjunto de palavras e dado um *nome* $s = a_1...a_k$, um outro *nome* $t = b_1...b_L$ é considerado para fins de calculo de similaridade de token no $2LM$ se houver uma palavra $b_i = a_j$ em t tal que $d(b_i, a_j) < \epsilon$, onde d é uma função de distância de edição e ϵ é definido como a distância máxima tolerada.

$$2LM(S,T,C) = \sum_{w \in RelaxCLOSE(\epsilon,S,T,C)} V(w,S) \cdot V(w,T) \cdot (1 - (fator_{2LM} * D(w,T))) \quad (3.3)$$

onde C é a coleção, $V(w,S) = V'(w,S)/\sqrt{\sum_{w'} V'(w,S)^2}$ e $V'(w,S) = log(TF_{w,S} + 1) \cdot log(IDF_w)$ e D(w,T) = $min_{v \in T} dist(w,v)$ e $fator_{2LM}$ é um número real onde, quanto maior o número, menor a flexibilidade do modelo.

Considerando o exemplo apresentado na Seção 3.3, onde str_q = "Antônio Casanova" e a coleção C possui diversos autores, dentre os quais: str_1 = "Antônio Silva", str_2 = "Antônnio Casanova", str_3 ="A Cassanova" e str_4 ="Pedro Casanova". Utilizando-se o modelo $2LM$ e tendo como distância de edição *Levenshtein* e $\epsilon = 1$, primeiramente seriam identificadas as palavras similares às da consulta, no caso, "Antônio" $\rightarrow$ {Antônio,Antônnio} e "Casanova" $\rightarrow$

{Casanova,Cassanova}. Todos os *nomes* similares são ponderados conforme a distância de edição em relação à palavra da consulta, dessa forma, garante-se que palavras com erro de edição como "Cassanova" sejam consideradas menos importantes que o casamento exato "Casanova". Após esse ajuste, a palavra "Cassanova" pode ser representada no Modelo Vetorial na dimensão do espaço Euclideano reservado à "Casanova". Como próximo passo, aplica-se ao Modelo Vetorial, todos os objetos com palavras consideradas similares.

3.5.1 Economizando Cálculos de Distância

Para identificar as palavras com distância inferior a ϵ em relação a uma palavra v da consulta, pode-se realizar uma varredura seqüencial no conjunto de palavras, também chamado $Vocabulário$ da coleção, e, para cada palavra do Vocabulário, computar a distância em relação a v. Contudo, esse processo torna-se extremamente caro para $Vocabulários$ extensos, que é o caso de bases de dados de nomes. Uma solução possível para esse problema é a adoção de estruturas métricas para indexar as palavras do $Vocabulário$.

Como o problema de junção por similaridade aqui abordado prevê que a coleção será freqüentemente alimentada com novos objetos, será utilizada uma estrutura métrica dinâmica. Assim, a medida que novas palavras forem acrescentadas ao Vocabulário, a estrutura métrica será reorganizada sem que haja necessidade de reconstruí-la.

Com o uso de estruturas métricas vinculadas a funções de distância de edição, a partir de uma palavra do *nome* do objeto a ser inserido, a pesquisa pelas palavras mais próximas na coleção pode ser realizada através de uma consulta por abrangência, ou *Range Query*. Nesse caso, para obter-se as palavras com distância de até uma operação de edição em relação a "Casanova" usando *Levenshtein*, por exemplo, bastaria realizar a consulta $RangeQuery(Q,rQ,S)$, onde Q é "Casanova", rQ é o raio de abrangência, no caso, 1 e S é o Vocabulário da base.

Em uma estrutura métrica como a M-Tree, a busca em abrangência é realizada segundo o algoritmo apresentado na Figura 3.1 [14].

No modelo proposto, as entradas são coleções que sofrerão junção por similaridade através de um atributo em comum. Como saída serão geradas *listas de similaridade* contendo os identificadores dos *nomes* ordenados segundo a similaridade.

Um exemplo é mostrado na Figura 3.2, onde está sendo inserido na coleção C o *nome* "adenilso simao". Primeiramente, é realizada uma busca por abrangência na árvore métrica que

RangeSearch(N,Q,ε)

```
1.  Seja O_p o objeto pai do nodo N
2.  Se N não for folha então:
3.      Para cada objeto O_i em N faça:
4.          Se |d(Q,O_p) - d(O_n,O_i) ≤ ε então:
5.              Compute d(Q, O_i);
6.              Se d(Q,O_i) ≤ ε então adicione O_i ao conjunto resultado
7.  Senão // N é um nodo interno
8.      Para cada nodo filho N_c de N faça:
9.          Se |d(Q,O_n) - d(O_n,O^[Nc])| ≤ ε + r^[Nc] então:
10.             Compute d(Q,O^[Nc]);
11.             Se d(Q,O^[Nc]) ≤ ε + r^[Nc] então:
12.                 Carregue nodo Nc e chame RangeSearch(Nc,Q, ε);
13. Fim.
```

Figura 3.1: Algoritmo de Range Query

indexa o Vocabulário de S. A indexação na árvore é feita através de uma função métrica de distância de edição, no caso, *Levenshtein*. Dessa forma, tem-se $RangeQuery(\text{“}adenilso\text{”}, 1, S)$, onde S é todo o Vocabulário de C, que retorna o termo “adenilson”, pois o mesmo tem $Levenshtein(\text{“}adenilson\text{”}, \text{“}adenilso\text{”}) = 1$. O mesmo é feito para o termo “simao”, que retorna como similar “simão”. Assim, todos os *nomes* que possuem os termos retornados serão considerados para fins de cálculo da distância de *token*. Nos experimentos realizados utilizou-se a similaridade do *cosseno* para calcular a similaridade final do modelo. Foram definidas duas versões de $2LM$, ambas tendo como métrica de edição a distância de *Levenshtein*, sendo que uma das versões utiliza a similaridade do *cosseno* ponderada segundo o IDF e outra não considera o IDF dos termos.

Os objetos considerados similares farão parte da *lista de similaridade* do objeto de consulta, sendo inserido na lista em ordem decrescente de similaridade.

No Capítulo 4 serão apresentados o resultado de experimentos realizados com métricas de edição, métricas de token, o modelo híbrido SoftTFIDF e o modelo 2LM proposto. Também será realizada uma análise comparativa dos resultados segundo precisão, revocação e média harmônica F1 das *listas de relevantes* obtidas.

