UNIVERSIDADE FEDERAL DE SANTA CATARINA

IMPLEMENTAÇÃO DE OTIMIZAÇÕES EM UMA TÉCNICA DE MATCHING POR SIMILARIDADE PARA WEB FORMS

FILIPE ROBERTO SILVA

Florianópolis - SC

2012 / 1

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

IMPLEMENTAÇÃO DE OTIMIZAÇÕES EM UMA TÉCNICA DE MATCHING POR SIMILARIDADE PARA WEB FORMS

Filipe Roberto Silva

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação.

Florianópolis – SC

2012 / 1

Filipe Roberto Silva

IMPLEMENTAÇÃO DE OTIMIZAÇÕES EM UMA TÉCNICA DE MATCHING POR SIMILARIDADE PARA WEB FORMS

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Ronaldo dos Santos Mello, Dr.

Banca examinadora

Profa. Carina Friedrich Dorneles

Prof. José Leomar Todesco

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

A Internet hoje possui uma grande quantidade de sites gerados dinamicamente, ou seja, a partir de dados fornecidos pelo usuário, são feitas consultas em bancos de dados e geradas páginas de resultado baseadas nos dados de entrada. É a chamada *Deep web*, ou *Hidden-Web* ou ainda Web oculta, que representa boa parte da Internet em termos de volume de dados. A *Deep web* é formada por bancos de dados "escondidos", acessíveis somente através de consultas definidas a partir de formulários web. Para tanto, formulários adequados devem ser disponibilizados ao usuário para que ele defina as suas buscas. Este trabalho está inserido neste tema e faz parte de um projeto de pesquisa, com financiamento do CNPq, denominado *WF-Sim*, que visa o desenvolvimento de um sistema de busca por similaridade para formulários Web. O núcleo deste sistema é uma abordagem de *matching* (casamento) de formulários similares. O foco deste trabalho está no desenvolvimento de otimizações para esta abordagem de *matching*, bem como experimentos que avaliam a qualidade deste *matching*. Para tanto, algumas modificações foram feitas no sistema já existente para que fosse possível identificar formulários semelhantes entre si de forma eficiente e eficaz.

Palavras Chave: *Deep web*, *Web forms*, *Matching*

1. Introdução

Quando a Internet surgiu, as páginas *web* eram criadas de forma estática. É a chamada *Web* 1.0 [1] onde as páginas servem somente para leitura e a adição de novos conteúdos é feita pelo programador. Esse modelo evoluiu, e com a integração de *scripts*, bancos de dados e outras linguagens de programação, a *web* passou para a forma atual de interação, onde o próprio usuário pode adicionar conteúdos. Essa é a chamada *Web* 2.0, onde os dados adicionados pelo usuário são armazenados em arquivos no servidor ou em bancos de dados, e esses dados são utilizados para gerar páginas de forma dinâmica.

Essa parte da Internet que é gerada dinamicamente, segundo o artigo da Bright Planet (Bergman 2001), é chamada de *Deep web* ou web oculta. Pelo fato de boa parte do conteúdo estar armazenado em Bancos de Dados, a busca e indexação desses dados por máquinas de busca é dificultada, pois o conteúdo é acessível somente através de formulários web (*Web forms*). Assim, ferramentas de busca comuns não tem acesso a esses dados, e esse é um problema em aberto, ou seja, como indexar e recuperar os dados da *Deep web*.

Uma forma de indexar e recuperar esse conteúdo da *Deep web* é indexando formulários. Com isso, é possível encontrar vários formulários que retornem conteúdos semelhantes e assim o usuário poderia realizar suas buscas sobre os *web forms* encontrados. Um desafio nessa abordagem é encontrar similaridades em formulários para poder indexá-los de acordo com suas características semelhantes.

