UNIVERSIDADE ESTADUAL PAULISTA
"Júlio de Mesquita Filho"
Pós-Graduação em Ciência da Computação

Juliano Augusto Carreira

Enriquecimento de dados: uma pré-etapa em relação à limpeza de dados

UNESP
2012

Juliano Augusto Carreira

Enriquecimento de dados: uma pré-etapa em relação à limpeza de dados

Orientador: Prof. Dr. Carlos Roberto Valêncio

Dissertação de Mestrado elaborada junto ao Programa de Pós-Graduação em Ciência da Computação – Área de Concentração em Engenharia de Software e Banco de Dados, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

UNESP

2012

JULIANO AUGUSTO CARREIRA

Enriquecimento de dados: uma pré-etapa em relação à limpeza de dados

Dissertação apresentada para obtenção do título de Mestre em Ciência da Computação, área de Engenharia de Software e Banco de Dados junto ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Campus de São José do Rio Preto.

BANCA EXAMINADORA

Prof. Dr. Carlos Roberto Valêncio
Professor Doutor
UNESP – São José do Rio Preto
Orientador

Prof. Dr. José Márcio Machado
Professor Doutor
UNESP – São José do Rio Preto

Profª. Dra. Marilde Terezinha Prado Santos
Professora Doutora
UFSCar – Universidade Federal de São Carlos

São José do Rio Preto, 12 de julho de 2012

Agradecimentos

Ao meu orientador, Prof. Dr. Carlos Roberto Valêncio, pela grande amizade desenvolvida ao longo de todos esses anos de convivência, pela orientação cuidadosa, pelos inúmeros conselhos pessoais e científicos, pelas correções e adequações referentes a este trabalho e que foram fundamentais para sua conclusão.

Aos meus pais, Doraci Costa Carreira e Osvaldo Carreira (*in memoriam*) por todo apoio, amor, carinho e, também, por todos os conselhos que me foram dados e as oportunidades que me foram oferecidas.

A toda minha família que, de uma forma geral, sempre me apoiou e me incentivou a seguir essa jornada.

À Simone Martins da Silva, pela dedicação, carinho e palavras amigas na fase final deste trabalho.

Aos Profs. Drs. Rogéria Cristiane Gratão de Souza e José Márcio Machado pelas contribuições em ocasião do Exame Geral de Qualificação.

A todos os professores do Programa de Pós-Graduação em Ciência da Computação do IBILCE – UNESP, que de forma direta ou indireta contribuíram para minha formação e conclusão deste trabalho.

Aos meus chefes da IBM, Ki Hwang e Humberto Trinca, que sempre permitiram que me ausentasse das atividades laborais para participar das atividades presenciais do programa de pós-graduação.

Aos amigos da pós-graduação, em especial a Cleriston Araujo Chiuchi, Carlos Henrique El Hetti Laurenti, Fernando Tochio Ichiba e Tiago Luiz de Andrade, por participarem de todos os momentos de alegrias e dificuldades vivenciados.

A todos que, direta ou indiretamente, contribuíram para a realização desta pesquisa.

“No meio da dificuldade encontra-se a oportunidade.”

Albert Einstein

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

KDD	*Knowledge Discovery in Databases*
SGBD	Sistema Gerenciador de Banco de Dados
NYSIIS	*New York State Identification and Intelligence System*
ONCA	*Oxford Name Compression Algorithm*
CEBMDC	*Clustering Ensemble Based Mixed Data Clustering*
Febrl	*Freely Extensible Biomedical Record Linkage*
HMM	*Hidden Markov Models*
SQL	*Structured Query Language*
GBD	Grupo de Banco de Dados
SIVAT	Sistema de Informação e Vigilância de Acidentes do Trabalho
CEREST	Centro de Referência em Saúde do Trabalhador

Resumo

A incidência de tuplas duplicadas é um problema significativo e inerente às grandes bases de dados atuais. Trata-se da repetição de registros que, na maioria das vezes, são representados de formas diferentes nas bases de dados, mas fazem referência a uma mesma entidade do mundo real, tornando, assim, a tarefa de identificação das duplicatas um trabalho árduo. As técnicas designadas para o tratamento deste tipo de problema são geralmente genéricas. Isso significa que não levam em consideração as características particulares dos idiomas o que, de certa forma, inibe a maximização quantitativa e qualitativa das tuplas duplicadas identificadas. Este trabalho propõe a criação de uma pré-etapa – intitulada “enriquecimento” – referente ao processo de identificação de tuplas duplicadas. Tal processo baseia-se no favorecimento do idioma e se dá por meio da utilização de regras de linguagem pré-definidas, de forma genérica, para cada idioma desejado. Assim, consegue-se enriquecer os registros de entrada, definidos em qualquer idioma, e, com a aproximação ortográfica que o enriquecimento proporciona, consegue-se aumentar a quantidade de tuplas duplicadas e/ou melhorar o nível de confiança em relação aos pares de tuplas duplicadas identificadas pelo processo.

Abstract

The incidence of duplicate tuples is a significant problem inherent in current large databases. It is the repetition of records that, in most cases, are represented differently in the database but refer to the same real world entity thus making the task of identifying duplicates a hard work. The techniques designed to treat this type of problem are usually generic. That means they do not take into account the particular characteristics of the languages that somehow inhibits the quantitative and qualitative maximization of duplicate tuples identified. This dissertation proposes the creation of a pre-step - called "enrichment" – in relation to the process of duplicate tuples identification. This process is based on the language favoring and is through the use of predefined language rules in a general way for each language. Thus, it is possible to enrich the input records defined in any language and considering the spell approximation provided by the enrichment process, it is possible to increase the amount of duplicate tuples and/or improve the level of trust in relation to the pairs of duplicate tuples identified by the process.

Capítulo 1
Introdução

1.1 Considerações iniciais

Em virtude da grande evolução tecnológica dos últimos anos, a proeminência de grandes volumes de dados – nas mais diversas áreas – faz-se muito presente na contemporaneidade. Além de somente se preocuparem com os procedimentos para armazenamento e recuperação desses dados, os detentores dos mesmos vêm se preocupando com algo a mais, algo que possa ajudá-los na agregação de valor aos seus negócios e também possa auxiliá-los nas tomadas de decisões. Essa necessidade, ainda desconhecida para muitos, pode ser atendida pelo processo de extração de conhecimento das bases de dados, "*Knowledge Discovery in Databases* – KDD" (PIATETSKY-SHAPIRO, 1990) que, por sua vez, aceita definição mediante um "processo não trivial de identificação de padrões válidos, desconhecidos, potencialmente úteis e compreensíveis a partir de um volume de dados" (FAYYAD;PIATETSKY-SHAPIRO; SMYTH, 1996).

O processo de KDD é composto por um conjunto de etapas interligadas, que devem ser respeitadas para alcançar seu objetivo final. Cada uma delas possui sua importância individual, portanto não deveriam ser deixadas de lado, por mais simples que possam parecer. Essas etapas podem, ainda, ser definidas de uma forma mais genérica por meio de três grandes estágios principais: pré-processamento de dados, *data mining* e pós-processamento dos dados.

Nas bases de dados existentes atualmente, seja por um projeto ruim ou por falhas no processo de alimentação das mesmas, são recorrentes os problemas relacionados à

inconsistência dos dados como: erros de digitação, valores nulos ou fora do domínio, tuplas duplicadas, inexistência de chave primária, etc. No sentido de solucioná-los, o processo de KDD fornece a fase de limpeza de dados dentro da ampla etapa de pré-processamento, cujo objetivo principal é garantir a integridade e a consistência dos dados para etapas futuras, mais precisamente a fase de *data mining* que é responsável por extrair conhecimento útil desses dados (MITRA; ACHARYA, 2003). Sendo assim, a não realização dessa etapa pode comprometer a confiabilidade dos resultados, e até prejudicar as organizações que se baseiam neles para tomada de decisão.

Dentre todas as atividades realizadas pela fase de limpeza de dados, podem-se destacar as técnicas para identificação de tuplas duplicadas não idênticas, que possuem a incumbência de identificar tuplas que são representadas de forma diferente na base de dados, mas que, ao mesmo tempo, fazem referência a objetos idênticos no mundo real. As técnicas utilizadas nesse tipo de atividade podem ser classificadas em duas categorias principais: *ad-hoc*, que são mais eficientes e lidam melhor com grandes volumes de dados; e as probabilísticas, que são mais precisas em seus resultados (ELMAGARMID; IPEIROTIS; VERYKIOS,2007).

Apesar de realizarem um ótimo trabalho na identificação de tuplas duplicadas, essas técnicas geralmente são propostas de forma genérica. Tal abordagem desmembra-se em lados positivo e negativo. Positivo, pois permite uma utilização global de suas técnicas sem restrições idiomáticas, além de criar uma base conceitual sólida que possibilita a ascensão desta área de pesquisa; negativo, pois desconsidera as características particulares de cada idioma, dificuldades específicas e erros comumente cometidos pelos seus falantes. Se o caso negativo fosse levado em consideração, seria possível maximizar a quantidade de tuplas duplicadas identificadas e/ou melhorar seu nível de confiabilidade.

O foco deste trabalho está no favorecimento do idioma. Para tanto, aplicam-se técnicas *ad-hoc* conhecidas e validadas na identificação de tuplas duplicadas, com o intuito de melhorar seus resultados apresentados quando expostos à forma genérica. Tal proposição oferece um enriquecimento do texto dos registros de uma base de dados, levando em consideração características típicas de cada um dos idiomas suportados pelo ambiente. Propõe-se, portanto, a construção de um ambiente estruturado para receber qualquer idioma desejado. Mediante tais explanações, é correto afirmar que este trabalho propõe uma pré-etapa para a realização do processo de identificação de tuplas duplicadas que constitui a fase de pré-processamento de dados no processo de KDD.

Consciente de que essa pré-etapa gera um custo adicional ao procedimento de identificação de tuplas duplicadas, acarretando sua possível não utilização dentro do processo, também se propõe, de forma antecipada, a utilização de dois esquemas de processamento paralelo com auxílio de programação *multithreading*. Tal prática tem o intuito de utilizar os vários núcleos de processamento oferecidos pelos processadores modernos para maximizar a eficiência de processamento e tornar essa pré-etapa mais atrativa.

1.2 Motivação e escopo

Embora as técnicas probabilísticas para identificação de tuplas duplicadas não idênticas sejam mais precisas, elas não lidam bem com grandes volumes de dados, fato este que as distanciam da tendência das bases de dados atuais que ficam maiores a cada dia. Por esse motivo, as técnicas *ad-hoc* são mais utilizadas para essa tarefa e também recebem uma maior atenção por parte dos pesquisadores, cujo intuito é de aproximar a precisão dos resultados fornecidos aos resultados conferidos pelas técnicas probabilísticas. De um modo geral, as pesquisas realizadas para as técnicas *ad-hoc* buscam melhorar sua eficiência e eficácia de uma forma ampla para que possam ser aplicadas em ambientes distintos, bases de dados distintas e também idiomas distintos, embora nem sempre seja possível (ELMAGARMID; IPEIROTIS; VERYKIOS,2007).

Ao analisar os registros presentes em bases de dados reais, o estado da arte dos ambientes de identificação de tuplas duplicadas e o fato de que os idiomas são diferentes e que cada um possui suas características peculiares, notou-se que a criação de uma etapa, anterior a de identificação de tuplas duplicadas não idênticas, em que o fator idioma seja favorecido, pode melhorar a qualidade e a precisão dos resultados apresentados pelas técnicas *ad-hoc*. Tal alusão refere-se ao tratamento dos dados de um idioma específico por meio de algumas operações de texto que, se utilizadas, podem: eliminar palavras sem muito significado; suprimir erros comuns cometidos pelos falantes do idioma; substituir palavras sinônimas, além de realizar algumas outras contribuições de menor expressão que, de um modo geral, aproximam ortograficamente os registros de uma base de dados. Essas são, portanto, operações de texto que podem ser desenvolvidas por meio da observação dos problemas mais comuns presentes nas bases de dados e de absorções da área de recuperação de informação, por exemplo:

tesauros, corretores ortográficos, *stemming* e eliminação de *stopwords* (YATES; RIBEIRO NETO, 2000).

Para essa tarefa de favorecimento do idioma, utilizar-se-á ao longo do trabalho o termo “enriquecimento”. Pois o que realmente ocorre nessa atividade é um enriquecimento do texto antes da aplicação de técnicas de identificação de tuplas duplicadas. Uma vez o texto enriquecido, acredita-se que o procedimento de identificação de tuplas duplicadas pode obter melhores resultados.

1.3 Objetivos

O objetivo do presente trabalho é desenvolver uma pré-etapa em relação ao processo de identificação de tuplas duplicadas – na forma de um ambiente – com o intuito de enriquecer bases de dados que estejam definidas em qualquer idioma juntamente com técnicas de identificação de tuplas duplicadas *ad-hoc*. Como consequência, busca-se que a quantidade de resultados encontrados maximize-se e/ou o nível de confiabilidade de classificação dos pares de tuplas duplicadas aumente. A ideia é construir um ambiente que esteja totalmente estruturado para receber regras de qualquer idioma para que, posteriormente, o ambiente possa recuperá-las e utilizá-las no enriquecimento do texto dos registros da base de dados. Assim, consequentemente, obter-se-iam melhores resultados no processo de identificação de tuplas duplicadas.

Com o intuito de aproveitar os vários núcleos de processamento oferecidos pelos processadores atuais e aprimorar a eficácia da tarefa de enriquecimento, pretende-se também acomodar as fases integrantes da pré-etapa proposta em *threads* paralelas.

1.4 Organização do trabalho

Este documento está estruturado em cinco capítulos, conforme explicitado a seguir.

O Capítulo 2 apresenta a fundamentação teórica utilizada para elaboração do trabalho, juntamente com as técnicas *ad-hoc* mais utilizadas no processo de identificação de tuplas duplicadas e o estado da arte relacionado aos ambientes de mesmo objetivo previamente propostos.

O Capítulo 3 apresenta uma descrição detalhada da pré-etapa proposta juntamente com o ambiente construído para concretizá-la.

O Capítulo 4 apresenta os testes realizados e os resultados obtidos pelo ambiente.

O Capítulo 5 apresenta uma conclusão sobre o trabalho como um todo e também, algumas propostas de atividades futuras que podem ser incluídas no mesmo.

Capítulo 2
Revisão Bibliográfica

2.1 Considerações iniciais

Diante da expressiva evolução das tecnologias de *hardware* e *software* nos últimos anos, tornaram-se possíveis o armazenamento e a recuperação de informações de maneira cada vez mais eficiente. Em consequência disso, o aparecimento de grandes volumes de informação nas mais diversas áreas, como engenharia, estatística, medicina, economia, comércio, indústria, entre outras, ocorre com muita frequência.

Atualmente, o interesse em se encontrar técnicas mais eficientes de armazenamento e recuperação de informações é substituído gradativamente pelo interesse em explorar esses grandes volumes de dados em busca de conhecimento útil e ainda desconhecido. Essa busca, evidentemente, torna-se inviável de ser feita de forma manual, em virtude da enorme quantidade de dados a serem analisados e também, por ser um processo lento, caro e muito subjetivo. Sendo assim, a recorrência às técnicas computacionais como *Knowledge Discovery in Database* (KDD) torna-se inevitável (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996).

O processo de KDD é formado por várias fases e, dentre elas, existe uma em especial denominada limpeza de dados (*data cleansing*) cujo objetivo principal é fornecer dados mais confiáveis para fase de *data mining*, também pertencente ao processo de KDD. Pelo fato de ser responsável por fornecer qualidade aos dados, a fase de limpeza de dados lida com vários problemas inerentes aos mesmos, como tuplas duplicadas (idênticas e não idênticas), valores nulos, valores fora do domínio e etc.

Neste capítulo, serão apresentados conceitos relacionados ao processo de KDD e à fase de limpeza de dados com foco nas técnicas envolvidas na tarefa de detecção de tuplas duplicadas, que é o propósito deste trabalho. Além disso, serão mostradas também algumas ferramentas existentes cuja finalidade envolve a realização da tarefa de identificação de tuplas duplicadas.

2.2 *Knowledge discovery in databases* (KDD)

"O processo de KDD é o processo não trivial de identificação de padrões válidos, desconhecidos, potencialmente úteis e compreensíveis a partir de um volume de dados" (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996).

Historicamente, a ideia de se encontrar padrões úteis em volumes de dados recebeu alguns nomes distintos como *data mining*, *knowledge extraction*, *information discovery*, *information harvesting*, *data archaeology*, *knowledge discovery in database* e *data pattern processing*, porém os dois termos que mais prevaleceram e que, muitas vezes, são utilizados como sinônimos até os dias atuais são: *data mining* e *knowledge discovery in databases*.

O conceito de *Knowledge Discovery in Databases* apareceu no evento "KDD-89 *Workshop"* organizado por Gregory Piatetsy-Shapiro (PIATETSKY-SHAPIRO, 1990) e faz referência ao processo de descoberta de conhecimento útil em bases de dados, trazendo o conceito de *data mining* como uma de suas etapas cujo objetivo principal é a aplicação de algoritmos para extração de padrões das bases de dados. A palavra "processo" possui um peso muito forte na definição do termo KDD, já que expressa a existência de fases sequênciais e que precisam ser executadas, sem exceção, para obter um nível mais elevado de sucesso na produção do conhecimento.

O processo de KDD pode ser descrito basicamente por três etapas como é apresentado a seguir e ilustrado pela figura 1.

- **Pré-processamento dos dados:** fase responsável por integrar e limpar os dados que servirão de matéria prima para as fases subsequentes. Essa é uma fase muito importante e de grande valia para o processo, pois está diretamente ligada à qualidade do resultado final. É ela que oferece maior integridade e consistência aos dados que serão utilizados para extração de conhecimento útil.

- ***Data Mining*:** fase em que ocorre a aplicação de algoritmos nos dados já integrados, consistentes e consolidados com o objetivo de extrair padrões.
- **Pós-processamento dos resultados:** fase em que são feitas a visualização e a interpretação dos padrões extraídos na fase de *data mining*, pois estes podem ser incompreensíveis e, por isso, necessitam de ajuda para serem interpretados.

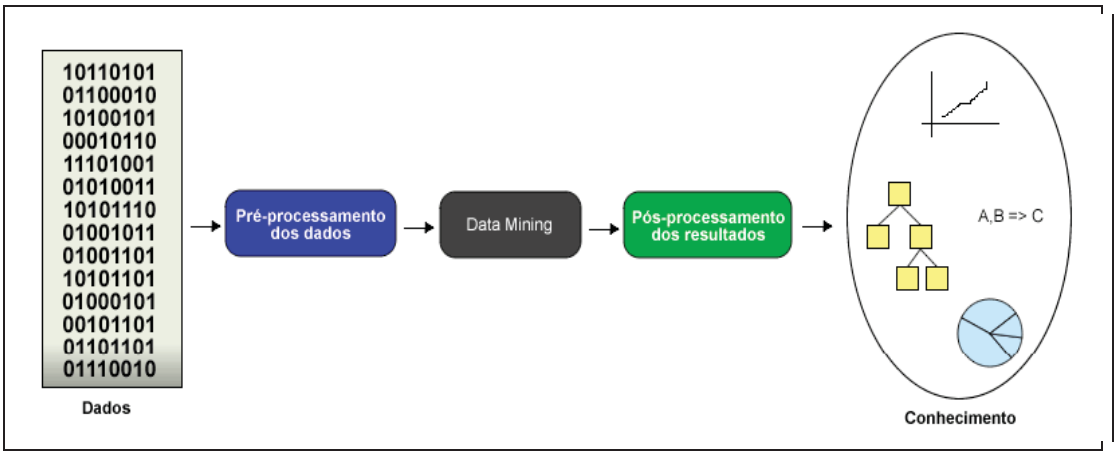

Figura 1 – Processo de KDD descrito em três fases.

Em (MITRA; ACHARYA, 2003) é apresentado um detalhamento maior dessas três etapas básicas do processo de KDD por meio de seis etapas mais detalhadas.

- **Compreensão do domínio da aplicação**: etapa responsável por desenvolver uma compreensão do domínio da aplicação e do conhecimento prévio relevante e identificar o objetivo do processo de KDD sob o ponto de vista do detentor do volume de dados a ser explorado. Essa atividade não pode ser desempenhada por qualquer pessoa, pois é preciso muita habilidade e *expertise* para entender o negócio ao qual se deseja criar conhecimento para ajudar nas tomadas de decisões.
- **Extração de um conjunto de dados alvo**: etapa responsável por selecionar um conjunto de dados ou concentrar-se em um subconjunto de variáveis ou amostra de dados nos quais a extração de conhecimento será aplicada.
- **Preparação de dados**: etapa importante para fornecer maior qualidade aos dados que serão analisados e ajudar na redução do tempo de prospecção, visto que muita informação inútil pode ser removida. Essa fase está subdividia nas seguintes subfases não mutuamente exclusivas (HAN; KAMBER,2006):

    - **Limpeza de dados**: consiste na realização de operações básicas, como padronização, remoção de ruídos, preenchimento de valores ausentes e redução de redundância, a fim de corrigir as inconsistências existentes nos dados.
    - **Integração de dados**: consiste em integrar múltiplos e heterogêneos volumes de dados a partir de diferentes fontes em um único e coerente volume de dados, como um *data warehouse*.
    - **Transformação de dados**: etapa que realiza transformação dos dados por meio de técnicas como normalização, e tem como objetivo melhorar a precisão e o desempenho dos algoritmos de *data mining*. Por exemplo, a normalização de números inteiros para valores entre 0 e 1.
    - **Redução de dados**: etapa que pode reduzir o volume de dados por meio de técnicas como agregação, redução de características redundantes e agrupamento.
- ***Data Mining***: etapa de maior destaque no processo de KDD a qual é responsável pela escolha e aplicação de um ou mais métodos para reconhecimento de padrões úteis.
- **Interpretação**: envolve a visualização e interpretação dos padrões descobertos e a remoção dos padrões redundantes e irrelevantes, visto que é muito difícil fazer uma limpeza 100% eficaz e traduzir os padrões úteis extraídos em termos compreensíveis para os usuários.
- **Utilização do conhecimento descoberto**: envolve a incorporação do conhecimento em um sistema de desempenho, realização de ações baseadas no conhecimento ou simplesmente a documentação e disseminação de informação para as partes interessadas.

