ADRIANA ZANELLA MARTINHAGO

CUSTOMIZAÇÃO EM AMBIENTES DE QUALIDADE DE DADOS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador: Prof. Dr. Marcos Sfair Sunye

CURITIBA

2006

ADRIANA ZANELLA MARTINHAGO

CUSTOMIZAÇÃO EM AMBIENTES DE QUALIDADE DE DADOS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador: Prof. Dr. Marcos Sfair Sunye

CURITIBA

2006

"Grandes realizações não são feitas por impulso, mas por uma soma de pequenas realizações."

(Vicent Van Gogh)

Dedico este trabalho a meus pais que amo muito,
José Carlos Martinhago e *Eide Zanella Martinhago*,
e aos meus irmãos pelo apoio e carinho que tiveram
comigo durante esta caminhada.
Também dedico ao meu namorado, *Douglas*, pela
compreensão, paciência e por estar sempre ao meu
lado em todos os momentos.

Agradecimentos

Inicialmente gostaria de agradecer ao meu orientador Marcos Sfair Sunye pela confiança depositada em mim e por todas as oportunidades.

Agradeço aos amigos que fiz durante o mestrado que foram muito importantes nesta etapa da minha vida e os quais sentirei muitas saudades: Andrea, Pryscila, Luciane, Regina, Ledyvania, Rodrigo, Marcelo, Ricardo , Pedro, Marcos, Eduardo, Sheila, ...

À todos que de uma maneira ou outra passaram pelo meu caminho e me ajudaram de alguma forma, principalmente a minha família que me sempre me apoiou em todos os momentos.

À Deus pela vida, por ter me dado forças para vencer mais esta batalha e por colocar no meu caminho todas estas pessoas maravilhosas.

Sumário

LISTA DE FIGURAS **vii**

LISTA DE TABELAS **viii**

RESUMO **ix**

ABSTRACT **x**

1 Introdução **1**

1.1 Organização do Trabalho 4

2 Qualidade de Dados **5**

2.1 Aproximações para o Estudo de Qualidade de Dados 6

2.1.1 Aproximação Intuitiva 6

2.1.2 Aproximação Teórica 7

2.1.3 Aproximação Empírica 7

2.2 Avaliação da Qualidade de Dados 7

2.3 Duplicação de Registros 8

3 O Ambiente FEBRL **10**

3.1 Limpeza de Dados 11

3.1.1 Limpeza 12

3.1.2 Identificação 13

3.1.3 Segmentação 15

3.2 Aproximação de Registros 19

3.2.1 Indexação 20
3.2.2 Comparação 21
3.2.3 Classificação 22
3.3 Formatos de Saída 23
3.3.1 Histograma 23
3.3.2 Lista de Detalhes 23
3.3.3 Lista de Identificadores 24

4 O Uso do FEBRL em Registros Brasileiros 26
4.1 Tabelas utilizadas pelo FEBRL 27
4.1.1 Listas de correção 28
4.1.2 Tabelas *look up* 29
4.2 Treinamento HMM 35
4.3 Estudo de Caso 39
4.3.1 Ambiente de Teste 41
4.3.2 Testes e Resultados 41

5 Processamento Paralelo 46
5.1 Bibliotecas de troca de mensagem 47
5.2 Módulo Pypar 48
5.3 O processamento paralelo no ambiente FEBRL 48
5.3.1 Problema encontrado 49

6 Conclusão 50
6.1 Trabalhos Futuros 51

REFERÊNCIAS BIBLIOGRÁFICAS 55

ANEXO A - ESTADOS DO MODELO ESCONDIDO DE MARKOV 56

APÊNDICE A - FORMATO DE SAÍDA: LISTA DE DETALHES 58

APÊNDICE B - TABELAS LOOK UP 60

APÊNDICE C - LISTA DE CORREÇÃO PARA ENDEREÇO **65**

APÊNDICE D - TESTES DAS VARIÁVEIS DE BLOCO **66**

APÊNDICE E - ARQUIVOS DE SAÍDA DO TESTE FINAL **68**

Lista de Figuras

3.1 Exemplo de uma parte da lista de correção de endereço. 13

3.2 Exemplo de uma parte da tabela *look up* para tipos de logradouros. 14

3.3 Campos de saída suportados pelo FEBRL. Fonte [6] 15

3.4 Exemplo simples de um HMM de endereço. Fonte [9] 17

3.5 Matriz de probabilidade de emissão. Fonte [9] 18

3.6 Exemplo de um histograma de peso. Fonte [6] 24

3.7 Exemplo de uma lista de identificadores. 25

4.1 Exemplo de um arquivo de lista de correção de nome 28

4.2 Exemplo de uma parte da tabela *look up* para nomes próprios femininos . . 30

4.3 Lista com os símbolos de identificação suportados pelo FEBRL. Fonte [6] . 31

4.4 Tabela *look up* de títulos. 32

4.5 Tabela *look up* de prefixos de nomes. 32

4.6 Tabela look up de miscêlanias para nomes. 33

4.7 HMM de nome resultante do treinamento de dados. 38

4.8 HMM de endereço resultante do treinamento de dados. 39

5.1 Diferença entre o processamento sequencial e paralelo 46

Lista de Tabelas

3.1 Uma variação de valores em conjunto de dados. 12

4.1 Tabela com exemplos de registros australianos 26

4.2 Tabela com exemplos de registros brasileiros 27

4.3 Relação das funções de comparação. 42

4.4 Resultados dos testes de métodos de indexação e classificação. 43

4.5 Comparação entre conjunto de dados SIBI e australiano. 45

Resumo

A qualidade de dados é um tema que cresce em importância a medida que aumentam em número e volume as bases de dados existentes. Entre os seus principais desafios está a deduplicação, que busca reduzir a existência de registros distintos na base que representam a mesma entidade do mundo real. Outro desafio igualmente importante é o desempenho, visto que o problema envolve a comparação entre milhões de registros. O ambiente FEBRL foi desenvolvido com o objetivo de apoiar as tarefas de deduplicação usando paralelismo. Este trabalho apresenta um estudo do ambiente FEBRL e as adaptações que foram feitas neste ambiente para que trabalhasse corretamente com conjunto de dados brasileiros, pois ele está padronizado para ser usado em conjunto de dados australianos. Devido a importância do paralelismo no processo de deduplicação de registros, é apresentado também neste trabalho o funcionamento do paralelismo no ambiente FEBRL e alguns problemas encontrados.

Abstract

The data quality is a theme that becomes more important as long as the quantity and volume of the extand databases increase. Among its major challenges is the deduplicate, which seeks for reduction of the distinct records in the base, but represents the same entity of the real world. Another challenge as important as the first one is the performance, since the problem involves the comparison between bases with millions of the records. The FEBRL environment was developed with the purpose of supporting the deduplicate tasks using parallelism. This research presents a study about the FEBRL environment and the adaptation that was made in this environment to make it work properly together with Brazilians data sets, because it is standardized to be used with Australians data sets. And due to the importance of the parallelism in the deduplicate process of the records, we also decided to present in this research the functioning of the parallelism in the FEBRL environment and the problems that were found.

Capítulo 1

Introdução

O crescimento das tecnologias nos últimos 20 anos fez com que uma infinidade de bytes fossem criados, tratados e armazenados por diferentes organizações e empresas. No início, sua principal preocupação relacionava-se aos custos associados com o armazenamento de dados e informações.

Hoje existe uma preocupação maior com a incerteza a respeito da qualidade de dados, pois uma qualidade de dados ruim significa que a informação é imprecisa. Informações imprecisas podem significar desperdício de dinheiro e de recursos.

Como sintomas de uma baixa qualidade de dados podemos citar: relatórios imprecisos, pedidos reprocessados, custos de compras superiores à média do mercado, desembolso inesperado de caixa, duplicação de informações, reclamação de clientes ou colaboradores, desperdício de dinheiro com mala direta e *telemarketing* entre outros. Essa baixa qualidade de dados prejudica uma organização em todos os sentidos, tanto internamente (ex: desperdício) quanto externamente (ex: relação com clientes).

Um estudo da *PrinceWaterhouse Coopers*, disponível em [12] foi feito com 600 empresas americanas, inglesas e australianas. Esse estudo revelou que cerca de 75% delas tiveram problemas inesperados em decorrência da baixa qualidade de dados. O *The Data Warehousing Institute* (TDWI) estimou que a baixa qualidade dos dados de clientes das organizações americanas gera custos de 611 bilhões de dólares por ano.

O problema da qualidade de dados começou a ser foco do desenvolvimento de sistemas

específicos na década de 80, quando surgiu a primeira geração de ferramentas específicas para resolvê-lo. Foi a partir daí que surgiram termos como 'limpeza de dados', 'duplicação' e 'higienização de banco de dados', e também de metodologias para eliminar dados redundantes entre outros. Assim, tem-se a utilização destes procedimentos para normatizar e padronizar dados, efetuar correções, validações e certificações de endereços, deixando o Banco de Dados coerente e consistente.

Um dos principais problemas em qualidade de dados é a duplicação de registros, principalmente duplicação de registros aproximados, ou seja, registros que não são sintaticamente iguais, mas que representam a mesma entidade no mundo real e onde um único identificador não está disponível. Duplicação de registros aproximados acontece quando mais de um registro corresponde a mesma pessoa em um cadastro de clientes por exemplo. A solução para este problema tem se tornado uma tarefa muito importante em qualidade de dados e vem sendo foco de muitas pesquisas nos últimos anos.

A duplicação de registros em um conjunto de dados pode acontecer por várias razões: erro de digitação, falta de padrão das abreviações (ex: R ou Rua para cadastrar um endereço; Dr ou Doutor usado em titulação), erro na entrada dos dados (ex: o dado pode ser digitado, capturado via reconhecimento de fala automático), entre outros.

A maioria dos erros contidos em um conjunto de dados são erros de digitação, ou seja, erros tipográficos. Um estudo feito em [11] descobriu que 80% dos erros tipográficos são erros simples como inserção, remoção ou substituição de um caractere ou a troca de dois caracteres adjacentes.

Uma das soluções encontradas na literatura para resolver o problema de duplicação de registros é a aproximação de registros. Trata-se do processo de comparação de registros em um (deduplicação) ou mais (*record linkage*) conjuntos de dados no esforço de determinar quais pares de registros representam a mesma entidade do mundo real [13]. O processo de aproximação de registros somente detecta os registros que se referem a mesma entidade (possíveis duplicados), este processo não elimina os mesmos.

A idéia da aproximação probabilística de registros foi introduzida por NewCome e Kennedy [20] em 1962 enquanto a fundamentação teórica foi produzida por Fellegi e

Sunter em 1969 [14]. A idéia básica é juntar registros por comparações de atributos comuns, que inclui identificadores pessoais (como nome, data de nascimento ou idade) e informações demográficas (como endereço ou cidade) [10].

Como exemplo para o uso de *record linkage* temos: um escritório de contabilidade que pode identificar se uma pessoa está registrada em múltiplos benefícios. Um exemplo interessante seria verificar dois conjuntos de dados, um com dados referentes a imposto de renda e outro com dados de pessoas cadastradas no projeto bolsa família. Seria possível usar *record linkage* para verificar se a pessoa que ganha bolsa família faz parte do conjunto de dados dos contribuintes do imposto de renda.

Neste trabalho foi utilizado apenas a deduplicação de registros, pois visa detectar registros duplicados em apenas um conjunto de dados.

São vários os algoritmos que se propõem a resolver o problema de duplicação de registros aproximados, mas dos algoritmos, frameworks e ambientes pesquisados, apenas o ambiente FEBRL propõe uma solução paralela, diminuindo assim o tempo de execução.

O ambiente de software FEBRL (*Freely Extensible Biomedical Record Linkage*), que foi desenvolvido na Austrália, tem como principal objetivo detectar registros relacionados com a mesma entidade no mundo real em um ou mais conjuntos de dados. A vantagem de se usar este ambiente para resolver o problema de detecção de registros é que ele é um sistema robusto, multiplataforma e pode ser executado paralelamente usando a tecnologia de clusters, workstations, entre outros.

Um problema observado em se usar a versão atual do FEBRL está no fato dele ser padronizado para trabalhar com o modelo australiano de identificação de pessoas. Isso dificulta a remoção dos registros duplicados no Brasil porque a escrita é realizada de forma diferente entre os países e o FEBRL segue toda uma estrutura de funcionamento que será mostrada no capítulo 3, tornando a detecção de registros duplicados complexa e ineficiente. Por exemplo, um endereço no estilo australiano seria '*17 epping street Smithfield*', ou seja número da residência, nome do logradouro, tipo do logradouro (*street, avenue*) e cidade. Um exemplo de endereço no estilo brasileiro seria 'Rua João Itiberê 295, Curitiba', ou seja, tipo do logradouro (rua, avenida), nome do logradouro, número da residência e cidade. Na

hora de executar a deduplicação, o sistema analisa a seqüência definida como se fosse do tipo australiana, que é totalmente diferente da brasileira, tornando o resultado impreciso.

O objetivo deste trabalho é mostrar uma solução de detecção de registros duplicados aproximados em conjunto de dados brasileiros usando o ambiente de software FEBRL. Para isso, algumas adaptações foram feitas neste ambiente de software para que o FEBRL fosse ajustado as necessidades dos conjuntos de dados brasileiros.

Inicialmente foi realizado um estudo da linguagem de programação *Phyton*, que é usada no código fonte do FEBRL. Em seguida foi realizado um estudo no código fonte dos módulos do ambiente FEBRL para saber como é o funcionamento, a padronização e aproximação dos registros, e por fim, foram realizadas as modificações necessárias.

Como estudo de caso para mostrar a viabilidade do uso do ambiente de software FEBRL brasileiros foi utilizado um conjunto de dados do Sistema de Bibliotecas da Universidade Federal do Paraná (SIBI-UFPR). Este conjunto de dados contém o cadastro dos alunos no sistema de bibliotecas da universidade.

Uma solução para o problema de duplicação de registros é de suma importância para organizações de todos os tipos que estão à procura de uma boa qualidade de dados, pois o problema de duplicação de dados aproximados pode atrapalhar no processo de qualidade da organização. E hoje, a qualidade é um fator competitivo no mercado para as organizações.

1.1 Organização do Trabalho

O trabalho está dividido da seguinte forma: no capítulo 2 serão apresentados os conceitos relativos a qualidade de dados; o capítulo 3 trata do funcionamento do pacote de software FEBRL; o capítulo 4 mostra como usar o ambiente FEBRL para buscar registros duplicados em conjunto de dados brasileiros e apresenta o estudo de caso; o capítulo 5 descreve com é o funcionamento do paralelismo no FEBRL e os problemas encontrados; no capítulo 6 são descritos a conclusão e os trabalhos futuros.

Capítulo 2

Qualidade de Dados

Comparando com os conceitos de Banco de Dados como segurança e integridade, que tem sido estudado em detalhes desde a introdução da tecnologia de Banco de Dados Relacionais, a noção de qualidade de dados apareceu somente durante os últimos 20 anos e mostra um firme aumento no interesse [15].

Esse aumento no interesse pela questão da qualidade de dados é devido à sua importância dentro da organização, a falta de qualidade de dados pode atrapalhar o desenvolvimento e o crescimento da organização. Entre os problemas que a falta de qualidade pode trazer podemos citar a falta de confiança do cliente, tomadas de decisões equivocadas, perda de oportunidades de negócios, entre outros.