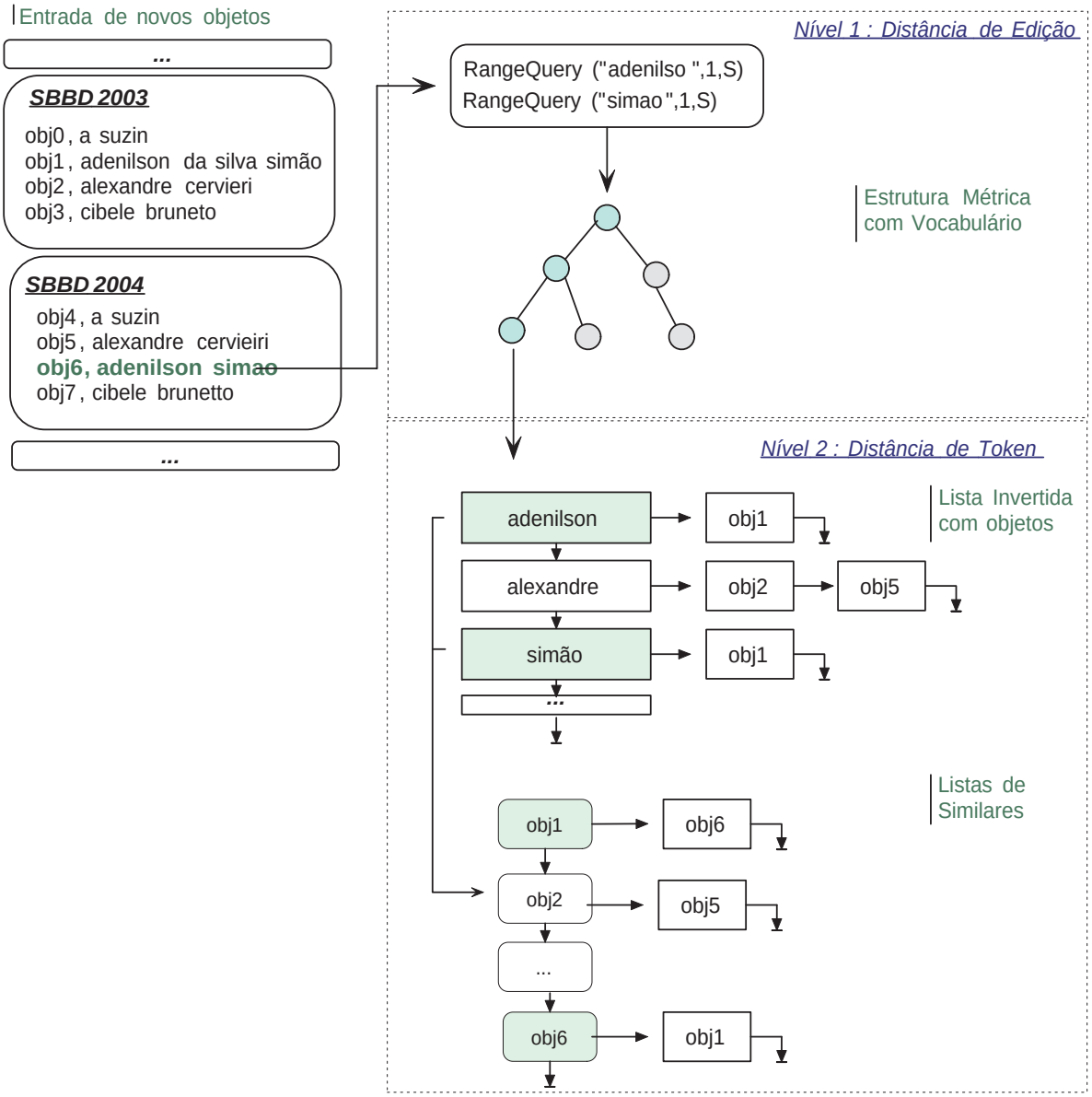

Figura 3.2: Exemplo de 2LM

Capítulo 4

Verificação Experimental

Para avaliar o desempenho do modelo proposto, $2LM$, foram realizados experimentos de junção por similaridade sobre uma coleção real e 4 coleções sintéticas.

Os resultados do modelo $2LM$ foram comparados com resultados obtidos com outros modelos apresentados na literatura. Como funções de distância de edição, foram utilizadas as métricas de *Levenshtein* e *Jaro-Winkler*, como funções de *token* adotamos a similaridade de *Jaccard*, a medida da intersecção entre os termos dos dois objetos (*Bag of Words*) e a similaridade do *cosseno*, com e sem ponderação de IDFs. Para os modelos híbridos, utilizamos uma implementação baseada na definição proposta por Cohen do *Soft-TFIDF*, utilizando como função secundária de distância *Jaro-Winkler*, como recomendado em [4]. Na implementação do Modelo $2LM$ usamos a função de *Levenshtein* como métrica de distância de edição e a *similaridade do cosseno*, com e sem ponderação por IDF, como distância de *token*. Também foram utilizadas as variações *Soft Bag*, *Soft Jaccard* e *Soft Cosseno* propostas na Seção 3.1.

Na Seção 4.1 serão apresentadas informações sobre as coleções utilizadas para teste dos modelos e a metodologia adotada na formação das bases sintéticas. Na Seção 4.2 serão apresentadas as ferramentas utilizadas. Na Seção 4.3 serão descritos os experimentos e analisados seus resultados.

4.1 As Coleções de Teste

Os experimentos realizados usaram as diferentes funções de similaridade e modelos sobre uma coleção real da *Biblioteca Digital Brasileira de Computação* (BDBComp)[1] e sobre 4 coleções sinteticamente geradas a partir do banco de dados da $DBLP$ [2]. Informações detalhadas sobre cada coleção serão apresentadas a seguir.

4.1.1 BDBComp

A BDBComp (Biblioteca Digital Brasileira de Computação) é uma biblioteca que tem como objetivo reunir e proporcionar o acesso a produção científica brasileira na área de computação. Como não há acervo brasileiro que concentre tais trabalhos, a BDBComp está cadastrando a produção científica brasileira publicada em eventos e periódicos nacionais. Uma das principais dificuldades encontradas é identificar *nomes* distintos que se referem ao mesmo autor do mundo real.

Buscando submeter o modelo proposto a uma situação real, foi utilizada a coleção da $BDBComp$ disponibilizada pelo *UFMG Database Group*, Universidade Federal de Minas Gerais, que contém 5.130 *nomes* distintos de autores, totalizando 7.348 associações de autores com publicações na área de Ciência da Computação.

O arquivo de origem dos dados da coleção $BDBComp$ foi disponibilizado em formato XML. A Figura 4.1 apresenta sua estrutura. Com o objetivo de promover a junção por similaridade dos *nomes* de autores, foi utilizado apenas o conteúdo da tag $< creator >$.

Por conveniência e sem perda de generalidade, antes de iniciar o processamento da coleção, os *nomes* dos autores foram submetidos a remoção da acentuação e caracteres da tabela ASCII estendida e convertidos para minúsculo (*lowercase*).

Os *nomes* de autores foram identificados manualmente por Jean Oliveira do LBD/DCC/UFMG, que elaborou listas de autores similares. Essas listas, aqui denominadas $RelevantesBDBComp$, serviram de referência para a avaliação da qualidade dos modelos junção implementados.

A partir da coleção $RelevantesBDBComp$, foi realizado processo manual para verificação da freqüência de erros de grafia, inversão de palavras, abreviações e supressões. A Tabela 4.1 resume

[1] http://www.lbd.dcc.ufmg.br/bdbcomp
[2] http://www.dblp.uni-trier.de

```
<record>
    <header>
        <identifier>SBRC1986meta0012</identifier>
        <datestamp>2003-03-20</datestamp>
        <setspec>bdbcomp:SBRC1986</setspec>
    </header>
    <metadata>
        <oai_dc:dc>
            <dc:title>DIMAC : Um Protocolo de Acesso por Passagem de Token</dc:title>
            <dc:creator>Clylton G.Fernandes </dc:creator>
            <dc:creator>Jaelson F.S.Castro </dc:creator>
            <dc:creator>Paulo R. F. Cunha </dc:creator>
            <dc:subject/>
            <dc:publisher/>
            <dc:contributor/>
            <dc:date>1986</dc:date>
            <dc:type>Text</dc:type>
            <dc:format/>
            <dc:identifier>SBRC1986article0012</dc:identifier>
            <dc:source>SBRC1986</dc:source>
            <dc:language>por</dc:language>
            <dc:reference/>
            <dc:coverage>Recife, PE, Brasil </dc:coverage>
            <dc:rights>Sociedade Brasileira de ComputaÃ§Ã£o</dc:rights>
        </oai_dc:dc>
    </metadata>
    <about>
        <provenance>http://www.dcc.ufmg.br/~sbrc/sbrc86.html</provenance>
    </about>
</record>
```

Figura 4.1: Exemplo da BDBComp

os dados sobre a coleção BDBComp, onde *Palavras distintas* se referem aos termos que compõem o vocabulário da coleção, por exemplo, “Agma”, “Juci” e “Traina”, e *nomes completos distintos* se referem ao *nome* do autor, por exemplo, “Agma Juci Traina”. Através de um histograma de freqüência dos termos, observou-se que 70.04% dos termos ocorrem em, no máximo, 10 objetos sendo que 46.33% dos termos são apresentados em apenas 1 *objeto*. Analisou-se ainda que os 20 termos mais freqüentes estão presentes em 60.26% dos objetos.

Ressalta-se que, através das características da coleção apresentadas na Tabela 4.1, é possível identificar as seguintes influências sobre o resultado dos experimentos:

1. O número de erros de grafia é baixo, favorecendo as métricas baseadas em palavras, uma vez que essas não tratam erros de escrita;

2. Como são poucos os erros de grafia, espera-se que as métricas híbridas sejam ligeiramente superiores às baseadas apenas em conjunto palavras;

3. O número de erros de abreviação e supressão é alto, o que prejudica as métricas baseadas unicamente em caracteres , pois promove o aumento considerável da distância entre as duas strings que representam o mesmo objeto.

Estatísticas	Quantidade	%
Objetos (autor/publicação)	7.348	100.00
Palavras distintas	4.237	-
Nomes completos distintos	5.130	-
Objetos com Erros de grafia	81	1.10
Objetos com Inversão de palavras	2	0.03
Objetos com Abreviação	2.608	35.49
Objetos com Supressão	766	10.43

Tabela 4.1: Estatísticas da coleção BDBComp

4.1.2 DBLP

Assim como a *BDBComp*, a *DBLP* é uma biblioteca Digital contendo publicações na área de computação.