Embora as fases de pré-processamento e pós-processamento dos dados sejam muito importantes no processo de KDD, é possível realizar a extração de conhecimento dos volumes de dados apenas utilizando a fase de *data mining*. Esse fato é o grande responsável por fazer com que essa fase receba mais atenção que as outras e pela criação da ideia de similaridade entre os conceitos *data mining* e KDD. Embora seja mais simples, não é recomendável que somente a fase de *data mining* seja utilizada como substituição de todo o processo de KDD, pois a ausência da fase de preparação dos dados, por exemplo, pode resultar na presença de informações irrelevantes

misturadas aos padrões úteis extraídos pelas técnicas de *data mining*, e, consequentemente, prejudicar a organização por meio de danos financeiros, insatisfação dos clientes, perda de mercado para concorrência e etc.

2.3 Limpeza de dados (*Data Cleansing*)

Data cleansing, também denominado como *data cleaning* ou *scrubbing*, é a área responsável por lidar com a detecção e remoção de erros e inconsistências dos dados com a intenção de melhorar a sua qualidade. Os problemas relacionados à qualidade dos dados geralmente estão presentes em coleções únicas de dados (como arquivos e bases de dados). Esses problemas geralmente são provenientes de erros de digitação, ausência de informações e também pela ocorrência natural de erros nos dados. Quando múltiplas fontes de dados precisam ser integradas, por exemplo, nos casos dos *data warehouses* e dos banco de dados federados, a necessidade da presença de técnicas de limpeza de dados aumenta significativamente, visto que, geralmente, essas fontes de dados distintas envolvidas no processo possuem dados redundantes e são representadas de maneira diferente.

Os *data warehouses*, frequentemente presentes no processo de KDD, requisitam um extenso suporte às estratégias de limpeza de dados. Eles integram e atualizam grandes quantidades de dados de múltiplas fontes o tempo todo, e a probabilidade de existir dados inconsistentes nessas fontes é muito alta. Além disso, mesmo com o advento das técnicas de *data mining* multi-relacional (DZEROSKY et al., 2003), os *data warehouses* ainda são muito utilizados nos processos de tomada de decisões nas organizações e, por isso, a confiabilidade e precisão dos dados que os constituem é fundamental para evitar conclusões precipitadas pelas organizações. Por exemplo, informações duplicadas e/ou ausentes podem produzir relatórios estatísticos não condizentes com a realidade e induzir de forma ilusória as decisões de uma organização.

Diante da grande quantidade de possíveis inconsistências nos dados, a limpeza de dados pode ser considerada como um dos maiores problemas nos projetos de *data warehouse* (RAHM; DO, 2000). Para comprovar a grande importância da fase de limpeza de dados para o processo de KDD, pode-se considerar alguns números apresentados sobre o assunto:

- “Problemas atuais referentes à qualidade dos dados custam mais de 600 bilhões de dólares para as organizações dos Estados Unidos” (ECKERSON, 2002);
- “Entre 30% e 80% do tempo no processo de KDD é gasto em limpeza e entendimento dos dados” (DASU; JOHNSON, 2003);
- “As organizações geralmente esperam taxas de erro entre 1% e 5% nos seus dados e algumas organizações chegam a ter uma expectativa acima dos 30%” (FAN; GEERTS; JIA, 2008).

Além dos problemas relacionados ao processo de KDD já mencionados anteriormente, a ausência da fase de limpeza de dados pode causar outras adversidades relacionadas às atividades triviais das organizações, como o aumento de custos relacionados ao envio de correspondências para clientes. Assim, uma situação imaginável seria a de uma organização que possua uma base de dados de clientes com uma ampla quantidade de tuplas duplicadas e que planeje realizar ofertas por meio de catálogos enviados aos seus clientes. Dependendo da quantidade de duplicatas e do grau de sofisticação destes catálogos, uma quantidade considerável de dinheiro pode ser desperdiçada nesta tarefa por meio da produção de catálogos extras desnecessários.

A limpeza de dados geralmente lida com os seguintes problemas: tuplas duplicadas, valores nulos, valores fora do domínio e valores fora do padrão (HAN; KAMBER,2006). Nas subseções a seguir, será apresentada uma visão mais detalhada destes problemas e também possíveis técnicas para solucioná-los.

2.3.1 Tuplas duplicadas

O problema de tuplas duplicadas pode ser caracterizado de duas formas: tuplas duplicadas idênticas e tuplas duplicadas não idênticas. A solução para o problema de tuplas duplicadas idênticas é bem simples e envolve apenas a utilização de expressões SQL do tipo “SELECT DISTINCT ...”. Entretanto a identificação de tuplas duplicadas não idênticas apresenta-se como uma atividade não trivial, porque tem a responsabilidade de identificar uma mesma entidade do mundo real que está representada de forma diferente na base de dados.

A identificação de tuplas duplicadas é considerada uma das atividades mais importantes na fase de limpeza de dados e também é o foco central deste trabalho.

Dentre as estratégias para tratar esse tipo de problema estão (ELMAGARMID; IPEIROTIS; VERYKIOS,2007).

- **Inferência bayesiana**: Fellegi e Sunter (1969) foram os responsáveis por formalizar e introduzir as notações que são utilizadas atualmente na literatura de identificação de tuplas duplicadas por meio de modelos probabilísticos, mais precisamente por meio de inferência bayesiana. De acordo com essas notações, cada par de tupla $<\alpha,\beta>$ é representado por um vetor randômico $\underline{x}=[x_1, \ldots, x_n]^T$ com *n* componentes que correspondem aos *n* campos comparáveis das duas tuplas. Cada x_i representa o nível de similaridade do *i-ésimo* campo das tuplas α e β. Muitas abordagens utilizam valores binários para preencher os valores dos $x_{i's}$ e atribuem $x_i=1$ quando os campos são similares e $x_i=0$ quando não são. O vetor $\underline{x}$ serve como entrada de um modelo de decisão que determina se o par de tuplas representado por $\underline{x}$ pertence à classe de pares similares (S) ou à classe de pares não similares (N).

  A principal suposição sobre esse tipo de modelo é que o vetor $\underline{x}$ é um vetor randômico cuja função de densidade é diferente para cada uma das classes S e N. Sendo assim, se a função de densidade para cada classe é conhecida, o problema de identificação de tuplas duplicadas torna-se um problema de inferência bayesiana.

- **Aprendizado supervisionado**: baseia-se em um conjunto de treinamento pré-existente formado por pares de tuplas já rotulados como similares ou não similares e trata cada par de tupla $<\alpha,\beta>$ de maneira independente. Baseando-se neste conjunto de treinamento, monta-se um modelo capaz de classificar as futuras tuplas que não estiverem atribuídas a nenhuma classe. Como exemplo dessa estratégia, pode-se citar (COCHINWALA et al., 2001) que utiliza o algoritmo CART (BREIMAN et al., 1984) para gerar árvores de regressão juntamente com a utilização de algoritmos de classificação e também (BILENKO et al., 2003) que propõe a utilização do *SVMlight* (JOACHIMS, 1999) para aprender como mesclar os resultados similares para os campos individuais das tuplas.

- **Aprendizado ativo**: um dos principais problemas relacionados às técnicas de aprendizado supervisionado é a necessidade de uma ampla quantidade de exemplos pré-rotulados no conjunto de treinamento. É uma tarefa fácil criar uma determinada quantidade de pares de treinamento sendo claramente similares ou

claramente não similares, porém a definição de pares ambíguos, aqueles que não se pode dizer muita coisa sobre o nível de similaridade entre eles, é uma tarefa bem complexa. Baseando-se nessa observação, alguns sistemas de identificação de tuplas duplicadas utilizam as chamadas “técnicas de aprendizado ativo” para identificar automaticamente esses casos ambíguos e repassá-los para serem classificados manualmente por seres humanos (COHN; LADNER; WAIBEL, 1994). Diferentemente dos classificadores comuns, que são treinados com base em um conjunto de treinamento estático, os classificadores “ativos” desempenham a atividade de seleção de pares de tuplas ainda não rotulados ativamente, de modo que, quando esses pares são rotulados manualmente fornecem uma maior precisão para o classificador. Como exemplo desse tipo de classificador pode-se citar o trabalho (SARAWAGI; BHAMIDIPATY, 2002) que apresenta uma ferramenta denominada ALIAS, cuja ideia principal consiste em identificar de forma automática os pares de tuplas que são claramente similares e não similares e utilizar a ajuda humana para classificar os pares de tuplas identificados automaticamente como ambíguos.

O ALIAS inicia-se com um pequeno conjunto de treinamento composto por pares de tuplas que são considerados claramente similares ou não similares e, com base nesse primeiro conjunto de treinamento, cria um classificador inicial que irá percorrer os pares de tuplas ainda não classificados com o intuito de classificá-los. Esse classificador inicial fará determinações muito claras em relação a alguns pares de tuplas, porém também fará determinações não tão claras assim em relação a outros pares. O objetivo principal dessa etapa é procurar por pares de tuplas ainda não rotulados, mas que, quando rotulados por um ser humano, possam contribuir de maneira significativa para a precisão do classificador.

De um modo geral, pode-se dizer que o ALIAS é capaz de aprender as peculiaridades de um conjunto de dados e, a partir daí, detectar rapidamente tuplas duplicadas com a utilização de apenas uma pequena quantidade de dados como conjunto de treinamento.

- **Técnicas *ad-hoc***: uma desvantagem considerável presente nas técnicas de aprendizado supervisionado e ativo é a necessidade da existência de um conjunto de treinamento prévio e também, no caso do aprendizado ativo, a necessidade de interação humana direta no processo. Em situações em que a

ausência desses recursos e habilidades se faz presente, a identificação de tuplas duplicadas por meio dessas técnicas se torna inviável. Em contrapartida, pode-se definir estratégias *ad-hoc* para realização da mesma tarefa, evitando, assim, a necessidade de criação de um conjunto de treinamento prévio. Ao utilizar essas técnicas, juntamente com um valor de *treshold*[1] apropriado, é possível identificar tuplas similares sem a necessidade de treinamento e interação humana durante o processo. As técnicas *ad-hoc* podem ser basicamente definidas como algoritmos que recebem duas *strings* α e β e, de alguma forma direta, decidem se podem ser consideradas similares ou não. Nas seções 2.5, 2.6 e 2.7 os algoritmos mais utilizados para esse propósito serão apresentados em detalhes.

Como abordagem mais simples desse tipo de estratégia, pode-se citar a utilização das tuplas (incluindo todos os campos considerados úteis na avaliação) como um grande campo de texto e, posteriormente, realizar a aplicação de uma das técnicas de similaridade para identificação de "campos" similares (MONGE; ELKAN, 1996). O problema dessa abordagem é que existe a possibilidade dela ignorar informações importantes para o processo de identificação de tuplas duplicadas e comprometer de forma significativa o resultado final.

Outra abordagem mais eficaz em relação ao tratamento de tuplas como grandes campos únicos envolve o cálculo de distância individual para cada campo correspondente, utilizando a técnica *ad-hoc* mais apropriada para cada caso, e, no final, realizar uma soma ponderada dessas distâncias para chegar a uma distância final mais precisa entre as tuplas (DEY; SARKAR; DE, 1998).

Existe também outra alternativa proposta por Guha et. al (2004), que consiste em criar uma métrica de distância baseada em combinação de *rankings*. A ideia principal por trás dessa estratégia está no fato de que, se for considerado apenas um campo das tuplas, é muito fácil identificar o grau de similaridade entre cada par e criar uma lista ordenada decrescente com essas informações. Ao aplicar essa ideia básica para todos os campos, obter-se-á *n* listas desse mesmo tipo, uma para cada campo. Uma vez que as *n* listas forem identificadas, a solução proposta realiza uma busca das *top-k* tuplas similares.

---

[1] O limitante que define o grupo ao qual um par de tuplas em análise pertence é denominado *treshold.*

Embora essa abordagem de identificação de tuplas duplicadas baseada nas técnicas *ad-hoc* apresente a vantagem de não precisar de nenhum conjunto de treinamento inicial e nem interação humana durante o processo, ela possui um problema relacionado aos *tresholds*. Visto que os *tresholds* desempenham um papel importante no processo, servindo como limites de definição para a classificação de pares de tuplas como similares ou não, faz-se necessário que esse valor seja definido por alguém e, se não for bem definido, pode ocasionar uma queda de qualidade do resultado final. Na presença de um conjunto de treinamento, consegue-se facilmente encontrar um valor considerado "ideal" para o *treshold*, porém essa técnica perderia sua qualidade principal que consiste na não necessidade de conhecimento prévio sobre os dados.

Em (CHAUDHURI; GANTI; MOTWANI, 2005) é proposta a ideia de um *framework* de identificação de tuplas duplicadas baseado em técnicas *ad-hoc* juntamente com a ideia de que os valores de *treshold* devem ser diferentes para cada par de tuplas presentes na base de dados, o que gera a necessidade do cálculo de valores de *treshold* diferentes para cada par de tuplas analisadas. Chaudhuri, Ganti e Motwani (2005) propõem a utilização de um algoritmo eficiente para realizar esse cálculo e mostram que os resultados obtidos com essa estratégia superam os resultados alcançados por estratégias que consideram apenas um valor de *treshold* global para todos os casos.

- **Agrupamento:** essa abordagem emprega algoritmos capazes de agrupar tuplas equivalentes em um mesmo grupo e tuplas não equivalentes em grupos diferentes. Dentre as técnicas mais recentes para a realização do agrupamento de tuplas, podem ser destacadas: *Accumulatively-Adaptative Sorted Neighborhood Method* (YAN et al., 2007), um *framework* desenvolvido por Shahri e Sharhri (2006), e o algoritmo CEBMDC (*Cluster Ensemble Based Mixed Data Clustering*) desenvolvido por He, Xu e Deng (2008).

2.3.2 Valores nulos

O problema de valores nulos está relacionado a várias razões, como: não conhecimento da informação, inexistência da informação, erro na aplicação responsável pela inserção de registros na base dados e etc. Algumas das técnicas utilizadas para lidar com o preenchimento desses valores nulos são (HAN; KAMBER, 2006):

- **Ignorar tupla**: pode ser utilizada quando, na fase de *data mining*, estiver presente alguma técnica de classificação. Dessa forma, as tuplas que não possuem a classe de rótulos para os atributos que serão utilizados na etapa de *data mining* são excluídas. Em outras situações, essa técnica não é muito desejável com exceção dos casos em que a tupla possui muitos atributos com valores ausentes.
- **Preenchimento manual**: consiste em preencher manualmente todos os atributos que possuem valores ausentes. É uma técnica que leva muito tempo e deve ser realizada apenas em casos em que a tabela possua poucos valores ausentes.
- **Constante global**: utiliza uma constante especificada pelo usuário para preencher todos os valores ausentes para um determinado atributo em uma tabela.
- **Média do atributo**: técnica utilizada para atributos do tipo numérico e leva em consideração todos os valores existentes para um atributo de uma tabela para calcular a média e preencher os valores ausentes.
- **Média por grupo**: também é utilizada para preenchimento de atributos numéricos e calcula a média baseando-se em algum grupo em que uma tupla pertence.
- **Valor com maior probabilidade**: técnica que se baseia em métodos como regressão, formalismo bayesiano e árvores de decisão para determinar qual o valor mais provável para preencher o valor ausente.

2.3.3 Valores fora do domínio

Para lidar com o tratamento de valores fora do domínio, além de consultas diretas na base de dados, utilizam-se principalmente métodos de agrupamento. Nesse caso, para cada tipo de atributo utiliza-se um método específico. Por exemplo, para atributos numéricos utilizam-se técnicas de agrupamento como *k-means*, e, para atributos categóricos, utilizam-se técnicas de agrupamento como *Squeezer* (HE; XU; DENG, 2002) e ROCK (GUHA; RASTOJI; SHIM, 1991).

2.4 Identificação de tuplas duplicadas não idênticas

O problema relacionado com a identificação de tuplas duplicadas não idênticas é, hoje, referenciado por vários nomes distintos em várias áreas diferentes, como: *record linkage*, *merge/purge problem*, *dedup*, duplicatas *fuzzy*, dentre outros. Neste trabalho, será adotado como padrão o termo "*record linkage*", que pode ser definido formalmente como "processo utilizado para comparar registros de duas ou mais fontes de dados em uma tentativa de determinar quais pares de registros representam a mesma entidade do mundo real" ou também como "processo de descoberta de registros duplicados em um único arquivo" (ELFEKY; VERYKIOS; ELMAGARMID, 2002).

O processo de *record linkage* é bem mais complexo que a atividade de identificação de tuplas duplicatas idênticas. No modelo relacional, por exemplo, a identificação de duplicatas idênticas é inteiramente baseada na expressão SQL *"select distinct ..."*. Duas tuplas só são consideradas idênticas se, e somente se, possuem dados exatamente iguais para todos os campos que são considerados na comparação, caso contrário, são consideradas distintas. Quando as tuplas estão duplicadas de forma não idêntica em uma base de dados, elas não compartilham total similaridade entre os valores de seus campos e acabam sendo rejeitadas pelo método de identificação de duplicatas idênticas. Sendo assim, de alguma forma, esses dados de um mesmo campo que referenciam uma mesma entidade no mundo real e que estão representados de forma diferente na base de dados precisam ser classificados como semelhantes. É aí que se encaixa o processo de *record linkage*.

Fellegi e Sunter (1969) foram os primeiros a introduzir os fundamentos matemáticos formais para a realização do processo de *record linkage*. O modelo proposto é caracterizado como um modelo probabilístico, pois é inteiramente baseado em teorias de probabilidade, mais precisamente em inferência bayesiana. Desde então, outras técnicas foram surgindo e ganhando espaço nessa área tão ampla, como: aprendizado supervisionado, aprendizado ativo, agrupamento (*clustering*) e técnicas *ad-hoc*, as quais servem de base para as anteriores e caracterizam-se por ser as mais utilizadas atualmente no processo de *record linkage*.

O processo de *record linkage* também pode ser visto como parte do problema de classificação, que é muito conhecido dentre as técnicas utilizadas na realização da atividade de *data mining*. Isso porque o problema de classificação possui o objetivo de atribuir corretamente os dados analisados para uma das classes pertencentes a um

conjunto finito de classes. Em analogia ao processo de *record linkage,* percebe-se que a ideia é a mesma, pois este processo classifica pares de tuplas não idênticas em duas possíveis classes: similares e não similares (ELFEKY; VERYKIOS; ELMAGARMID, 2002).

Como já visto anteriormente, há várias técnicas para realizar o tratamento do problema de *record linkage*, porém pode-se dividi-las basicamente em duas categorias principais: técnicas *ad-hoc*, que trabalham de forma eficiente em bases de dados relacionais, e também técnicas baseadas em modelos de inferência probabilística. Enquanto as técnicas probabilísticas são melhores que as técnicas *ad-hoc* em termos de precisão, as técnicas *ad-hoc* trabalham de forma mais eficiente em relação ao tempo de resposta e escalabilidade que as técnicas probabilísticas. Atualmente, as técnicas probabilísticas são viáveis somente para bases de dados pequenas, o que se opõe à tendência das bases de dados atuais, que estão cada vez maiores. Sendo assim, as técnicas *ad-hoc* são mais utilizadas atualmente que as técnicas probabilísticas e, por consequência, recebem mais atenção acadêmica (ELMAGARMID; IPEIROTIS; VERYKIOS, 2007).

Uma das questões mais relevantes relacionadas às técnicas de *record linkage* versa sobre qual seria a melhor técnica para identificação de tuplas duplicadas não idênticas. Porém, não há nenhuma reposta muito clara para tal questionamento. A detecção de tuplas duplicadas não idênticas é uma tarefa muito dependente dos dados e, dessa forma, não há como definir uma técnica que sempre será demarcada como a melhor para todas as situações (ELMAGARMID; IPEIROTIS; VERYKIOS,2007).

Como é sabido, este trabalho atua na maximização da precisão das técnicas *ad-hoc* para bases de dados relacionais definidas em qualquer idioma. Nas seções a seguir, serão apresentadas as técnicas *ad-hoc* mais utilizadas e consolidadas no processo de *record linkage*.

2.5 Técnicas *ad-hoc* baseadas em caracteres

As técnicas *ad-hoc* baseadas em caracteres são geralmente utilizadas para lidar com erros léxicos, pois são bem eficientes nesse tipo de problema. Nesta seção, as técnicas que serão apresentadas em tal categoria são: *Edit Distance*, *Affine Gap Distance*, *Smith-Waterman Distance*, *Jaro Distance Metric* e *Q-gram Distance*.