O termo "qualidade de dados" relaciona-se a vários conceitos, tornando difícil sua definição. Segundo [5], qualidade de dados é a descrição de um dado completo, consistente, exato e preciso. Além disso, a qualidade de dados pode ser descrita com base no que é bom o suficiente para nossas necessidades, ou seja, qualidade de dados é simplesmente garantir que esses dados atendam as necessidades do negócio, onde são utilizados.

Existem dois tipos de avaliações da qualidade de dados que podem ser feitas em Banco de Dados: a avaliação quantitativa (ou objetiva) e a avaliação qualitativa (ou subjetiva) [25].

Na avaliação quantitativa, o grau de qualidade depende somente do dado que esta sendo observado e não do ponto de vista do observador. Geralmente este tipo de avaliação

é realizado por ferramentas automatizadas.

Na avaliação qualitativa, o grau de qualidade depende do ponto de vista do observador e também dos dados que estão sendo observados. Esta avaliação pode ser feita por especialistas ou usuários do Banco de Dados ou em conjunto com o uso de ferramentas automatizadas.

2.1 Aproximações para o Estudo de Qualidade de Dados

As pesquisas sobre qualidade de dados apresentam diferentes abordagens quanto a definição dos atributos para a avaliação da qualidade [2]. Três aproximações são usadas na literatura [29]: intuitiva, teórica e empírica.

2.1.1 Aproximação Intuitiva

A aproximação intuitiva é realizada quando a seleção dos atributos de qualidade de dados para algum estudo particular é baseado em experiências de pesquisas ou conhecimentos intuitivos sobre quais atributos são importantes [29]. A maioria dos estudos sobre qualidade de dados se encaixa nesta aproximação.

A vantagem de se usar uma aproximação intuitiva está em cada estudo poder selecionar os atributos mais relevantes para seus objetivos particulares [29].

Entre os atributos convencionais, também chamados de métricas de qualidade de dados, freqüentemente encontrados na literatura podem ser citados:

- acurácia (precisão): o grau de exatidão e precisão com que os dados de interesse do mundo real são representados em um sistema de informação;
- completeza: porcentagem de elementos de dados que tem valores instanciados;
- atualidade: o grau de identificação para saber se o dado gravado é atual;
- consistência: porcentagem de valores combinados entre tabelas/registros/arquivos;
- validação: porcentagem de dados que possuem valores que condizem com seu respectivo domínio de valores permitidos.

Segundo [5], geralmente, o uso de atributos para determinar a qualidade de dados requer a execução de cinco atividades principais:

1. Determinar a abordagem a ser usada para medir qualidade de dados;
2. Aplicar o conjunto de regras aos arquivos/tabelas/registros que devem ser validados;
3. Indicar dados suspeitos em relatório de erros;
4. Validar e refinar o conjunto de regras;
5. Desenvolver relatórios de métricas para categorizar os problemas de qualidade de dados.

Esse ciclo de atividades pode ser realizado de uma forma geral, independente do mecanismo usado para implementar qualidade de dados. O que deve ser realizado com cuidado é a definição do conjunto de regras a serem usadas [5].

2.1.2 Aproximação Teórica

Uma aproximação teórica para qualidade de dados foca em como os dados podem tornar-se deficientes durante o processo de produção dos dados [29]. Embora essa aproximação seja freqüentemente recomendada, as pesquisas oferecem poucos exemplos.

2.1.3 Aproximação Empírica

Uma aproximação empírica para análise de qualidade de dados determina as características que os clientes usam para avaliar se dados são adequados para serem usados em suas tarefas [29]. Por isso, essas características não podem ser teoricamente determinadas ou intuitivamente selecionadas pelos pesquisadores.

2.2 Avaliação da Qualidade de Dados

Medir qualidade de dados é determinar a natureza e magnitude exata dos problemas dos dados a partir de valores armazenados nas tabelas/arquivos de uma aplicação [28].

Segundo [23], a qualidade de dados é medida de acordo com as visões apresentadas por uma informação do sistema e o mesmo dado no mundo real. Um sistema de qualidade de dados 100%, ou seja com qualidade total, indica que as visões de dados estão de perfeito acordo com o mundo real e uma avaliação de qualidade de 0%, indica uma qualidade de dados muito ruim, quando a maioria dos registros cadastrados não representam as respectivas entidades do mundo real. A real dificuldade em qualidade de dados está na mudança, pois os dados em nosso Banco de Dados são estáticos enquanto os dados referentes no mundo real são dinâmicos. Essa mudança faz com que os dados em nosso Banco de Dados acabem ficando desatualizados gerando assim uma qualidade de dados ruim.

2.3 Duplicação de Registros

São muitos os problemas que geram a falta de qualidade de dados em um Banco de Dados entre eles podemos citar informações fictícias ou incompletas, desatualização de registros, duplicação de registros, duplicação de registros aproximados, entre outros.

Dos problemas citados um dos principais e que vem sendo muito pesquisado é a duplicação de registros aproximados, ou seja, registros que não são necessariamente idênticos textualmente, mas que representam a mesma entidade no mundo real.

Quando um Banco de Dados contem registros que foram coletados de múltiplas fontes de informação, ou quando houve a integração de vários Bancos de Dados em um só, este freqüentemente inclui valores duplicados [4]. Um Banco de Dados também pode conter registros duplicados por outros motivos como: erros tipográficos, erros na captura de dados, o uso de diferentes códigos ou falta de padrão em abreviações ou diferenças entre interfaces de gravação. O Banco de Dados deve ser limpo, ou seja, ter os registros duplicados removidos.

Existem diversos algoritmos, *frameworks*, pacotes, ambientes e métodos para detectar de registros duplicados publicados na literatura.

Em [4] os autores apresentam um *framework* para melhorar a detecção de registros duplicados usando medidas de texto similares. Eles propuseram empregar funções de

distância de texto para cada campo e introduzir uma variação da distância de texto, editada em um algoritmo de instrução '*Expectation-Maximization*'.

Existe também o método de combinação fonética, o qual é usado em aplicações como recuperação de nome, onde a escrita pode ser similar a pronuncia. Sendo assim, é feita uma aproximação dos registros que são similares foneticamente e que na maioria das vezes são registros que se parecem gramaticalmente. Mais detalhes podem ser vistos em [30].

Um método bem conhecido, chamado '*sorted neighborhood*' [16], consiste em três passos: criar chaves, selecionar dados e juntar os registros. O método não se mostra muito eficiente em detectar registros duplicados em casos onde os dados fontes não contêm um campo de chave primária de referência. Uma variação para esse método é chamada '*multi-pass sorted neighborhood*'. Nesta variação o problema anterior é resolvido.

Como mostrado são vários os métodos e algoritmos para detecção de registros duplicados, todos os pesquisados possuem um tempo de execução alto quando a base de dados possui milhares de registros, mas nenhum dos algoritmos e dos métodos pesquisados possui uma versão que trabalhe paralelamente diminuindo o tempo de execução.

Capítulo 3

O Ambiente FEBRL

O ambiente FEBRL (*Freely Extensible Biomedical Record Linkage*), está atualmente sendo desenvolvido na Austrália como parte de um projeto colaborativo sendo contratado pelo *Australian National University* (ANU), *Data Mining Group*, *Centre for Epidemology* e por pesquisas no departamento de saúde de *New South Wales* [6].

O objetivo deste ambiente é desenvolver melhores técnicas de probabilidades de limpeza de dados também chamada de padronização, deduplicação e aproximação de registros que combinam métodos probabilísticos clássicos com métodos determinísticos. Visa também uma melhor qualidade da aproximação de registros em conjuntos de dados de todos os tamanhos reduzindo o tempo de execução desses procedimentos.

O FEBRL está implementado em um projeto orientado a objeto com vários módulos com rotinas para tarefas específicas [6].

FEBRL está escrito na linguagem de programação *Python*, uma linguagem de código fonte aberto [17]. *Python* é uma linguagem de alto nível, interpretada, orientada a objetos com uma semântica dinâmica. Sua estrutura de alto nível, combinada com sua tipagem de amarração dinâmica, a faz muito atrativa para o desenvolvimento de aplicativos de grande porte assim como para seu uso como linguagem de *script* [22].

O FEBRL suporta paralelismo fazendo bom uso de modernas plataformas de computação paralela de alto desempenho, tais como *clusters* ou *workstations*, servidores de multiprocessadores ou supercomputadores [6].

O paralelismo acontece de forma transparente para o usuário e executando o FEBRL em paralelo os problemas são resolvidos com um tempo de execução bem menor que em modo seqüencial. O padrão de troca de mensagens utilizado no FEBRL é o *Message Passing Interface* (MPI) uma biblioteca de suporte a programação baseada em trocas de mensagens [19].

O FEBRL utiliza várias técnicas de aproximação de registros para detectar registros duplicados, pois na maioria dos casos, não existe um identificador comum ou chave primária compartilhada pelos conjuntos de dados [6].

A aproximação de registros se refere ao processo de juntar registros que se relacionam com a mesma entidade ou evento em um ou mais conjuntos de dados [9]. O FEBRL usa o termo deduplicação (*deduplication*), quando se quer buscar a mesma entidade em apenas um conjunto de dados. Usa o termo *record linkage* quando a busca pela entidade é feita em dois ou mais conjuntos de dados.

Os atributos usados para aproximação de registros podem ser categorizados em cinco classes: nomes, endereços, datas e horas, atributos categóricos (tais como sexo ou nacionalidade) e quantidades escalares (tais como altura ou peso) [6]. A versão atual do FEBRL possui facilidades específicas para o processamento de nomes, endereços e datas.

Dois passos devem ser seguidos para detectar registros duplicados no FEBRL. O primeiro passo é o de limpeza dos dados também chamado de padronização e o segundo passo é a aproximação de registros (deduplicação ou *record linkage*) propriamente dita, ou seja, a busca pelos registros duplicados.

3.1 Limpeza de Dados

O exemplo da Tabela 3.1, mostra uma tabela com uma evidente variação na formatação dos registros, muito comum em Bancos de Dados que foram integrados de outros Banco de Dados. Esta variação junto com diversos outros tipos de erros de digitação ou de falta de padrões em abreviaturas pode frustrar as tentativas de se detectar corretamente os registros duplicados. Por esse motivo deve-se utilizar a limpeza de dados para padronizar os registros, deixando a busca pelos registros duplicados mais simples.

Ao analisar a Tabela 3.1 pode-se perceber que os registros com Id 01 e 04 correspondem a mesma entida do mundo real gerando uma duplicação de informações.

Id	Nome	Sexo	Endereço	Cidade
01	Mariane da Silva Batista	F	Rua das Flores, 333	Florianópolis
02	Eduardo do Nascimento	M	Rua Prof Roberto Souza	Curitiba
03	Ana Carolina Pereira	2	Rua João Negrão	São Paulo
04	Mariane S. Baptista	feminino	R. Flores, 333	Floripa
05	Roberto F. Afonso	1	Av. Pres. Getúlio Vargas, 111	Vitoria

Tabela 3.1: Uma variação de valores em conjunto de dados.

A limpeza ou padronização dos dados tem como objetivo transformar a informação armazenada no conjunto de dados original em uma forma definida [6], ou seja, consiste em limpar e padronizar todos os campos de forma que os registros fiquem consistentes.

Para fazer a limpeza e padronização dos registros no FEBRL são seguidos três etapas sequenciais: limpeza, identificação e segmentação [6].

3.1.1 Limpeza

Primeiramente na etapa de limpeza dos dados, a entrada (que pode ser um nome ou endereço) é convertida para letras minúsculas. O próximo passo é utilizar as listas de correções para corrigir eventuais erros e remover caracteres indesejados.

Estas listas de correções são formadas por pares de palavras 'valor substituto:valor atual'. Se uma palavra da entrada é encontrada na lista de correção como 'valor atual', esta palavra será substituida pelo 'valor substituto'. Por exemplo, podemos substituir todas as variações para a palavra 'sem número' (usada em endereço) como 's/n', 'sn', 'sno' pela própria palavra 'sem número', ou seja, a palavra 'sem número' será o 'valor substituto' enquanto as suas variações serão o 'valor atual'. A Figura 3.1 mostra um exemplo da parte da lista de correção para endereço que especifica o exemplo apresentado. A lista de correção completa para endereço pode ser consultada no Apêndice C.

A saída desta etapa é uma nova lista de palavras, onde todas as ocorrências de palavras encontradas nas listas de correções foram substituídas pela palavra correta correspondente. O tamanho da lista de palavras de saída pode ser diferente da palavra de entrada.

```
' sem numero ' := ' s/n ', ' s / n ', ' sn ', 'sno'
        ' e ' := '+', '&'
        ' - ' := '-', '/'
        ' | ' := '<', '>', '(', ')', '[', ']', '{', '}', '"', "'", '|'
```

Figura 3.1: Exemplo de uma parte da lista de correção de endereço.

Supondo que nossa entrada seja : 'Av. Pres. Getúlio Vargas, 111' no final desta etapa ficará : 'av pres getulio vargas 111'. Como se pode perceber a saída desta etapa possui as palavras com letras minúsculas sem os pontos e acentos apresentados na entrada.

3.1.2 Identificação

Na etapa de identificação a entrada (saída da etapa anterior) é separada em uma lista removendo os espaços em branco. Por exemplo, a entrada 'av pres getulio vargas 111' é separada em uma lista contendo cinco palavras ['av', 'pres', 'getulio', 'vargas', '111'].

Usando tabelas chamadas *look up* e algumas regras de codificação, cada elemento da lista é especificada com um ou mais símbolos de identificação (*tags*). As regras de codificação incluem símbolos de identificação para elementos como vírgula, hífen, número, palavras alfanuméricas, enquanto as tabelas *look up* incluem os símbolos de identificação para nomes próprios, sobrenomes, cidades, nome da logradouro, cep, entre outros.

As regras de codificações usadas pelo FEBRL são implementadas em seu código fonte enquanto as tabelas *look up* são arquivos separados que podem facilmente ser modificados pelo usuário.

As tabelas *look up* funcionam da mesma maneira que as listas de correções, possuem pares de palavras 'valor substituto:valor atual'. Se uma palavra da entrada é encontrada na tabela *look up* como 'valor atual', esta será substituida pelo 'valor substituto' e será atribuida a ela o símbolo de identificação correspondente a tabela *look up*, onde a palavra da entrada foi encontrada.

Os símbolos de identificação determinam o que aquela palavra significa. Por exemplo: o símbolo de identificação utilizado para a palavra 'avenida' é 'WT' que significa '*wayfare type*', ou seja, o tipo do logradouro. Uma lista dos símbolos de identificação suportados

pelo FEBRL pode ser vista no Anexo A.

Cada símbolo de identificação possui a sua tabela *look up*. Um exemplo de uma parte da tabela *look up* para tipo de logradouro pode ser visto na Figura 3.2. A tabela *look up* completa para tipo de logradouro usada pode ser consultada no Apêndice B.

```
<WT>  # Símbolo de identificação para tipo de logradouro

      acesso :
     alameda : al
        area :
     avenida : av, aven
     arraial :
```

Figura 3.2: Exemplo de uma parte da tabela *look up* para tipos de logradouros.

É possível que uma palavra da lista esteja presente em mais de uma tabela *look up*, conseqüentemente esta palavra receberá mais que um símbolo de identificação. Por exemplo, a palavra 'Morretes' pode estar presente na tabela *look up* de nomes de logradouros e pode estar presente também na tabela *look up* nome de localidades.