Para os experimentos realizados, foi utilizada uma amostra de 10.000 objetos extraídos de arquivo XML contendo 1.124.200 *nomes* de autores associados à publicações na DBLP. A coleção de 10.000 objetos contém um vocabulário de 8.572 termos distintos. Esta coleção será citada aqui como *DBLP_Original*. Por conveniência e sem perda da generalidade, os *nomes* tiveram um prévio tratamento para remoção de acentuação e de caracteres da tabela ASCII estendida, assim como conversão para maiúsculo (*uppercase*).

Para simular bancos de dados com quantidades crescentes de erros, foram geradas 4 coleções sintéticas baseadas na *DBLP_Original*, essas coleções serão denominadas DBLP_Sintética1, DBLP_Sintética2, DBLP_Sintética3 e DBLP_Sintética4, onde a DBLP_Sintética2 contém mais erros que a DBLP_Sintética1 e assim sucessivamente.

O objetivo de utilizar uma coleção com tantos erros é testar o comportamento dos modelos em situações extremas, onde a ocorrência de erros, principalmente de edição, é crescente.

Informações sobre as coleções sintéticas são apresentadas na Tabela 4.2.

Para a avaliação dos resultados foi gerada a lista de *RelevantesDBLP*. A identificação dos objetos relevantes foi feita através do casamento exato dos *nomes* da coleção original *DBLP_Original*.

Erros Introduzidos	*Sint 1*		*Sint 2*		*Sint 3*		*Sint 4*	
	Obj	%	Obj	%	Obj	%	Obj	%
Grafia incorreta	1.010	13.75	2.020	27.49	3.030	41.24	4.040	54.98
Inversão	405	5.51	910	12.38	1.415	19.26	1.920	26.13
Abreviação	303	4.12	505	6.87	707	9.62	1.010	13.75
Supressão	130	1.77	220	2.99	375	5.10	790	10.75

Tabela 4.2: Estatísticas da coleção DBLP

4.2 Ferramentas Utilizadas

Para realização dos experimentos foram implementados programas em linguagem C e C++, utilizando o gcc e g++, em ambiente Linux, *SuSe* 9.02. Também foram desenvolvidos scripts em Perl para geração das bases sintéticas e avaliação dos resultados gerados.

Adotou-se como estrutura métrica para o primeiro nível do $2LM$ uma árvore métrica dinâmica, a $M-Tree$. Foi utilizada a versão 0.911, cuja implementação encontra-se disponível no site do Projeto M-Tree [3]. A implementação da M-Tree utiliza o pacote *GiST Generalized Search Tree* [4]. Embora tenha sido usada a M-Tree, ressalta-se que outras estruturas métricas dinâmicas também podem ser utilizadas no modelo proposto [18, 19, 25].

Algumas funções utilizadas têm sua implementação disponível na WWW, tais como $Jaro-Winkler$, disponibilizada pelo U.S. Census Bureau[5], e de *Levenshtein* disponíbilizada por Michael Gilleland [6].

Embora o pacote *secondstring*, contendo diversas funções de similaridade, inclusive Soft-TFIDF, encontre-se disponível para *download* [7], o mesmo foi desenvolvido em Java, motivo pelo qual optou-se por não utilizá-lo, uma vez que a implementação da árvore métrica e demais programas adotou C e C++.

4.3 Descrição dos experimentos

Os experimentos realizados foram divididos em dois grupos, no primeiro grupo, denominado *Experimentos em coleções previamente conhecidas*, foram avaliadas métricas com pré-processamento de IDFs. No segundo grupo, chamado *Experimentos em coleções dinâmicas*, fo-

[3] http://www-db.deis.unibo.it/Mtree/
[4] http://epoch.cs.berkeley.edu:8000/gist/
[5] http://www.census.gov/geo/msb/stand/strcmp.c
[6] http://www.merriampark.com/ld.htm
[7] http://secondstring.sourceforge.net/

ram avaliadas apenas as métricas que não consideravam a freqüência inversa dos termos (IDF), uma vez que esse valor varia a medida que novos *nomes* são incluídos na coleção.

Em cada grupo de experimentos são avaliados os resultados sobre a coleção BDBComp e as 4 coleções sintéticas da DBLP.

A implementação das funções de similaridade baseadas em conjunto de palavras utilizou uma *lista invertida* [1] contendo os termos do vocabulário e os objetos onde esses termos são encontrados, conforme ilustra a Figura 4.2.

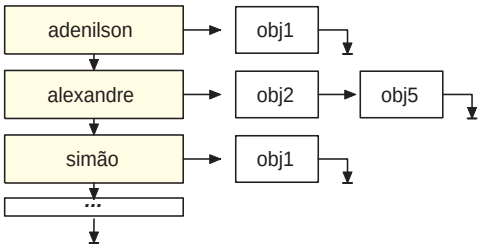

Figura 4.2: Lista Invertida utilizada em funções de similaridade de token

A lista invertida é utilizada para podar os objetos que não possuam pelo menos um termo em comum com o objeto de consulta. As funções híbridas *Soft Bag*, *Soft Jaccard*, *Soft Cosseno* e *SoftTFIDF* também utilizam a lista invertida para poda.

No modelo $2LM$, os termos do objeto de consulta passam por uma busca em abrangência na árvore métrica (M-Tree), onde são localizados os termos que não excedam um limiar ϵ pré-determinado (*RangeSearch*). Após a identificação dos termos similares, é realizada uma busca dos *nomes* em que eles ocorrem, para isso também é utilizada a lista invertida.

Nas funções híbridas, a distância de edição é calculada sobre cada termo do *nome* e não sobre o *nome* completo do objeto. No caso de *2LM-TFIDF* e *2LM-Cosseno*, apenas os termos com mais de 4 caracteres foram submetidos para o processo de cálculo de distância de edição.

Nos modelos baseados somente em distância de edição, (*Levenshtein* e *Jaro-Winkler*), apenas os objetos exatamente iguais não foram submetidos ao cálculo de similaridade de edição, sendo-lhes atribuído o valor zero para *Levenshtein* e um para *Jaro-Winkler*.

4.3.1 Definição de Limiares (thresholds)

Nos modelos híbridos, foi utilizado como limiar para a função de distância de Edição de Levenshtein $\epsilon = 1$. Não encontramos na literatura estatísticas específicas sobre erros de edição

em nomes, entretanto Dohnal cita genericamente que 80% dos erros de edição que ocorrem em texto apresentam apenas uma deleção, inserção ou alteração [19].

Limiar Jaro-Winkler	*F1*
0.90	96.65
0.92	96.86
0.94	96.85
0.95	96.92
0.96	96.91
0.98	96.73

Tabela 4.3: Escolha do Limiar para a função Jaro-Winkler

Em modelos híbridos que adotam a função de similaridade de *Jaro-Winkler* foi adotado $\epsilon = 0.95$, que mostrou ser o melhor valor entre os demais testados $\{0.90, 0.92, 0.94, 0.95, 0.96, 0.98\}$ (vide Tabela 4.3). Nessas funções, o valor da similaridade parcial é multiplicado pelo valor retornado por *Jaro-Winkler*, que encontra-se entre $[0, 1]$, sendo 1 o casamento exato entre as *strings*. No entanto, ao aplicar *Levenshtein* em modelos híbridos o mesmo procedimento não pode ser adotado, pois trata-se de uma função de distância, onde seu valor cresce a medida que a similaridade entre as *strings* diminui, sendo 0 atribuído quando ocorre a igualdade entre as *strings*. Dessa forma, nas implementações de $2LM$, onde é utilizada a distância de *Levenshtein* como distância de edição, o valor da similaridade parcial entre as *strings* é multiplicado por k, onde $k = (1 - fator_{2LM} * Levenshtein(string1, string2)$ para todo $Levenshtein(string1, string2) \leq \epsilon$. O valor adotado para $fator_{2LM}$ foi escolhido experimentalmente aplicando-se os valores $\{0.14, 0.16, 0.18, 0.30, 0.50\}$ sobre as coleções BDBComp e DBLP_Sintética1. Como pode ser observado na Tabela 4.4, quanto maior o valor de k menos tolerante a erros torna-se a função, o que é bom para a BDBComp por ter poucos erros; contudo, na DBLP_Sintética1, que apresenta muitos erros de edição, pode-se perceber que o Fator Multiplicador oscila no ponto 0.18 a partir do qual F_1 volta a cair. Adotou-se, portanto, o valor 0.18 para uso em todas as coleções visando equilibrar os ganhos do $2LM$ entre coleções com muitos e poucos erros.