2.5.1 *Edit Distance*

O algoritmo *Edit Distance* (LEVENSHTEIN, 1966) possui sua essência na programação dinâmica e tem como objetivo calcular a menor distância entre duas *strings* α e β. Tal distância é mensurada pela quantidade de operações de edição de caracteres necessárias para transformar a *string* α em β ou vice-versa. As operações de edição a serem consideras são:

- **inserção:** pode-se adicionar um caractere em uma *string* para transformá-la em outra: DISSERTÇÃO → DISSERT**A**ÇÃO;
- **remoção:** pode-se remover um caractere de uma *string* para transformá-la em outra: DISSER**R**TAÇÃO → DISSERTAÇÃO;
- **substituição:** pode-se alterar um caractere de uma *string* para transformá-la em outra: DISSERTA**S**ÃO → DISSERTA**Ç**ÃO.

A versão mais simples do algoritmo, também conhecida como distância de *Levenshtein*, lida com todas as operações apresentadas anteriormente de forma igualitária. Isso significa que ela atribui para cada uma dessas operações o mesmo peso: um (1). Essa versão possui complexidade de tempo $O(|\alpha|.|\beta|)$, sendo |α| e |β| o comprimento das *strings* α e β respectivamente. Sendo assim, quando |α| = |β| = n, pode-se dizer que o algoritmo possui complexidade $O(n^2)$ (ELMAGARMID; IPEIROTIS; VERYKIOS, 2007). Em (NEEDLEMAN; WUNSCH, 1970) é proposta uma variação interessante do algoritmo *Edit Distance*, de modo que os pesos atribuídos às operações não são sempre os mesmos. Essa ideia permite valorizar erros mais frequentes e desvalorizar erros mais raros, resultando em uma maior precisão no resultado final.

De um modo geral, o algoritmo *Edit Distance* funciona bem quando se trata da resolução de problemas relacionados a identificação de erros léxicos. Em contrapartida, não se faz tão efetivo quando se trata de outros tipos de problemas como: truncamento de *strings*, abreviações e mudança na ordenação de palavras.

2.5.2 *Affine Gap Distance*

Ao perceber o benefício da utilização do algoritmo *Edit Distance* para o tratamento de erros léxicos e, também, suas dificuldades relacionadas ao truncamento de *strings* e abreviações, Waterman, Smith e A.Beyer (1976) propõem uma estratégia

derivada do algoritmo *Edit Distance* padrão denominada *Affine Gap Distance*. Essa estratégia propõe incrementar a ideia original do algoritmo, que consiste em três operações básicas, com mais duas operações extras para realizar o tratamento dos casos de abreviações e truncamentos, que são denominadas: *open gap* e *extend gap*.

O custo de estender um *gap* (ausência de caracteres) é geralmente menor que o custo para abrir um *gap*, e estes dois custos são considerados menos importantes que os custos das operações apresentadas pelo *Edit Distance* padrão. O algoritmo *Affine Gap Distance* possui complexidade de tempo $O(a.|\alpha|.|\beta|)$ quando o tamanho máximo do *gap* $a << \min\{|\alpha|.|\beta|\}$. Sendo assim, de forma genérica, o algoritmo é executado com complexidade $O(a^2.|\alpha|.|\beta|)$.

2.5.3 *Smith-Waterman Distance*

Em (SMITH; WATERMAN, 1981) analisa-se as ideias por trás do algoritmo *Edit Distance* e da extensão *Affine Gap Distance*, ambos apresentados anteriormente, e levanta-se uma nova questão relacionada a essa classe de problemas de erros léxicos. Smith e Waterman (1981) apresentam a teoria de que as inconsistências encontradas no início e no final das *strings* não deveriam ter o mesmo peso que as inconsistências encontradas no meio das *strings*, e sim, pesos menores. Seguindo essa linha de raciocínio, propõem uma nova extensão denominada *Smith-Waterman Distance*, cujo objetivo principal é, de certa forma, desvalorizar a importância dos prefixos e sufixos das *strings* e valorizar os seus núcleos para determinar similaridade entre *strings* distintas.

Como exemplo de aplicação dessa estratégia, pode-se identificar com maior precisão – valor de similaridade – que as *strings* "Prof. João da Silva, UNESP" e "João da Silva, Prof." são similares, em virtude da desvalorização dos prefixos e sufixos e valorização dos núcleos de ambas as *strings*, que seria uma *string* muito próxima de "João da Silva".

O algoritmo *Smith-Waterman Distance* possui complexidade de tempo e espaço $O(|\alpha|.|\beta|)$, tal que $|\alpha|$ e $|\beta|$ são os comprimentos das *strings* α e β envolvidas na comparação.

2.5.4 Jaro *Distance Metric*

Em (JARO, 1976), é proposto um algoritmo de comparação de *strings* denominado *Jaro Edit Distance.* Essa estratégia é um pouco mais especializada que as outras, pois tem como objetivo principal a identificação de similaridade entre campos de nomes.

A ideia principal do algoritmo para calcular a distância entre duas *strings* α e β inclui os seguintes passos:

1. Calcular o comprimento das *strings* α e β que são representados respectivamente por |α| e |β|;
2. Encontrar caracteres comuns nas duas *strings*. Neste contexto, caracteres comuns são definidos da seguinte forma: α[*i*] e β[*j*], tal que α[*i*] = β[*j*] e $|i - j| \leq \frac{1}{2} \min\{|\alpha|, |\beta|\}$;
3. Encontrar a quantidade de transposições *t*, que é calculada da seguinte maneira: compara-se o *i-ésimo* caractere da *string* α com o *i-ésimo* caractere da *string* β e sempre que os caracteres forem diferentes, contabiliza-se uma transposição.

Após a execução dos três passos apresentados anteriormente, calcula-se a distância que é definida pela seguinte fórmula:

$$Jaro(\alpha, \beta) = \frac{1}{3}\left(\frac{c}{|\alpha|} + \frac{c}{|\beta|} + \frac{\frac{c-t}{2}}{c}\right)$$

De acordo com a descrição anterior do algoritmo, conclui-se que sua complexidade de tempo pode ser definida como $O(|\alpha|.|\beta|)$.

Diante do fato dessa estratégia ser especificamente proposta para identificação de campos de nomes, contrariando a ideia apresentada em (SMITH; WATERMAN, 1981), Winkler e Thibaudeau (1991) apresentaram uma modificação na estratégia *Jaro Distance Metric* a qual propõe maior valorização dos prefixos, pois acreditam que as similaridades encontradas no prefixo dos campos de nomes são mais significativas que as outras similaridades, que geralmente fazem referência a um nome de meio ou sobrenome.

2.5.5 *Q-grams*

Os *q-grams* são *substrings* de tamanho *q* retiradas das *strings* em análise pelas técnicas *ad-hoc*. A ideia por trás dos *q-grams* é que, quando duas *strings* α e β são similares, elas possuem um alto número de *q-grams* em comum (ULLMANN, 1977) (UKKONEN, 1992). Dada uma *string* α, os seus *q-grams* são obtidos por meio de uma janela deslizante de tamanho fixo *q* sobre os caracteres de α. Considerando-se que os *q-grams* do início e do fim podem possuir uma quantidade de caracteres menor, esses *q-grams* são preenchidos por algum símbolo especial fora do alfabeto.

Para exemplificar a extração de *q-grams* descrita anteriormente, considere a palavra "dissertação" e *q*=3. Os *3-grams* para essa palavra seriam: "$$d", "$di", "dis", "iss", "sse", "ser", "ert", "rta", "taç", "açã", "ção", "ão$" e "o$$". O que essa técnica propõe é simples e basicamente consiste na identificação dos *q-grams* das duas *strings* que estão sendo comparadas e, depois, faz uma avaliação da quantidade de *q-grams* que se sobrepõem. Pode-se se definir um *treshold* e, com base nesse limite, definir se as duas *strings* são similares ou não. A complexidade de tempo dessa estratégia é dada por $O(max\{|\alpha|,|\beta|\})$, tal que |α| e |β| representam respectivamente os comprimentos das duas *strings* α e β envolvidas no processo de identificação de similaridade.

Uma variação dessa estratégia de *q-grams* é proposta em (GRAVANO et al., 2001), que apresenta o conceito de utilizar *q-grams* juntamente com a posição em que se encontram para que, de forma eficiente, possa localizar *strings* similares dentro de uma base de dados relacional. Essa ideia é muito interessante, pois utiliza todos os recursos de otimização que um Sistema Gerenciador de Banco de Dados (SGBD) pode oferecer, diminuindo de forma considerável o tempo utilizado para identificação de duplicatas não idênticas.

2.6 Técnicas *ad-hoc* baseadas em *tokens*

As métricas de similaridade apresentadas anteriormente são consideradas eficientes quando se trata de erros léxicos. Porém não são tão eficientes quando os problemas estão relacionados ao posicionamento das palavras, como "John Smith" e "Smith, John". Para esse tipo de problema, as estratégias baseadas em caracteres

geralmente falham ao identificar as similaridades entre *strings*, retornando um valor alto para marcar a similaridade entre as mesmas.

Em contrapartida, as técnicas baseadas em *tokens* tentam compensar esse problema de ordenação das *strings* e são, geralmente, utilizadas para esse tipo específico de tarefa. A seguir, serão apresentadas algumas técnicas baseadas em *tokens* como: *Atomic Strings*, WHIRL e *Q-Grams* com *tf.idf*.

2.6.1 *Atomic Strings*

Monge e Elkan (1996) propõem um algoritmo que identifica similaridade entre campos de texto com base em *strings* atômicas (*tokens*), que são uma sequência de caracteres alfanuméricos delimitados por caracteres de pontuação ou espaços em branco. Duas *strings* atômicas são consideradas similares se, e somente se, forem exatamente iguais ou se uma for um prefixo da outra. Sendo assim, a similaridade calculada pelo algoritmo proposto é dada pelo número de *strings* atômicas similares dividido pela média do número de *strings* atômicas.

Para classificar se dois campos de texto são similares ou não, pode-se definir um *treshold*. Dessa forma, caso a similaridade resultante do algoritmo seja maior que esse *treshold*, os campos podem ser classificados como similares. Caso contrário, são classificados como não similares.

2.6.2 WHIRL

Cohen (1998) apresenta um sistema denominado WHIRL que busca, na área de recuperação de informação, a técnica de similaridade *Cosine* juntamente com o esquema de pesos *tf.idf* (YATES; RIBEIRO NETO, 1999) para calcular a similaridade entre campos de texto. Cohen separa cada *string* α em palavras *w* e atribui a cada uma dessas palavras um peso:

$$V_{\alpha}(w) = \log(tf_w + 1) * \log(idf_w)$$

Em que o termo tf_w corresponde ao número de vezes que a palavra *w* aparece no campo de texto e o termo idf_w é calculado por $\frac{|D|}{n_w}$, tal que |D| representa a quantidade de

registros na base de dados D e n_w o número de registros na base de dados D que contém a palavra *w*.

O valor *tf.idf* de uma palavra *w* é alto se essa palavra *w* aparece muitas vezes dentro de um campo de texto específico (tf_w) e *w* é suficientemente rara dentro da base de dados (idf_w). Isso significa que a *string w* pode ser considerada como um termo importante na caracterização do campo de texto, pois é um termo raro dentro da base de dados e forte dentro do campo de texto.

A similaridade *Cosine* entre duas *strings* α e β pode ser calculada como segue:

$$\text{Sim}(\alpha,\beta) = \frac{\sum_{j=1}^{|D|} v_\alpha(j).v_\beta}{||v_\alpha||^2.||v_\beta||^2}$$

A estratégia de identificação abordada por essa técnica apresenta resultados satisfatórios para uma grande variedade de entradas e lida de forma eficaz com os problemas relacionados ao posicionamento das palavras. Entretanto essa estratégia não consegue lidar com os erros léxicos apresentados na seção 2.5 pelo fato de identificar os *tokens* como um todo, sem realizar nenhuma verificação prévia de ortografia. Pensando nisso, (BILENKO et al., 2003) propuseram uma estratégia denominada *SoftTF-IDF* para resolver esse problema. Nessa estratégia, os *tokens* não são extraídos de forma bruta. Utilizam-se técnicas de similaridade baseadas em caracteres para identificação de *tokens* similares ao invés de considerar apenas *tokens* idênticos.

2.6.3 *Q-grams* com *tf.idf*

Gravano et al. (2003), estenderam o sistema WHIRL com o intuito de manipular erros léxicos por meio de *q-grams* ao invés de palavras inteiras, como *tokens*. Percebeu-se que, com essa configuração, um erro léxico afetaria bem menos o conjunto de *q-grams* comuns das duas *strings* que o conjunto de palavras comuns. Sendo assim, duas *strings* como “Projeto Mestrado” e “Mestrado Projeto” teriam um alto grau de similaridade com o uso dessa técnica.

Além de lidar com os erros léxicos, essa estratégia também lida de forma eficaz com a inserção e remoção de palavras. Por exemplo, as *strings* “Projeto Mestrado” e “Mestrado Projeto Unesp” teriam um alto grau de similaridade pelo fato de que os *q-grams* da palavra “Unesp” aparecem pouco nos conjuntos de *q-grams* das duas *strings*,

ocasionando em um baixo grau de sobreposição e consequentemente em um baixo valor *tf.idf*.

2.7 Técnicas *ad-hoc* baseadas em fonética

2.7.1 *Soundex*

Esse algoritmo, desenvolvido por Newcombe (1967), tem como objetivo encontrar o grau de similaridade entre *strings* por meio dos sons fonéticos avaliados pelas consoantes das mesmas. Dessa forma, o algoritmo é capaz de dizer se a pronúncia de duas *strings* é parecida apenas com a utilização de suas consoantes. A proposta do algoritmo *Soundex* é analisar uma *string* de entrada e produzir como saída um código de tamanho fixo de quatro caracteres, denominado código *soundex*, que pode ser utilizado para identificar similaridade fonética entre *strings* distintas. As etapas para o cálculo do código *soundex* são descritas a seguir (ELMAGARMID; IPEIROTIS; VERYKIOS, 2007).

1. Mantenha a primeira letra da *string* como prefixo e ignore todas as ocorrências das letras H ou W.
2. Assuma os seguintes códigos para as letras restantes:
    1. B, F, P, V → 1;
    2. C, G, J, K, Q, S, X, Z → 2;
    3. D, T → 3;
    4. L→ 4;
    5. M, N → 5;
    6. R→6.
3. A, E, I, O, U e Y não são codificados, mas podem ser utilizados como separadores.
4. Elimine códigos duplicados.
5. Elimine os separadores (vogais + Y).
6. Mantenha a letra prefixo (definida no passo 1), e os três primeiros códigos seguintes formando um código *soundex* com quatro posições. Caso existam menos que três dígitos após a letra prefixo, preencha os dígitos ausentes com 0's

para conseguir o código *soundex* de tamanho quatro e com o seguinte formato: <letra><dígito><dígito><dígito>.

Diante das etapas desse procedimento, conclui-se que a complexidade de tempo do algoritmo pode ser definida como *O(n)*, tal que *n* é o comprimento da *string* de entrada. É interessante ressaltar que esse algoritmo não apresenta bons resultados para *strings* grafadas em idiomas orientais, pois nessas línguas as vogais são grandes diferenciadoras e, como foi visto, o *soundex* ignora completamente as vogais (NEWCOMBE, 1967).

2.7.2 Sistema de identificação do estado de Nova Iorque

O sistema *New York State Identification and Intelligence System* (NYSIIS) foi proposto por Taft (1970) com o intuito de ajudar na rotina de trabalho da Divisão de Justiça Criminal do Estado de Nova Iorque. A ideia por trás dessa técnica baseia-se na extensão do algoritmo *Soundex* apresentado anteriormente, mais especificamente, consiste em duas modificações consideráveis na versão original:

1. Retém informação sobre a posição das vogais no código *soundex*, convertendo grande parte das mesmas pela letra A.
2. Não utiliza números para substituir as letras, mas converte uma consoante em outra com uma fonética similar. Pelo fato de não utilizar números na etapa de conversão, o código retornado pelo algoritmo não possui nenhum número, é puramente alfabético.

Geralmente, o código gerado pelo NYSIIS utiliza, no máximo, nove letras da *string* analisada e limita-se a seis caracteres de comprimento. De acordo com o trabalho de Taft (1970), o sistema NYSIIS tem uma maior eficácia na busca de resultados em comparação com *Soundex*, girando em torno de 3%.

As regras utilizadas pelo algoritmo são as seguintes (GÁLVEZ, 2006):

1. O primeiro caractere da palavra corresponde ao primeiro caractere do código;
2. Transforma os primeiros caracteres da palavra:
    - MAC = MCC;
    - PH = FF;
    - KN = NN;

- K = C;
- SCH = SSS.

3. Transforma os últimos caracteres da palavra:
    - EE = Y;
    - IE = Y;
    - DT, RT, RD, NT, ND = D;
    - Se o ultimo caractere é S, eliminá-lo;
    - Se o último caractere é A, eliminá-lo;
    - Se os últimos caracteres são AY, substituir por Y.

2.7.3 *Oxford Name Compression Algorithm* (ONCA)

O algoritmo ONCA (GILL, 1997) é uma técnica baseada em dois estágios a qual foi desenvolvida com o intuito de melhorar algumas características não satisfatórias da versão original do algoritmo *Soundex*. Tal estratégia caracteriza-se como uma simples combinação do algoritmo *Soundex* com o algoritmo NYSIIS.

No primeiro estágio, recebe-se uma *string* de entrada e gera-se um código igual ao código gerado pelo NYSIIS, com a exceção de que se utiliza a versão do inglês britânico. No segundo estágio, recebe-se como entrada o código do estágio anterior e aplica-se a versão original do algoritmo *Soundex*, mantendo-se assim um código final cujo tamanho é sempre quatro caracteres.

Assim, como nas propostas anteriores, essa técnica é utilizada com sucesso na identificação de nomes similares.

2.7.4 *Metaphone* e *Double Metaphone*

O algoritmo *Metaphone* foi desenvolvido por Philips (1990) como uma alternativa mais interessante em relação ao algoritmo *Soundex*. Sugere-se a utilização de 16 sons de consoantes que podem descrever uma grande variedade de sons, sejam eles pertencentes à língua inglesa ou não. Esse algoritmo remove todas as vogais das palavras, salvo o caso em que a vogal é a primeira letra, e mantém todas as consoantes (GÁLVEZ, 2006).

Posteriormente à proposta do *Metaphone*, Philips (2000) propõe uma versão melhorada da mesma, denominada *Double Metaphone*. Essa nova versão diferencia-se

da primeira pelo fato de conseguir aprimorar algumas das escolhas de codificação e permitir múltiplas codificações para palavras que possuem várias pronúncias possíveis.

As substituições previstas pelo algoritmo *Metaphone* funcionam da seguinte forma:

- B → B, exceto em final de uma palavra;
- C → X, se aparecer como “cia”, “ch”;
  S, se aparecer como “ci”, “ce”, “cy”;
  K para o resto dos casos;
- D → J se aparecer como “dge”, “dgy”, “dgi”;
  T para o resto dos casos;
- F → F;
- G → silêncio, se aparecer como “gh”;
  J se estiver à frente de “i”, “e”, “y”;
  K para o resto dos casos;
- H → silêncio, se aparecer depois de vogal e não seguida por vogal;
  H para o resto dos casos;
- J → J;
- K → silêncio se aparecer depois de “c”;
  K para o resto dos casos;
- L → L;
- M → M;
- N → N;
- P → F se aparecer antes de “h”;
  P para o resto dos casos;
- Q → K;
- R → R;
- S → X se aparecer antes de -h- ou como “sio”, “sai’;
  S para o resto dos casos;
- T → X se aparecer como “tia”, “tio”;
  O se aparecer antes de “h”;
  T para o resto dos casos;
- V → F;
- W → silêncio, se não está seguida por vogal;

W se está seguida por vogal;

- X → KS;
- Y → silêncio, se não está seguida por vogal
  Y se está seguida por vogal;
- Z → S.

2.8 Ferramentas para identificação de tuplas duplicadas

Nos últimos anos, uma grande quantidade de ferramentas estão sendo propostas para atuar na fase de *data cleansing*. Dentre elas, existem desde ferramentas mais amplas, que abordam uma ampla variedade de problemas, até ferramentas inteiramente especializadas em tratar informações relacionadas a nomes, endereços, e-mail e etc. Nesta seção, serão apresentadas algumas das ferramentas mais relevantes na realização da tarefa de identificação de tuplas duplicadas.

O Febrl (*Freely Extensible Biomedical Record Linkage*) é uma ferramenta de *data cleaning open-source* desenvolvida em *Python* composta basicamente por dois componentes principais: um para lidar com a padronização de dados e outro para tratar de identificação de tuplas duplicadas. O componente referente à padronização baseia-se em Modelos de Markov Ocultos (HMMs – *Hidden Markov Models*) para analisar as entradas de uma base de dados. A utilização de HMMs é uma indicação da necessidade de conjuntos de treinamentos para realização dessa etapa com sucesso. O segundo componente, referente à identificação de tuplas duplicadas, aborda uma variedade de técnicas de identificação de similaridade entre *strings* (como *Jaro Distance*, *Edit Distance* e *Q-gram Distance*) e também algumas técnicas de codificação fonética (como *Soundex*, NYSIIS e *Double Metaphone*). Uma característica interessante do Febrl é a utilização das codificações fonéticas ao contrário. Isso porque, as codificações fonéticas consideradas são sensíveis à primeira letra, ou seja, sempre a mantém intocável, o que limita que haja a identificação de erros relacionados às mesmas. Quando se faz o uso da estratégia fonética ao contrário, aumenta-se consideravelmente a precisão da técnica (CHRISTEN; CHURCHES; HEGLAND, 2004).