Quando uma palavra não é encontrada em nenhuma tabela *look up* e não se encaixa em nenhuma das regras de codificação, ela recebe como símbolo de identificação as letras UN (do inglês *unknown*, desconhecido).

Esta etapa de identificação é importante para corrigir erros de digitação e erros relacionados à falta de padrão de abreviaturas.

A saída desta etapa é uma lista de palavras e a correspondente lista de símbolos de identificação como mostra o exemplo abaixo:

lista de palavras: ['avenida', 'presidente_getulio_vargas', '111']

lista de símbolos: ['WT', 'WN', 'NU']

Onde WT (*wayfare type*) significa tipo do logradouro, WN (*wayfare name*) é o nome do logradouro e NU é o número da residência. No exemplo mostrado as palavras 'presidente getulio vargas' aparecem juntas na tabela *look up* de nomes de logradouros, cujo símbolo de identificação é WN. Quando isso acontece, o FEBRL considera essas palavras como se fossem uma só, com apenas um símbolo de identificação para elas.

3.1.3 Segmentação

A terceira e última etapa da limpeza dos dados é a segmentação. Nesta etapa os símbolos de identificação são usados para segmentar os elementos da palavra de entrada dentro de campos de saídas corretos [6]. Os campos de saídas suportados pelo FEBRL são mostrados na Figura 3.3.

A etapa de segmentação é necessária porque várias tabelas *look up* são usadas e pode acontecer dos símbolos de identificação finais ficarem incorretos, principalmente quando uma palavra recebe mais de um símbolo de identificação. Resumindo: esta etapa de segmentação é usada para corrigir possíveis símbolos de identificação incorretos.

Name	Address	Date	Phone number
Title	Wayfare number	Day	Country code
Gender guess	Wayfare name	Month	Country name
Given name	Wayfare type	Year	Area code
Alternative given name	Wayfare qualifier		Number
Surname	Unit number		Extension
Alternative surname	Unit type		
	Property name		
	Institution name		
	Institution type		
	Post address number		
	Post address type		
	Postcode		
	Locality name		
	Locality qualifier		
	Territory		
	Country		
	Address HMM probability		

Figura 3.3: Campos de saída suportados pelo FEBRL. Fonte [6]

Para fazer a segmentação pode ser usado um dos dois modelos implementados no FEBRL: o Modelo Baseado em Regras de Aproximação ou o Modelo Escondido de Markov (*Hidden Markov Model* - HMM). Ambos os modelos trabalham com os campos de saída da Figura 3.3.

Neste trabalho foi utilizado o Modelo Escondido de Markov porque apresenta melhores resultados. Experimentos feitos em [7] e mostram que se comparado com os Modelos Baseado em Regras de Aproximação, o Modelo Escondido de Markov é mais fácil e mais

flexível de se usar. Além disso em dados complexos produz resultados mais exatos.

O HMM é uma teoria matemática que começou sendo muito utilizada no problema de reconhecimento de fala. Desde então, essa teoria vem sendo utilizada em vários tipos de aplicações [3], principalmente no problema de padronização e limpeza de registros.

Os HMMs foram introduzidos e estudados no ano de 1960 e no início de 1970 e tem se tornado popular de modo crescente nos últimos anos. Segundo [27] há duas fortes razões para isto, os modelos são muitos ricos em estruturas matemáticas e por isso podem formar as bases teóricas para usar uma grande extensão de aplicações; os modelos, quando aplicados corretamente trabalham muito bem na prática para diversas aplicações importantes.

Segundo [9] o Modelo Escondido de Markov é uma máquina de estados finitos probabilísticos, compreendendo um conjunto de símbolos de identificação, um conjunto de estados não observados (escondidos), uma matriz de probabilidades de transição entre os estados escondidos e uma matriz de probabilidades de emissão que é uma relação entre os símbolos de identificação e os estados escondidos. Existem dois estados especiais no HMM, o estado inicial e o estado final. Este são estados virtuais, ou seja, eles não se encontram no arquivo relacionado ao HMM que é gerado pelo treinamento de dados.

A mesma sequência de saída pode ser gerada por vários caminhos conectados a um Modelo Escondido de Markov com diferentes probabilidades.

O Modelo Escondido de Markov tem o objetivo de determinar a seqüência de estados escondidos mais parecida com a seqüência de símbolos apresentada [9].

No caso de endereços, podemos supor que o HMM usado para modelar possui estados finitos para cada segmento de um endereço, tais como: número da residência, nome do logradouro, tipo do logradouro, localidade e cep.

A entrada é tratada como uma seqüência ordenada de símbolos de identificação (saída da etapa de identificação) e assumimos que cada símbolo é emitido por um estado escondido. Depois são feitos testes para determinar qual, de um grande número de possibilidades de arranjos e estados escondidos, é a mais parecida com a seqüência de símbolos de identificação [9].

A Figura 3.4 mostra um exemplo simples de HMM para endereço, onde os nós retangulares representam os estados escondidos e os números indicados nas arestas são referentes as probabilidade de transição.

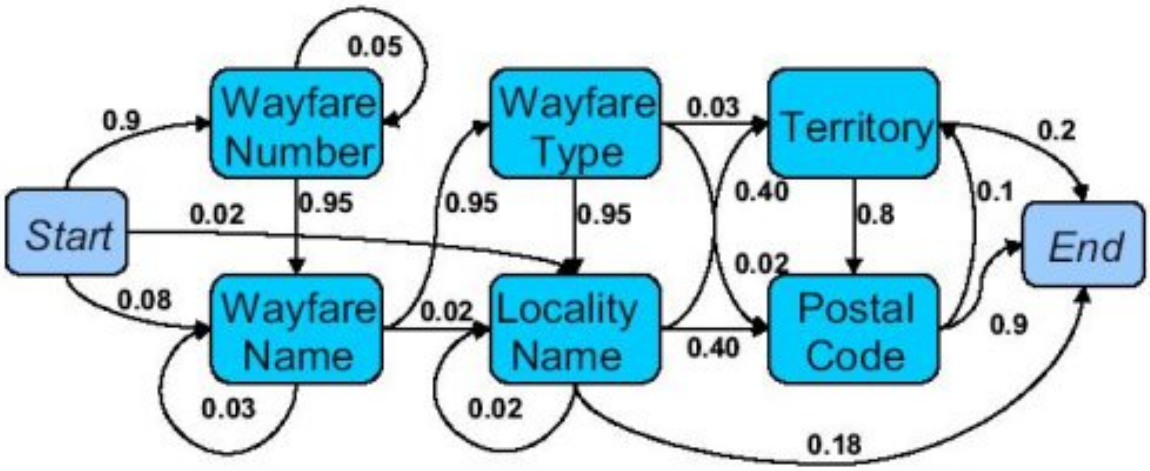


Figura 3.4: Exemplo simples de um HMM de endereço. Fonte [9]

Para melhor entender o funcionamento dos Modelos Escondidos de Markov no FEBRL, considere o exemplo de endereço australiano: '17, Epping st Smithfield New South Wales 2987'.

Após as etapas de limpeza e de identificação, a saída ficaria como mostrada abaixo:

['17', 'epping', 'st', 'smithfield', 'new south wales', '2987']

['NU', 'LN', 'WT', 'LN', 'TR', 'PC']

onde:

NU: número da residência

LN: nome da localidade (cidade)

WT: tipo do logradouro (ex: rua, avenida, travessa)

TR: território (estado)

PC: cep.

Note que há um problema nos símbolos de identificação em '*epping*' como 'LN'. Isso ocorreu porque '*Epping*' é um subúrbio na cidade de '*Sidney*' no estado de '*New South Wales*' e provavelmente estava na tabela *look up* do símbolo LN. Se nós olharmos o

endereço do exemplo saberemos que '*epping*' está claramente se referindo ao nome de um logradouro (*wayfare name* - WN) e não uma localidade (LN).

Para o exemplo simples de HMM apresentado há $8^6 = 262.144$ possíveis combinações de estados escondidos que podem ter gerado esta seqüência de símbolos. Onde 8 é o número de estados do HMM e 6 é o número de símbolos de identificação da seqüência apresentada.

Uma dessas 262.144 combinações possíveis poderia ser:

Início → Wayfare Name (NU) → Locality Name (LN) → Postal Code (WT) → Territory (LN) → Postal code (TR) → Territory (PC) → Final

O senso comum nos diz que esta seqüência de estados escondidos é uma combinação improvável para a seqüência de símbolos de identificação (que estão entre parênteses). Para provar isso vamos calcular a probabilidade usando a matriz de probabilidade de emissão mostrada na tabela da Figura 3.5.

	State							
Observation Symbol	*Start*	Wayfare Number	Wayfare Name	Wayfare Type	Locality Name	Territory	Postal Code	*End*
NU	-	0.9	0.01	0.01	0.01	0.01	0.1	-
WN	-	0.01	0.5	0.01	0.1	0.01	0.01	-
WT	-	0.01	0.01	0.92	0.01	0.01	0.01	-
LN	-	0.01	0.1	0.01	0.8	0.01	0.01	-
TR	-	0.01	0.07	0.01	0.01	0.94	0.01	-
PC	-	0.04	0.01	0.01	0.01	0.01	0.85	-
UN	-	0.02	0.31	0.03	0.06	0.01	0.01	-

Figura 3.5: Matriz de probabilidade de emissão. Fonte [9]

Probabilidade = $0.08 * 0.01 * 0.02 * 0.8 * 0.4 * 0.01 * 0.1 * 0.01 * 0.8 * 0.01 * 0.1 * 0.01 * 0.2 = 8.19 * 10^{-}17$

Agora vamos calcular a probabilidade para a seqüência de estados escondidos mais parecida:

Início → Wayfare Number (NU) → Wayfare Name (LN) → Wayfare Type (WT) → Locality Name (LN) → Territory (TR) → Postal Code (PC) → Final

Probabilidade = $0.9 * 0.9 * 0.95 * 0.1 * 0.95 * 0.92 * 0.95 * 0.8 * 0.4 * 0.94 * 0.8 * 0.85 * 0.9 = 1.18 * 10^{-}2$

A segunda seqüência tem uma probabilidade bem maior que a primeira, indicando que essa seqüência de estados escondidos é a mais parecida com a seqüência de símbolos. O cálculo da probabilidade é resolvido através de um algoritmo de Viterbi que é um eficiente caminho para computar a sequência mais provável para uma dada sequência de símbolos de identificação. Este algoritmo não será abordado neste trabalho, mas pode ser consultado em [27].

A distribuição das probabilidades de transição e de emissão são aprendidas através do treinamento de dados, ou seja, a matriz de probabilidade de transição e a matriz de probabilidade de emissão são geradas a partir do treinamento de dados que deve ser realizado antes do usuário começar a limpeza ou padronização dos dados. Dois Modelos Escondidos de Markov (um para endereço e outro para nome) devem ser gerados com o treinamento de dados.

O treinamento de dados deve ser realizado para gerar um Modelo Escondido de Markov de acordo com o conjunto de dados que será usado no processo de deduplicação ou *record linkage*, assim a aproximação de registros ficará mais precisa.

A desvantagem de se utilizar o treinamento de dados no FEBRL é o fato de ser um processo totalmente manual e demorado. A vantagem está em poder usar o mesmo HMM gerado pelo treinamento para conjunto de dados semelhantes.

Mais detalhes sobre como foi realizado o treinamento de dados no estudo de caso é mostrado no capítulo 4.

3.2 Aproximação de Registros

O segundo passo para detectar registros duplicados consiste em decidir se os pares de registros são iguais não iguais, ou se esta decisão não pode ser feita pelo sistema de aproximação de registros, necessitando da intervenção humana para decidir o estado de igualdade dos pares de registro [6].

Segundo [6], o passo para a aproximação de registros dos conjuntos de dados consiste em três etapas: indexação, comparação e classificação.

3.2.1 Indexação

O objetivo da etapa de indexação é reduzir potencialmente o grande número de comparações por eliminação de comparações entre registros que obviamente não são iguais [6], ou seja, esta etapa agrupa os registros similares. Somente estes registros serão comparados na etapa de comparação que é a etapa com o processamento mais caro.

Isto deve ser feito porque se dois conjuntos A e B vão ser deduplicados, o número possível de comparações será igual ao produto do número de registros dos conjunto de dados A e B. Sem a indexação o número de comparações cresce quadraticamente com o número de registros contidos nos conjuntos de dados que serão comparados.

Por exemplo, executando a deduplicação em dois conjuntos de dados com 100.000 registros cada um, resultaria em 10 bilhões de possíveis comparações, o que seria totalmente inviável, por isso a indexação é necessária.

Atualmente o FEBRL tem implementado os seguintes métodos de indexação: método tradicional de indexação de blocos (*blocking index*), método *sorting* e o método *bigram*.

O método tradicional de indexação de blocos agrupa registros dentro de blocos onde eles compartilham uma mesma chave. A chave de bloco é definida pelo usuário e é composta de atributos dos registros no conjunto de dados. Um exemplo de chave pode ser os primeiros quatro caracteres de um campo chamado sobrenome. O usuário também pode utilizar uma combinação de várias chaves de blocos.

O método de indexação *sorting* estende a idéia do método tradicional explicado anteriormente. Com este método blocos maiores são formados por combinações de blocos consecutivos [6]. O número de blocos depende do tamanho da variável "janela deslizante" que dever ser fornecida pelo usuário. Esta variável nada mais é do que o número de blocos a mais que serão criados. Se o tamanho da janela for igual a 1 este modelo passa a ser igual ao método de indexação tradicional de blocos.

A idéia por trás deste método de indexação é a de que blocos vizinhos podem conter registros com valores similares [6].

Já no método de indexação *bigram* os valores da chave de blocos são convertidas dentro de uma lista de *bigrams* (que são substrings contendo dois caracteres) e sublistas

de todas as possíveis permutações serão contruídas usando uma variável limite que deve ser fornecida pelo usuário. Esta variável pode tomar valores entre 0.0 a 10.0.

A lista de *bigrams* resultante são classificadas e inseridas dentro de um índice invertido, que será usado para pesquisar os números de registros correspondentes em um bloco.

Este método é bastante usado por ser robusto para erros tipográficos pequenos.

3.2.2 Comparação

A etapa de comparação é o coração do processo de aproximação de registros e consiste em comparar registros do mesmo bloco de índices que foram criados na etapa anterior (de indexação).

O FEBRL possui um grande número de funções de comparação de campos que são usadas nesta etapa. Os campos de comparação permitem várias comparações de *strings*, números, datas, idades e tempo (hora). As funções de comparação implementadas no FEBRL e suas descrições apresentadas em [6] são mostradas a seguir.

FieldComparatorExactString - Compara dois campos *string* e retorna um peso de acordo com a igualdade entre eles.

FieldComparatorTruncateString - Permite a comparação de *strings* que podem ser truncadas a uma certa posição fornecida pelo usuário.

FieldComparatorApproxString - Utiliza vários métodos de comparações aproximadas de *string*. O método deve ser fornecido pelo usuário. Os métodos disponíveis no FEBRL são: *Jaro*, *Winkler*, *Bigram*, *Editdist*, *Bagdist*, *Seqmatch*, *Compression*, *Sortwinkler* e *Permwinkler*.

FieldComparatorEncodeString - Utiliza codificação fonética para realizar a comparação. Os métodos de codificação atualmente implementados no FEBRL são: *Soundex*, *Mod_soundex*, *Nysiis*, *Phonex* e *Dmetaphone*. O método deve ser escolhido pelo usuário.