[1] http://pt.wikipedia.org/wiki/World_Wide_Web

Este trabalho é uma continuação dos resultados obtidos com o projeto WF-Sim (Gonçalves et. al. 2011), projeto este que está em andamento e visa tornar possíveis as buscas por similaridade para *web forms* através de uma máquina de busca. Atualmente, o sistema já permite o agrupamento de formulários conforme a semelhança, indexação de dados de formulários e, a partir de formulários de entrada, encontrar outros semelhantes. Foram realizados testes sobre o sistema WF-Sim e o mesmo já tem apresentado resultados satisfatórios de qualidade da busca para alguns domínios. Porém, existem melhorias a serem feitas, como a adição de persistências nas clusterizações e melhorar a técnica de agrupamento. Este trabalho visa implementar algumas dessas melhorias.

O objetivo desse trabalho é realizar melhorias no algoritmo de clusterização de dados de formulários utilizado pelo WF-Sim, bem como a persistência desses *clusters*. O WF-Sim, como foi implementado, realiza clusterizações de *web forms* e somente as guarda em memória, não as persistindo e, por isso, realiza agrupamentos dos formulários a cada vez que é executado. Neste trabalho, foram implementadas duas soluções para a persistência desses dados: uma em Bancos de Dados e outra em arquivos. Cada alternativa foi testada visando verificar a melhor solução para o problema.

O WF-Sim já possuía um algoritmo de clusterização, o Median Shift. Para fins de comparação, foi implementado e integrado ao WF-Sim o K-Means (MacQueen 1967) que é um algoritmo de clusterização popular na literatura e considerado eficiente. Foram realizados testes para avaliar a qualidade dos dois algoritmos. Esta é outra contribuição deste trabalho.

Os demais capítulos detalham trabalho. O Capítulo 2 apresenta a fundamentação teórica sobre *Deep web* e como são estruturados os *web forms*. O Capítulo 3 descreve o projeto WF-Sim. O Capítulo 4 mostra como foram feitas as persistências dos *clusters*. O Capítulo 5 fala sobre a implementação do K-Means. No Capítulo 6 são apresentados os experimentos diversos que foram realizados neste trabalho e, por fim, o Capítulo 7 apresenta a conclusão e trabalhos futuros.

2. *Deep web*

A Internet está presente na vida de muitas pessoas. Qualquer dúvida, pesquisa, notícia ou outra necessidade de informação pode ser buscada na Internet através das populares máquinas de busca, como o Google ou o Bing. Porém, o que poucos sabem é que essas buscas atingem apenas a superfície da web, ou seja, ignoram uma grande quantidade de dados que se encontra escondido.

Michael K. Bergman em seu artigo para a Bright Planet (Bergman 2001) faz uma analogia da web atual com um oceano, conforme mostra a Figura 1. Nesse caso, máquinas de busca tradicionais conseguem "pescar" somente os "peixes" da superfície (que indicam os dados visíveis nas páginas *web*), porém existem muito mais "peixes" nas profundezas do oceano. A esses "peixes" (ou conteúdos) “nas profundezas” é dado o nome de *Deep web* ou *web* oculta (Halevy 2009).

Com essa informação, alguém pode se perguntar onde se encontra esse conteúdo e por que ele não pode ser acessado facilmente. Para responder essas questões é interessante entender o que são páginas dinâmicas e estáticas.

Páginas estáticas, como o nome já diz, são páginas que não mudam. Elas podem ser criadas diretamente em HTML e colocadas na Web. Por isso, são facilmente indexadas por buscadores, pois elas estão sempre visíveis na web.