Em (ELFEKY; VERYKIOS; ELMAGARMID, 2002) é proposta uma ferramenta de *record linkage* interativa denominada TAILOR (***R**ec**o**rd **Li**nk**a**ge **T**oolbox*). Ela recebeu esse nome porque segundo os autores, a ferramenta pode ser

adaptada (*tailored* – em inglês) para suportar qualquer modelo de *record linkage* futuro. Em função disso, sua estrutura foi definida em camadas, separando as funções de similaridade da lógica de detecção de tuplas similares, o que torna o sistema extensível em relação aos outros sistemas monolíticos. Além de implementar o estado da arte relacionado às técnicas *ad-hoc*, TAILOR propõe uma abordagem nova baseada em aprendizado de máquina, mais precisamente três técnicas: modelo de indução, modelo de agrupamento e modelo híbrido. Por fim, outra característica interessante relacionada à ferramenta TAILOR refere-se à emissão de relatórios estatísticos com o intuito de informar o usuário e fazer com que ele tenha um melhor entendimento da qualidade que as técnicas aplicadas proporcionaram a sua base de dados.

O WHIRL (COHEN, 1998) é um sistema de identificação de duplicatas gratuito para fins de pesquisa. Nele, utilizam-se estratégias oriundas da área de recuperação de informação, mais especificamente o algoritmo de similaridade *Cosine* combinado com o esquema de pesos *tf.idf*, apresentado em detalhes na seção 2.6, para identificar similaridade entre dois campos. Este sistema mostrou-se mais rápido que todos os outros métodos de inferência probabilística.

O *WinPure Clean & Match* (PIATETSKY-SHAPIRO, 2011) é uma ferramenta proprietária que possui a função de limpar listas de dados e também realizar a identificação de tuplas duplicadas. Para realizar tal tarefa, a ferramenta é composta por cinco módulos sendo um deles responsável pelo processo de *record linkage*. O módulo responsável pela eliminação de tuplas duplicadas é denominado "*Dedup/Match Module*" e funciona basicamente em três etapas: a primeira etapa envolve a identificação das tabelas e colunas desejáveis para participar do processo de identificação de tuplas duplicadas; a segunda etapa serve para identificar quais técnicas devem ser utilizadas no processo, que podem ser técnicas mais básicas (para identificação de e-mails, números de telefone e etc.) ou mais avançadas envolvendo ou não a identificação de duplicatas *fuzzy*[2]; a terceira etapa envolve a escolha de como a visualização dos dados duplicados deve ser apresentada ao usuário.

O *BigMatch* (YANCEY, 2002) também é um sistema especializado na identificação de tuplas duplicadas utilizado pelo US *Census Bureau*, que baseia-se em

[2] Duplicatas *Fuzzy* pode ser outra denominação para tuplas duplicadas não idênticas.

estratégias de *blocking*[3] para identificar possíveis tuplas similares entre duas tabelas. Sua única restrição é que pelo menos uma dessas duas tabelas caiba inteiramente na memória principal. O objetivo da ferramenta não é realizar a tarefa de detecção de tuplas duplicadas de maneira sofisticada, mas funcionar como uma peneira para filtrar possíveis candidatos que posteriormente serão analisados por técnicas mais elaboradas.

2.9 Considerações finais

Diante do estado da arte apresentado, pode-se notar a característica genérica das técnicas de identificação de tuplas duplicadas e ter uma ideia de como os ambientes propostos utilizam-nas para concretização dessa tarefa. A não consideração das características dos idiomas em que as bases de dados estão definidas faz com que estas estratégias não alcancem o máximo de seu potencial, pois os problemas de linguagem são reais e a sua ocorrência nos volumes de dados impacta diretamente no distanciamento ortográfico dos registros.

[3] Função utilizada para subdividir um arquivo em um conjunto de subconjuntos de dados mutuamente exclusivos (blocos), de modo que os dados pertencentes a um determinado bloco não estarão presentes em outro bloco diferente.

Capítulo 3
Ambiente de Enriquecimento de Dados para Identificação de Tuplas Duplicadas

3.1 Considerações iniciais

Neste capítulo, é apresentada uma descrição detalhada da implementação da pré-etapa de enriquecimento proposta, cujo objetivo principal é realizar a aproximação ortográfica entre os registros de uma base de dados. O ambiente é independente de idiomas e está estruturado para receber bases de dados em qualquer idioma disponível, concentrando-se em enriquecê-las no intuito de obter resultados melhores e mais consistentes no processo de identificação de tuplas duplicadas, que precede a etapa de *data mining* no processo de descoberta de dados em bases de dados.

3.2 Descrição do problema

Como pôde ser visto no capítulo anterior, existem várias técnicas que podem ser utilizadas para identificação de tuplas duplicadas em banco de dados. De um modo geral, elas podem ser divididas em duas grandes categorias principais: probabilísticas e *ad-hoc*. As técnicas denominadas *ad-hoc* ganham, atualmente, mais espaço no mercado e também no meio acadêmico em relação às técnicas probabilísticas já que são menos complexas, não utilizam um conjunto de dados de treinamento e possuem a vantagem de lidar de forma eficiente com volumes de dados maiores. Entretanto, as técnicas *ad-hoc* geralmente são propostas de forma genérica, ou seja, de tal modo que possam ser

aplicadas em qualquer base de dados ao redor do globo independentemente do idioma em que tal base esteja definida. Este fato, embora não seja relevante para muitos, implica na desconsideração de todas as características e problemas específicos que cada idioma carrega consigo.

Seguindo essa linha, os ambientes propostos para cumprir a tarefa de identificação de tuplas duplicadas, em sua maioria, baseiam-se nas técnicas *ad-hoc* para alcançar seu objetivo e, por conseguinte, tratam o problema da mesma forma genérica com que as mesmas são propostas. Apesar de a melhoria dessas técnicas ser cada vez mais evidente qualitativa e quantitativamente, os problemas relacionados a cada idioma específico ainda são esquecidos. Isso ocorre ou por não serem considerados relevantes ou, ainda, pelo fato de acrescentarem uma quantidade extra de processamento dos dados no processo de identificação de tuplas duplicadas.

Levando em consideração o idioma "Português", são apresentados a seguir alguns problemas comuns, facilmente encontrados em qualquer volume de dados.

- **Erros ortográficos:** trata-se da grafia distinta de uma palavra, considerada errônea, em relação à norma padrão da língua. Por exemplo, é muito comum aos falantes da língua portuguesa a confusão sobre a grafia de algumas palavras como: enxada e enchada, berinjela e beringela, exceção e excessão, entre outras.
- **Utilização de termos sinônimos:** todos os idiomas possuem palavras sinônimas, ou seja, palavras diferentes que possuem um mesmo significado. Levando em consideração que pessoas diferentes pensam de forma diferente e possuem aparatos lexicais diferenciados, é comum identificar divergências deste tipo em volumes de dados criados por pessoas distintas. Alguns exemplos de palavras sinônimas: ajudante e auxiliar, carro e automóvel, piso e azulejo, etc.
- **Inversão de nomes:** trata-se de um deslocamento ou troca dos nomes para representar um mesmo tipo de informação. Alguns exemplos possíveis em volumes de dados reais: Supermercado Laranjão ↔ Laranjão Supermercado, José Silva ↔ Silva José, entre outros.
- **Utilização de abreviações:** toda língua é passível de abreviações, mediante seu uso oral e escrito. Atualmente, com a inclusão digital e a explosão de aplicativos de comunicação via Internet, o uso de abreviações torna-se cada vez mais comum e usual, até mesmo quando se trata de criar volumes de dados

importantes nas organizações. Alguns exemplos: você → vc; limitada → ltda; doutor → dr.; professor → prof.; mesmo → msm; entre outras.

Considerando os problemas supracitados e sua ocorrência em um único volume de dados, apresenta-se, como consequência principal, o distanciamento ortográfico entre as sentenças que compõem os registros desse volume. Tal distanciamento tem impacto direto no processo de identificação de tuplas duplicadas, pois a maioria das técnicas *ad-hoc* são baseadas em algum tipo de similaridade (léxica ou fonética) que realiza comparações letra a letra ou entre os sons das sílabas.

Diante dos fatos apresentados, sentiu-se a necessidade de propor uma pré-etapa em relação ao processo de identificação de tuplas duplicadas em que o fator idioma receba a maior parte das atenções. Tem-se a ideia de construir um ambiente capaz de lidar com qualquer idioma existente e, por conseguinte, capaz de enriquecer as sentenças de qualquer base de dados ao redor do globo terrestre, em busca da melhoria dos resultados do processo de identificação de tuplas duplicadas. Além disso, antecipando-se a carga extra de processamento que essa pré-etapa acrescenta aos padrões atuais de identificação de tuplas duplicadas, e pensando no caráter sequencial de execução das tarefas de enriquecimento, propõe-se a utilização de processamento paralelo, juntamente com programação *multithreading*, para codificação desta pré-etapa de uma forma mais otimizada. Como resultado, tem-se a maximização do rendimento dos vários núcleos de processamento que compõem a organização dos processadores atuais.

3.3 O Ambiente de enriquecimento

O ambiente proposto neste trabalho consiste basicamente em dois grandes módulos: o módulo enriquecedor e o módulo de identificação de duplicatas. Como pode ser visto no diagrama de blocos ilustrado na figura 2, o primeiro tem a função de enriquecer os dados de uma base de dados de entrada e o segundo a função de identificar tuplas duplicadas em uma base de dados já enriquecida. Juntos, formam o núcleo do ambiente proposto e assumem a responsabilidade de realizar o processo de identificação de tuplas duplicadas de um banco de dados de modo que seu idioma seja favorecido e os resultados apresentados sejam mais eficientes e íntegros.

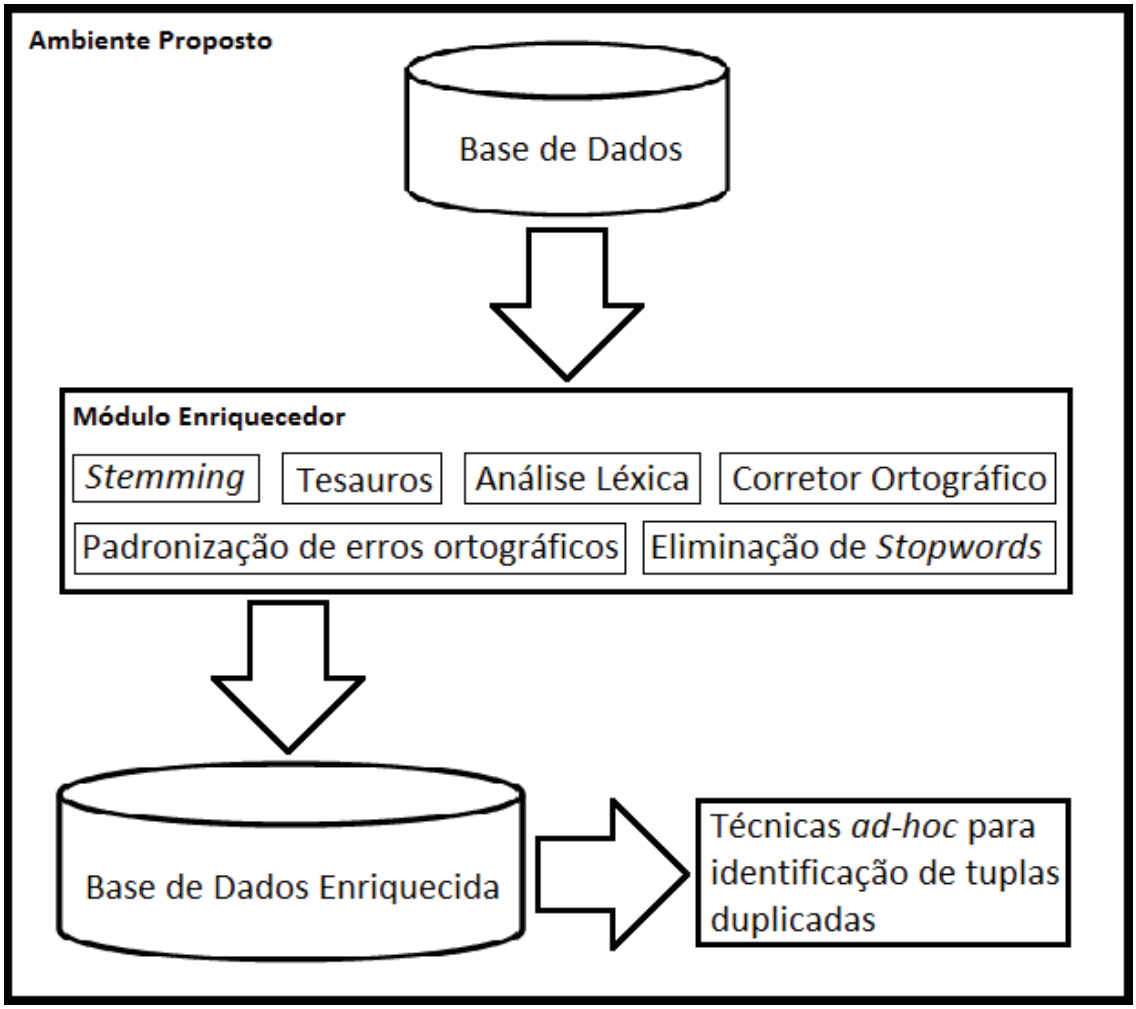

Figura 2 – Ambiente proposto contendo a pré-etapa de enriquecimento.

O fluxo exibido pela figura 2 é simples. Primeiramente, a base de dados que será enriquecida deverá passar pelo "Módulo Enriquecedor", no qual estão localizados os submódulos, que contém as técnicas responsáveis pelo enriquecimento dos dados. Uma vez que a base de dados sai desse módulo, ela está pronta – enriquecida – para que as técnicas de identificação de duplicatas possam ser aplicadas sobre a mesma. É muito importante ressaltar que a tabela que participa do processo de enriquecimento não terá os seus registros modificados. O procedimento é descrito passo a passo a seguir.

1. Adiciona-se uma coluna numérica, do tipo inteiro, chamada "ident_enri" na tabela original e preenche-se essa coluna com valores inteiros de 1 até *n*, tal que *n* representa a quantidade total de registros na tabela. Essa coluna tem a função de suprir a possível ausência de definição de chaves primárias nas tabelas.
2. Cria-se uma cópia da tabela original cujo nome é modificado pela inclusão do sufixo "_E$". O sufixo indica que a nova tabela é a versão enriquecida da versão

anterior sem o sufixo. Supondo que a tabela “funcionários” irá participar do processo de enriquecimento, após executar o passo dois (2), existirá uma tabela enriquecida denominada “funcionários_E$”, e a tabela original “funcionários” continuará com seus dados intactos.

3. Aplicam-se as técnicas de identificação de tuplas duplicadas na tabela enriquecida. No caso da tabela “funcionários”, apresentada como exemplo no passo anterior, a, agora, enriquecida é denominada “funcionários_E$”.

O ambiente proposto pode ser considerado extensível, pois propõe a criação de módulos para que sempre seja possível a inclusão de novas estratégias de enriquecimento dos dados e também a inclusão de novas técnicas de identificação de tuplas duplicadas. Dessa forma, pode-se manter o ambiente sempre alinhado com o estado da arte e também permitir que diferentes grupos de pessoas possam alterá-lo.

As seções a seguir descreverão, em detalhes, cada um dos dois módulos principais ilustrados pela figura 2.

3.4 Módulo de identificação de duplicatas

O módulo de identificação de duplicatas possui o encargo de abrigar o estado da arte relacionado às técnicas de identificação de tuplas duplicadas. Porém, como já mencionado anteriormente, a tendência dos grandes volumes de dados atuais se alinha com a utilização das técnicas *ad-hoc*. Como consequência, optou-se por absorvê-las nesse módulo.

Atualmente, este módulo conta com uma das versões do algoritmo *Edit Distance* e a versão original do algoritmo *Soundex*, ambos apresentados no capítulo 2, para identificação de tuplas duplicadas.

3.5 Módulo enriquecedor

O módulo enriquecedor possui a incumbência de enriquecer o texto dos registros de uma base de dados de acordo com o seu idioma. Esse enriquecimento acontece por

meio da aplicação de “regras de idioma” as quais podem ser definidas previamente na própria ferramenta e possui como objetivo principal a aproximação ortográfica desses registros. Tal aproximação tem a finalidade de auxiliar as técnicas definidas no módulo de identificação de tuplas duplicadas para que possam desempenhar melhor o seu trabalho.

As regras de idioma mencionadas no parágrafo anterior podem ser vistas como um conjunto de características que possibilitam, de alguma forma, a aproximação dos registros da base de dados. Essas regras são oriundas de observações de problemas pontuais existentes em bases de dados reais e também da área de recuperação de informações, mais precisamente das operações realizadas sobre textos como: análise léxica, tesauros, *stemming*, corretores ortográficos e padronização textual. É necessário manter em mente que essas regras devem ser definidas de forma agregada a um idioma específico, pois, dessa forma, o ambiente poderá utilizá-las posteriormente de acordo com o idioma definido na base de dados. A figura 3 ilustra uma das interfaces do ambiente com o campo em que o idioma desejado é escolhido (destacado em vermelho). Esse campo é de suma importância para o módulo enriquecedor, pois todas as vezes que o ambiente precisar tomar decisões sobre quais regras devem ser utilizadas em determinado momento, é ele que fornecerá tal informação.

As regras de idioma utilizadas neste módulo são absorvidas pelos submódulos, ilustrados na figura 2, que realizam o processamento designado a eles com auxílio de uma base de dados auxiliar denominada “DCmanut”. Esta base de dados tem como principal responsabilidade o armazenamento de todos os idiomas suportados pelo ambiente, juntamente com suas respectivas regras pré-definidas. Dessa forma, com a utilização dessa base de dados e do campo em destaque, apresentado na figura 3, pode-se garantir que a ferramenta tenha sempre ao seu alcance os recursos necessários para trabalhar com o enriquecimento de uma base de dados definida em qualquer idioma.

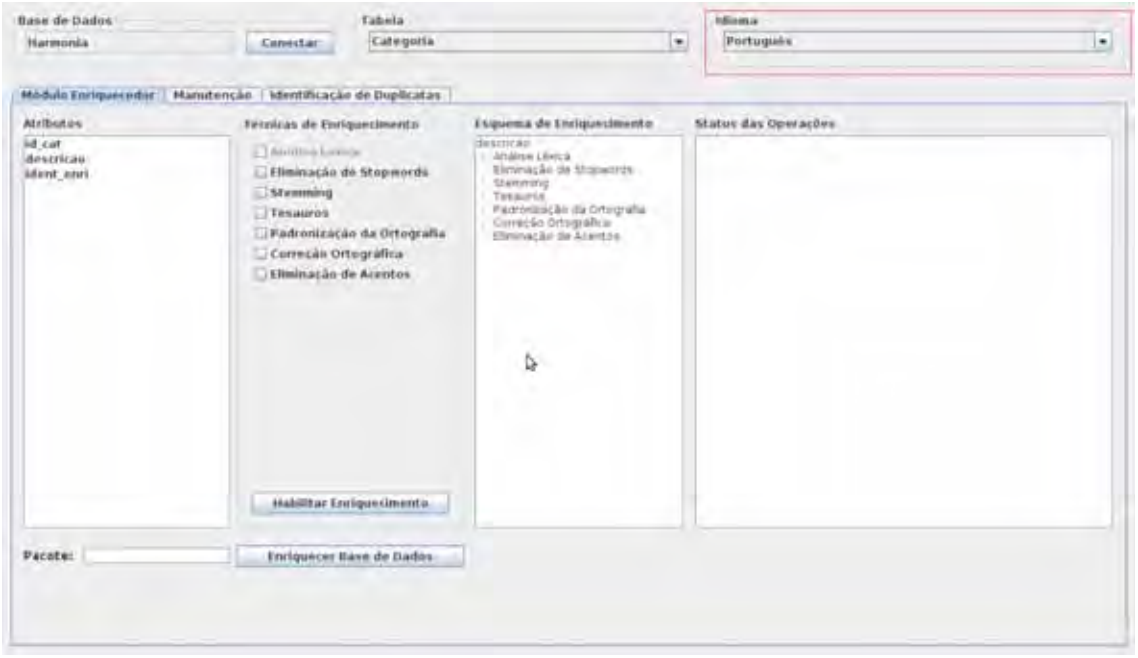

Figura 3 – Interface de apresentação do ambiente.

A figura 4 ilustra a estrutura da base de dados “DCmanut”.