4.3.2 Experimentos em coleções previamente conhecidas

Quando as coleções são previamente conhecidas é possível o treinamento dos modelos através do cálculo da freqüência inversa dos termos na coleção (IDF). Dessa forma, além de funções independentes de IDF, como *Levenshtein*, *Jaro-Winkler*, *Bag of Words*, *Jaccard*, *Cosseno não*

Fator Multiplicador	*F1*	
	BDBComp	*DBLP Sint 1*
0.14	97.01	93.86
0.16	97.04	93.89
0.18	97.08	93.91
0.30	97.27	93.86
0.50	97.31	93.12

Tabela 4.4: Escolha do Fator Multiplicador

ponderado, *Soft Bag*, *Soft Cosseno*, *Soft Jaccard*, e $2LM-Cosseno$, é possível usar funções como *TF-IDF*, *Soft-TFIDF* e $2LM-TFIDF$.

Neste grupo de experimentos, abordaremos todas as funções que utilizam IDFs para cálculo da similaridade das coleções da $BDBComp$ e da $DBLP$.

Na execução dos programas, é primeiramente lida a coleção completa, montada a *lista invertida* e calculadas as normas dos documentos e IDFs dos termos do Vocabulário da coleção. Em seguida, é realizada leitura seqüencial dos objetos da coleção, calculada sua similaridade com os demais objetos da *lista invertida* e, finalmente, gerada a *lista de similares*.

Para o Modelo Vetorial (TFIDF) e $2LM-TFIDF$, as informações contidas na lista invertida são suficientes para cálculo da similaridade final entre os objetos. Contudo, em modelos que requerem o conhecimento de todos os termos dos objetos, como é o caso de *Soft-TFIDF* foi utilizado um *array* para armazenar a lista com todos os termos de cada *nome*. Para evitar esse armazenamento em memória, poderia ser implementado *hashing* em disco, o que não adotou-se neste trabalho.

4.3.3 Experimentos em coleções dinâmicas

Considerando que tanto a BDBComp quanto a DBLP são coleções dinâmicas, apresentaremos experimentos que não farão uso do IDF dos termos para fins de cálculo da similaridade entre os objetos.

Dessa forma, serão apresentados resultados referentes às funções de *Levenshtein*, *Jaro-Winkler*, *Bag of Words*, *Jaccard*, *Cosseno não ponderado*, *Soft Bag*, *Soft Cosseno*, *Soft Jaccard*, e $2LM-Cosseno$.

Para realização dos experimentos simulando coleções dinâmicas, ao ler um novo objeto da coleção é calculada a similaridade deste com os demais objetos já pertencentes à *lista invertida*. A partir dos valores de similaridade obtidos é gerada uma *lista de similares parcial* e o novo

objeto é acrescentado à *lista invertida*. A *lista de similares parcial* do objeto será atualizada a medida que novos objetos similares vão sendo inseridos.

Ressalta-se que as funções de *Jaccard*, *Soft Cosseno* e *Soft Jaccard* requerem o conhecimento do *nome* completo do objeto, nesses casos, os mesmos foram armazenados em memória.

4.4 Resultados

Nesta seção serão apresentados e analisados os resultados dos experimentos sobre as coleções BDBComp e DBLP. Primeiramente serão reportados os experimentos realizados sobre a BDB-Comp com funções que utilizam IDF e depois com funções independentes de IDF. O mesmo será feito em seguida sobre as coleções sintéticas da DBLP, onde além disso, será analisado o comportamento das funções a medida que aumenta a quantidade de erros na coleção.

4.4.1 BDBComp

Sobre a coleção da BDBComp, a função $Soft{-}TFIDF$ apresentou a melhor precisão média entre todos os modelos, entretanto, no último ponto de revocação o modelo proposto $2LM-TFIDF$ apresentou o melhor valor de F_1. Até o ponto de 70% de revocação, os resultados obtidos com o Modelo Vetorial tradicional (TF-IDF) foram muito próximos aos obtidos pela $SOFT-TFIDF$, que apresentou maior diferença nos últimos pontos de revocação. A proximidade de TF-IDF com relação às duas abordagens híbridas se dá porque a coleção BDBComp apresenta apenas 81 objetos (1.1%) onde ocorrem erros de edição, caso que em as métricas híbridas apresentam ganhos bastante signficativos.

O modelo $2LM-TFIDF$ obteve resultados inferiores ao $Soft-TFIDF$ na maioria dos pontos de revocação, como pode ser visto na Figura 4.3. Esse resultado já era esperado pelo fato de ser o modelo voltado à coleções onde ocorram muitos erros de edição. Além disso, a vantagem em relação ao modelo híbrido $Soft-TFIDF$ está em considerar para fins de cálculo da similaridade objetos onde todos os termos apresentem erros de grafia e na BDBComp não há nenhuma ocorrência desse caso.

Os valores de precisão média, de F_1 calculado no último ponto de revocação e média dos valores F_1 são apresentados na Tabela 4.5.

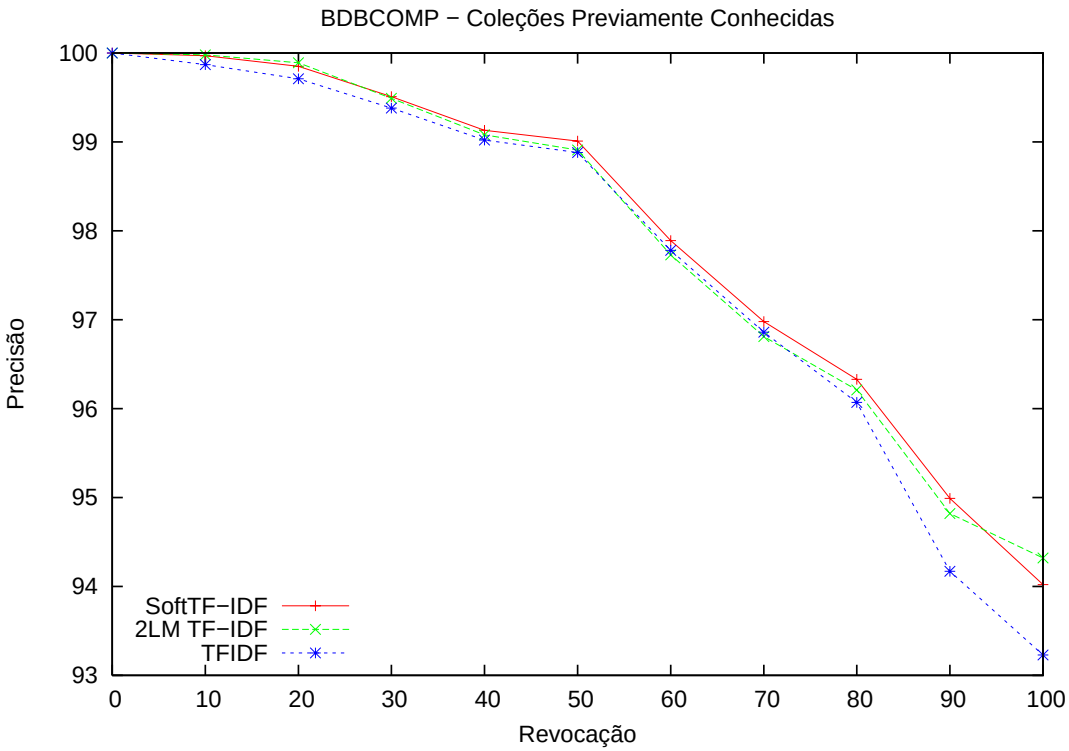

Figura 4.3: BDBComp - Resultado utilizando IDF

Métrica	*Precisão Média*	F_1	F_1 *Média*
Soft TFIDF	97.97	96.92	65.35
2LM TFIDF	97.93	97.08	65.34
TF-IDF	97.72	96.50	65.24

Tabela 4.5: Avaliação de Métricas com IDF sobre a BDBComp

Na Tabela 4.6, são apresentados os resultados obtidos com métricas independentes de prévio treinamento, ou seja, sem IDF, sobre a coleção da BDBComp. Observa-se o fraco desempenho de funções baseadas somente em distância de edição, tendo $Jaro - Winkler$ e *Levenshtein* os piores resultados. Isso era previsível uma vez que a ocorrência de abreviações e supressões, dois casos críticos para funções de distância de edição, estão presentes em 45.95% dos *nomes* da BDBComp.