FieldComparatorKeyDiff - Compara dois campos (ou lista de campos) com um número máximo de caracteres diferentes que pode ser tolerado que deve ser fornecido pelo usuário.

FieldComparatorNumericPer - Usado para comparar campo númerico, onde um

diferencial máximo de porcentagem deve ser tolerado. O valor do percentual deve ser fornecido pelo usuário e deve variar de 0.0 à 100.0. O valor default é 0.0.

FieldComparatorNumericAbs - Compara dois compos numéricos, onde o diferencial absoluto é dado e pode ser tolerado. Este valor deve ser fornecido pelo usuário e deve ser um valor positivo.

FieldComparatorDate - Compara duas datas que devem ser fornecidas em três campos separados dia, mês e ano.

FieldComparatorAge - Compara idades.

FieldComparatorHour - Compara horas, assumindo que o formato dado seja em 24 horas como HHMM ou HH:MM.

FieldComparatorDistance - Pode ser calculado quando ceps ou campos similares estão disponíveis no conjunto de dados. Este comparador usa tabelas geográficas (onde para cada entrada, latitude e longitude são armazenadas) para calcular a distância em quilômetros entre dois valores.

Mais informações sobre as funções de comparações apresentadas podem ser encontradas em [6].

Uma função de comparação é escolhida para um campo do conjunto de dados e as funções devem ser inicializadas na ordem em que os campos se encontram.

O resultado desta etapa é um vetor de peso para cada par de registros comparados independentemente da função de comparação usada. Uma informação adicional no vetor de pesos é o identificador dos registros comparados [6].

3.2.3 Classificação

A terceira e última etapa do processo de aproximação de registros, mostra a classificação baseada nos vetores de pesos (saída da etapa anterior), se um par de registros constitui uma igualdade, uma não igualdade ou uma possível igualdade.

A versão atual do FEBRL possui dois tipos de classificadores implementados: o classificador '*Felligi and Sunter*' e o classificador flexível.

O classificador *Felligi and Sunter* simplesmente soma log2-pesos de um vetor de pesos

(vetor que foi retornado na etapa anterior de comparação) e então usa dois limites, que são fornecidos pelo usuário, para classificar um par de registros em uma das três classes: iguais, não iguais ou possivelmente iguais .

O classificador flexível permite que diferentes métodos possam ser usados para calcular o peso da aproximação final. Para o vetor de peso, dois limites, fornecidos pelo usuário, são usados para classificar o par de registros. O vetor final é então calculado usando uma função (*min, max, add, mult e arvg*) que também precisa ser definida pelo usuário [6].

Ambos os classificadores armazenam os resultados em uma estrutura de dados que pode ser usada para produzir vários tipos de formatos de saída que são apresentados a seguir.

3.3 Formatos de Saída

Na versão atual do pacote FEBRL existem três formatos de saída que podem ser mostrados na tela ou salvos em um arquivo de texto (dependendo da escolha do usuário):

1. um histograma das comparações dos pesos;
2. uma lista de detalhes de todos os tópicos relacionados aos pares;
3. uma lista de identificadores de registros de todos os pares comparados e seus pesos correspondentes.

3.3.1 Histograma

O histograma é feito de caracteres simples, com os pesos estando no eixo vertical. Para cada peso, uma barra é exibida horizontalmente indicando o número de pares de registros com esse peso. Um exemplo de um histograma pode ser visto na Figura 3.6.

3.3.2 Lista de Detalhes

Na lista de detalhes, cada par de registro é visualizado no formato de três colunas, com os nomes dos campos na primeira coluna, os registros na coluna do meio e os valores do

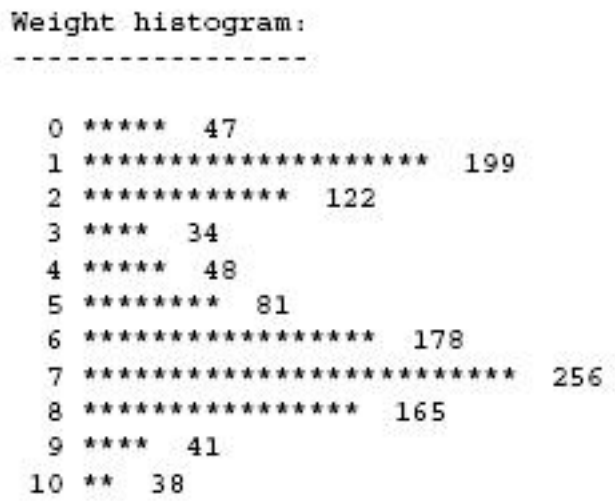


Figura 3.6: Exemplo de um histograma de peso. Fonte [6]

segundo registro na terceira coluna. Um exemplo deste formato de saída é mostrado no Apêndice A.

3.3.3 Lista de Identificadores

Na última forma de exibição, lista de identificadores, somente os números dos registros e o peso total correspondente serão exibidos ou salvos. A primeira coluna mostra os identificadores do registro no primeiro conjunto de dados e a segunda coluna contem os identificadores de registros iguais no segundo conjunto de dados. Os pesos são escritos na terceira coluna. Se existe mais que dois registros que são possivelmente iguais, a *string* '*assigned*' será mostrada na coluna quatro. Por exemplo: se um registro A é considerado igual à um registro B com um peso de 42,21 e o registro A também é considerado igual à um registro C com um peso de 39,01. Neste caso a palavra '*assigned*' será escrita na linha referente ao registros A e B, porque o peso de A e B é maior que o peso de A e C. Um exemplo deste formato de saída é apresentada na Figura 3.7.

```
Rec_ID_A, Rec_ID_B, Weight, Assigned
1,4811,11.351950,
1,3373,11.351950,
2,3870,11.569103,assigned
2,142,11.567734,
7,3186,11.569103,assigned
18,2519,11.569103,assigned
20,2699,11.383410,
20,2041,11.383410,
26,3751,11.569103,
26,1550,11.569103,
26,1742,11.567734,
27,2455,11.569103,
31,1013,11.569103,assigned
35,3725,11.417040,assigned
38,2391,11.569103,assigned
41,3347,11.569103,assigned
44,677,11.569103,
44,115,11.569103,assigned
44,1951,11.500749,
52,2914,11.569103,assigned
52,1015,11.569103,
55,3427,11.569103,assigned
55,2990,11.374640,
56,779,11.569103,assigned
58,4458,11.383410,
58,2700,11.383410,
58,2501,11.383410,
70,2769,11.567734,
70,1232,11.567734,
70,351,11.499068,
88,4577,11.569103,assigned
```

Figura 3.7: Exemplo de uma lista de identificadores.

Capítulo 4

O Uso do FEBRL em Registros Brasileiros

O ambiente de software FEBRL é desenvolvido na Austrália. A versão mais recente deste pacote foi desenvolvida e padronizada para ser usada em conjuntos de dados com registros contendo nomes e endereços australianos.

O problema de se usar o FEBRL para detectar registros duplicados aproximados em conjunto de dados com registros brasileiros é a diferença da escrita entre os registros brasileiros e australianos.

A Tabela 4.1 mostra exemplos de registros australianos enquanto a Tabela 4.2 mostra exemplos de registros brasileiros.

Name	**Address**	**Locality**	**Postcode**
James Whiteway	2, Maribyrnong, Ave	Red Hill	2611
Mitchell Devin	26, knox, St	Holder	2606
Isaac White	73, Chauncy, Pl	Watson	2913
Elle Webb4	3, Burnie, St	Bruce	2617

Tabela 4.1: Tabela com exemplos de registros australianos

Observando as tabelas acima, percebe-se a diferença entre os registros brasileiros e australianos. No campo *address* (endereço), por exemplo, os registros australianos apresentam primeiro o número da residência, depois o nome do logradouro e por último o tipo

Nome	Endereço	Cidade	Cep
Mariana da Silva Batista	Rua das Flores, 333	Florianópolis	88000-000
Eduardo do Nascimento	Rua Prof Roberto Souza	Curitiba	85230-200
Roberto F. Afonso	Av. Pres. Getúlio Vargas, 111	Vitória	29000-000
Ana Carolina Pereira	Rua João Negrão	São Paulo	01000-000

Tabela 4.2: Tabela com exemplos de registros brasileiros

do logradouro. Já no campo endereço dos registros brasileiros o tipo de logradouro vem primeiro, depois o nome do logradouro e por último o número da residência.

Estas diferenças podem ser resolvidas modificando algumas tabelas utilizadas pelo FEBRL. Essas tabelas são chamadas *look up* e contem listas de nomes, sobrenomes, endereços, etc. O uso de treinamento dos Modelos Escondidos de Markov (HMM) e algumas modificações no código fonte também ajudaram a resolver os problemas apresentados.

4.1 Tabelas utilizadas pelo FEBRL

Tabelas são arquivos texto que contêm palavras usadas na limpeza ou padronização de registros. Existem cinco tipos de tabelas apresentadas em [6] e que são usadas pelo FEBRL. São elas:

- Listas de correção: contem palavras (ou caracteres) e suas respectivas substituições;
- Tabelas *look up*: contem os símbolos de identificação (*tags*) relacionadas as palavras e suas substituições;
- Tabelas de frequências: contem palavras e números inteiros que correspondem a frequência com que a palavra pode aparecer em um conjunto de dados;
- Tabelas de localização geográfica: contem palavras e sua localização geográfica correspondente como um par numérico que indica latitude e longitude;
- Tabelas de regiões vizinhas: contem valores de regiões (endereços, ceps), e para cada um deles uma lista com suas regiões vizinhas.

Dos cinco tipos de tabelas apresentadas acima as duas primeiras listas de correção e tabelas *look up* foram modificadas com valores brasileiros para serem usadas no processo de deduplicação do conjunto de dados brasileiro.

4.1.1 Listas de correção

Oa arquivos da lista de correção contêm caracteres ou palavras e suas correspondentes correções. Os dados do arquivo são convertidos dentro de uma lista *Python* que é usada na etapa de limpeza e padronização de registros. Estas listas são usadas para substituir um caracter ou palavras pela correção correspondente. Arquivos de lista de correção devem ter a extensão 'lst'. Um exemplo do conteúdo de uma lista de correção pode ser visto na Figura 4.1.

```
# Remove caracteres e palavras da entrada
              ' ' := ' ', '?', '~', '_', '-', ';', ':', '^', '=', ' na ',
                 ' n/a ', ' n.a.', ' ! ', '*', 'on ', ' off ',
                 ' of ', '``'


# Correct words and symbols
          ' e ' := '+', '&'
          ' - ' := '-', '/'
          ' | ' := '<', '>', '(', ')', '[', ']', '{', '}', '"', "'", '|'

# Remove o apostofro (')
             ' o' := " o'"
             ' a' := " a'"
             ' l' := " l'"
             ' i' := " i'"
             '-o' := "-o'"
             '-a' := "-a'"
             '-l' := "-l'"
             '-i' := "-i'"

# Numeros romanos corretos
#                ' i ' := ' primeiro '
                ' ii ' := ' segundo '
               ' iii ' := ' terceiro '

# Remove os acentos das palavras
                 'a' := "ã", "â", "á", "à"
                 'o' := "ó", "õ", "ô"
                 'u' := "ú", "ü"
                 'c' := "ç"
                 'e' := "é", "ê"
                 'i' := "í"
```

Figura 4.1: Exemplo de um arquivo de lista de correção de nome

As listas de correções foram modificadas para excluir caracteres considerados inúteis na busca de registros duplicados aproximados (ex: '.', '?', '-', ';'), para retirar a acentuação das palavras (facilitando assim o processo de deduplicação), retirar apóstofros e corrigir alguns números.

Foram criadas duas listas de correções uma para nomes e outra para endereços. Ambas as listas contem praticamente a mesma informação e são usadas no processo de limpeza e padronização de registros. A lista de correção completa de endereço pode ser consultada no Apêndice C.

4.1.2 Tabelas *look up*

Chamadas também de tabela *look up* de identificação, as tabelas *look up* contêm um ou mais blocos de entradas. Cada bloco de entrada possui um símbolo de identificação. Os arquivos de tabelas *look up* devem ter a extensão 'tbl' e o formato desses arquivos deve seguir as seguintes especificações:

- Começar um bloco com uma linha que contém o símbolo de identificação entre os símbolos maior menor (< >). EX: tag=<TI>. Este símbolo de identificação deve ser escrito no início da linha;
- Escrever abaixo do símbolo de identificação somente as entradas que serão identificadas com o símbolo de identificação do bloco atual, ou seja, se o símbolo de identificação é TI, então as linhas abaixo só deverão conter informações relacionadas a títulos;
- Inserir cada entrada do bloco na forma 'valor substituto : valor atual', onde 'valor substituto' é uma ou várias palavras e 'valor atual' é a lista de uma ou mais palavras separadas por vírgulas. Cada valor da lista de 'valor atual' será substituído pelo seu respectivo 'valor substituto';
- Inserir # para linhas comentadas.

Se a lista de 'valor atual' estiver vazia, então somente o 'valor substituto' será inserido na tabela *look up*. Se um 'valor atual' ocorre em vários blocos e este valor é substituído pelo mesmo 'valor substituido' mas com símbolos de identificação diferentes, todos os símbolos de identificação serão armazenadas em uma lista para esse valor atual.

A Figura 4.2 mostra um exemplo de uma parte da tabela *look up* de nomes proprios femininos.

```
tag=<GF>  # Símbolo de Identificação para Nomes Proprios Femininos

      abgail : abigail
         ada :
       adail : adal
    adelaide :
      adelia : adelha
     adriana : adriane, adriani
       agata : agatha
        aida :
     alcione :
        alda :
  alessandra : alexandra, alejandra, alesandra
       alice : alicia, alica
       aline : alina
      amalia : amalia
      amanda :
      amelia : amelha
         ana :
      andrea : andreia
     andresa : andressa
      angela :
    angelica : angel
       anita : annita
   aparecida : aparicida
       ariel :
      arlete :
       aurea : auria
     aurelia : aurelha
      aurora :
```

Figura 4.2: Exemplo de uma parte da tabela *look up* para nomes próprios femininos

Para o desenvovimento deste trabalho foram criadas dezessete tabelas *look up* com informações de registros brasileiros.

Os símbolos de identificação padronizados pelo FEBRL não foram modificados, eles apenas foram adaptados. A lista com os símbolos de identificação suportados pelo FEBRL pode ser consultada na Figura 4.3.

Para limpeza ou padronização de nomes foram criados seis tabelas *look up*: *titulos*, *nomes_proprios_f*, *nomes_proprios_m*, *sobrenomes*, *nome_misc* e *prefixo_nome*.

A tabela *look up nomes_proprios_f* é usada para identificar os nomes próprios femininos. Para criar essa tabela foi realizada uma pesquisa em vários sites especializados em nomes e em alguns livros com informações sobre nomes próprios. Ao todo foram coletados 500 nomes femininos e as variações foram criadas inserindo alguns erros comuns de digitação. O símbolo de identificação usado nesta tabela é o GF (*given name female*). Uma parte desta tabela foi apresentada na Figura 4.2.