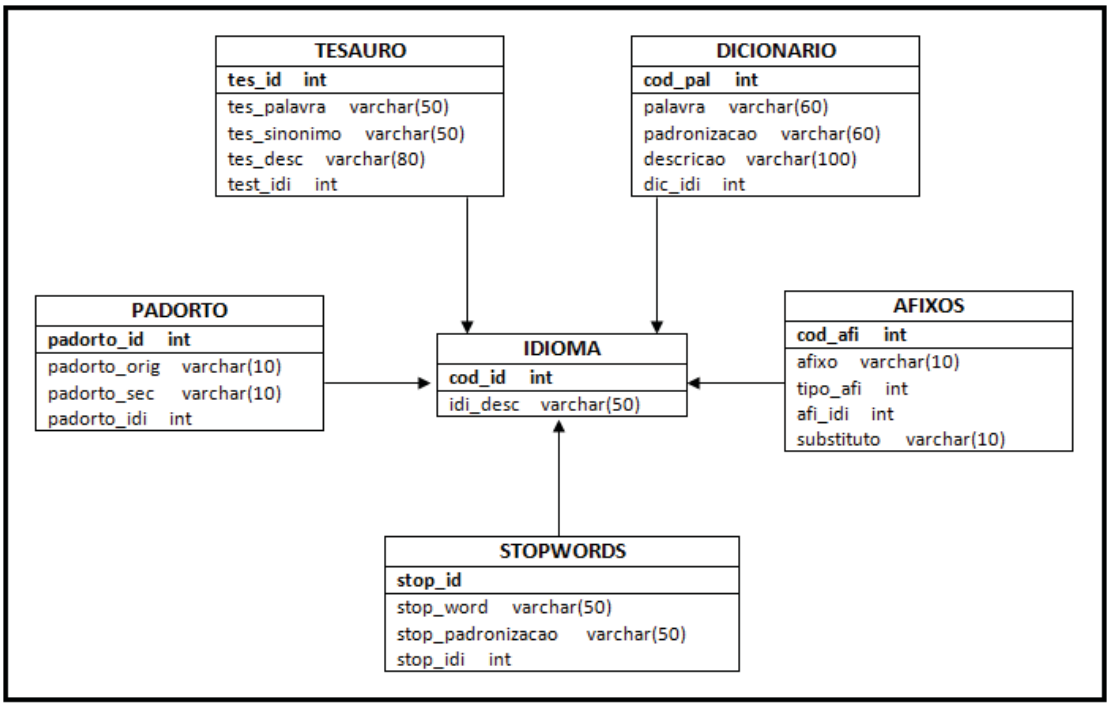

Figura 4 – Base de dados auxiliar para implementação da pré-etapa.

Em relação à figura 4, é válido ressaltar que a tabela “Idioma” é o centro da base de dados auxiliar “DCmanut”. É nessa tabela que os idiomas suportados pelo ambiente

são registrados e é por meio dela (chaves estrangeiras) que as regras de idioma são associadas aos idiomas suportados pelo ambiente para uso posterior.

Em relação à execução dos submódulos do módulo enriquecedor, ao analisar-se o trabalho que cada um executa, nota-se que não é possível processar uma mesma palavra paralelamente em vários submódulos. Sendo assim, há a necessidade de executá-los de forma sequencial. Isso significa que uma tabela de uma base de dados que participa do processo de enriquecimento é percorrida palavra por palavra em relação aos campos definidos para procedimento e que cada palavra passa de forma sequencial por todos os submódulos definidos para se tornar enriquecida.

Considerando-se o fato de que os submódulos devem estar sequenciados para realizar o enriquecimento dos registros com sucesso, surge uma pergunta intrigante sobre o tema e que versa sobre qual seria a ordem de aplicação correta dos submódulos para realização desta tarefa. Certamente há controvérsias em relação a esta resposta, mas, com a análise dos problemas reais inerentes às bases de dados, definiu-se a seguinte ordenação com o intuito de contribuir neste sentido.

1. **Análise Léxica/Eliminação de acentos:** organização do fluxo de caracteres em palavras com o intuito de analisá-las e eliminação da acentuação para prevenir possíveis erros;
2. **Padronização ortográfica:** tentativa de eliminar alguns erros previsíveis de linguagem;
3. **Correção ortográfica:** diante de palavras não pertencentes ao dicionário e, consequentemente, consideradas alvo de maior investigação, procuram-se as substituições mais plausíveis dentro de suas zonas de significância e norma padrão da língua em questão;
4. **Eliminação de *stopwords*:** uma vez que o processo de correção de palavras já está concluído, tenta-se reconhecer e excluir *stopwords*;
5. **Tesauros:** uma vez que o processo de correção de palavras já está concluído, tenta-se reconhecer e substituir palavras que possuam sinônimos;
6. ***Stemming*:** tentativa de transformar as palavras já tratadas pelas etapas anteriores em um radical comum.

A figura 5 ilustra o procedimento de enriquecimento de uma sentença utilizando todos os submódulos definidos para o módulo enriquecedor, os quais, por sua vez, serão

apresentados detalhadamente nas subseções subsequentes. Os submódulos estão destacados em negrito e, a cada vez que uma sentença de entrada é processada, o seu resultado – sentença de saída – é passado como parâmetro de entrada para o submódulo seguinte. Dessa forma, ao final do procedimento, tem-se uma sentença reduzida e considerada enriquecida pelas regras de idioma definidas na base de dados "DCmanut" ilustrada na figura 4.

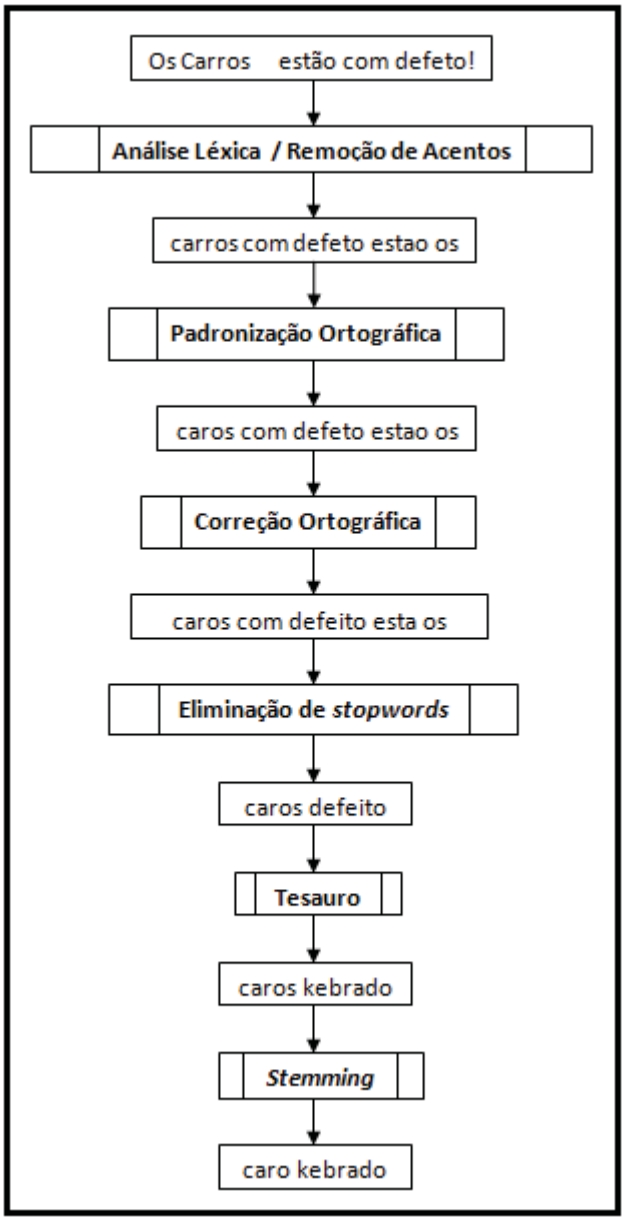

Figura 5 – Exemplo de enriquecimento de uma sentença.

É importante observar que o trabalho realizado, palavra por palavra, em um grande volume de dados exige uma alta carga de processamento. Adicionando a outra

carga de processamento gerada pelo procedimento de identificação de tuplas duplicadas, faz com que sua soma total possa tornar-se proibitiva. Como consequência, essa atividade de enriquecimento de dados, proposta como pré-etapa do processo de identificação de tuplas duplicadas, pode ser vista como um elemento extra de consumo de processamento e não ser incluída como um recurso no processo de KDD.

Ao analisar a carga de processamento extra, gerada pela execução dos submódulos para enriquecimento da base de dados, e seu caráter sequencial de processamento, sentiu-se a necessidade de recorrer a alternativas que pudessem otimizar esta fase, no caso o processamento paralelo. Tal esquema de processamento foi implantado com o auxílio de programação *multithreading* e tem o propósito de explorar ao máximo os vários núcleos de processamento oferecidos pelos processadores atuais, os quais, geralmente, tendem a ficar ociosos durante a execução de uma aplicação sem este recurso.

As subseções 3.5.1 e 3.5.2 apresentarão os dois modelos de processamento paralelo adotados por este trabalho e as subseções posteriores apresentarão os submódulos de enriquecimento propostos.

3.5.1 Processamento utilizando *threads* paralelas

O primeiro modelo de processamento, denominado *threads* paralelas, consiste em dividir o trabalho em partes iguais e distribuí-las de acordo com a quantidade de *threads* definidas. Considerando a aplicação deste modelo de processamento no presente trabalho, cada uma das *threads* definidas conterá todos os submódulos de enriquecimentos e os executará em sequência para uma determinada porção da base de dados envolvida no processo de enriquecimento.

Na figura 6, este modelo de processamento é ilustrado para duas *threads* supostamente definidas em um processador de dois núcleos.

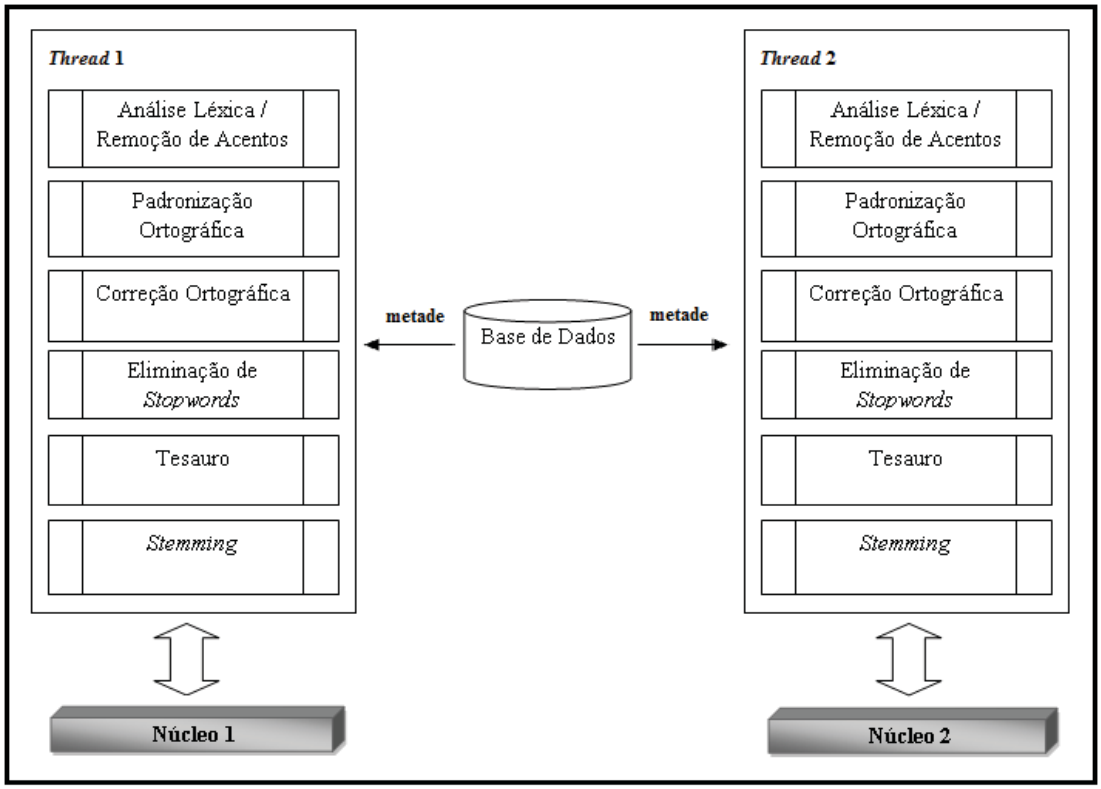

Figura 6 – Modelo de processamento com *threads* paralelas.

Como pode ser notado na figura 6, cada *thread* recebe metade dos registros da base de dados envolvida no processo de enriquecimento e os processa utilizando todos os submódulos de enriquecimento de forma sequencial. Vale ressaltar que cada *thread* trabalha com metade dos registros da base de dados pelo fato de que somente duas *threads* foram definidas. Caso houvesse três *threads* definidas, cada uma receberia 1/3 dos registros da base de dados.

3.5.2 Processamento utilizando *threads* em *pipeline*

O segundo modelo de processamento, denominado *threads* em *pipeline*, possui inspiração no esquema de processamento de instruções em *pipeline* realizado pelos processadores e consiste basicamente na realização do processamento paralelo dividido em estágios. Um *pipeline* de instruções é semelhante a uma linha de montagem de uma indústria. A linha de montagem tira proveito do fato de que um determinado produto passa por vários estágios de produção, um de cada vez. Consequentemente, vários destes produtos podem ser trabalhados em estágios de produção distintos simultaneamente. Assim, como em uma linha de montagem, em um *pipeline* de

instruções, novas entradas são aceitas em uma extremidade antes que entradas aceitas previamente apareçam como saídas na outra extremidade (STALLINGS, 2002).

Com a utilização destes conceitos dentro do ambiente proposto, pode-se organizar um ou mais submódulos em cada um dos estágios do *pipeline*, de forma que seja possível realizar o processamento paralelo de palavras com o objetivo final de enriquecê-las, como se fosse uma "linha de montagem de palavras".

A figura 7 ilustra um modelo de como pode ser realizado o trabalho do *pipeline* no ambiente, utilizando cada um dos seus submódulos como um único estágio deste *pipeline*.

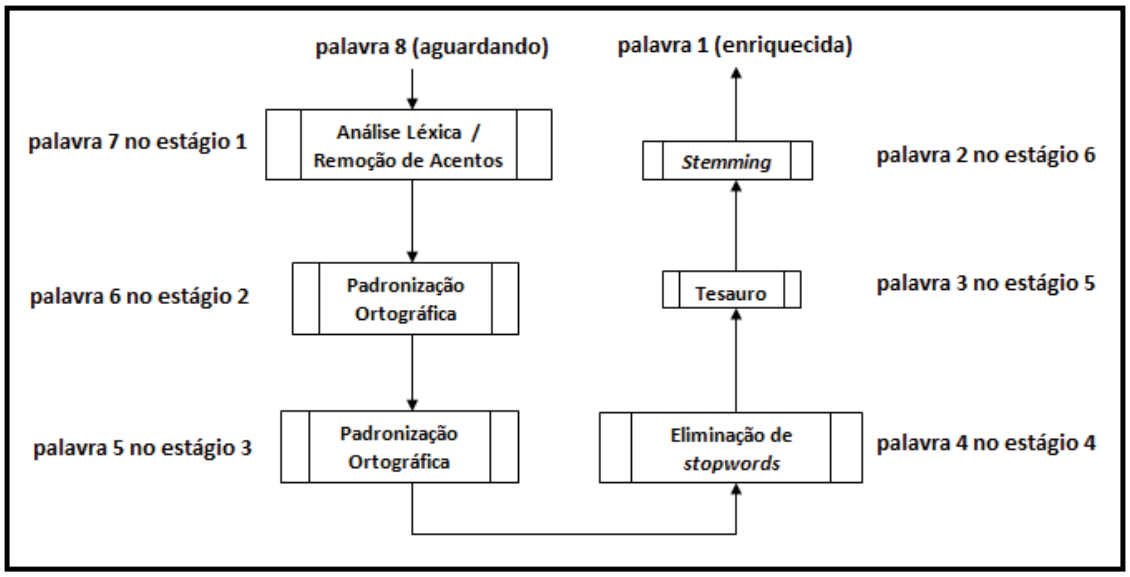

Figura 7 – Modelo de *pipeline* de seis estágios no ambiente proposto.

Para caracterizar o *pipeline*, a tabela não será enriquecida toda de uma vez, mas sim por partes. Isso significa que pequenos pacotes serão extraídos da tabela e serão vistos como "instruções", as quais passarão em cada um dos estágios do *pipeline*. Por exemplo, supondo que a tabela X contenha 10.000 registros. Pode-se dividi-la em 100 pacotes com 100 registros cada. Dessa forma, têm-se 100 "instruções" para serem processadas em paralelo na linha de produção imposta de acordo com a quantidade de estágios definidos pelo próprio ambiente.

A figura 8 ilustra um modelo da ideia de *pipeline multithreading* discutido anteriormente para um processador de dois núcleos que faz uso de dois estágios de *pipeline*.

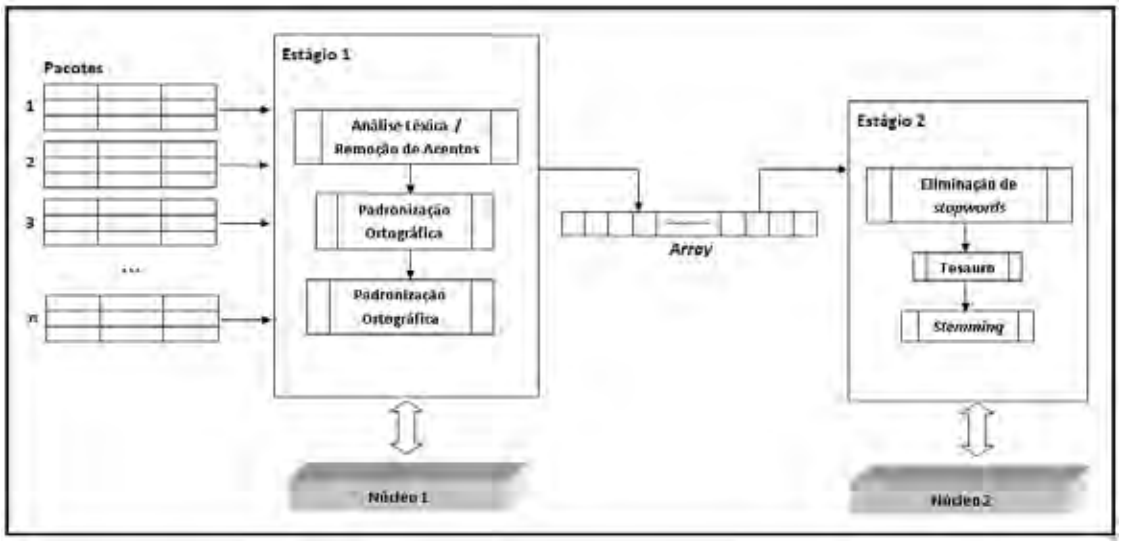

Figura 8 – Modelo de *pipeline multithreading* de dois estágios.

O ambiente proposto por este trabalho visa implantar o modelo apresentado na figura 8 como estratégia principal de processamento. Dessa forma, de modo antecipado, pretende-se minimizar os problemas de desempenho com a tentativa de enriquecer grandes bases de dados. Como consequência, a pré-etapa proposta pelo trabalho pode se tornar mais atrativa e, consequentemente, ser incluída como um recurso indispensável ao processo de KDD.

Às próximas subseções cabe a apresentação detalhada de cada um dos submódulos do módulo enriquecedor.

3.5.3 Análise léxica

O submódulo “Análise léxica”, assim como nos compiladores, possui a responsabilidade de converter um fluxo de caracteres em um fluxo de palavras. Para isso, leva-se em consideração espaços em branco e caracteres de pontuação como separadores para identificação e isolamento dessas palavras, que podem ser denominadas *tokens*.

O que acontece dentro deste submódulo é a reconstrução dos registros da base de dados com o intuito de eliminar espaços em branco excedentes, caracteres de pontuação e também padronizar todas as letras para minúsculo. Tudo com o objetivo de aproximar ortograficamente os registros. Além da reconstrução, o submódulo também possui a opção de ordenar ortograficamente as palavras dentro do registro para auxiliar nessa

aproximação, pois há vários casos em que palavras são deslocadas para representar o mesmo tipo de informação.

A seguir, é apresentado um algoritmo que representa o funcionamento do procedimento.

Algoritmo:

Entradas:

T – Tabela em que as tuplas duplicadas devem ser identificadas.

Procedimento:

Para cada tupla t_i de T

elimina espaços em branco excedentes

remove caracteres de pontuação

transforma todas as letras em minúsculas

ordena as palavras dentro do registro (opcional)

salva essa nova cópia em T_E$

Para exemplificar o funcionamento deste submódulo, pode-se considerar a seguinte sentença:

"Existe apenas um bem, o SABER, e apenas um mal, a IGNORÂNCIA."

Como saída será produzida a seguinte sentença:

"a apenas apenas bem e existe ignorância mal o saber um um"

3.5.4 Tesauro

Os tesauros, também considerados como uma ontologia simples, podem ser definidos, amiúde, como uma lista pré-definida de palavras com os seus respectivos sinônimos associados. Ao definir esta lista, é muito importante que os termos que a compõem sejam palavras essenciais no domínio em questão para um melhor resultado. Embora os tesauros aparentem ser uma estrutura muito simples, geralmente suas implementações envolvem a normalização do vocabulário e incluem estruturas mais complexas que uma lista de palavras e sinônimos, como a utilização de sentenças.

A motivação para se construir um tesauro é baseada na ideia de utilizar um vocabulário controlado nas atividades de indexação e busca. O vocabulário controlado apresenta muitas vantagens como normalização de conceitos de indexação, redução de divergências, identificação de termos de indexação com um claro significado de semântica e recuperação baseada em conceitos (YATES; RIBEIRO NETO, 2000).

Ao analisar as vantagens que uma estrutura de tesauro pode proporcionar, nota-se que a substituição de palavras similares pode ajudar de forma significativa no processo de aproximação de *strings*, e, consequentemente, melhorar os resultados das estratégias de identificação de tuplas existentes. Como já ilustrado na figura 4, o ambiente utiliza uma tabela "DCmanut.Tesauro" para armazenar uma lista de palavras e seus respectivos sinônimos, e os associa por meio de uma chave estrangeira que se refere a um idioma específico para que o ambiente saiba qual conjunto de palavras utilizar ao processar uma tabela de base de dados.