Dentre as métricas que utilizam apenas distância entre conjuntos de palavras, *Bag of Words* foi a que obteve pior desempenho, distanciando-se bastante da similaridade de *Jaccard* e do *Cosseno*. Isso se dá porque essas últimas normalizam os resultados, considerando também as palavras que não pertencem à intersecção dos conjuntos; um fator importante, uma vez que a supressão de termos ocorre em 10.43% dos objetos da coleção. Também pode-se observar que a similaridade de *Jaccard* e do *Cosseno* apresentam resultados quase idênticos, o que era previsível

pois a normalização dos objetos é adotada por ambas.

Métrica	*Precisão Média*	F_1	F_1 *Média*
2LM Cosseno	94.79	92.35	63.89
Soft Cosseno	94.77	92.29	63.88
Soft Jaccard	94.76	91.95	63.87
Jaccard	94.67	92.18	63.84
Cosseno	94.55	92.03	63.79
Soft Bag	93.46	90.84	63.30
Bag of Words	93.17	90.42	63.17
Levenshtein	78.97	76.34	56.52
Jaro-Winkler	75.89	75.06	55.04

Tabela 4.6: Avaliação de Métricas sem IDF sobre a BDBComp

Como pode ser observado na Figura 4.5, métricas que adotam tanto distância de edição quanto de palavras, apresentaram melhores resultados, embora devido aos poucos erros de edição, não haja muita diferença entre elas e as versões que utilizam apenas distância de token (vide Tabela 4.7).

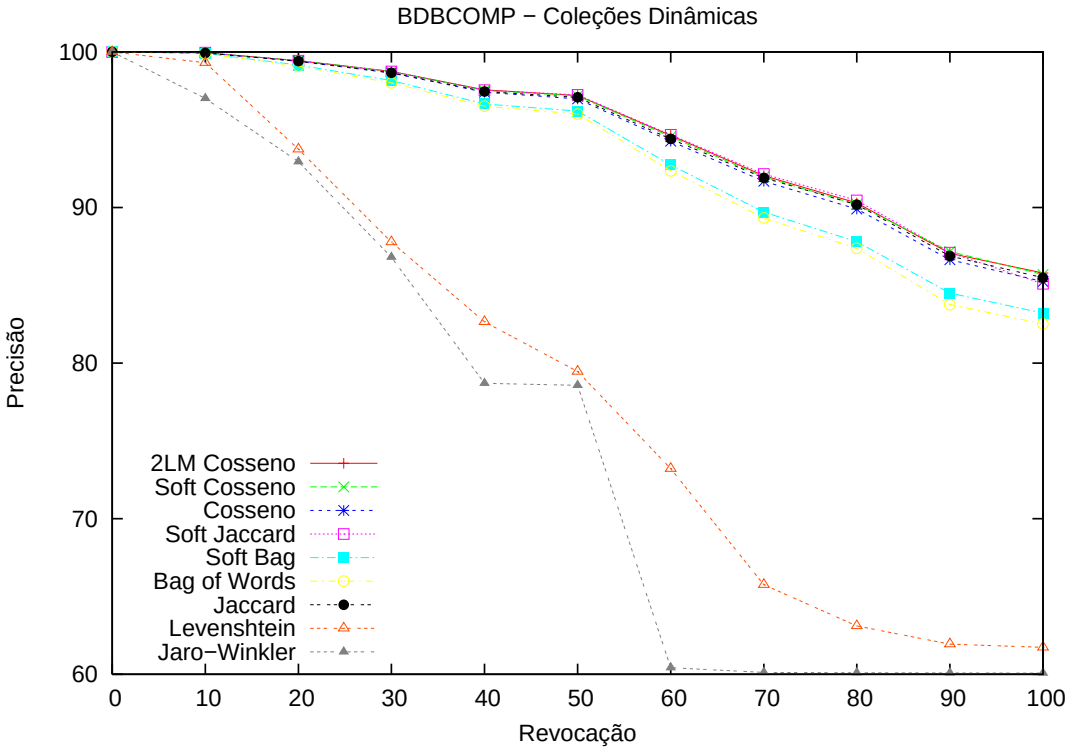

Figura 4.4: BDBComp - Resultados sem IDF

Em uma comparação de todas as métricas, é possível observar que as métricas que utilizam IDFs têm eficácia significativamente superior às demais, como pode ser notado na Figura 4.5.

Na Tabela 4.8 é apresentada a avaliação de todas as métricas sobre a BDBComp, consoli-

Métrica	F_1		*Diferença*
	Token	Híbrida (Soft)	
TFIDF	96.50	96.92	0.42
Cosseno	92.03	92.29	0.26
Jaccard	92.18	91.95	0.22
Bag of Words	90.42	90.84	0.42

Tabela 4.7: Comparação de Métricas de Token com suas variantes Híbridas - BDBComp

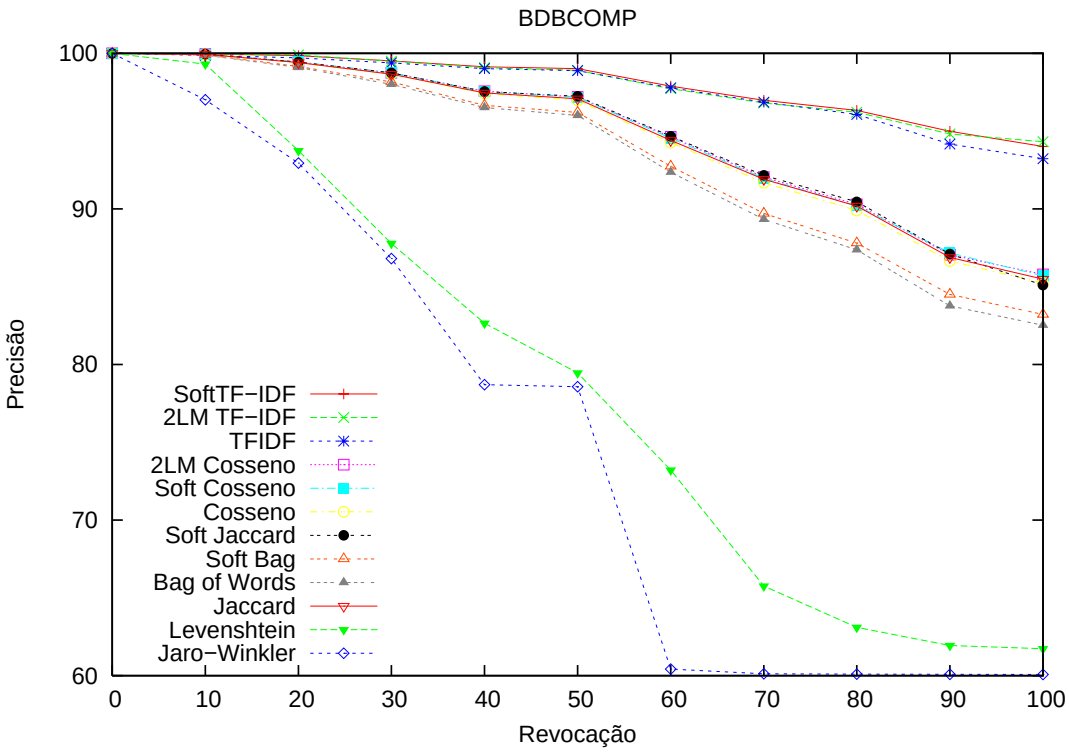

Figura 4.5: BDBComp - Resultados

dando os resultados das Tabelas 4.5 e 4.6.

Os tempos de processamento de todas as métricas podem ser visualizados na Tabela 4.9, onde observa-se o alto custo de métricas unicamente baseadas em distâncias de edição. Ressalta-se que, em *Jaro-Winkler* e *Levenshtein*, por não haver poda, as *listas de similares* são consideravelmente maiores que as demais e sua manutenção impacta no tempo de execução do programa. Também é signficativa a diferença do tempo de execução entre métricas de distância de token e as métricas híbridas, que sofrem a influência do custo de distância de edição. Além disso, assim como as métricas que utilizam apenas distância de edição, as métricas híbridas, com exceção de $2LM$, necessitam que sejam armazenados todos os termos do *nome*, não bastando a lista invertida para realizar o cálculo de similaridade.