Nomes_proprios_m é uma tabela similar à tabela *look up* explicada acima, a diferença é que esta tabela identifica nomes próprios masculinos. A pesquisa foi feita da mesma

Tag	Description	Name	Address
TI	Tag for title words	Yes	–
GF	Tag for female given names	Yes	–
GM	Tag for male given names	Yes	–
SN	Tag for surnames	Yes	–
II	Tag for one-letter words (initials)	Yes	–
PR	Tag for name prefix words (like *de*, *la*, *van*, etc.)	Yes	–
ST	Tag for saint words	Yes	Yes
NE	Tag for the word *nee*, which can be a surname but may also mean *born* (in which case it becomes a separator)	Yes	–
BO	Tag for *baby of*, *daughter of* and *son of* sequences	Yes	–
SP	Tag for a separator, like *known as*	Yes	–
PC	Tag for postcodes	–	Yes
CR	Tag for country words	–	Yes
TR	Tag for territory (state) words	–	Yes
LN	Tag for locality name words	–	Yes
LQ	Tag for locality qualifier words	–	Yes
IN	Tag for institution name words	–	Yes
IT	Tag for institution type words	–	Yes
WT	Tag for wayfare type words	–	Yes
WN	Tag for wayfare name words	–	Yes
UT	Tag for unit type words	–	Yes
PA	Tag for postal address type words	–	Yes
VB	Tag for vertical bars (which are the processed form of various brackets and quotes)	Yes	Yes
HY	Tag for hyphens	Yes	Yes
CO	Tag for commas	Yes	Yes
SL	Tag for slashes	–	Yes
NU	Tag for numbers (all numbers in names, but only numbers that do not have 4-digits in the address	Yes	Yes
N4	Tag for four-digit numbers (that are not listed in the postcode look-up table)	–	Yes
AN	Tag for alphanumeric words, i.e. words that contains both letters and digits	Yes	Yes
UN	Tag for unknown words (i.e. words not listed in any look-up table)	Yes	Yes
RU	Rubbish tag (i.e. words that will be removed from the input)	Yes	Yes

Figura 4.3: Lista com os símbolos de identificação suportados pelo FEBRL. Fonte [6]

maneira e ao todo foram coletados 400 nomes próprios masculinos. O símbolo de identificação usado nesta tabela *look up* é o GM (*given name male*). Uma parte desta tabela pode ser consultada no Apêndice B.

A tabela *look up sobrenome* contém uma relação dos sobrenomes mais usados no Brasil. Para a criação da tabela foram pesquisados vários sites e livros relacionados a brasões de família. Ao todo esta tabela possui 1050 sobrenomes diferentes e suas variações. O símbolo de identificação utilizado pra identificar sobrenomes é o SN (*surname*). Uma parte desta tabela pode ser consultada no Apêndice B.

No Brasil geralmente não colocamos títulos antes do nome em um cadastro, já na Austrália, onde o FEBRL foi desenvolvido isso é muito comum. Mesmo não sendo muito utilizado no Brasil, foi criada uma tabela *look up* contendo alguns títulos. Isto foi feito

pois pode ocorrer o aparecimento de títulos como doutor em conjunto de dados médicos, professor em conjunto de dados acadêmicos, entre outros exemplos. Seria uma maneira de garantir que o FEBRL reconhecesse o que está cadastrado se aparecer algum título. Para criar esta tabela foram usados apenas os títulos mais comuns. O símbolo de identificação utilizado para identificação de títulos é TI (*title*). O conteúdo da tabela *look up* de títulos utilizada é mostrado na Figura 4.4.

```
tag=<TI>  # Símbolo de identificação para título

         bel : bacharel
           d : dona
          dr : doutor, doc, phd
         dra : doutora, phd
         eng : engenheiro, engenhero
         mme : madame
        prof : professor
       profa : professora
          sr : senhor
         sra : senhora
        srta : senhorita
```

Figura 4.4: Tabela *look up* de títulos.

Prefixo_nome contem palavras usadas para identificar prefixos usados em nomes completos (ex: da, do, de, etc). Para criar essa tabela *look up* foi usado a tabela *look up* de prefixo original do FEBRL. O simbolo de identificação usado é PR e a tabela *look up* completa pode ser vista na Figura 4.5.

```
tag=<PR>  # Símbolo de identificação para prefixos

         da :
         de :
        del :
        den :
        des :
         di :
         do :
        dos :
         du :
         el : le
         il :
         in :
         la :
         le :
         li :
         lo :
         na :
         no :
         ol :
        san :
```

Figura 4.5: Tabela *look up* de prefixos de nomes.

A tabela *look up nome_misc* contém uma miscelânia que é usada para identificar algumas palavras relacionadas com nomes, mas que não estão listadas nas tabelas *look up* mostradas anteriormente. Esta tabela é mostrada na Figura 4.6. A tabela é divida em 2 blocos com símbolos de identificação diferentes: BO usados para identificar palavras como 'filho', 'junior', 'neto' entre outras e o símbolo de identificação SP é usado para identificar separadores (e, ou).

```
tag=<SP>  # Símbolo de identificação para elementos separadores

          e :
         ou :
         da :
         de :
         di :

tag=<BO>  # Símbolo de identificação para sequencias similares a filho

       filho : filho de
       filha : filha de
        neto : neto de
        neta : neta de
      junior : jr
    sobrinho : sobrinho de
    sobrinha : sobrinha de
```

Figura 4.6: Tabela look up de miscêlanias para nomes.

Da mesma maneira que foram criadas as tabelas *look up* para nomes, foram criadas as tabelas *look up* para endereços. Ao todo foram criadas onze tabelas *look up* para fazer a limpeza dos endereços: *endereco_misc*, *endereco_bairro*, *tipo_rua*, *tipo_instituicao*, *complemento*, *nome_cidades_brasil*, *nome_cidades_parana*, *ceps_brasil*, *ceps_parana*, *estados* e *paises*.

Endereco_misc é a tabela *look up* que identifica nomes de logradouros (nomes de ruas). Devido o grande número de nomes de logradouros existentes foi concentrado somente nos nomes de logradouros da cidade de Curitiba. Essa decisão foi tomada porque o estudo de caso é um conjunto de dados do Sistema de Bibliotecas da Universidade Federal do Paraná, que está localizada na cidade de Curitiba, sendo assim a maioria dos cadastrados no conjunto de dados é da cidade de Curitiba. Para criar este arquivo foi usada a base de dados dos correios (www.correios.com.br). O símbolo de identificação usado nesta tabela é o WN (*wayfare name*).

A tabela *look up endereco_bairro* é usada para identificar os bairros em um endereço. Neste caso ficaria difícil e trabalhoso cadastrar todos os bairros das principais cidades, por isso primeiro foi feita uma pesquisa referente a todos os bairros das capitais nacionais depois foi selecionada as principais palavras (palavras chaves) que relacionam um nome de bairro, ou seja, palavras que se repetem com mais frequência em nomes dos bairros pesquisados, por exemplo: 'alto', 'bela', 'campo', 'céu', 'floresta', 'jardim', 'primavera', 'planalto', 'serra', 'velha', 'vargem', 'vista' entre outras. Como o estudo de caso é um conjunto de dados da Universidade Federal do Paraná, que está localizada na cidade de Curitiba no estado do Paraná, foi acrescentado nesta tabela *look up* todos os nomes dos bairros de Curitiba. Estas informações foram retiradas do site da Prefeitura Municipal de Curitiba (www.curitiba.pr.gov.br). O simbolo de identificação usado nesta tabela *look up* é o LQ (*locality qualifier*).

Tipo_de_rua é a tabela *look up* que identifica o tipo de logradouro em um endereço como por exemplo: 'alameda', 'avenida', 'largo', 'praça', 'rua', 'vila', entre outros. A lista completa pode ser vista no Apêndice B. As informações contidas nesta tabela foram pesquisadas no site dos correios (www.correios.com.br).O simbolo de identificação utilizado para identificar tipo de logradouro é o WT (*wayfare type*).

Tipo_instituicao é uma tabela *look up* com palavras institucionais como por exemplo: 'aeroporto', 'clube', 'delegacia', 'hotel', 'mercado', 'rodoviário', 'universidade' entre outros. São palavras que servem como informações adicionais à um endereço. A lista completa está disponível no Apêndice B. As informações contidas neste arquivo foram conseguidas com a tradução da tabela *look up* original do FEBRL. O simbolo de identificação usado para identificar tipos de instituições é o IT (*institution*).

A tabela *look up complemento* com várias palavras que identificam complementos de endereços como por exemplo: 'apartamento', 'bloco', 'condomínio', 'fundos', entre outras. A lista completa pode ser encontrada no Apêndice B. Para criar esta tabela foram pesquisadas informações em listas telefônicas e catálogos de endereços. O símbolo de identificação usado é o UT (*unity*).

Tabelas *look up* que contêm relações de cidades do Brasil e cidades do estado do Paraná são as *nomes_cidades_brasil* e *nomes_cidades_parana* respectivamente. Seria inviável criar uma tabela contendo realmente todas as 5561 cidades do Brasil. Por esse motivo foi criado dois arquivos de cidades, um contendo todas as cidades do Brasil com mais de 100.000 habitantes e outro contendo todas as cidades do estado do Paraná com acima de 20.000 habitantes. O estado Paraná foi escolhido por causa do estudo de caso que possui vários registros de todo o estado. Foi escolhido colocar várias cidades do Brasil e principalmente do estado do Paraná porque aparecem pessoas de todo o Brasil para estudar na Universidade Federal do Paraná e geralmente eles cadastram o endereço de origem no primeiro cadastro ao sitema de bibliotecas. Ao todo foram coletados 237 cidades de outros estados e 85 cidades do estado do Paraná. As informações referentes as cidades foram adquiridas no site do Instituto Brasileiro de Geografia Estatística - IBGE (www.ibge.gov.br) dados de 2005. O símbolo de identificação utilizados em ambas as tabelas *look up* é o LN (*locality name*).

Estados é a tabela *look up* que identifica os estados brasileiros. Esta tabela contêm todos os estados com suas siglas e algumas variações referentes a erros tipográficos. O símbolo de observação utilizado é o TR (*territory*).

Ceps_brasil e *ceps_parana* são tabelas *look up* com números de ceps da cidade de Curitiba e algumas faixas de ceps das principais cidades do Brasil. As informações contidas nestas tabelas foram retiradas da base de dados dos correios (www.correios.com.br). O símbolo de identificação utilizado nestas tabelas *look up* é o PC (*postcode*).

Países de todo o mundo e suas respectivas siglas e variações podem ser encontrados na tabela *look up paises*. O símbolo de identificação utilizado para identificar países é o CR (*country*).

4.2 Treinamento HMM

No processo de limpeza de registros foi usado o Modelo Escondido de Markov (HMM) na etapa de segmentação para fazer a limpeza e padronização, como descrito na seção 3.2.

O treinamento de dados deve ser feito com o conjunto de dados que será utilizado na

aproximação de registros, antes da execução do módulo correspondente no FEBRL. O treinamento de dados tem o objetivo de criar um Modelo Escondido de Markov com as características do conjunto de dados que será utilizado.

Os arquivos de treinamento de dados consistem de sequências com um ou mais pares do tipo 'símbolo de identificação : estadohmm'. Cada sequência é um registro correto, e o HMM aprende as características de um conjunto de dados usando todos os exemplos de treinamento apresentados durante o processo [7].

Um conjunto simples de exemplos de treinamento é parecido com:

GF:gname1, SN:sname1

UN:gname1, SN:sname1

GF:gname1, GM:gname2, UN:sname1

GF:gname1, GM:sname1

Onde GF é o símbolo de identificação e gname1 é o estado do HMM. Uma lista dos símbolos de identificação e dos estados do HMM atualmente suportados pelo FEBRL podem ser vistos no Anexo B.

Cada linha no exemplo acima corresponde a um registro de treinamento e contém uma sequência que corresponde a um caminho particular através de vários estados (escondidos) do HMM junto com o símbolo de identificação correspondente [7].

O treinamento de dados no FEBRL é um processo totalmente manual e muito demorado de ser realizado. E para que seja bem feito as tabelas *look up* e as listas de correções devem estar bem completas. Quanto mais completas as tabelas *look up* e as listas de correções mais fácil será o treinamento de dados.

Para fazer o treinamento no conjunto de dados do Sistemas de Bibliotecas da Universidade Federal do Paraná, foram utilizadas as listas de correções da seção 4.1.1 e as tabelas *look up* da seção 4.1.2. O treinamento foi feito seguindo os nove passos apresentados a seguir:

Passo 1:

Foi criado um arquivo com 100 registros de treinamento. Para isso foi usado um programa próprio chamado 'tagdata.py' que faz parte do pacote FEBRL. Para rodar

este programa algumas informações devem ser fornecidas pelo usuário como: o nome do conjunto de dados, o número de registros do conjunto, o número de registros que serão selecionados, o nome do arquivo de saída, o tipo do componente (nome ou endereço), o nome dos campos do conjunto de dados, nome das tabelas *look up* e o nome das listas de correções;

Passo 2:

Depois de criado o arquivo inicial de treinamento chamado 'treinamento_sibi.csv', este arquivo foi editado. Na edição do arquivo foram modificados os símbolos de identificação incorretos e foram comentadas as linhas incorretas (para registros com mais de um símbolo de identificação por palavra). Como este foi o nosso primeiro arquivo de treinamento e nós não nos baseamos em nenhum Modelo Escondido de Markov já pronto, então para cada símbolo de identificação apresentado, foi acrescentado o estado do HMM correspondente;

Passo 3:

Foi criado um HMM inicial usando um programa próprio chamado 'trainhmm.py', que também faz parte do FEBRL. Algumas informações devem ser fornecidas pelo usuário como: o nome do arquivo hmm, nome do arquivo contendo os registros treinados, o tipo do componente (nome ou endereço);

Passo 4:

Depois foi criado um segundo arquivo de treinamento como no passo 1, mas desta vez com 1000 registros. Como este é o segundo arquivo de treinamento, então foi usado o HMM criado anteriormente. Para isso é só configurar a variável 'hmm_file' para o nome do HMM que foi criado no passo 3;

Passo 5:

O passo 2 foi repetido com o segundo arquivo de treinamento. Como foi usado um HMM para gerar o arquivo no passo 4, então este arquivo já possui os estados do hmm correspondentes ao símbolo de identificação. O trabalho neste passo foi corrigir os símbolos e estados incorretos;

Passo 6:

Foi criado o segundo HMM usando o segundo arquivo de treinamento como foi no

passo 3;

Passo 7:

Depois um terceiro arquivo de treinamento com 1000 registros foi criado, mas desta vez reprocessando o segundo arquivo de treinamento corrigido no passo 5 e usando o HMM criado no passo 6. O reprocessamento é realizado configurando a variável 'retag_file_name' para o nome do segundo arquivo de treinamento;

Passo 8:

Depois foi feito um exame no terceiro arquivo de treinamento onde as sequências modificadas estavam em destaque com a palavra '*changed*'. Essas sequências modificadas foram corrigidas.

Passo 9:

Os três passos anteriores foram repetidos até que nenhuma mudança fosse detectada no arquivo de treinamento.

Estes nove passos foram realizados para gerar dois Modelos Escondidos de Markov, um de nome e outro de endereço, os HMM criados foram usados no processo de deduplicação do conjunto de dados do estudo de caso.

Os Modelos Escondidos de Markov de nome e endereço resultantes do treinamento do conjunto de dados do Sistema de Bibliotecas da Universidade Federal do Paraná são apresentados graficamente nas Figuras 4.7 e 4.8 respectivamente.