A seguir, é apresentado um algoritmo que descreve o funcionamento deste submódulo.

Algoritmo

Entrada:

T - Tabela em que as tuplas duplicadas devem ser identificadas.

O - Tesauro contendo a lista de sinônimos

Procedimento:

Para cada registro *ti* da tabela T

    Para cada palavra *pj* do registro *ti*

        busca palavra *pj* na tabela O

        Se palavra *pj* for encontrada, substitui *pj* pelo sinônimo *s* no registro *ti*

Para exemplificar o funcionamento deste algoritmo, pode-se considerar o seguinte par de palavras definidos na tabela "DCmanut.Tesauro": "assistente" e "auxiliar". Ao aplicar este procedimento em uma tabela que armazena registros de profissões, por exemplo, todas as palavras "assistente" que forem encontradas serão substituídas pelo seu sinônimo "auxiliar". Sendo assim, cargos como "assistente de escritório" ou "assistente de projetos" seriam convertidos para "auxiliar de escritório" e "auxiliar de projetos".

Com esse exemplo, reforça-se a importância de se conhecer o domínio da base de dados para que seja possível definir termos que realmente contribuirão nas

substituições. Dessa forma, consegue-se realizar uma aproximação ortográfica dos registros, e, assim, melhorar os resultados proporcionados pelas técnicas de identificação de tuplas duplicadas baseadas em similaridades.

3.5.5 Padronização ortográfica

É inerente a qualquer idioma que os seus falantes possuam dúvidas em relação à grafia de determinadas palavras e cometam erros ao escrevê-las. Diante deste fato, observou-se a existência de muitos erros ortográficos presentes em bases de dados reais, que, de certa forma, atrapalham no processo de identificação de tuplas duplicadas, pois os algoritmos não são especializados nos idiomas, e, por isso, não são capazes de julgar se determinadas grafias estão corretas ou erradas, simplesmente julgam os itens analisados.

A proposta deste submódulo é utilizar regras previamente definidas para um idioma específico a fim de padronizar a grafia das palavras, e, com isso, minimizar a distância ortográfica entre elas. Por exemplo, com base na regra de substituição $ch \rightarrow x$, todas as palavras grafadas com “ch” de uma tabela específica passariam a ser grafadas com “x”, mesmo que a grafia correta da palavra seja utilizando “ch”. O importante nesta funcionalidade não é encontrar a grafia correta, mas sim levar todas as palavras para um ambiente controlado e livre destes possíveis erros. Ademais, dúvidas sobre o emprego de “x” e “ch” podem ser citadas como um exemplo de erro muito comum na língua portuguesa.

Como ilustrado na figura 4, o ambiente utiliza uma tabela denominada “DCmanut.PadOrto” para armazenar as regras de padronização ortográfica definidas para cada idioma.

Na tabela 1, é possível visualizar todas as regras já definidas para o idioma português no ambiente.

Tabela 1 – Padronização de ortografia para língua portuguesa.

Origem	Destino	Origem	Destino	Origem	Destino	Origem	Destino
ce	se	rr	r	cao	cao	nb	mb
ci	si	ge	je	que	ke	np	mp
z	s	gi	ji	qui	ki	xc	s
ss	s	y	i	ha	a	xd	sd
ç	s	ph	f	he	e	xp	sp
sci	si	ca	ka	hi	i	xq	sq
ch	x	co	ko	ho	o	xt	st
sh	x	cu	ku	hu	u		

Como se pode notar na tabela 1, as regras de padronização são definidas com base em dois parâmetros: origem e destino. O parâmetro “origem” representa o que se deseja encontrar nas palavras analisadas pelo módulo enriquecedor. Uma vez que este parâmetro é encontrado, o mesmo é substituído pelo parâmetro definido como “destino”.

A seguir, é apresentado um algoritmo que descreve o funcionamento deste submódulo:

Algoritmo

Entradas:

T - Tabela em que as tuplas duplicadas devem ser identificadas.

P - Tabela DCmanut.PadOrto que contém as regras de padronização

Procedimento:

Seleciona os padrões definidos na tabela P para um idioma específico

Para cada registro r_i da tabela T

Para cada palavra p_j do registro r_i

Procura-se cada um dos parâmetros origem e substitui quando necessário

A tabela 2 exibe alguns exemplos de substituição realizados por essa funcionalidade.

Tabela 2 – Exemplos de palavras padronizadas na língua portuguesa.

Antes	Depois
amazonas	amasonas
cuba	kuba
thaciane	tasiane
abscesso	abseso
acolchoamento	akolxoamento

3.5.6 Remoção de *stopwords*

É muito comum, em qualquer tipo de texto, deparar-se com palavras cuja frequência de aparição é alta. Uma palavra que aparece em 80% dos registros de uma base de dados, por exemplo, pode ser considerada não muito útil em processos de recuperação de informações, pois não é considerada como uma boa discriminadora. Isso significa que não desempenha bem o papel de caracterizar o contexto do documento ou tuplas que representa. Essa classe de palavras sem muito significado é frequentemente referida como *stopwords*. Artigos, preposições e conjunções são candidatos naturais para a lista de *stopwords*. Além disso, alguns verbos, advérbios e adjetivos também podem ser incluídos nesta lista (YATES; RIBEIRO NETO, 2000).

O processo de remoção de *stopwords* consiste em analisar as palavras de um texto de entrada, seja ele um documento ou uma base de dados, e compará-las com uma lista pré-definida de *stopwords*. Caso a palavra comparada exista nessa lista, a mesma deve ser removida do texto de entrada.

Como pôde ser visto na figura 4, este submódulo conta com uma tabela chamada “DCmanut.Stopwords” para auxiliar em seu funcionamento, cujo objetivo consiste em armazenar uma lista de palavras consideradas *stopwords* para um idioma específico registrado na tabela de idiomas.

A figura 9 ilustra uma lista de *stopwords* já cadastradas no ambiente para o idioma “Português”.

a, à, agora, ainda, alguém, algum, alguma, algumas, alguns, ampla, amplas, amplo, amplos, ante, antes, ao, aos, após, aquela, aquelas, aquele, aqueles, aquilo, as, até, através, cada, coisa, coisas, com, como, contra, contudo, da, daquele, daqueles, das, de, dela, delas, dele, deles, depois, dessa, dessas, desse, desses, desta, destas, deste, deste, destes, deve, devem, devendo, dever, deverá, deverão, deveria, deveriam, devia, deviam, disse, disso, disto, dito, diz, dizem, do, dos, e, é, ela, elas, ele, eles, em, enquanto, entre, era, essa, essas, esse, esses, esta, está, estamos, estão, estas, estava, estavam, estávamos, este, estes, estou, eu, fazendo, fazer, feita, feitas, feito, feitos, foi, for, foram, fosse, fossem, grande, grandes, há, isso, isto, já, la, la, lá, lhe, lhes, lo, mas, me, mesma, mesmas, mesmo, mesmos, meu, meus, minha, minhas, muita, muitas, muito, muitos, na, não, nas, nem, nenhum, nessa, nessas, nesta, nestas, ninguém, no, nos, nós, nossa, nossas, nosso, nossos, num, numa, nunca, o, os, ou, outra, outras, outro, outros, para, pela, pelas, pelo, pelos, pequena, pequenas, pequeno, pequenos, per, perante, pode, pôde, podendo, poder, poderia, poderiam, podia, podiam, pois, por, porém, porque, posso, pouca, poucas, pouco, poucos, primeiro, primeiros, própria, próprias, próprio, próprios, quais, qual, quando, quanto, quantos, que, quem, são, se, seja, sejam, sem, sempre, sendo, será, serão, seu, seus, si, sido, só, sob, sobre, sua, suas, talvez, também, tampouco, te, tem, tendo, tenha, ter, teu, teus, ti, tido, tinha, tinham, toda, todas, todavia, todo, todos, tu, tua, tuas, tudo, última, últimas, último, últimos, um, uma, umas, uns, vendo, ver, vez, vindo, vir, vos, vós

Figura 9 – *Stopwords* da língua portuguesa.

A seguir, é apresentado um algoritmo que descreve o funcionamento deste submódulo.

Algoritmo

Entradas:

T - Tabela em que as tuplas duplicadas devem ser identificadas.

S – Tabela “DCmanut.Stopwords” que contém as *stopwords* registradas.

Procedimento:

Para cada registro *ri* da tabela T

Para cada palavra *pj* do registro *ri*

Procura palavra *pj* na tabela S

Se palavra *pj* for encontrada, remove do registro *ri*

Para exemplificar o funcionamento do algoritmo acima no tocante ao idioma em questão (Português), utilizar-se-á a sentença “São José do Rio Preto” – que poderia ser um exemplo de registro de uma tabela de cidades – considerando-se as *stopwords* apresentadas na figura 9. Após a aplicação do algoritmo, ter-se-ia como resultado a sentença “José Rio Preto”.

3.5.7 Remoção de acentos

A funcionalidade “Remoção de Acentos” caracteriza-se por realizar uma tarefa simples: a troca de letras acentuadas por letras não acentuadas. Isso porque, para os algoritmos de identificação de tuplas duplicadas, letras acentuadas são diferentes de letras não acentuadas e, sendo assim, a ausência ou acréscimo desnecessário de acentos por qualquer motivo pode influenciar diretamente no distanciamento ortográfico dos registros. Para implementar esta funcionalidade, utilizou-se uma classe disponibilizada pela linguagem de programação Java com a finalidade de efetuar a troca de qualquer letra acentuada pela sua correspondente sem acento.

3.5.8 Correção ortográfica

A funcionalidade “Correção Ortográfica” possui a mesma incumbência que os corretores ortográficos encontrados em editores de texto (i.e. Microsoft Word, Open Office, Symphony e etc.), que é identificar palavras desconhecidas em relação a um dicionário e propor possíveis substituições para as mesmas. A padronização ortográfica, apresentada na seção 3.5.4, resolve alguns problemas previsíveis relacionados à grafia das palavras, deixando os erros não previsíveis de lado. Dessa maneira, cabe aos corretores ortográficos a tentativa de complementar o trabalho realizado pela funcionalidade “Padronização Ortográfica”.

O funcionamento deste submódulo consiste em verificar a existência de cada palavra em um dicionário específico do idioma trabalhado. Caso a palavra não seja encontrada, assume-se que a mesma esteja grafada com algum erro não resolvido pela padronização de ortografia, e se faz necessária a identificação de um grupo de palavras candidatas para substituir essa palavra em investigação. Para identificar uma possível palavra substituta para o vocábulo em questão de forma automática no ambiente proposto, utilizam-se duas etapas:

- **Identificação de grupo de palavras candidatas:** selecionar palavras do dicionário cuja quantidade de caracteres é definida pela mesma quantidade de caracteres da palavra suspeita com variação de ± 1. Além disso, faz-se necessário que a letra inicial das palavras do dicionário seja a mesma letra inicial da palavra em investigação.
- **Definição da palavra substituta:** após a identificação do grupo de palavras

candidatas, aplica-se o algoritmo *Edit Distance* entre cada uma delas e a palavra em investigação. Uma vez que todas as palavras candidatas foram comparadas com a palavra em investigação, define-se como “palavra substituta” aquela que possuir a menor distância em relação à palavra em investigação de acordo com o algoritmo, e cuja distância seja menor ou igual ao *treshold* definido. O valor de *treshold* escolhido para este procedimento é definido, por padrão, como dois. No caso de haver mais de uma palavra que respeite as condições impostas em relação ao algoritmo *Edit Distance* e em relação ao valor de *treshold* igual a dois, a palavra em investigação não é substituída, pois acredita-se que não há um nível adequado de certeza para realização da troca.

É importante ressaltar que as regras definidas no corretor ortográfico não possuem fundamentos científicos, mas compreende-se que colaboram de forma significativa para a implementação de um corretor ortográfico cujas decisões sejam automáticas, sem a utilização de um contexto definido. A seguir, é apresentado um algoritmo que descreve o funcionamento desse submódulo.

Algoritmo

Entradas:

T - Tabela em que as tuplas duplicadas devem ser identificadas.

D - Tabela “DCmanut.Dicionario” que contém as palavras de determinado idioma.

Procedimento:

Para cada registro r_i da tabela T

Para cada palavra p_j do registro r_i

Busca palavra p_j na tabela D

Se p_j não existir em D

Seleciona grupo de palavras candidatas

Aplica *Edit Distance* entre todas as palavras candidatas e p_j

Identifica palavras com menor distância

Verifica se menor distância é menor que *treshold*=2

Se existir somente 1 palavra que satisfaça as condições anteriores

Realiza substituição

3.5.9 *Stemming*

Além de todos os pormenores citados anteriormente, os quais podem causar o distanciamento ortográfico entre os registros de uma base de dados, deve-se também considerar o problema causado pelas variações sintáticas das palavras. Plurais, formas de gerúndio e sufixos indicativos de passado são exemplos dessas variações que também podem impactar nos resultados do processo de identificação de tuplas duplicadas. Uma das formas de lidar com esse problema e melhorar parcialmente a qualidade do texto envolve a substituição dessas variações de palavras pelos seus radicais ou também conhecidos como *stems*, na área de recuperação de informação. Um *stem* é uma porção da palavra que resta após a remoção de seus afixos (prefixos e sufixos). Por exemplo, o *stem* da língua inglesa *connect* pode ser considerado a base das seguintes variações: *connected*, *connection*, *connecting* e *connections* (YATES; RIBEIRO NETO, 2000).

O processo de *stemming* pode ser elucidado como um método de redução de uma palavra em seu radical por meio da remoção de seus afixos, de forma que as palavras morfologicamente relacionadas sejam representadas por uma única forma comum. Aplicando-se este conceito no domínio de identificação de tuplas duplicadas, trata-se da redução das palavras que compõem uma base de dados em seus respectivos radicais, com o intuito de reduzir a quantidade de suas variações, e assim minimizar a distância ortográfica entre os registros.

Frakes e Baeza-yates (1992) apontam quatro estratégias diferentes para realização do processo de *stemming*: consulta em dicionário; variedade de sucessores; *n*-gramas e remoção de afixos. De acordo com a proposição deste trabalho, utilizar-se-á somente o método de remoção de afixos, pois é a estratégia que se alinha de forma mais adequada ao propósito de implementação independente de idioma proposto pela ferramenta.

A parte mais importante no processo de remoção de afixos é a remoção dos sufixos, pois a maioria das variações de palavras é gerada pela introdução de um sufixo no seu radical. Existem alguns algoritmos propostos na literatura cujo objetivo é a remoção de sufixos, sendo o mais popular deles o algoritmo de Porter (PORTER, 1980). Embora o algoritmo de Porter seja o mais popular, o mesmo não se identifica com a finalidade desta pesquisa, pois suas regras precisam ser adaptadas para cada idioma em que for utilizado. Já o algoritmo RSLP (ORENGO; HUYCK, 2001), que é proposto

para lidar especificamente com o idioma português, mostra-se mais eficiente que o de Porter, pois apresenta, nesse idioma, um formato de conjunto de regras de idioma o qual se mostra interessante para ser adaptado ao propósito deste ambiente. Basicamente, este algoritmo é executado em oito etapas consecutivas de tal forma que os sufixos mais extensos devem ser removidos primeiro. É, ainda, desenvolvido com base nos sufixos mais comuns da língua portuguesa.

A seguir, estão apresentadas as oito etapas do algoritmo RSLP.

1. **Redução de plural:** consiste em remover o final “s” das palavras que não estão listadas como exceções. Isso porque nem todas as palavras com terminação em “s” estão no plural (i.e. lápis).
2. **Redução do gênero feminino:** todos os substantivos e adjetivos na língua portuguesa possuem um gênero. Esta etapa consiste em transformar palavras que estejam na forma feminina em sua respectiva forma masculina. Somente palavras terminadas em “a” são testadas nessa etapa, mas nem todas são reduzidas. Só serão reduzidas as palavras com sufixos mais comuns.
3. **Redução de advérbio:** é a menor etapa do processo, pois trabalha com somente um sufixo que denota advérbio: mente. Porém, nem todas as palavras que terminam em “mente” são reduzidas. Uma lista de exceções se faz necessária.
4. **Redução de aumentativo e diminutivo:** remove os sufixos que representam aumentativo e diminutivo nas palavras.
5. **Redução de substantivo como sufixo:** verifica se o sufixo da palavra coincide com um dos 61 substantivos (ou adjetivos) de uma lista pré-definida. No caso de o sufixo ser removido nesta etapa, as etapas 6 e 7 não serão executadas.
6. **Redução de sufixos de verbos:** os verbos regulares da língua portuguesa possuem cerca de 50 formas diferentes. Cada um possui um sufixo específico e pode variar de acordo com o tempo, pessoa, número e modo. A estrutura das formas verbais pode ser representada como: raíz + vogal temática + desinência de tempo + desinência de pessoa (i.e. and + a + ra + m = andaram). As formas verbais são reduzidas até sua raiz.
7. **Redução de vogal:** consiste em remover a última vogal (a, e, i, o, u) das palavras que não foram reduzidas pelas etapas 5 e 6.
8. **Remoção de acentos:** A remoção de acentos se faz necessária, pois em alguns casos formas variantes são acentuadas. Por exemplo, é o caso das palavras “psic**ó**logo” e “psicologia”. Para essas duas palavras, o radical “psicolog” deve

> ser gerado. É importante que a remoção de acentos seja efetuada ao final do algoritmo e não no início, pois a presença dos acentos no decorrer do algoritmo pode ser importante para algumas regras como: óis -> ol e sóis -> sol.

Como foi mencionado anteriormente, o algoritmo RSLP apresenta uma estrutura de regras de idioma que pode ser facilmente absorvida pelo ambiente para realizar o tratamento de bases de dados independentemente do idioma. Sendo assim, para compor este submódulo, utilizar-se-á o algoritmo RSLP como base. Ele não será implementado como fora designado por Orengo e Huyck (2001), mas contribuirá com o formato proposto para as regras de idioma. Isso significa que somente as seguintes fases do algoritmo foram consideradas: redução de plural, redução de advérbio, redução de vogal e remoção de acentos.

Embora o processo de *stemming* aparente ser um procedimento benéfico para recuperação de informações, existem algumas controvérsias na literatura em relação a sua eficácia. Em (FRAKES; BAEZA-YATES, 1992), pode-se identificar um estudo comparativo de oito trabalhos que discutem os reais benefícios do *stemming*. Embora os autores se mostrem favoráveis ao uso deste procedimento, os estudos, por eles comparados, não são tão conclusivos a ponto de afirmar a eficácia do tópico em questão. Como exemplo de uma das desvantagens do processo de *stemming*, pode-se citar a redução de precisão no processo de recuperação de informação e também a perda de contexto das informações, a qual tem como consequência a produção de *stems* iguais para palavras com significados diferentes. Para constatar a veracidade da discussão, pode-se citar como exemplo a palavra “verão” (estação do ano), que, ao ser reduzida ao seu respectivo radical, se transforma no *stem* “ver” (verbo).

Mesmo que ainda não exista um consenso geral sobre o real benefício da técnica de *stemming*, o ambiente conta com este submódulo. Para o propósito de identificação de tuplas duplicadas, em que geralmente se utiliza algoritmos de similaridade textual, o significado não é o fator mais importante, mas sim a aproximação ortográfica. Sendo assim, acredita-se que este submódulo pode contribuir de forma positiva para o enriquecimento dos registros em determinadas situações.

Capítulo 4
Testes e Resultados

4.1 Ambiente de testes

Para realização dos testes do ambiente proposto, utilizou-se um sistema computacional com a seguinte configuração: processador Intel *Core* 2 *Duo* T5800 (2.0 GHz, 800 MHz FSB), memória RAM DDR2 de 2GB (667 Mhz) e sistema operacional Ubuntu Linux 11.04. O ambiente foi desenvolvido com o uso da linguagem de programação Java, juntamente com o SGBD MySQL 5.1.54-1ubuntu4.

O volume de dados utilizado nos testes caracteriza-se como uma base de dados real, sob o domínio do Grupo de Banco de Dados (GBD) da UNESP de São José do Rio Preto, e objeto de convênio com a Prefeitura Municipal de São José do Rio Preto, que suporta o Sistema de Informação e Vigilância de Acidentes do Trabalho (SIVAT). A base de dados SIVAT armazena registros de acidentes de trabalho que ocorrem na cidade de São José do Rio Preto e região. A utilização desta base de dados em específico mostrou-se atrativa, pois o procedimento utilizado para inserção de informações favorece a criação de tuplas duplicadas. Uma vez que os acidentes de trabalho são identificados nas unidades de saúde, preenche-se manualmente uma ficha física com informações do tipo: dados do paciente, características do acidente, informações relacionadas ao médico que atendeu a ocorrência, informações relacionadas à pessoa que preencheu tal ficha, empresa em que o acidentado trabalha, cargo que o acidentado ocupa dentro da empresa, entre outras. De tempos em tempos, pessoas contratadas pelo Centro de Referência em Saúde do Trabalhador (CEREST) transferem as informações de cada uma das fichas para o sistema computacional

alocado em um dos servidores mantidos pelo GBD. Dessa forma, seja por problemas com letras ilegíveis, falta de concentração das pessoas que transferem as informações para o sistema, erros ortográficos na ficha, erros de digitação ou forma como o referenciado relata os dados para o preenchimento, cria-se uma enorme quantidade de tuplas duplicadas nas tabelas da base de dados do sistema.