Métrica	*Precisão Média*	F_1	F_1 *Média*
Soft TFIDF	97.97	96.92	65.35
2LM TFIDF	97.93	97.08	65.34
TF-IDF	97.72	96.50	65.24
2LM Cosseno	94.79	92.35	63.89
Soft Cosseno	94.77	92.29	63.88
Soft Jaccard	94.76	91.95	63.87
Jaccard	94.67	92.18	63.84
Cosseno	94.55	92.03	63.79
Soft Bag	93.46	90.84	63.30
Bag of Words	93.17	90.42	63.17
Levenshtein	78.97	76.34	56.52
Jaro-Winkler	75.89	75.06	55.04

Tabela 4.8: Avaliação de Métricas sobre a BDBComp

Métrica	*Tempo (ms)*	*Cálculos de Distância*	
		Token	*Edição*
2LM - Cosseno	539,780	2,575,362	18,192,790
2LM - TFIDF	833,780	2,619,204	39,109,161
Bag of Words	78,880	2,413,288	-
Cosseno	191,730	2,413,288	-
Jaccard	96,730	2,413,288	-
Jaro-Winkler	4,288,770	-	26,986,584
Levenshtein	16,499,040	-	26,986,584
Soft Bag	84,420	2,413,288	14,663,992
Soft Cosseno	366,160	2,413,288	14,663,992
Soft Jaccard	106,310	2,413,288	29,327,984
Soft TF-IDF	568,430	2,421,170	19,661,499
TF-IDF	1,310	2,421,170	-

Tabela 4.9: Tempo de Processamento e Cálculos de Distância - BDBComp

Para comparar o custo em relação ao número de distâncias computadas, ou seja, independente da manutenção das *listas de similares*, coletou-se a quantidade de distâncias de edição e de *token* de cada uma das métricas, também apresentadas na Tabela 4.9.

Ressalta-se que uma operação de cálculo de distância de edição nas métricas *Levenshtein* e $Jaro-Winkler$ equivalem ao cálculo sobre um *nome* completo, enquanto que nas métricas híbridas, referem-se ao cálculo sobre um termo.

Como pôde ser observado na Tabela 4.9, $2LM$ apresenta maior número de distâncias de token que TF-IDF. Isso ocorre porque enquanto o TF-IDF e demais métricas baseadas unicamente em token reduzem a dimensionalidade do problema de D (termos do Vocabulário) para $d < D$ (termos da consulta), o $2LM$ relaxa essa redução para D' ($d < D' < D$) que representa os termos do vocabulário similares aos termos da consulta. Dessa forma, um maior número de objetos é considerado.

Embora a eficácia do $2LM - TFIDF$ seja muito próxima do $Soft - TFIDF$, este último necessita de menos cálculos de distância tanto de edição quanto de *token* para realizar junções por similaridade, pois só calcula a similaridade entre os termos dos objetos com pelo menos um termo em comum com a consulta.

4.4.2 DBLP

A seguir, serão apresentados os resultados com as 4 bases sinteticamente geradas a partir da $DBLP\_Original$. Ressalta-se que os erros nas coleções são crescentes, tendo a $DBLP\_Sintética1$ menos erros que a $DBLP\_Sintética2$ e assim sucessivamente.

Primeiramente serão observadas as métricas aplicadas sobre coleções previamente conhecidas, ou seja, aquelas que permitem a utilização de IDF.

Na Figura 4.6 observa-se que a medida que os erros aumentam, a eficácia de *TF-IDF* e *Soft-TFIDF* caem e $2LM - TFIDF$ mantém-se com pouca variação. Isso se dá porque nas coleções sintéticas foram acrescentados muitos erros de edição, sendo que na maioria dos objetos com erros de edição todas as palavras apresentavam erros. Quando nenhuma palavra casa exatamente com uma palavra do objeto de consulta, o objeto é ignorado pelo TF-IDF e Soft-TFIDF. Observa-se que a eficácia de Soft-TFIDF é ligeiramente superior à da função TF-IDF, uma vez que há objetos na coleção que, embora apresentem erros de edição em alguns termos, apresentam casamento exato de outros.

Embora nem todos os termos dos objetos com erros de edição apresentem erros, como pode ser visto na Tabela 4.10, a diferença não foi suficiente para que o Soft-TFIDF apresentasse grande vantagem em relação ao TF-IDF no que se refere a eficácia. Isso ocorre porque os termos da DBLP são mais raros que os da BDBComp. Ao analisar a $DBLP\_Original$ observou-se que 97.04% dos termos aparecem em no máximo 10 *nomes* enquanto que na BDBComp esse percentual é de 70.04%. Além disso, 65.37% dos termos da $DBLP\_Original$ aparecem apenas em um objeto, enquanto que na BDBComp esse número cai para 46.33%.

Na Tabela 4.11 são apresentados os valores de F_1 no último ponto de revocação de todas das coleções sintéticas, utilizando todas as métricas. Comparando-se as métricas que não utilizam IDF sobre as mesmas coleções, observa-se que $2LM - Cosseno$ como era esperado apresenta maior eficácia, seguido da distância de edição de *Levenshtein*. As métricas baseadas em token que utilizaram a poda através da *lista invertida*, apresentaram valores de $F1$ muito similares,

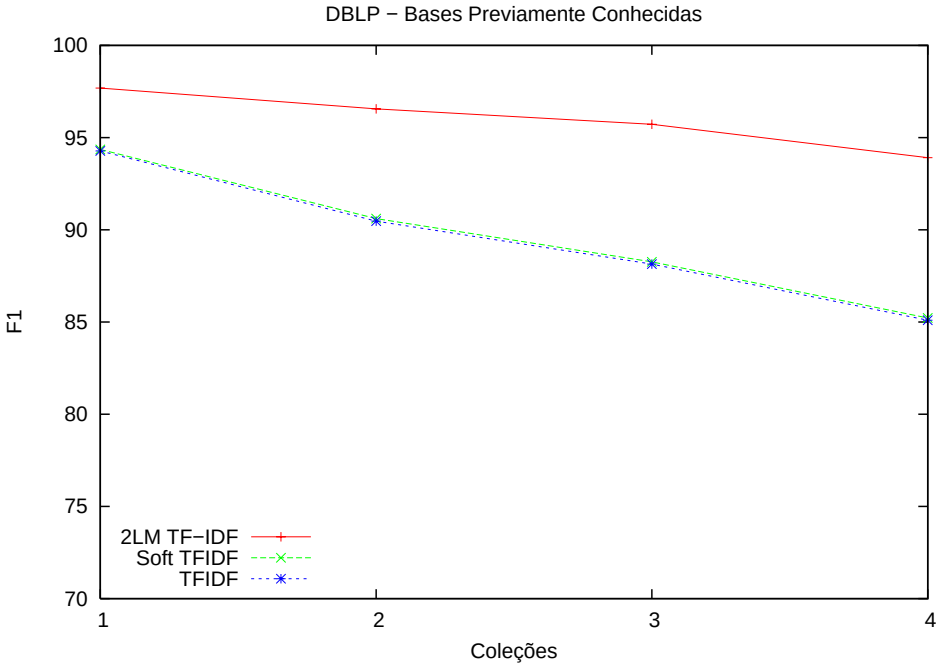

Figura 4.6: DBLP - Resultado utilizando IDF

Coleção	*Em algum termo*		*Em todos os termos*	
	Obj	%	Obj	%
DBLP$_{S}intetica1$	1010	13.75	914	12.44
DBLP$_{S}intetica2$	2020	27.49	1794	24.41
DBLP$_{S}intetica3$	3030	41.24	2717	36.98
DBLP$_{S}intetica4$	4040	54.98	3646	49.62

Tabela 4.10: DBLP - Erros de Edição

como pode ser visto na Figura 4.7. Embora muito próximos, é possível observar a melhor eficácia das funções que associam token com métricas de edição (vide Tabela 4.11).

Os resultados de todas as métricas estão apresentados na Figura 4.8, onde nota-se claramente os ganhos utilizando a estrutura métrica para relaxar a restrição de igualdade dos termos.