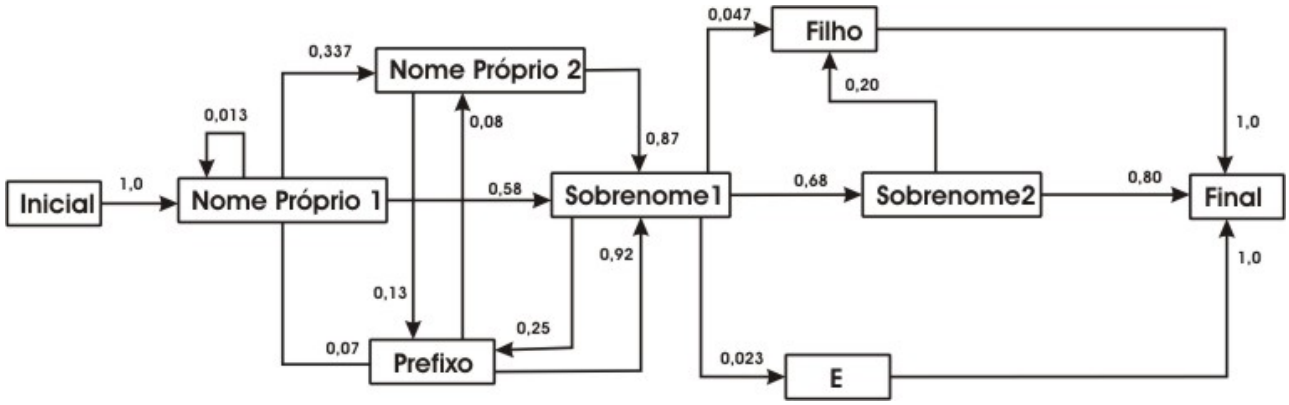


Figura 4.7: HMM de nome resultante do treinamento de dados.

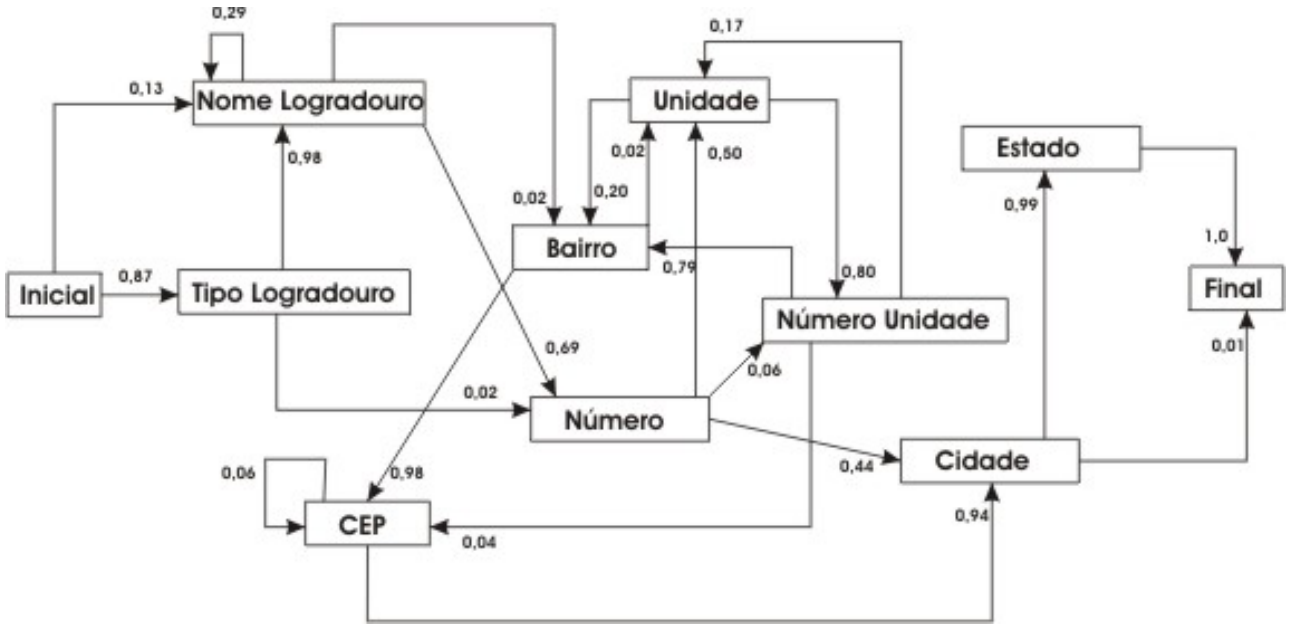


Figura 4.8: HMM de endereço resultante do treinamento de dados.

Durante os passos de edição dos arquivos de treinamento, todas as informações novas que apareciam e que não estavam nas tabelas *look up*, foram acrescentadas nas tabelas *look up* correspondentes.

4.3 Estudo de Caso

Para estudar a viabilidade do uso do ambiente FEBRL em detectar registros duplicados em base de dados brasileiras, foi usado um conjunto de dados real. Este conjunto de dados é o cadastro de usuários do Sistema de Bibliotecas da Universidade Federal do Paraná (SIBI).

O Sistema de Bibliotecas da UFPR é constituído por uma sede administrativa, doze bibliotecas universitárias e uma biblioteca de ensino médio. Todas as bibliotecas trabalham de forma integrada.

Este conjunto de dados foi escolhido pois havia muita reclamação por parte de alguns funcionários em relação a quantidade de registros duplicados neste conjunto de dados que é o principal do Banco de Dados da biblioteca.

O problema de duplicação de registros no cadastro de usuários do sistema de bibliotecas

acontece por vários motivos, entre eles:

- A identificação dos usuários atualmente é feita pelo número de matrícula. Existem padrões diferentes do número de matrícula para alunos da escola técnica, alunos da graduação e alunos da pós-graduação;
- Alunos cadastrados na biblioteca da escola técnica (ensino médio) passam no vestibular, ou seja, eles já estão cadastrados na escola técnica, passando no vestibular, eles terão um novo cadastro;
- O cadastro dos calouros de graduação é feito por eles mesmos *online*, o que dificulta mais ainda na hora de verificar se ele já está cadastrado;
- Alunos que passam no vestibular, trancam ou abandonam o curso e depois passam novamente no vestibular, ou seja, eles já estão cadastrados no sistema (porque já tinham passado no vestibular e se cadastrado no sistema), mas acabam cadastrados novamente no sistema de bibliotecas com um número de matrícula diferente;
- Pesquisas mal feitas e falta de informação das bibliotecárias que são responsáveis pelo cadastros de novos usuários;
- Alunos de graduação ou pós-graduação que fazem cursos técnicos na escola técnica da UFPR. Estes alunos acabam fazendo o cadastro em ambas as bibliotecas com números de matrículas diferentes.

O conjunto de dados SIBI possui as seguintes características:

Campos: Nome, Endereço, Bairro, Cep, Cidade e Estado;
Números de registro: 60.247

Para facilitar a identificação dos possíveis registros duplicados foi criado um campo a mais no conjunto de dados de teste. Este campo foi chamado de Identificador e seus valores são números inteiros incrementais que variam de 0 à 60.246.

Para fazer os testes com o estudo de caso foi utilizado o formato de base de dados csv. Este formato foi escolhido porque o FEBRL trabalha muito bem com este tipo de formato. Outros formatos como sql ainda estão em fase de testes no FEBRL.

4.3.1 Ambiente de Teste

Para realizar os testes com o conjunto de dados brasileiro foi utilizado um computador com as seguintes configurações:

Hardware:

AMD Athlon (tm) XP 1700 GHz

Memória RAM: 3GB

Software:

Linux Debian 4.0.2-6 Kernel versão 2.6.15.5

Python 2.4

FEBRL 0.3

4.3.2 Testes e Resultados

Os testes foram realizados utilizando o módulo '*project-deduplicate.py*' do FEBRL. Este módulo é responsável pelo deduplicação em um conjunto de dados, ou seja, ele é responsável pela detecção dos registros duplicados aproximados em um conjunto de dados apenas.

O FEBRL possui vários tipos de configurações, por isso, inicialmente foram feitos alguns testes com apenas 10.000 dos 60.247 registros do conjunto de dados SIBI para ser analisada a melhor configuração. Isso foi realizado para se ter um melhor desempenho do FEBRL.

Os tipos de configurações estão relacionados a: arquivo de log, base de dados, tipo de padronizador (HMM ou regras de aproximação), tabelas *look up*, listas de correções, método de indexação de blocos (*blocking*, *sorting* ou *bigram*) e as variáveis de blocos, funções de comparações, método de classificação (*Fellegi and Sunter* ou flexível) e os formatos dos arquivos de saída.

Os primeiros testes foram para escolher as melhores funções de comparações dentre as funções apresentadas na seção 3.2.2. Para isso, foram realizados cinco testes com diferentes funções de comparações e diferentes tipos de variáveis de função. Foram escolhidos cinco campos principais para serem usados nas funções testes: *given_name* (nome próprio), *surname* (sobrenome), *wayfare_name* (nome do logradouro), *locality_name* (cidade) e *postcode* (cep). Estes campos foram escolhidos porque são os mais significativos para detectar registros duplicados, não faria sentido colocar *wayfare_type* (tipo da rua) ou *unity* (unidade) por exemplo. Em todos os testes feitos, o tempo de execução foi praticamente o mesmo. As melhores funções foram escolhidas pela análise dos arquivos saída de todos os testes e escolhemos o conjunto de funções que apresentaram um melhor resultado. A relação das funções escolhidas para o teste final são apresentadas na Tabela 4.3.

Função de Comparação	**Campo**
FieldComparatorEncodeString	given_name (nome próprio)
FieldComparatorTruncateString	surname (sobrenome)
FieldComparatorApproxString	wayfare_name (nome logradouro)
FieldComparatorKeyDiff	locality_name (cidade)
FieldComparatorKeyDiff	postcode (cep)

Tabela 4.3: Relação das funções de comparação.

Depois de escolhidas as funções de comparação, foram feitos alguns testes para escolher as variáveis de bloco que seriam usadas na etapa de indexação. Os testes realizados são apresentados com detalhes no Apêndice D. Analisando os resultados, o conjunto de variáveis de bloco escolhido para nosso teste final foi:

[[('*surname*','*truncate*',4), ('*wayfare_name*','*nysiis*')],
[('*locality_name*','*soundex*'), ('*given_name*','*phonex*')]]

Em seguida foram feitos vários testes para escolher o melhor método de indexação e classificação. As configurações padrão para os testes realizados foram:

- **Número de registros:** 10.000

- **Tipo de padronizador:** Modelo Escondido de Markov para nomes e endereços;
- **Funções de comparações:** Apresentadas na tabela 4.3;
- **Variáveis de blocos:** Apresentadas acima;
- **Tamanho do bloco :** 500;
- **Formatos de saída :** histograma, lista de detalhes e lista de identificadores.

O tipo de padronizador utilizado em todos os testes foi o Modelo Escondido de Markov, não foram feitos testes com o modelo de Regras de Aproximação.

Os resultados destes testes para escolher os métodos de indexação e classificação são mostrados na tabela 4.4.

Método de Indexação	**Método de Classificação**	**Tempo Total**
Blocking	Fellegi and Sunter	16 min. e 10 seg.
Sorting	Fellegi and Sunter	20 min. e 34 seg.
Bigram	Fellegi and Sunter	27 min. e 21 seg.
Blocking	Flexível	11 min. e 38 seg.
Sorting	Flexível	16 min. e 38 seg.
Bigram	Flexível	21 min. e 53 seg.

Tabela 4.4: Resultados dos testes de métodos de indexação e classificação.

Analisando a tabela 4.4 foi concluído que o melhor metódo de indexação é o blocking. O melhor método de classificação é o flexível para conjunto de dados brasileiros, pois ambos os métodos trabalhando juntos apresentam um tempo inferior aos demais métodos.

Para realizar os testes finais com o conjunto de dados de cadastro do Sistema de Biblioteca da UFPR foram utilizadas as configurações finais mostradas abaixo:

Tamanho do conjunto de dados: 60.247 registros;

Tipo de padronizador: Modelo Escondido de Markov para nome e endereço;

Variáveis de blocos: Apresentadas na tabela 4.3;

Método de indexação: *Blocking*;

Funções de comparação: Apresentadas na tabela 4.3;

Método de classificação: Flexível;

Tamanho do bloco de indexação: 1000;

Fomatos de Saída: histograma, lista de detalhes e lista de identificadores;

Peso limite (usado nos formatos de saída): 30.0;

O processo de deduplicação do ambiente FEBRL executando com o conjunto de dados SIBI e com as configurações mostradas acima demorou 3 horas, 27 minutos e 40 segundos para ser executado. Este é o tempo total que é dividido em três passos principais e seus tempos: (1) padronização - 29 minutos e 40 segundos; (2) deduplicação - 2 horas, 57 minutos e 45 segundos; (3) saída - 14 segundos.

Foram analisados os arquivos de saída e o processo de deduplicação do FEBRL encontrou aproximadamente 3000 pares de registros duplicados reais que foram conferidos no conjunto de dados, ou seja, cerca de 5% do conjunto todo estava realmente duplicado. Entre os outros pares de registros que o FEBRL considerava duplicado, mas não eram duplicados no conjunto de dados do estudo de caso estavam: irmãos com nomes parecidos e com o mesmo endereço, pessoas com nomes parecidos e que moram no mesmo prédio ou mesma rua e homônimos, ou seja, nomes iguais que respresentam pessoas diferentes.

Os formatos de saída histograma e uma parte da lista de identificadores finais são apresentadas no Apêndice E. A lista de detalhes não foi apresentada para preservar as informações dos usuários cadastrados no Sistema de Bibliotecas da UFPR.

É apresentado na Tabela 4.5 uma comparação de tempo do processo de deduplicação realizado em nosso conjunto de dados SIBI com registros brasileiros e com um conjunto de dados australiano fictício com o mesmo número de registros (60.247). Este conjunto de dados australiano foi criado através de um programa chamado *generate.py* que também está disponível no FEBRL. Este programa permite criar um conjunto de dados fictícios utilizando tabelas *look up* e tabelas de frequência. Este programa também permite inserir registros duplicados durante sua criação. As configurações usadas no conjunto de dados australianos foi a configuração *default* do FEBRL no processo de deduplicação.

Observando os resultados apresentados, é verificado que o FEBRL pode ser utilizado para detecção de registros duplicados em conjunto de dados brasileiros. Os resultados

Passos	**Conjunto de Dados SIBI**	**Conjunto de Dados Australianos**
Padronização	29 min. e 40 seg.	30 min. e 10 seg.
Deduplicação	2 horas, 57 min. e 45 seg.	3 horas 10 min. e 24 seg.
Saída	14 segundos	20 segundos
Total	3 horas, 27 min. e 40 seg.	3 horas 40 min. e 54 seg.

Tabela 4.5: Comparação entre conjunto de dados SIBI e australiano.

deixam claro que o FEBRL é um ambiente completo e muito eficiente.

O uso do FEBRL não é muito fácil e exige muito estudo do manual para entender tudo que ele é capaz de fazer e todas as suas configurações. Ele é um ambiente muito complexo que exige um certa experiência em programação por parte do usuário.

O ambiente FEBRL não necessita de instalação, para usá-lo basta baixar o arquivo compactado e descompactar em uma máquina independente do Sistema Operacional, pois o FEBRL é multiplataforma. Para executar os programas contidos no ambientes deve-se ter instalado na máquina a linguagem de programação *Python* e usar a linha de comando: *python* <nome_programa.py>.