Diante da base de dados SIVAT, utilizou-se duas tabelas que apresentam problemas com a criação de tuplas duplicadas: “Ocupação” (699 registros) e “Responsável_Preenchimento” (516 registros). A tabela “Ocupação” armazena informações referentes aos cargos que os acidentados ocupam dentro das empresas em que trabalham; a tabela “Responsável_Preenchimento” armazena informações referentes às pessoas que preenchem as fichas manualmente nas unidades de saúde. É importante ressaltar que as duas tabelas utilizadas nos testes apresentam domínios bem diferentes. A tabela “Ocupação”, por exemplo, pode ser de grande importância para a aplicação das técnicas de *data mining*, pois pode contribuir para a geração de padrões relevantes que relacionam os acidentes de trabalho registrados às funções desempenhadas pelos acidentados.

O processo de testes aplicado nas tabelas citadas envolve tanto o módulo enriquecedor quanto o módulo de identificação de duplicatas. Primeiramente, enriqueceram-se as tabelas utilizando os submódulos pertencentes ao módulo enriquecedor (análise léxica, eliminação de *stopwords*, *stemming*, tesauros, padronização ortográfica, correção ortográfica e eliminação de acentos) e geraram-se as tabelas enriquecidas “Ocupação_E$” e “Responsavel_preenchimento_E$”. Para identificar os registros duplicados, utilizou-se o módulo de identificação de duplicatas, que, atualmente, conta com a implementação de duas estratégias: *Edit Distance* (similaridade léxica) e *Soundex* (similaridade fonética).

A dinâmica de comparação dos registros foi realizada da forma $i \rightarrow [i+1, n]$, em que i representa um contador que percorre todos os registros da tabela alvo até o penúltimo registro, e n representa a quantidade total de registros na tabela alvo. Isso significa que, quando $i=1$, compara-se i com todos os outros registros a partir de 2, quando $i=2$, compara-se i com todos os outros registros a partir de 3, e assim por diante. Fez-se a opção por esta estratégia de avaliação da contribuição do enriquecimento textual para os métodos de identificação de tuplas duplicadas, pois deste modo são realizadas todas as comparações possíveis entre os registros e evita-se comparações repetidas.

Com base no esquema de comparação de tuplas proposto, a utilização de grandes tabelas no processo de enriquecimento mostrou-se inviável, pois a quantidade de pares de tuplas a serem manualmente analisados para determinação da eficiência do processo sempre será (n-1) fatorial, tal que n é a quantidade de registros da tabela. Assim, as tabelas utilizadas para os testes, apesar de serem reais, podem ser consideradas como amostras para evidenciar os benefícios de se utilizar a pré-etapa em relação ao processo de tuplas duplicadas.

A figura 10 ilustra o paradigma proposto (quadro da direita) em relação ao paradigma encontrado atualmente na literatura (quadro da esquerda).

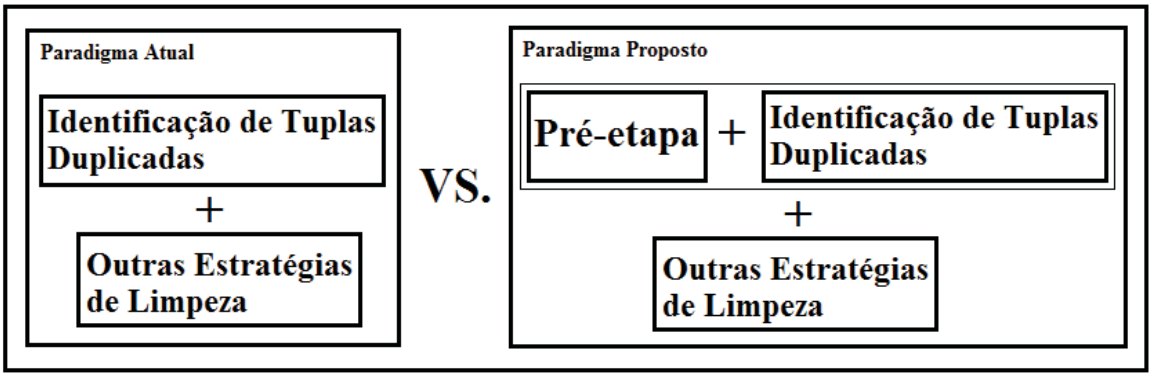

Figura 10 – Paradigma de identificação de duplicatas proposto vs. paradigma de identificação de duplicatas atual.

4.2 Testes utilizando o algoritmo *Edit Distance*

Para configuração do algoritmo *Edit Distance* utilizou-se *treshold* igual a dois, pois, após vários testes realizados na base de dados, esse valor mostrou-se ideal por não ser tão restritivo quanto o valor de *treshold* igual a um e nem tão tolerante com resultados indesejáveis para o valor de *treshold* igual a três ou superior. Isso significa que, quando dois registros comparados podem ser transformados um no outro por meio de até duas operações textuais, os mesmos são considerados duplicados. Sendo assim, contabilizou-se a quantidade de registros considerados duplicados para cada um dos valores de *treshold* possíveis (0, 1 e 2), antes e depois do enriquecimento.

As figuras 11 e 12 ilustram os resultados obtidos para as duas tabelas utilizadas no processo, sendo a coluna azul a responsável por contabilizar a quantidade de

duplicatas após o enriquecimento (_E$) e a coluna vermelha a responsável por contabilizar a quantidade de duplicatas antes do enriquecimento.

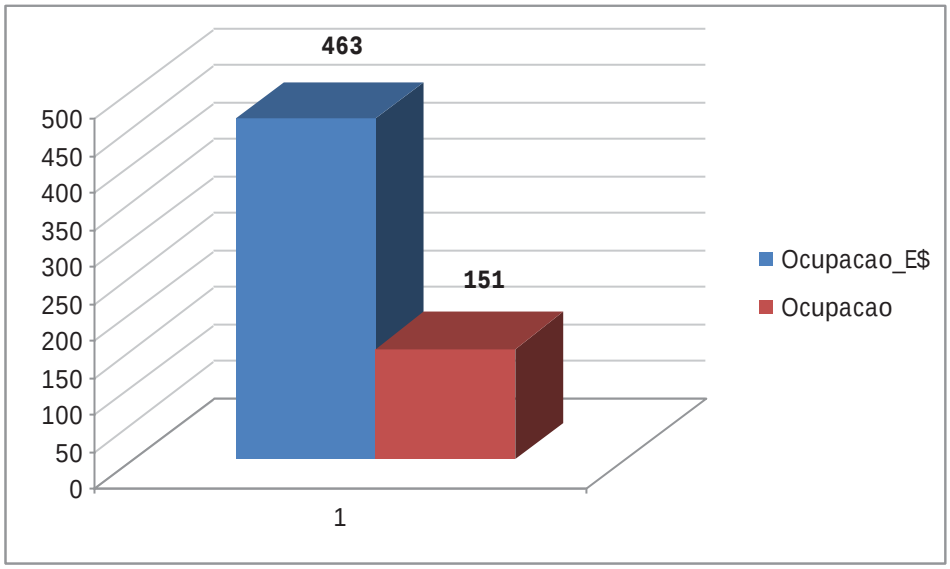

Figura 11 – Duplicatas identificadas na tabela "Ocupação" com o algoritmo *Edit Distance*.

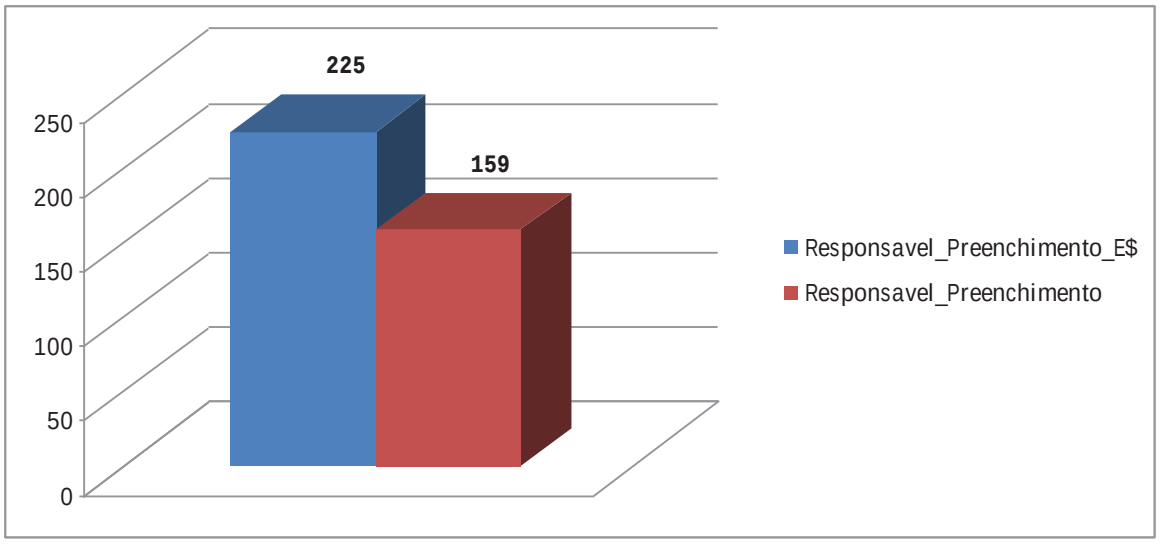

Figura 12 – Duplicatas identificadas na tabela "Responsavel_preenchimento" com o algoritmo *Edit Distance*.

De acordo com as figuras 11 e 12, pode-se notar um aumento significativo de pares de duplicatas encontrados no processo de identificação de tuplas duplicadas após o enriquecimento dos dados nas tabelas utilizadas. Isso significa que o enriquecimento dos dados gerou uma aproximação ortográfica dos registros e, consequentemente,

aumentou o número de tuplas duplicadas identificadas no processo. Porém, além de aumentar o número de tuplas duplicadas, era esperado a manutenção ou o aumento do nível de eficiência do procedimento. Para avaliar tal nível de eficiência, os resultados obtidos foram analisados manualmente, antes e depois da etapa de enriquecimento, com o intuito de se verificar a quantidade real de tuplas duplicadas identificadas. O resultado de tal avaliação pode ser verificado na tabela 3 e resulta da seguinte razão: [pares duplicados reais] / [todos pares duplicados encontrados].

Tabela 3 – Análise de eficiência para o algoritmo *Edit Distance*.

	Duplicatas Encontradas	**Duplicatas Reais**	**Eficiência**
Ocupacao	151	105	69,5%
Ocupacao_E$	463	371	80,1%
Responsável_preenchimento	159	123	77,3%
Responsável_preenchimento_E$	225	179	79,5%

Para ilustrar os benefícios da pré-etapa de enriquecimento proposta, a tabela 4 exibe alguns exemplos de casos identificados após tal procedimento utilizando o algoritmo *Edit Distance*.

4.3 Testes utilizando o algoritmo *Soundex*

Para realização dos testes com o algoritmo *Soundex*, não foi necessário realizar nenhum tipo de configuração como a definição de *treshold* no algoritmo *Edit Distance*. Como já foi apresentado no capítulo 2, o algoritmo *Soundex* gera um código de quatro caracteres para cada *string* comparada no processo e depois compara os códigos para determinar se as *strings* são similares.

As figuras 13 e 14 ilustram os resultados obtidos ao se aplicar o algoritmo *Soundex* nas mesmas tabelas avaliadas na seção anterior. Ao analisar estas novas figuras, pode-se notar que a tabela “Responsavel_preenchimento” manteve o padrão avaliado na seção anterior, aumentando a quantidade de tuplas duplicadas identificadas. Porém, ao se analisar a figura 13, nota-se a diminuição de tuplas duplicadas identificadas na tabela “Ocupação” (coluna azul) após o enriquecimento dos dados.

Tabela 4 – Exemplos de tuplas duplicadas identificadas após o enriquecimento para o algoritmo *Edit Distance*.

Registro A	Registro B
NÃO INFORMADO	NAO INFORMADO
NÃO INFORMADO	(NÃO INFORMADO)
SOLDADOR	SOLDADORA
SOLDADOR	SOLDADOR
(NÃO INFORMADO)	NAO INFORMADO
AJUDANTE GERAL	AJ. GERAL
AJUDANTE GERAL	AUXILIAR GERAL
AJUDANTE GERAL	AUX. GERAL
AJUDANTE GERAL	ASSISTENTE GERAL
SERVIÇOS GERAIS	SERV. GERAIS
AJUDANTE DE MOTORISTA	AJ. MOTORISTA
AJUDANTE DE MOTORISTA	AUXILIAR DE MOTORISTA
AJUDANTE DE MOTORISTA	AUX. MOTORISTA
AJUDANTE DE MOTORISTA	AJUDANTE MOTORISTA
MOTOBOY	MOTOTAXI
MOTOBOY	MOTOBOY
MOTOBOY	MOTOCICLISTA
MOTOBOY	MOTOTAXI
MOTOBOY	MOTOTAXISTA
MOTOBOY	MOTOTAXISTA

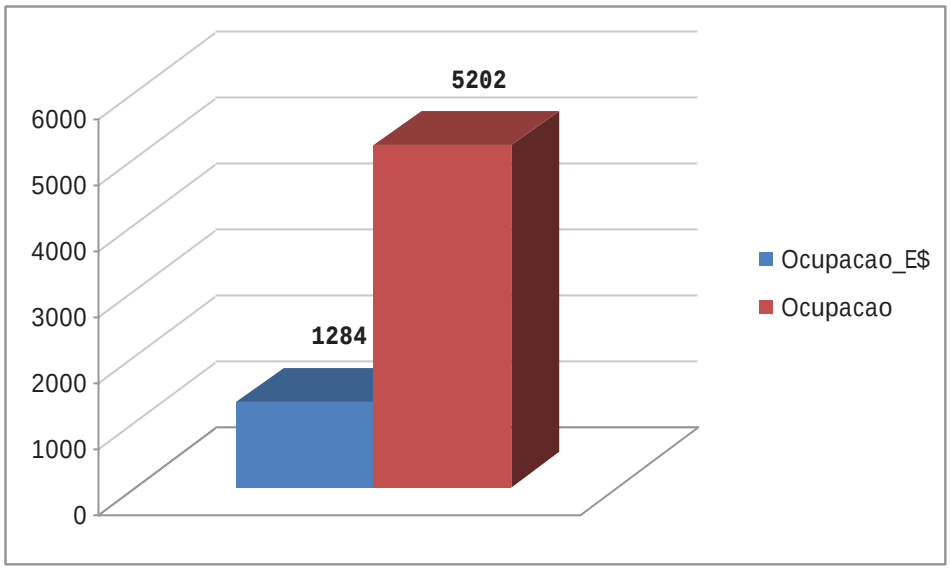

Figura 13 – Duplicatas identificadas na tabela “Ocupação” com o algoritmo *Soundex*.

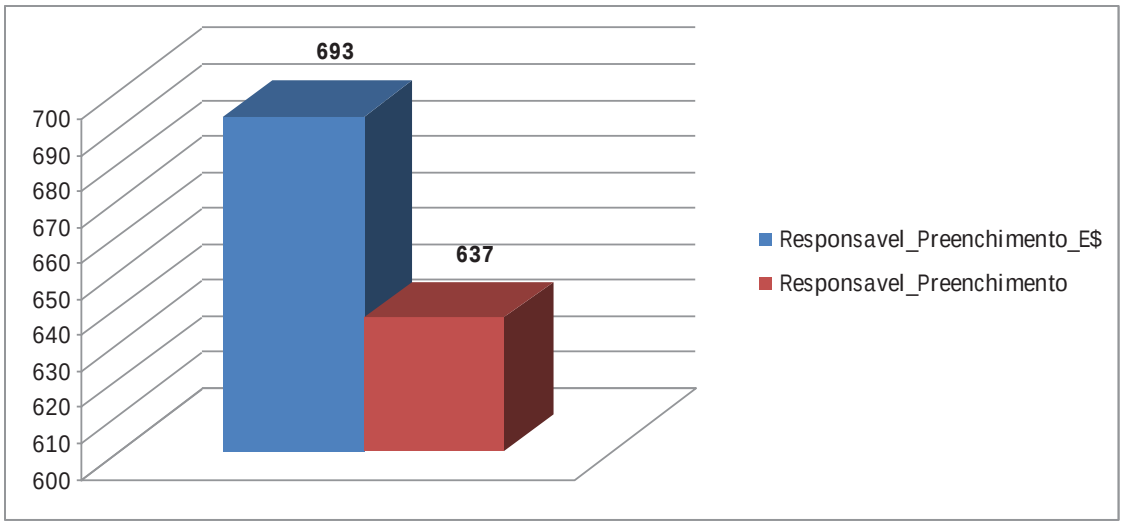

Figura 14 – Duplicatas identificadas na tabela "Responsavel_preenchimento" com o algoritmo *Soundex*.

Embora a quantidade de tuplas duplicadas identificadas na tabela "Ocupação" tenha diminuído, a quantidade de tuplas reais identificadas aumentou e, consequentemente, a eficiência do procedimento foi acrescida significativamente.

A tabela 5 exibe a eficiência calculada para cada uma das tabelas, antes e depois do procedimento de enriquecimento, utilizando a seguinte razão: [pares duplicados reais] / [todos pares duplicados encontrados].

Tabela 5 – Análise de eficiência para o algoritmo *Soundex*.

	Duplicatas Encontradas	**Duplicatas Reais**	**Eficiência**
Ocupação	5202	609	11,7%
Ocupacao_E$	1284	676	52,6%
Responsável_preenchimento	637	238	37,3%
Responsável_preenchimento_E$	693	289	41,7%

Para ilustrar os benefícios da pré-etapa de enriquecimento proposta, a tabela 6 exibe alguns exemplos de casos identificados após tal procedimento com o uso do algoritmo *Soundex*.

Tabela 6 – Exemplos de tuplas duplicadas identificadas após o enriquecimento para o algoritmo *Soundex*.

Registro A	Registro B
LÍVIA HELENA	LÍVIA HELENA
LÍVIA HELENA	MARLENE ELI MAIOLI
LÍVIA HELENA	Livia Helena
NAIARA GOMES GUIMORA	NAIARA GOMES GUIMARAES
NAIARA GOMES GUIMORA	MARIA GOMES GUIMARÃES
NAIARA GOMES GUIMORA	NAIARA GOMES GUIMARAES
NAIARA GOMES GUIMORA	MARIA GOMES GUIMARÃES
NAIARA GOMES GUIMORA	NAIARA GOMES GUIMARAES
NAIARA GOMES GUIMARAES	MARIA GOMES GUIMARÃES
NAIARA GOMES GUIMARAES	NAIARA GOMES GUIMARAES
NAIARA GOMES GUIMARAES	MARIA GOMES GUIMARÃES
NAIARA GOMES GUIMARAES	NAIARA GOMES GUIMARAES
SILENE JOSUE	SILENE JOSUÉ
SILENE JOSUE	CELIANE CUELLAR DE JESUS
SILENE JOSUE	CELIANE CUELHAR DE JESUS
SILENE JOSUE	CILIANI DE JESUS
SILENE JOSUE	SILENE JOSUÉ
FERNANDA NOGUEIRA	FERNANDO W.
FERNANDA NOGUEIRA	FERNANDO MANZOCH

4.4 Testes de performance

De forma antecipada, como já apresentado no capítulo 3, são previstos problemas de desempenho ao enriquecer bases de dados com muitos registros. Sendo assim, implementou-se dois esquemas de processamento paralelo com auxílio de *threads* para melhorar a utilização dos núcleos dos processadores e, consequentemente, aumentar a eficiência de processamento.

A primeira estratégia de processamento paralelo consiste na utilização de duas *threads* paralelas que executam o mesmo trabalho. Isso significa que cada uma das *threads* executa todos os submódulos exibidos na figura 2 para uma determinada “fatia” da tabela. Como são utilizadas duas *threads* em um processador de dois núcleos, cada *thread* fica responsável pelo enriquecimento de 50% dos registros da tabela escolhida. Cada uma das *threads* fica responsável por coletar dados da tabela, enriquecê-los e devolvê-los enriquecidos à base de dados.

A segunda estratégia de processamento paralelo consiste na utilização de duas *threads* em *pipeline*. Isso significa que cada uma das *threads* executa uma parte dos passos ilustrados na figura 2 para todos os registros da tabela. A primeira *thread* é

responsável por coletar os registros da tabela e realizar os primeiros passos de enriquecimento. Uma vez que essa primeira etapa esteja concluída, a primeira *thread* envia os dados para a segunda *thread* (relação de produtor/consumidor) para que esta possa realizar a segunda parte do enriquecimento. Dessa forma, enquanto a segunda *thread* termina de processar os dados em seu domínio, a primeira *thread* inicia o processamento de novos dados. O esquema citado usa a mesma ideia empregada no processamento de instruções realizado pelos processadores.

A divisão dos submódulos de enriquecimento dentre as *threads* foi feita com base estatística. Primeiramente, calculou-se a média de tempo, em milissegundos, que cada submódulo necessitava para processar uma única palavra. Com base nos tempos coletados e em testes com diversas configurações, optou-se por dividir os submódulos como exibido na tabela 7.

Tabela 7 – Aferição do tempo médio de processamento por palavra.

	Submódulos	**Tempo médio de processamento**
***Thread* 1**	Análise Léxica	7094.46
	Eliminação de Acentos	11677.93
	Padronização da Ortografia	31259.06
	Corretor Ortográfico	953359.04
***Thread* 2**	*Stopwords*	161060.10
	Tesauro	144648.61
	Stemming	5860.15

A captura dos dados da tabela, como mencionado no capítulo anterior, é feita por meio de pacotes contendo uma determinada quantidade de registros, que nesse caso foi definida como 100. A razão de o número 100 ter sido escolhido e fixado para a demonstração destes resultados deve-se ao fato de que o mesmo apresentou um melhor desempenho em todos os esquemas propostos de processamento e também pelo fato de as tabelas analisadas conterem uma quantidade pequena de registros.