Métrica	F_1			
	Sint 1	*Sint* 2	*Sint* 3	*Sint* 4
2LM - Cosseno	98.05	96.36	95.16	92.84
2LM - TFIDF	97.69	96.56	95.72	93.91
Bag of Words	93.48	89.62	87.34	84.28
Cosseno	93.96	89.94	87.71	84.54
Jaccard	93.96	89.94	87.70	84.53
Jaro-Winkler	77.44	77.27	77.19	76.89
Levenshtein	96.28	92.81	90.91	88.23
Soft Bag	93.56	89.82	87.50	84.46
Soft Cosseno	94.04	90.10	87.82	84.69
Soft Jaccard	94.05	90.14	87.86	84.73
Soft TF-IDF	94.36	90.61	88.26	85.22
TF-IDF	94.28	90.47	88.14	85.10

Tabela 4.11: Comparação de F_1 sobre Coleções Sintéticas da DBLP

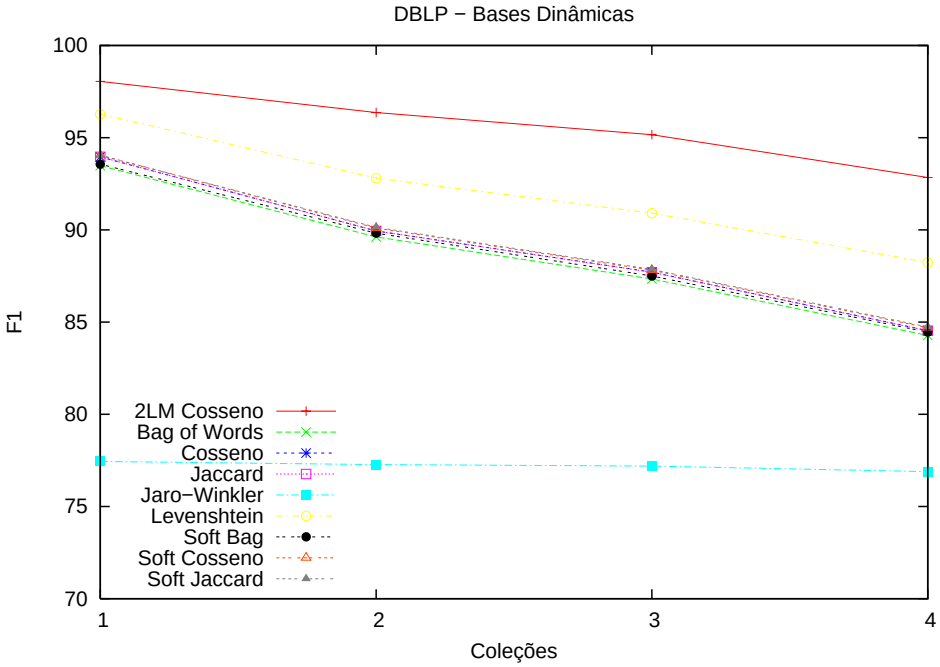

Figura 4.7: DBLP - Resultados sem IDF

Métrica	*Sint 1*			*Sint 2*			*Sint 3*			*Sint 4*		
	Token	*Soft*	*Dif*	*Token*	*Soft*	*Dif*	*Token*	*Soft*	*Dif*	*Token*	*Soft*	*Dif*
TF-IDF	94.28	94.36	0.08	90.47	90.61	0.14	88.14	88.26	0.12	85.10	85.22	0.12
Cosseno	93.96	94.04	0.09	89.94	90.10	0.16	87.71	87.82	0.11	84.54	84.69	0.15
Bag of Words	93.48	93.56	0.08	89.62	89.82	0.19	87.34	87.50	0.16	84.28	84.46	0.19
Jaccard	93.96	94.05	0.09	89.94	90.14	0.20	87.70	87.86	0.16	84.53	84.73	0.19

Tabela 4.12: Comparação de Métricas de Token com suas variantes Híbridas - DBLP

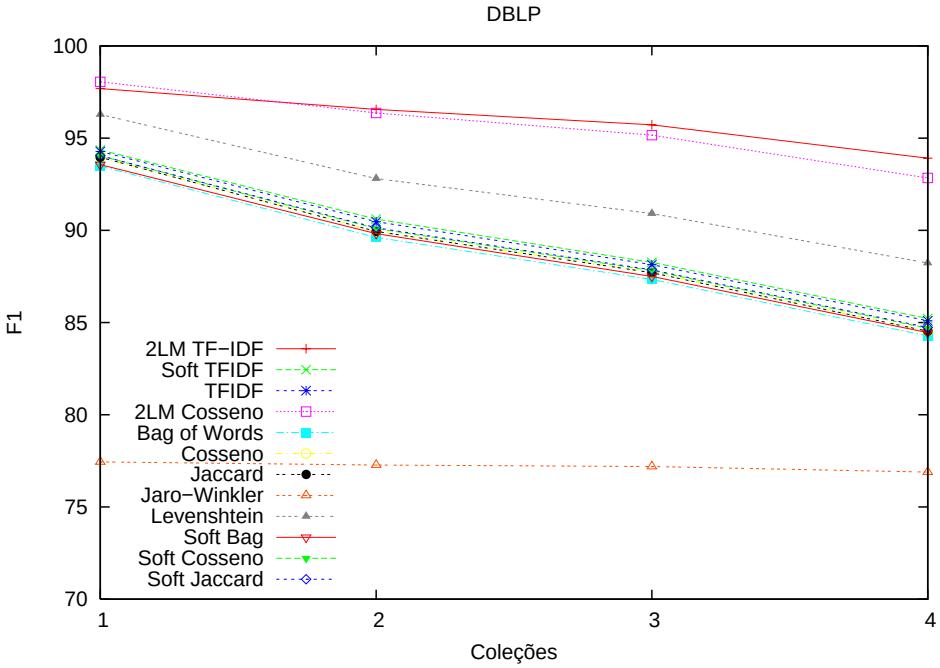

Figura 4.8: DBLP - Resultados

Capítulo 5

Conclusão e Trabalhos Futuros

Este trabalho realizou estudos sobre a utilização de funções de similaridade no processo de junção de diferentes *nomes* que representam o mesmo objeto do mundo real. Também foi verificado o uso de estruturas métricas para a poda de distâncias de computação.

Tomando-se como base estas pesquisas, a proposta desta dissertação foi realizar a junção por similaridade a partir do modelo em dois níveis $2LM$, onde relaxa-se a igualdade dos termos utilizando uma função de distância de edição no primeiro nível e calcula-se a similaridade final do objeto a partir de uma distância de *token* no segundo nível. Para isso, foi utilizada uma estrutura métrica para indexar os termos do vocabulário da coleção, possibilitando a poda por desigualdade triangular e assim economizando cálculos de distância.

Visando comparar os modelos propostos com os aqueles existentes na literatura, foram implementadas duas versões do $2LM$, ambas usando como métrica de distância de edição *Levenshtein*. Na primeira versão foi implementado o $2LM - TFIDF$ usando a similaridade do cosseno ponderada segundo a freqüência inversa dos termos na coleção (IDF). Neste caso, há necessidade de prévio conhecimento da coleção. A segunda versão $2LM - Cosseno$ utilizou a similaridade do cosseno não ponderada, não requerendo o pré-processamento dos IDFs.

Para comparar o modelo com os demais disponíveis na literatura, utilizou-se como métricas de distância de edição *Levenshtein* e $Jaro - Winkler$; como métricas de *token*, *Bag of Words*, *Jaccard*, *Cosseno não ponderado* e *TF-IDF*; e como métricas híbridas: *Soft Bag*, *Soft Cosseno*, *Soft Jaccard* e *Soft-TFIDF*.

Na coleção real da BDBComp, as métricas híbridas que utilizaram IDFs foram as que apresentaram melhores resultados. $Soft - TFIDF$ foi a que apresentou melhor eficácia, sendo

seguida bem de perto pelo modelo proposto $2LM - TFIDF$. Os resultados de TF-IDF foram ligeiramente inferiores aos demais, principalmente nos últimos pontos de revocação. Ressalta-se, que o modelo proposto, $2LM - TFIDF$ necessita de mais cálculos de distância para obter praticamente os mesmos resultados que $Soft - TFIDF$. Dentre as métricas que não requerem o pré-processamento de IDFs, o modelo proposto $2LM - Cosseno$ foi o que melhores resultados obteve, sendo seguido das demais métricas hibrídas e de token. As métricas baseadas em distância de edição, $Levenshtein$ e $Jaro - Winkler$ apresentaram os piores resultados nessa coleção.

Usando as coleções sintéticas geradas a partir da DBLP, às quais foram atribuídos erros crescentes de edição, supressão e abreviação, os modelos propostos obtiveram os melhores resultados, sendo $2LM - TFIDF$ seguido por $2LM - Cosseno$. Os resultados foram melhores nessas bases devido ao fato de terem sido atribuídos erros de edição a todos os termos de vários objetos, sendo esses casos ignorados pela política de poda adotada pelos demais modelos de token e híbridos. Entretanto, no mundo real não espera-se que bases estejam tão corrompidas; e, nesse caso, quando há possibilidade de pré-processamento de IDFs, seria mais viável a utilização de $Soft - TFIDF$. Contudo, quando as coleções são dinâmicas e não pretende-se reprocessá-la a cada inclusão de novos objetos, a utilização do modelo $2LM - Cosseno$ obteve bons resultados, mesmo na coleção BDBComp, que apresenta pouquíssimos erros.