O conjunto de dados testado possui um baixo número de registros, por isso não foi necessário o uso do paralelismo, mas quando se trabalha com conjunto de dados possuindo milhões de registros (que é muito comun), este tipo de solução se torna inviável, pois o tempo de execução vai aumentar muito. Para fazer a deduplicação em conjunto de dados grandes, o paralelismo é essencial e o FEBRL tem um bom suporte para paralelismo e pode ser muito bem explorado.

O funcionamento do paralelismo no ambiente FEBRL é apresentado no próximo capítulo.

Capítulo 5

Processamento Paralelo

Enquanto a computação de alto desempenho (paralela) foi historicamente restrita à ciência e a engenharia, avanços tecnológicos das últimas décadas permitiram a disseminação desta tecnologia dentro do mundo comercial na tecnologia da informação (TI) [8].

O processamento paralelo tem como objetivo dividir um problema entre diversos processadores, diminuindo assim o tempo de execução. Geralmente o processamento paralelo é destinado a solução de problemas que utilizam grande poder computacional.

Em outras palavras, o processamento paralelo é o método utilizado em tarefas grandes e complexas para obter resultados na forma mais rápida possível. Consiste em dividir uma grande tarefa em tarefas pequenas que são distribuídas em vários processadores para serem executadas simultaneamente [26].

A Figura 5.1 exemplifica a diferença entre o processamento seqüencial e o processamento paralelo.

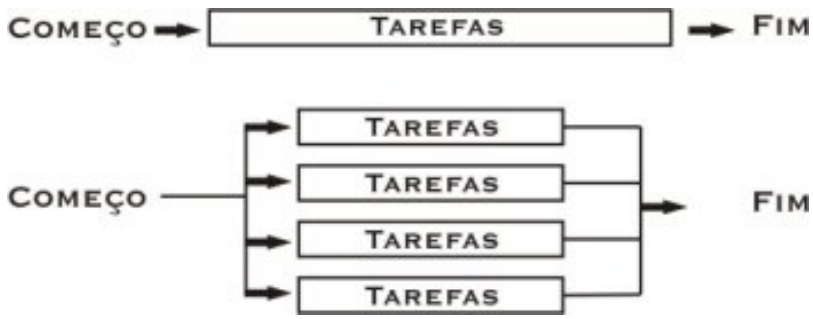


Figura 5.1: Diferença entre o processamento sequencial e paralelo

As duas tecnologias mais comuns para usar o processamento paralelo são: multiprocessadores simétricos (SMP-*Symmetric Multi-Processors*) e os *clusters* (também chamado de agrupamentos em algumas pesquisas).

O SMP é um computador composto de múltiplos processadores, de forma que cada processador tem a mesma habilidade e aproximadamente o mesmo tempo de acesso a qualquer localização de memória [24].

A tecnologia de *cluster* segundo [18] é caracterizado como um sistema de processamento paralelo ou distribuído, que é composto por um conjunto de computadores distintos trabalhando como um recurso computacional único e integrado. Em outras palavras, cluster é um sistema montado com mais de um computador e tem como objetivo distribuir o processamento entre os computadores de forma que pareça um só computador.

Cada computador que faz parte de um cluster é chamado de nó. Para que o sistema funcione corretamente é preciso um nó servidor, vários nós clientes, uma biblioteca de troca de mensagem e uma conexão de rede entre os nós. O sistema operacional utilizado deve ser o mesmo em todos os nós, pois cada sistema operacional trabalha o paralelismo de uma forma diferente.

5.1 Bibliotecas de troca de mensagem

As bibliotecas de troca de mensagens são responsáveis pela comunicação entre os nós do *cluster*, ou seja, elas procuram oferecer ao programador o suporte necessário para o desenvolvimento de aplicações paralelas eficiente possibilitando que programas com características paralela sejam construídos.

As bibliotecas de troca de mensagens mais conhecidas são: PVM (*Parallel Virtual Machine*) [1] e a MPI (*Message Passing Interface*) [19].

A biblioteca PVM é um ambiente que oferece uma máquina virtual que agrega os recursos de processamento dos nós de um *cluster*.

A biblioteca MPI apresenta um desempenho muito melhor se comparada com a biblioteca de troca de mensagens PVM. Por causa desse melhor desempenho, a biblioteca MPI é atualmente adotada pela maioria dos centros de pesquisas e fabricantes do mundo.

5.2 Módulo Pypar

A biblioteca MPI possui implementações para linguagens C, C++, Fortran e Java. Como o ambiente (FEBRL) utilizado no trabalho foi desenvolvido em *Python*, é necessário instalar um módulo chamado *pypar*.

Este módulo permite que *scripts* na linguagem de programação *Python* sejam executados em paralelo utilizando a biblioteca de troca de mensagens MPI. Este módulo é de código fonte aberto e não realiza modificações no interpretador Python.

Atualmente este módulo está na versão 1.9.2 e segundo [21] o módulo pypar é caracterizado pelas seguintes caracteríticas:

- Flexibilidade: qualquer objeto Python pode ser transmitido;
- Facilidade de Uso: o usuário precisa somente especificar o que enviar e para qual processador;
- Eficiência: a troca de mensagem é muito eficiente;
- Facilidade de instalação: para rodar o módulo pypar é necesário ter instalado a linguagem Python, uma biblioteca MPI, o módulo Numeric (Python) e um compilador C.

5.3 O processamento paralelo no ambiente FEBRL

Para ser capaz de processar conjuntos de dados com grandes volumes de registros o processamento paralelo torna-se essencial.

O paralelismo no FEBRL é atualmente implementado baseado na biblioteca de troca de mensagens MPI.

No passo de limpeza e padronização de dados (descrito no capítulo 3) cada registro do conjunto de dados de entrada pode ser padronizado e limpo independentemente de todos os outros registros.

Assumindo que P processadores estão disponíveis no *cluster*, cada um destes processadores processam [número de registros/P] registros de entrada para então realizarem a

etapa de limpeza e padronização.

Por exemplo: se 4 processadores estão disponíveis no *cluster* e um conjunto de dados com 100.000 registros será processado, então, o primeiro processador pega os registros de 1 até 25.000, o segundo processador pega os registros de 25.001 até 50.000, o terceiro pega os registros de 50.001 até 75.000 e o quarto processador pega o restante dos registros [8].

Cada processador então abre o arquivo de entrada, pula para o primeiro registro para o qual é responsável carrega os registros e padroniza parte do arquivo de entrada, depois cada processador escreve no arquivo de saída o resultado.

No passo de aproximação de registros, a etapa de indexação é o ponto de partida da paralelização. Os dados de entrada são separados de acordo com os valores de uma ou mais variáveis de blocos (definidas pelo usuário). Por exemplo: se a variável de bloco definida foi o 'cep', todos os registros que contenham o mesmo cep são movidos para dentro do mesmo bloco. Somente os registros do mesmo bloco serão comparados.

Como nenhuma comparação é conduzida dentro de diferentes blocos, cada bloco pode ser processado independente dos outros blocos [8], ou seja, cada processador fica responsável por um bloco para então realizar a etapa de comparação.

Como cada bloco (na etapa de comparação) retorna os pares de registros comparados e seus respectivos pesos, na etapa de classificação os processadores recebem estas informações, analisam os pesos e escrevem os resultados nos arquivos de saída. Cada processador é responsável pelos resultados que ele gerou na etapa de comparação.

5.3.1 Problema encontrado

O problema encontrado no paralelismo do FEBRL está na etapa de classificação. O FEBRL não consegue realizar esta etapa paralelamente. Cada processador fica esperando o processador responsável pelo bloco anterior terminar a execução para depois ele executar o seu bloco, tornando o processo sequencial.

Capítulo 6

Conclusão

A medida que o número e volume das bases de dados aumentam, cresce a importância do termo qualidade de dados, pois qualidade de dados ruim significa informações imprecisas, incompletas, redundantes ou até fictícias. Isso pode prejudicar todos os tipos de organizações.

Registros duplicados, principalmente aproximados, é um dos principais problemas na qualidade de dados e sua detecção está sendo muito discutida nos últimos anos. Por estes motivos foi desenvolvido este trabalho e apresentada uma solução para detectar registros duplicados em conjunto de dados brasileiros.

Foi realizada uma adaptação no ambiente FEBRL para ser utilizado em conjunto de dados brasileiros. Essa adaptação foi realizada modificando as tabelas *look up*, listas de correções e algumas regras de codificação no código fonte. Com essas adaptações nós mostramos que é viável usar este ambiente para detectar duplicações em conjunto de dados brasileiros.

O FEBRL foi escolhido por suportar paralelismo, porque na deduplicação de registros o uso do paralelismo é essencial, pois como foi mostrado nos resultados, mesmo com um conjunto de dados pequeno, como no estudo de caso, o FEBRL demora um tempo considerável para executar a deduplicação. Deve-se pensar em milhões de registros, pois é o que temos em grandes organizações e fazer a deduplicação sem o paralelismo, com esta quantidade de registros, é totalmente inviável. Por isso, testes e melhorias no paralelismo

no FEBRL são tão importantes.

6.1 Trabalhos Futuros

Há vários trabalhos futuros possíveis. O principal trabalho seria melhorar o paralelismo do FEBRL, pois um tempo de execução baixo é muito importante no processo de deduplicação de registros em conjunto de dados com milhares de registros . O mais interessante do paralelismo é uso de tecnologias como *clusters*, que são tecnologias aparentemente de baixo custo (se comparado com outros tipos de tecnologias paralelas) com um poder de processamento enorme.

Adaptar os algoritmos fonéticos utilizados no FEBRL para reconhecer palavras brasileiras, também é um trabalho interessante. Estes algoritmos fonéticos são utilizados pela deduplicação na etapa de indexação e de comparação.

Um outro trabalho seria fazer testes com a ferramenta *geocoding* disponível no FEBRL. Esta ferramenta permite a localização de endereços usando coordenadas geográficas (latitude e longitude).

Fazer testes com vários conjunto de dados brasileiros utilizando o módulo para *record linkage* disponível no FEBRL para detectar registros duplicados entre os arquivos também são testes interessantes de serem feitos.

Melhorar o treinamento de dados é uma tarefa importante, pois este processo é totalmente manual e muito demorado de se realizar. Tentar automatizar esta tarefa seria essencial.

Referências Bibliográficas

[1] Jack Dongarra Weicheng Jiang Robert Manchek Vaidy Sunderam Al Geist, Adam Beguelin. *PVM: Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994.

[2] Glenda Carla Moura Amaral. Aquaware: Um ambiente de suporte à qualidade de dados em datawarehouse. Dissertação de Mestrado, Universidade Federal do Rio de Janeiro, Rio de Janeiro – RJ.

[3] R. V. Andreao. Implementação em tempo real de um sistema de reconhecimento de dígitos conectados. Dissertação de Mestrado, Universidade Estadual de Campinas, Campinas – São Paulo.

[4] Mikhail Bilenko e Raymond J. Mooney. Employing trainable string similarity metrics for information integration. Páginas 67–72, 2003.

[5] Ana Carolina de Almeida Carneiro. O uso de estruturas de metadados na implementação da qualidade dos dados de um datawarehouse. Relatório técnico, Universidade Federal do Rio de Janeiro, outubro de 2001.

[6] Peter Christen e Tim Churches. Febrl: Freely extensible biomedical record linkage. Relatório técnico, Australian National University, abril de 2005.

[7] Peter Christen, Tim Churches, e Justin X. Zhu. Probabilistic name and address cleaning and standardization. *Australasian Data Mining Workshop*, 2002. Disponível em: http://datamining.anu.edu.au/projects/linkage-publications.html.

[8] Peter Christen, Justin Xi Zhu, Martus Hegland, Stephen Roberts, Ole N. Nielsen, Tim Churches, e Kim Lim. High-performance computing techniques for record linkage. *Australian Helath Outcomes Conference (AHOC-2002)*, 2002. Disponível em: http://datamining.anu.edu.au/talks/2002/ahoc2002-4up.pdf.

[9] Tim Churches, Kim Lim Christen, e Justin Xi Zhu. Preparation of name and address data for record linkage using hidden markov models. *BMC - Biomed Central Medical Informatics and Decision Making*, 2(9), 2002. Disponível em: http://www.biomedcentral.com/1472-6947/2/9/.

[10] Peter Cristen e Tim Churches. A probabilistic deduplication, record linkage and geocoding system. *ARC Health Data Mining workshop*, 2005. Disponível em: http://acrc.unisa.edu.au/groups/health/hdw2005/Christen.pdf.

[11] Fred J. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964.

[12] Edgar D'Andréa. Cuide de seus dados. 2003. Disponível em: http://www.itweb.com.br/noticias. Último acesso em 26/10/2005.

[13] Mohamed G. Elfeky e Vassilios S. Verykios. Tailor: A record linkage toolbox. *International Conference on Data Engineering (ICDE)*, Páginas 17–28, 2002. Disponível em: http://www.cs.purdue.edu/homes/mgelfeky/Papers/icde02.pdf.

[14] I. Fellegi e A. Sunter. A theory for record linkage. *Journal of the American Statistical Society*, 64(328):1183–1210, 1969.

[15] Michael Gertz, M. Tamer Ozsu, e Gunter Saake. Report on the dagstuhl seminar: 'data quality on the web'. *ACM SIGMOD Record*, 33(1), 2004. Disponível em: http://www.acm.org/sigmod/record/issues/0403/R1.report4-tamer.pdf.

[16] M. A. Hernandex e S. J. Stolfo. The merge/purge problem for large databases. *ACM SIGMOD Conference*, 24(2):127–138, 1995.

[17] Fred. L. Drake Jr. Python tutorial (guido van rossum). Relatório técnico, Python Software Foundation, 2006. Release 2.4.3. Disponível em: http://docs.python.org/tut/.

[18] Luís Moura e Silva e Rajkumar Buyya. Parallel programming models and paradigms. *High Performance Cluster Computing: Architectures and Systems*, 2, 1999. Disponível em: http://www.buyya.com/cluster/v2chap1.pdf.

[19] Forum MPI. Mpi: A message-passing inteface standard. Relatório técnico, University of Tennessee, maio de 1995.

[20] H. B. NewCombe e J.M. Kennedy. Record linkage: Making maximum use of the discrimination power of identifying information. *Communications of the ACM*, 5(11):563–566, 1962.

[21] Ole Nielsen. Parallel programing in the spirit of python, 2004. Disponível em: http://datamining.anu.edu.au/~ole/pypar/.

[22] Marcelo Pereira Nunes. Aspectos formais da linguagem python. Relatório técnico, Fundação Universidade Federal do Rio Grande - FURG, 1998.

[23] Ken Orr. Data quality and systems theory. *Communications of the ACM*, 41(2):66–71, 1998.

[24] Gregory F. Pfister. *In Search of Clusters - The Ongoing Battle in Lowly Parallel Computing*. Prentice Hall, 1998. Segunda Edição.

[25] S. F. Pinho. Avaliacao da qualidade de dados pela nao conformidade. Dissertação de Mestrado, Universidade Federal do Rio de Janeiro, Rio de Janeiro – RJ.

[26] Marcos Pitanga. *Construindo Supercomputadores com Linux*. Brasport, 2004. Segunda Edição.

[27] L. R. A. Rabiner. Tutorial on hidden markov models and selected applications in speech recognition. *IEEE*, 77(2):257–286, 1989.