Para aferição dos tempos de execução, foram realizadas três seções de testes para cada esquema de processamento (simples, *threads* paralelas e *threads* em *pipeline*) e, posteriormente, foi feito uma média desses tempos em segundos. Com o intuito de se obter uma unidade de medida mais facilmente compreensível e extensível para os

leitores deste trabalho, criou-se uma relação do tamanho da tabela testada com a média dos tempos auferidos e conseguiu-se descobrir a quantidade de KB/s processados por cada estratégia.

A tabela 8 exibe os resultados obtidos para cada esquema de processamento. A última coluna da tabela representa quantos KB/s cada esquema de processamento realiza.

Tabela 8 – Desempenho aferido para os esquemas de processamento propostos.

	Teste 1	**Teste 2**	**Teste 3**	**KB/s**
Processamento simples	490563	488125	484304	**0,85304636**
Threads paralelas	425088	433842	428232	**0,96957493**
Threads em *pipeline*	349292	355760	346846	**1,18642682**

Com base em uma breve análise dos resultados, pode-se notar que o esquema de processamento o qual utiliza *threads* em *pipeline* mostrou-se mais eficiente que as outras estratégias utilizadas, sendo possível processar cerca de 0,33338046 KB/s a mais que o esquema de processamento sem utilização de *threads* e 0,21685189 KB/s a mais que o esquema tradicional de processamento de *threads* paralelas. Isso significa um aumento de eficiência no processamento em cerca de 39,1% e 22,4% respectivamente.

4.5 Testes com dados definidos no idioma Inglês

Como já mencionado no capítulo anterior, o ambiente desenvolvido para representar a pré-etapa proposta em relação à identificação de tuplas duplicadas é projetado para funcionar com bases de dados definidas em qualquer idioma, em virtude do esquema de armazenamento genérico das regras de idioma em uma base de dados denominada “DCmanut”. Diante desse fato, criou-se uma tabela contento títulos de filmes no idioma inglês para testar a adequabilidade do ambiente para outros idiomas diferentes do português, que não demonstrou dificuldades na realização dos testes anteriores. Assim como para as tabelas testadas no idioma português, não se julgou necessário utilizar uma grande quantidade de registros, pois esta configuração só proporcionaria trabalho manual extra na atividade de verificação de eficiência e na criação de regras de idioma mais completas da língua inglesa (i.e. um dicionário).

A tabela gerada, denominada "*movies*", é composta por 30 títulos de filmes em inglês. Para que a realização dos testes pudesse acontecer com sucesso, inseriu-se mais 18 títulos com problemas pontuais e pré-considerados pelas regras de idioma definidas para o idioma inglês. Dentre essas regras encontram-se: dicionário, *stopwords*, regras de padronização e *stemming*. Para testar a eficiência do processo de identificação de tuplas duplicadas, utilizaram-se os algoritmos *Edit Distance* e *Soundex*, antes e depois do enriquecimento. Para o algoritmo Edit Distance, definiu-se o valor de *treshold* igual a dois como nos testes realizados nas seções anteriores para o idioma Português.

As tabelas 9 e 10 exibem os resultados obtidos para os testes propostos.

Tabela 9 – Análise de eficiência para o algoritmo *Edit Distance* com o idioma inglês

	Duplicatas Encontradas	**Duplicatas Reais**	**Eficiência**
movies	5	5	100%
movies_E$	13	13	100%

Tabela 10 – Análise de eficiência para o algoritmo *Soundex* com o idioma inglês

	Duplicatas Encontradas	**Duplicatas Reais**	**Eficiência**
movies	24	16	66%
movies_E$	44	30	68%

Em ambos os casos, embora a eficiência tenha se mantido com um valor bem próximo ou igual para os testes realizados antes e depois do enriquecimento, conseguiu-se um aumento expressivo de duplicatas reais encontradas. Este resultado mostra que, mesmo com o uso de bases de dados definidas em idiomas diferentes, o ambiente funciona como esperado e auxilia de maneira eficiente no processo de identificação de tuplas duplicadas.

A tabela 11 exibe alguns registros de filmes duplicados encontrados após o enriquecimento.

Tabela 11 – Exemplos de tuplas duplicadas identificadas após o enriquecimento para o algoritmo *Edit Distance.*

The Shashank Redenption	The Shawshank Redemption
GoodFather	The Godfather
Pulpi Fiction	Pulp Fiction
The God, the Bed and Ugly	The Good, the Bad and the Ugly
Schindler List	Schindler's List
Se7en samurai	Seven Samurai
Matrix	The Matrix
once upon time in west	Once Upon a Time in the West
silence of lambs	The Silence of the Lambs
usual suspects	The Usual Suspects
Seven	Se7en
Psycho	Psycho
it is a wonderfull life	It's a Wonderful Life

4.6 Considerações finais

Com base nos testes e resultados apresentados neste capítulo, pode-se considerar que a hipótese levantada no início do trabalho foi concluída. Com o desenvolvimento de uma pré-etapa em relação à etapa de limpeza de dados, mais precisamente em relação às estratégias de identificação de tuplas duplicadas, conseguiu-se maximizar a quantidade de tuplas duplicadas e/ou melhorar o nível de confiança para as duplicatas identificadas por algumas das estratégias propostas e implementadas atualmente no estado da arte.

Além das estratégias para o enriquecimento dos dados, que sempre foi o objetivo principal deste trabalho, também foram propostos dois esquemas de processamento extra, cuja ideia principal é tornar a utilização da pré-etapa mais atrativa. Com base nos resultados apresentados na seção anterior, nota-se que houve ganhos significativos ao se utilizar o esquema de processamento em *pipeline* em relação aos demais e que, com essa estratégia, é possível adicionar o processamento extra de enriquecimento ao processo de limpeza de dados de uma forma otimizada.

De um modo geral, diante de todos os resultados apresentados, pode-se concluir que a pré-etapa de enriquecimento de dados deveria ser incorporada ao processo de KDD, ou ao processo de limpeza de dados quando realizado fora do processo de KDD, pois mostrou-se eficiente no enriquecimento dos dados e, consequentemente, melhorou a qualidade dos resultados obtidos pelas estratégias de identificação de tuplas duplicadas consideradas.

Capítulo 5
Conclusão

5.1 Conclusão

Mediante o conteúdo exposto neste trabalho, nota-se que a grande variedade de problemas relacionados aos dados é comum nas bases de dados atuais. Conclui-se, ainda, sobre a fase inicial do processo de KDD, que baseia-se, principalmente, na limpeza dos dados, que é de fundamental importância para a consistência e confiabilidade desses dados nas futuras etapas do processo. A não realização desta etapa pode ocasionar descobertas de padrões não confiáveis, e, consequentemente, prejudicar o processo de tomadas de decisão das organizações que possuem os dados.

Dentre os possíveis problemas inerentes às bases de dados, destacou-se a identificação de tuplas duplicadas não idênticas como um dos principais problemas a serem tratados pela fase de limpeza de dados. Embora existam algumas técnicas para atuar na solução desses problemas, as denominadas *ad-hoc* – apresentadas em detalhes – mostraram-se as mais eficazes, pois conseguem lidar com grandes volumes de dados de forma mais eficiente que as outras técnicas, não necessitam de nenhum conhecimento prévio sobre os dados e nem de interação humana durante o processo.

Como pôde ser visto no estado da arte relacionado, as técnicas *ad-hoc* tendem a tratar o problema de identificação de tuplas duplicadas de forma genérica, de modo que possam atender a situações diferentes e também às bases de dados que contenham idiomas diferentes. Essa é uma estratégia interessante, pois permite a evolução dessa área de pesquisa como um todo e não especializa as técnicas somente para uma única região do globo. Porém sabe-se que cada idioma possui suas características e

particularidades especiais que, se forem levadas em consideração, podem melhorar os resultados obtidos no processo de identificação de tuplas duplicadas.

A proposição de técnicas *ad-hoc* especializadas poderia caracterizar um retrocesso no processo pormenorizado. Neste caso, percebeu-se a necessidade de adicionar uma pré-etapa em relação ao processo de identificação de tuplas duplicadas cuja proposta é enriquecer o texto dos registros das bases de dados, de acordo com o idioma específico trabalhado e, com isso, maximizar os resultados obtidos pelas técnicas genéricas de identificação de tuplas duplicadas *ad-hoc*.

A tabela 12 exibe a diferença entre a pré-etapa proposta e as estratégias abordadas pelos trabalhos correlatos.

Tabela 12 – Pré-etapa proposta versus ambientes de mesmo propósito.

	Febrl	**TAILOR**	**WHIRL**	***WinPure Clean & Match***	***Big Match***	**Pré-etapa Proposta**
Eliminação de acentos	**X**					**X**
Análise Léxica	**X**					**X**
Stemming						**X**
Tesauros						**X**
Corretor Ortográfico						**X**
Padronização de Erros						**X**
Eliminação de *Stopwords*						**X**
Técnicas *ad-hoc*	**X**	**X**	**X**	**X**	**X**	**X**

Com base nos resultados apresentados no capítulo anterior, pode-se formalizar a importância da atividade de enriquecimento dos dados que ocorre antes da aplicação das técnicas de identificação de tuplas duplicadas. Quando esta primeira etapa foi aplicada às amostras de dados consideradas, conseguiu-se, em todos os casos, aumentar o

número de tuplas duplicadas e/ou aumentar o nível de confiança na determinação de pares realmente duplicados. Além dos números positivos a favor do processo de identificação de duplicatas, também foi possível tornar a utilização da pré-etapa proposta mais atrativa por meio do processamento de dados em *pipeline* com o uso de programação *multithreading*. Este esquema de processamento mostrou-se 39,1% mais eficiente que o modelo de processamento de dados trivial.

De um modo geral, a pré-etapa para enriquecimento de dados mostrou-se útil ao ser adicionada ao processo de identificação de tuplas duplicadas. Embora, essa tarefa adicional aumente o custo de processamento do processo como um todo, foi apresentado um esquema de codificação mais eficiente com o intuito de torná-la mais atrativa. Além disso, os números obtidos nos resultados também mostram que sua inclusão pode compensar o processamento extra com a qualidade final dos resultados. Sendo assim, acredita-se que tal procedimento deve ser considerado para integrar o processo de identificação de tuplas duplicadas de forma definitiva a fim de se alcançar uma maior integridade das bases de dados e também resultados mais confiáveis ao final da aplicação do processo de KDD.

5.2 Sugestões para trabalhos futuros

Com o intuito de aperfeiçoar a atividade de enriquecimento realizada pela pré-etapa sugerida, faz-se necessário dar continuidade aos trabalhos em relação aos submódulos de enriquecimento propostos. Diante deste fato, são recomendadas a seguir, algumas ideias que podem melhorar ainda mais os resultados obtidos no processo de identificação de tuplas duplicadas.

1) Adição de novos submódulos de enriquecimento que possam ser úteis para a aproximação ortográfica dos registros encontrados nos volumes de dados analisados;
2) Aperfeiçoamento dos submódulos já existentes com o intuito de melhorar o nível de enriquecimentos dos registros;
3) Incremento dos conjuntos de regras para idiomas diversos.
4) Adaptação do ambiente para outras atividades, como integração de dados e análise de dados genéticos.

Referências Bibliográficas

BILENKO, Mikhail et al. **Adaptative Name Matching in Information Integration.** IEEE Intelligent Systems, p. 16-23. out. 2003.

BREIMAN, Leo et al. **Classification and Regression trees.** CRC Press, Jul. 1984.

CHAUDHURI, Surajit; GANTI, Venkatesh; MOTWANI, Rajeev. **Robust Identification of Fuzzy Duplicates.** Proc. 21st IEEE Int, p. 865-876. 2005.

CHRISTEN, Peter; CHURCHES, Tim; HEGLAND, Markus. **A Parallel Open Source Data Linkage System.** Proceedings Of The 8th Pacific-asia Conference On Knowledge Discovery And Data Mining (PAKDD '04), Sydnei, maio 2004.

COCHINWALA, Munir et al. **Efficient Data Reconciliation.** Information Sciences, p. 1-15. set. 2001.

COHN, David; LADNER, Richard; WAIBEL, Alex. **Improving Generalization with Active Learning.** Machine Learning, p. 201-221. 1994.

COHEN, William W.. **Integration of Heterogenous Databases without Common Domains Using Queries Based on Textual Similarity.** Proceedings Of The 1998 Acm Sigmod International Conference On Management Of Data (SIGMOD '98), p. 201-212. jun. 1998.

DASU, Tamraparni; JOHNSON, Theodore. **Exploratory Data Mining and Data Cleaning.** New Jersey: John Willey & Sons, 2003. 203 p.

DEY, Debabrata; SARKAR, Sumit; DE, Prabuddha**. Entity Matching in Heterogeneous Databases: A Distance Based Decision Model.** Thirty-first Annual Hawaii International Conference On System Sciences, Kohala Coast, p. 305-313. 1998.

DZEROSKI, S. **Multi-Relational Data Mining: An Introduction.** ACM SIGKDD Explorations Newsletter, v. 5, n. 1, p.1-16, Jul. 2003.

ECKERSON, Wayne W.. **Data quality and the bottom line: Achieving business success through a commitment to high quality data.** Chatsworth: The Data Warehousing Institute, 2002. 36 p.

ELFEKY, Mohamed; VERYKIOS, Vassilios; ELMAGARMID, Ahmed**. TAILOR: A Record Linkage Toolbox.** Proceeding ICDE, p. 17-28. 2002.

ELMAGARMID, Ahmed K.; IPEIROTIS, Panagiotis G.; VERYKIOS, Vassilios S. **Duplicate Record Detection: A Suvey.** IEEE Transactions On Knowledge And Data Engineering, Los Angeles, p. 1-16. jan. 2007.

FAYYAD, Usama; PIATETSKY-SHAPIRO, Gregory; SMYTH, Padhraic. **From Data Mining to Discovery Knowledge in Databases.** AI Magazine, p.37-54, 1996.

FAYYAD, Usama; PIATETSKY-SHAPIRO, Gregory; SMYTH, Padhraic. **The KDD Process for Extracting Useful Knowledge from Volumes of Data.** Communications Of The ACM p. 27-34. nov. 1996.

FAN, Wenfei; GEERTS, Floris; JIA, Xibei. **A Revival of Integrity Constraints for Data Cleaning.** VLDB ‘08, Auckland, 24 ago. 2008.

FELLEGI, Ivan P.; SUNTER, Alan B.. **A Theory for Record linkage.** Journal Of The American Statistical Association, p. 1183-1210. 11 dez. 1969.

FRAKES, William Bill; BAEZA-YATES, Ricardo. **Information Retrieval: Data Structures & Algorithms.** Englewood Cliffs, Nj, Usa: Prentice Hall, 1992.

GÁLVEZ, Carmen**. Identificación de Nombres Personales por Medio de Sistemas de Codificación Fonética.** Encontros Bibli: Revista Eletrônica de Biblioteconomia e Ciência da Informação – Universidade Federal de Santa Catarina – UFSC, n. 22, p. 105-116, 2006.

GILL, Leicester E.. **OX-LINK: The Oxford Medical Record Linkage System.** Proc. Int'l Record Linkage Workshop and Exposition, p. 15-33. 1997.

GRAVANO, Luis et al. **Texts Joins in an RDBMS for Web Data Integration.** World Wide Web Conference, Budapeste, p. 90-101. 20 maio 2003.

GRAVANO, Luis et al. **Approximate String Joins in a Database (Almost) for Free.** Proceedings Of The 27th International Conference On Very Large Data Bases (VLDB'01), p. 491-500. 2001.

GUHA, Sudipto; RASTOJI, Rajeev; SHIM, Kyuseok. **ROCK: A Robust Clustering Algorithm for Categorical Attributes.** In Proc. 1999 Int. Conf. Data Engineering, Sidney, p.512-521, mar. 1991.

GUHA, Sudipto et al. **Merging the Results of Approximate Match Operations.** Proceedings Of The 30th VLDB Conference, Toronto, p. 636-647. 2004.

HAN, Jiawei; KAMBER, Micheline. **Data Mining: Concepts and Techniques.** 2. ed. San Francisco: Elsevier, 2006. 743 p.

HE, Zengyou; XU, Xiaofei; DENG, Shengchun. **Clustering Mixed Numeric and Categorical Data: A Cluster Ensemble Approach.** Disponível em: <http://arxiv.org/abs/cs/0509011>. Acesso em: 13 maio 2010.

HE, Zengyou; XU, Xiaofe I; DENG, Shengchun. **Squeezer: An Efficient Algorithm for Clustering Categorical Data.** Jornal Of Computer Science And Technology, v. 17, n. 5, p.611-625, 2002.

JARO, M. A.. Unimatch**: A record Linkage System: User's Manual.** Washington, D.C.: Us Bureau Of The Census, 1976.

JOACHIMS, Thorsten. **Making large-Scale SVM Learning Practical: Advances in Kernel Methods - Support Vector Learning.** B. Schölkopf and C. Burges and A. Smola, MIT-Press, 1999.

LEVENSHTEIN, Vladimir I. **Binary Codes Capable of Correcting Deletions, Insertions and Reversals.** Soviet Physics Doklady, p. 707-710. fev. 1966.

MITRA, Sushmita; ACHARYA, Tinku. **Data Mining: Multimedia, Soft Computing and Bioinformatics.** Hoboken: John Wiley & Sons, Inc., 2003. 401 p.

MONGE, Alvaro; ELKAN, Charles. **The field matching problem: Algorithms and Applications.** In Proceedings Of The Second International Conference On Knowledge Discovery And Data Mining, p. 267-270. 1996

NEEDLEMAN, Saul B.; WUNSCH, Christian D.. **A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins.** Journal Of Molecular Biology, p. 443-453. 28 mar. 1970.

NEWCOMBE, Howard B.. **Record Linking: The Design of Efficient Systems for Linking Records into Individual and Family Histories.** American Journal Of Human Genetics, p. 335-359. maio 1967.

ORENGO, Viviane Moreira; HUYCK, Christian. A Stemming Algorithm for the Portuguese Language. **8th International Symposium On String Processing And Information Retrieval (SPIRE)**, Laguna de San Raphael, Chile. p.183-193, 2001.

PHILIPS, Lawrence**. The Double Metaphone Search Algorithm.** C/c++ Users Journal, p. 38-43. jun. 2000.

PHILIPS, L.. **Hanging on the Metaphone.** Computer Language Magazine, v. 7, n. 12, p.39-44, dez. 1990.

PIATETSKY-SHAPIRO, Gregory. **KDnuggets:** Data Mining Community's Top Resource . Disponível em: <http://www.kdnuggets.com/>. Acesso em: 25 out. 2011.

PIATETSKY-SHAPIRO, Gregory. **Knowledge Discovery in Real Databases: A Report on the IJCAI-89 Workshop.** AI Magazine, v. 11, n. 5, p.68-70, 1990.

PORTER, M. F.. **An algorithm for suffix stripping**. Program, Londres, v. 14, n. 3, p.130-137, 1980.

RAHM, Erhard; DO, Hong Hai. **Data Cleaning: Problems and Current Approaches.** IEEE Data Engineering Bulletin, p. 3-13. dez. 2000.SARAWAGI, Sunita;

BHAMIDIPATY, Anuradha. **Interactive Deduplication Using Active Learing.** Proc. Eighth ACM SIGKDD International Conference: Knowledge Discovery and Data Mining (KDD '02), p. 269-278. 2002.

SHAHRI, Hamid Haidarian; SHARHRI, Saied Haidarian. **Eliminating Duplicate in Information Integration: An Adaptive, Extensible Framework.** IEEE Intelligent Systems, v. 21, n. 5, p.63-71, set/out. 2006.

SMITH, T. F.; WATERMAN, M. S.. **Identification of Common Molecular Subsequences.** Journal Of Molecular Biology, p. 195-197. 25 mar. 1981.

STALLINGS, William. **Arquitetura e Organização de Computadores.** 5. ed. São Paulo: Prentice Hall, 2002. 786 p.

TAFT, R. L.. **Name Search Techniques.** New York State Identification And Intelligence System. Albany, Nova York. 1970.

UKKONEN, E.. **Approximate String Matching with q-Grams and Maximal Matches.** Theoretical Computer Science, p. 191-211. 1992.

ULLMANN, J. R.. **A Binary n-Gram Technique for Automatic Correction of Substitution, Deletion, Insertion and Reversal Errors in Words.** The Computer Journal, p. 141-147. 1977.

WATERMAN, M. S.; SMITH, T. F.; A.BEYER, W.. **Some biological sequence metrics.** Advances In Mathematics, p. 367-387. jun. 1976.

WINKLER, W. E.; THIBAUDEAU, Y.. **An Application of the Fellegi-Sunter Model of Record Linkage to the 1990 US Decennial Census.** Washington, D.C.: Us Bureau Of The Census, 1991.

YAN, Sun et al. Adptative **Sorted Neighborhood Methods for Efficient Record Linkage.** JCDL'2007, Vancouver, p.17-22, jun. 2007.

YANCEY, William E.. Bigmatch: **A Program for Extracting Probable Matches from a Large File for Record Linkage.** U.S. Bureau Of The Census, Washington D.C. 2002.

YATES, Ricardo Baeza; RIBEIRO NETO, Berthier. **Modern Information Retrieval.** Harlow: Addison Wesley, 1999.

Autorizo a reprodução xerográfica para fins de pesquisa.

São José do Rio Preto, ____/____/____

_____
Assinatura