Em resumo, pode-se concluir deste trabalho que a utilização de métricas que combinem tanto distâncias de edição quanto métricas de token são mais indicadas para a tarefa de junção por similaridade. Conclui-se ainda que o modelo proposto $2LM - TFIDF$ acrescenta melhorias na junção de coleções com muitos erros de edição. Dessa forma, quando não é possível saber antecipadamente a qualidade das coleções, esse modelo poderia ser adotado.

Observa-se que os modelos propostos também são aplicáveis a buscas por similaridade, podendo ser uma alternativa para buscas que, utilizando-se o Modelo Vetorial convencional, apresentaram poucos ou mesmo nenhum resultado.

5.1 Trabalhos futuros

Apresentaremos nesta seção algumas sugestões para trabalhos futuros, que poderão dar continuidade aos experimentos realizados para esta dissertação.

Sugere-se o uso de técnicas de *cluster* para a criação de *agrupamentos de similares* substituindo as *listas de similares*, o que diminuiria o tempo de manutenção das listas e o espaço em memória utilizado.

Também sugere-se a realização de experimentos com o uso de outras estruturas métricas, principalmente D-Index e família OMNI, que apresentaram bons resultados de poda em problemas de alta dimensionalidade [19, 22].

Além disso, seria interessante o uso de outras coleções reais de referência para avaliações do desempenho dos modelos, uma vez que a coleção da BDBComp apresentava pouquíssimos erros.

A combinação de diferentes métricas apresentadas sobre diferentes tipos de atributo e consolidação das similaridades parciais seria um trabalho muito útil para integração de bases XML.

Como o número de termos abreviados em nomes de pessoas é grande, sugere-se que sejam considerados similares, com um certo grau de ponderação, palavras abreviadas.

Referências Bibliográficas

[1] Baeza-Yates, R.; Ribeiro-Neto, B. *Modern Information Retrieval.*, Addison Wesley, 1999.

[2] Santon, G.; Buckley, C. *Term-Weighting Approaches in Automatic Text Retrieval*, In Proceedings of Information Processing Management, 24(5):513-523,1988.

[3] G. Salton. *The SMART Retrieval System: Experiments in Automatic Document Processing.* Prentice-Hall, Inc. Englewood Cliffs, 1971.

[4] Cohen, W.; Ravikumar, P.; Fienberg, S. *A Comparison of String Metrics for Matching Names and Records.* In Proceedings of KDD Workshop on Data Cleaning, Record Linkage and Object Consolidation, 13-18, 2003.

[5] Winkler, W. *The state of record linkage and current research problems.* Statistical Research Division, U.S. Census Bureau, Internal Revenu Service Publication, 1999.

[6] Cohen, W. *Data Integration using Similarity Joins and a Word-based Information Representation Language*, In Proceedings of ACM Transactions on Office Information Systems, 18(3):288-321, 2000.

[7] Bilenko, M.; Mooney, R.; Cohen, W.; *et al. Adaptive Name Matching in Information Integration.* In Proceedings of IEEE Intelligent Systems Special Issue on Information Integration on the Web, 18(5):16-23, 2003.

[8] Cohen, W. *Knowledge Integration for Structured Information Sources Containing Text*, In The SIGIR-97 Workshop on Networked Information Retrieval,1997.

[9] Tejada, Sheila; Knoblock, Craig; Minton, Steven. *Learning Object Identification Rules for Information Integration*, Information Systems Journal, 26(8):635-656, 2001.

[10] Beyer K., Goldstein J., Ramakrishnan R., Shaft U. *When is Nearest Neighbors Meaningful?*, In Proceedings of 7th International Conference of Database Theorie, 217-235, 1999.

[11] Cohen, W. *WHIRL: A Word-based Information Representation Language*, in Journal of Artificial Intelligence, 118:(163-196), Elsevier Science Publishers Ltd., 2000.

[12] Johnson, T. ; Dasu, T. *Data Quality nd Data Cleaning: An Overview*, In Proceedings of the 2003 ACM SIGMOD international conference on Management of data, 681-681, 2003.

[13] Dorneles, C.; Heuser, Carlos A.; Silva, Altigran; Moura, Edleno *et al. Measuring similarity between collection of values*, In Proceedings of 6th ACM International Workshop on Web Information and Data Management, 12-13, 2004.

[14] Ciaccia, P; Patella, M.; Zezula, P. *M-tree: An efficient access method for similarity search in metric spaces*, In Proceedings of the 23rd International Conference on Very Large Data Bases, 426-435, 1997.

[15] Chávez, E.; Navarro, G.; Baeza-Yates, R.; *et al. Searching in Metric Spaces*, In ACM Computing Surveys, 33(3):273-321, 2001.

[16] Monge, A.; Elkan, C.; *An efficient domain-independent algorithm for detecting approximately duplicate database records*, In Proceedings of the SIGMOD 1997 workshop on data mining and knowledge discovery, 1997.

[17] Smith, T. F.; Waterman, M. S. *Identification of Common Molecular Subsequences*, Journal of Molecular Biology, 147:195-197, 1981.

[18] Traina Jr.,C; Traina, A., *et al. Slim-trees: High Performance Metric Trees Minimizing Overlap Between Nodes* C., In Proceedings of the 7th International Conference on Extending Database Technology, 51-65, 2000.

[19] Dohnal, V.; Gennaro, C.; Savino, P. *et al. A Metric Index for Approximate Text Management*, In Proceedings of Information Systems and Databases. Anaheim - Calgary - Zurich : ACTA Press, 37-42, 2002.

[20] Dohnal, V.; Gennaro, C.; Zezula, P. *D-Index: Distance Searching Index for Metric Data Sets*, in Proceedings of Multimedia Tools Appl: Kluwer Academic Publishers, 21:9-33, 2003.

[21] Yianilos, P. *Data Structures and Algorithms for Nested Neighbor Search in General Metric Spaces*. Proc. 5th ACM-SIAM Symp. Discrete Algorithms, 311-321, 1993.

[22] Santos Filho, R; Traina, A. ; Traina Jr., C; Faloutsos,C. *Similarity search without tears: The OMNI family of all-purpose access methods*, In Proceedings of the 17th International Conference on Data Engineering, 623-630, 2001.

[23] Bozkaya, T.; Özsoyoglu, Z. emphDistance-based indexing for high-dimensional metric space. In Proceedings of the 1997 ACM SIGMOD international conference on Management of data, 357-368, 1997.

[24] Burkhard, W. ; Keller, R. *Some approaches to bestmatch file searching*. In Communications of ACM, 16(4):230-236, 1973.

[25] Digout, Christian; Coman, Alexandru; Nascimento, Mario. *Similarity Search and Dimensionality Reduction: Not all Dimensions are Equally Useful*, In Proceedings of 9th International Conference on Database Systems for Advanced Applications, 831-842, 2004.

[26] Kukich, K. *Techniques for automatically correcting words in text*, In Proceedings of the 1993 ACM conference on Computer science, 515, 1993.

[27] Golgher, Paulo B. ; Silva, Altigran ; Laender, Alberto; Ribeiro-Neto,Berthier. *Bootstrapping for example-based data extraction*. In Proceedings of 10th ACM International Conference on Information and Knowledge Management, 371-378, 2001.

[28] Gotoh, O. ; *An Improved Algorithm for Matching Biological Sequences* Journal of Molecular Biology. 162:705-708, 1982.

[29] Guttman, Antonin. *R-Trees: A Dynamic Index Structure for Spatial Searching*. In Proceedings of the ACM SIGMOD Conference, 47-57, 1984.

[30] Sellis, Timos;Roussopoulos, Nick; Christos Faloutsos, Christos.*The R+-Tree: A Dynamic Index For Multi-Dimensional Objects*, In Proceedings of the 15th International Conference on Very Large Data Bases, 507-518, 1987.

[31] Beckmann, Norbert; Kriegel, Hans-Peter; Schneider, Ralf; Seeger,Bernhard . *The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles*, SIGMOD Records, 19(2):322-331, 1990.

[32] Elmasry, Navathe; *Fundamentals of Database Systems*, 3a. Edição. Editora LTC, Addison-Wesley Pub, 2002.

[33] Rijsbergen, C. Van; *Information Retrieval*, Segunda Edição, Dept. of Computer Science, University of Glasgow, 1979.