[28] Departament of Defense U.S. Dod guidelines on data quality management (summary). *Defence Information System Agency*, 2003. Disponível em: http://www.tricare.osd.mil/ocfo/_docs/DoDGuidelinesOnDataQualityManagement.pdf.

[29] Richard W. Wang e Diane M. Strong. Beyond accuracy: What data quality means to data consumers. *Journal of Management Information Systems*, 12(4):5–34, 1996.

[30] Justin Zobel e Philip Dart. Phonetic string matching: Lessons from information retrieval. *19th annual international ACM SIGIR conference on Research and Development in Information Retrieval*, Páginas 166–172, 1996. Disponível em: http://www.cs.rmit.edu.au/ jz/fulltext/sigir96.pdf.

Anexo A - Estados do Modelo Escondido de Markov

As duas tabelas abaixo contém todos os estados possíveis para nome e endereço dos Modelos Escondidos de Markov. Fonte [6].

Tabela de estados possíveis do Modelo Escondido de Markov para nome.

State	Description
titl	Title state
baby	State for *baby of*, *son of* or *daughter of*
knwn	State for *known as*
andor	State for *and* or *or*
gname1	Given name state 1
gname2	Given name state 2
ghyph	Given name hyphen state
gopbr	Given name opening bracket state
gclbr	Given name closing bracket state
agname1	Alternative given name state 1
agname2	Alternative given name state 2
coma	State for comma
sname1	Surname state 1
sname2	Surname state 2
shyph	Surname hyphen state
sopbr	Surname opening bracket state
sclbr	Surname closing bracket state
asname1	Alternative surname state 1
asname2	Alternative surname state 2
pref1	Name prefix state 1
pref2	Name prefix state 2
rubb	Rubbish state, for elements to be thrown away

Tabela de estados possíveis do Modelo Escondido de Marcov para endereço.

State	Description
titl	Title state
baby	State for *baby of*, *son of* or *daughter of*
knwn	State for *known as*
andor	State for *and* or *or*
gname1	Given name state 1
gname2	Given name state 2
ghyph	Given name hyphen state
gopbr	Given name opening bracket state
gclbr	Given name closing bracket state
agname1	Alternative given name state 1
agname2	Alternative given name state 2
coma	State for comma
sname1	Surname state 1
sname2	Surname state 2
shyph	Surname hyphen state
sopbr	Surname opening bracket state
sclbr	Surname closing bracket state
asname1	Alternative surname state 1
asname2	Alternative surname state 2
pref1	Name prefix state 1
pref2	Name prefix state 2
rubb	Rubbish state, for elements to be thrown away

Apêndice A - Formato de Saída: Lista de Detalhes

Este apêndice mostra um exemplo do formato de arquivo de saída lista de detalhes.

Resulting record pairs:

Output threshold: 10.000000

Data set A: example2tmp

Data set B: example2tmp

Weight: 11.567734

Fields	[RecID A: 4890/example2tmp]	[RecID B: 4935/example2tmp]
address_hmm_	1.04217665748e-07	1.04217665748e-07
gender_guess	female	female
given_name	carla	carla
rec_id	rec-851-dup-171	rec-851-dup-243
surname	alvse	alvsd
unit_type	para	para
wayfare_name	pedro_calisto merces salvador	pedro_calisto merces salvador
wayfare_numb	2370	2370
wayfare_type	rua	rua

Weight: 11.567734

Fields	[RecID A: 1724/example2tmp]	[RecID B: 3584/example2tmp]
address_hmm_	1.20540435338e-14	1.20540435338e-14
gender_guess	male	male
given_name	hugo	hugo
rec_id	rec-1642-dup-29	rec-1642-dup-24
surname	manzatyl	manzwtao
unit_type	para	para
wayfare_name	jardim_alegre ganchinho	jardim_alegre ganchinho
wayfare_numb	133	133
wayfare_type	rua	rua

Weight: 11.315827

Fields	[RecID A: 1041/example2tmp]	[RecID B: 3529/example2tmp]
address_hmm_	1.36752978293e-13	1.36752978293e-13
gender_guess	female	female
given_name	monica	monica
rec_id	rec-329-dup-26	rec-329-dup-0
surname	resende	resende
unit_type	paui	piaui
wayfare_name	presidente_getulio_vargas	presidente_getulio_vargas
wayfare_numb	766	766
wayfare_type	avenida	avenida

Esta lista de detalhes é o resultado de alguns testes com conjunto de dados brasileiros fictícios criado com o programa *generate.py* que faz parte do ambiente de software FEBRL.

Apêndice B - Tabelas Look Up

Tabelas *look up* desenvolvidas nesse trabalho.

Parte da tabela *look up* de nomes próprios masculinos

```
tag=<GM>  # Símbolo de Identificação para nomes proprios masculinos

        abel : abl, abell
    abelardo : abellardo
      abilio : abilho
      abraao : abrao, habraao, habrao
   adalberto : adaberto, adalbertho, ababertho
     adalton : adaltom
        adao : hadao
      ademar :
      ademir : admir, adimir
     adilson : adilson
     adriano : adrian
      afonso : affonso, afonzo
      agenor : argenor
     agnaldo : aginaldo, aguinaldo
      ailton : ailton
      airton : airthon, airto, airto, ailton, ailthon
        alan : alam, halan, halam
     alberto : albeto
       alceu : halceu
  alessandro : alessandre, alexandre, alexandro
        alex : alexi
     alfredo :
       almir :
     aloisio : aloisio, aluisio
      alvaro : alvaro
      amauri : amaury
    anderson : andersom
       andre : andreh
      angelo : angello
     anselmo :
     antonio : antonio, antonho, anthony
```

Parte da tabela *look up* de sobrenomes

```
tag=<SN>  # Símbolo de identificação para sobrenomes

       abbot : abot
   abranches : abranche
       abreu :
      acosta :
       adaid : adaidi
      afonso :
     agostini : agostinni
       aguiar :
      aguilar :
     alberico :
     alberton :
     albornoz : albornos
  albuquerque : alburquerque
      alencar :
    alcantara : alcantra
   alcoforado :
      alegria :
       aleixo :
       aliaga :
      almeida : aumeida
    alvarenga :
      alvares : alvarez
        alves :
        alvim :
       amaral :
       amorim : amorin
    amstalden :
      andrade :
        annes : anes
   antoniazzi : antoniazi
```

Tabela look up de tipo de logradouro

tag=<WT> # Símbolo de identificação para tipo de rua

acesso :

alameda : al

area :

avenida : av, aven

arraial :	
beco :	
caminho :	
campo :	
arraial :	
chacara :	
colonia :	
distrito :	
estacao :	est
estrada :	estr
favela :	
fazenda :	
largo :	lgo
nucleo :	
parque :	
patio :	
praca :	pc
quadra :	
recanto :	
residencial :	
rodovia :	rod
rua :	r. ru
ruela :	
sitio :	
travessa :	trav, tv
vale :	
vereda :	
via :	vai
viaduto :	
viela :	
vila :	

Tabela look up de instituição

tag=<IT> # Símbolo de Identificação para palavras institucionais

aeroporto :	
albergue :	
acampamento :	alojamento
asilo :	casa de repouso
companhia :	industria, fabrica
convento :	
cooperativa :	coperativa
delegacia :	casa de detencao
departamento :	dept
distribuidor :	estr
empresa :	
estacionamento :	
faculdade :	
hospital :	hosp
hotel :	
pensao :	pensionato
posto de saude :	
universidade :	
unidade :	

Tabela look up de tipo de unidade

tag=<UT> # Símbolo de Identificação para complemento de endereço

apartamento :	apt, ap, apto
anexo :	
andar :	
bloco :	bl
casa :	residencia

cobertura :	
condominio :	
conjunto :	cj, conj, conjunto habitacional
caixa postal :	cp
edificio :	predio, ed
entrada :	
escritorio :	
flat :	
frente :	
fundos :	fundo, casa dos fundos, casa do fundo
garagem :	
lote :	terreno
mansao :	
modulo :	
porao :	
portao :	
proximo :	
quarto :	
quilometro :	km
sala :	
sobrado :	sob
subsolo :	
terreo :	andar terreo, solo

Apêndice C - Lista de Correção para Endereço

Abaixo segue a lista de correção para endereço utilizada no processo de limpeza dos registros.

```
# Remove caracteres e palavras da entrada
              ' ' := ' ', '?', '~', '_', ' ', ':', ';', '^', '=', ' na ',
                ' n/a ', ' n.a.', ' ! ', '*', '`'

# Palavras e simbolos corretos
' sem numero ' := ' s/n ', ' s / n ', ' sn ', 'sno'
          ' e ' := '+', '&'
          ' - ' := '-', '/'
          ' | ' := '<', '>', '(', ')', '[', ']', '{', '}', '"', "'", '|'

# Remove o apostofro (')
             ' o' := " o'"
             ' a' := " a'"
             ' l' := " l'"
             ' i' := " i'"
             '-o' := "-o'"
             '-a' := "-a'"
             '-l' := "-l'"
             '-i' := "-i'"
#                 's ' := "'s "
#                 's ' := "''s "

# Remove os acentos das palavras
                 'a' := "ã", "â", "á", "à", "Ã", "Â", "Á", "À"
                 'o' := "ó", "õ", "ô", "Ó", "Õ", "Ô"
                 'u' := "ú", "ü", "Ú", "Ü"
                 'c' := "ç", "Ç"
                 'e' := "é", "ê", "É", "Ê"
                 'i' := "í", "Í"


#Numeros romanos
          'quinze' := "xv", "XV"
```

Apêndice D - Testes das Variáveis de Bloco

Este apêndice apresenta os detalhes dos testes realizados para escolher as variáveis de bloco que foram usadas nos testes finais. Para a escolha das variáveis de bloco, foram realizados 5 testes com tipos de variáveis diferentes.

1º Teste

Variáveis usadas:

[[('*surname*','*dmetaphone*', 4),('*wayfare_type*','*nysiis*')],

[('*given_name*','*truncate*', 3), ('*postcode*','*direct*')],

[('*locality_name*','*nysiis*'),('*unit_type*','*truncate*', 2)],]

2º Teste

Variáveis usadas:

[[('*given_name*','*nysiis*'),('*locality_name*','*truncate*', 3)],

[('*surname*','*dmethaphone*', 3), ('*postcode*','*direct*')],]

3º Teste

Variáveis usadas:

[[('*surname*','*truncate*', 4),('*wayfare_name*','*nysiis*')],

[('*locality_name*','*soundex*'), ('*given_name*','*phonex*')],]

4º Teste

Variáveis usadas:

[[(‘*given_name*’,‘*nysiis*’),(‘*locality_name*’,‘*truncate*’, 3)],

[(‘*postcode*’,‘*direct*’), (‘*surname*’,‘*mod_soundex*’)],]

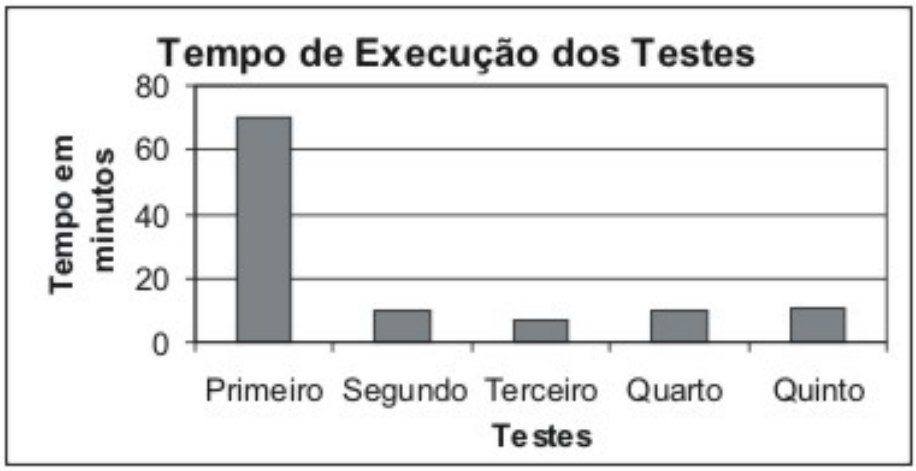
5º Teste

Variáveis usadas:

[[(‘*surname*’,‘*truncate*’, 4),(‘*postcode*’,‘*direct*’)],

[(‘*locality_name*’,‘*truncate*’, 3), (‘*given_name*’,‘*nysiis*’)],]

Para ficar mais fácil a visualização, a seguir é apresentado um gráfico relacionado ao tempo de execução dos cinco testes realizados.



Apêndice E - Arquivos de Saída do Teste Final

Neste apêndice são apresentados os arquivos de saída do teste final realizado no estudo de caso com o conjunto de dados do Sistema de Bibliotecas da Universidade Federal do Paraná.

HISTOGRAMA DO CONJUNTO DE DADOS SIBI

Weight histogram:

---

20 * 152

21 ********** 18967

22 ** 386

23 ** 505

24 ***** 1183

25 ********** 11225

26 ** 385

27 ** 351

28 **** 888

29 ** 463

30 *** 705

31 *** 736

32 ** 349

33 ** 320

34 ** 503

35 ********** 5828

36 * 6

37 * 3

38 * 9

39 * 9

40 * 42

41 * 9

42 * 27

43 * 11

44 * 13

45 * 18

46 * 11

47 * 10

48 * 16

49 * 4

50 *** 659

LISTA DE IDENTIFICADORES DO CONJUNTO DE DADOS SIBI

Observação: é apresentada somente uma parte da lista de identificadores devido sua extensão.

Rec_ID_A, Rec_ID_B, Weight, Assigned

0,33602,32.777724,assigned

3,26293,35.291504,assigned

3,7,35.291504,

4,29193,31.647333,assigned

5,8478,31.800772,assigned

7,26293,35.291504,

8,56911,35.291504,

8,47525,35.291504,

8,41857,35.291504,

8,36426,35.291504,

8,22423,35.291504,

11,15299,33.870278,assigned

13,18220,35.291504,assigned

14,51561,35.291504,

14,24812,35.291504,

23,40098,35.291504,assigned

23,472,31.390097,

25,40769,35.291504,assigned

32,24690,31.390097,assigned

33,15545,35.291504,assigned

39,19119,35.291504,assigned

42,29024,35.291504,assigned

45,29123,35.291504,

45,16312,35.291504,

51,3222,31.403728,assigned

64,45832,35.291504,

64,32578,35.291504,assigned

64,36062,30.614818,

66,39863,35.291504,assigned

67,34199,44.487678,assigned

67,21488,30.275410,

69,40161,35.291504,assigned

75,48025,32.449051,assigned

85,21877,35.299721,assigned

89,44870,35.291504,

89,35992,35.291504,assigned

89,19380,35.291504,

89,3050,35.291504,

92,21581,32.299448,assigned

121,25991,35.291504,assigned

122,24931,35.291504,assigned

131,50558,35.291504,

131,25421,32.133223,

143,31378,30.522914,assigned

150,14784,33.108170,assigned

155,47504,30.188386,assigned

159,42733,35.291504,assigned

166,10516,31.975308,assigned

166,6893,31.975308,

180,26140,34.389138,assigned

199,49634,35.291504,assigned

207,46925,35.291504,assigned

210,7652,30.198775,assigned

228,4832,44.450522,assigned

228,7854,35.291504,

228,3502,30.238253,

248,38870,35.291504,assigned

265,43586,35.291504,assigned

A lista de detalhes do conjunto de dados sibi não será apresentada neste apêndice para preservar as informações dos usuários cadastrados.