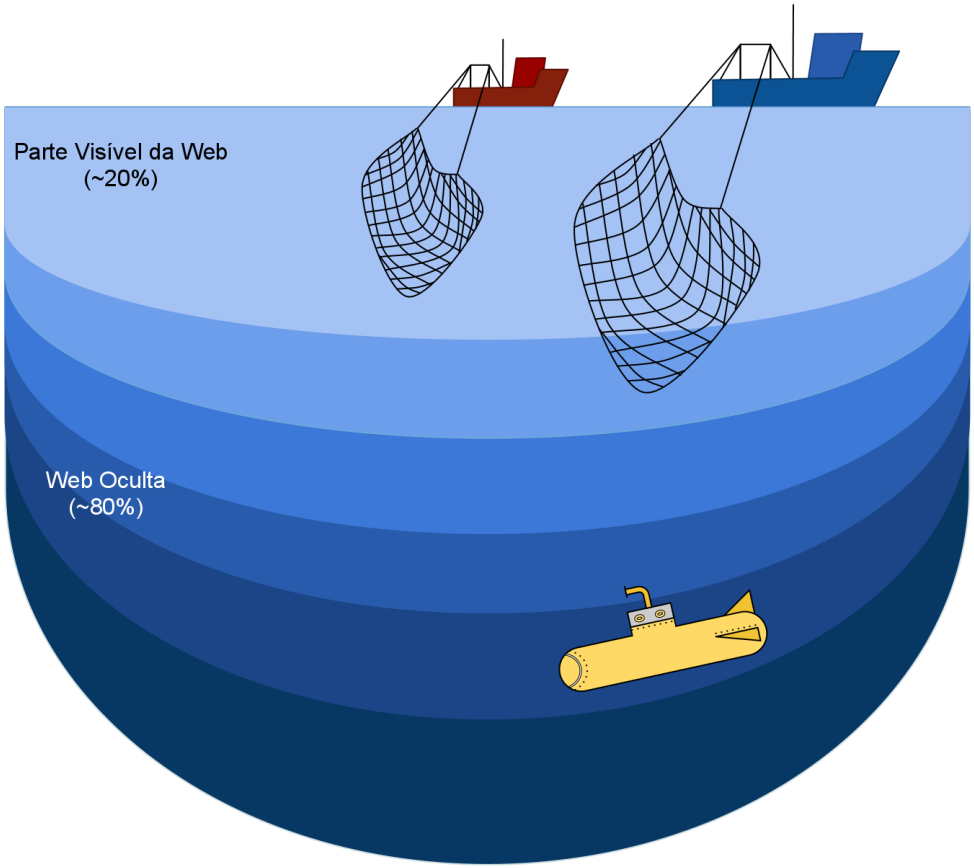

Figura 1 - Representação da *Deep web* (Adaptado de JUANICÓ, 2006).

Páginas dinâmicas, por sua vez, não existem permanentemente na web. Elas são geradas dinamicamente conforme requisições do usuário, ou seja, dependendo da interação do usuário uma nova página é gerada. Com isso, estas páginas não podem ser indexadas por buscadores comuns, já que elas não estão na web até que o usuário peça para gerá-las. Assim, conclui-se que as páginas dinâmicas fazem parte da *Deep web*.

O conteúdo a ser mostrado nessas páginas, por sua vez, pode vir de diferentes fontes, dentre elas estão os bancos de dados. As interações com os bancos de dados

podem ser de diversas formas, porém neste trabalho, serão focados os formulários *web*. Neste contexto, o usuário entra com alguns dados no formulário e, a seguir, é feita uma busca no banco de dados para retornar o conteúdo desejado. Esse conteúdo é formatado numa página web para ser exibido de forma clara ao usuário. Assim, para um buscador indexar essas páginas, ele teria que fazer o preenchimento dos *web forms* com todas as combinações possíveis de dados dos atributos dos formulários e salvar seus resultados, o que seria um trabalho muito custoso.

Outra abordagem é indexar os *web forms* que estão em páginas estáticas e assim, permitir que o usuário preencha um formulário com os dados desejados, a seguir, poder-se-ia propagar essa pesquisa pelos formulários indexados e isso retornaria os resultados. O grande desafio dessa abordagem é saber para quais formulários propagar essa pesquisa, ou seja, descobrir quais formulários são relevantes para a pesquisa do usuário e como reutilizar os mesmos dados de entrada em *web forms* com estruturas heterogêneas.

Outra abordagem possível de busca na *Deep web*, que é o foco deste trabalho, é pesquisar somente por campos de formulários, porém indexando os campos destes formulários e permitindo a busca por outros formulários similares, ou seja, considerando um processo de *matching* de *web forms*. Para entender melhor esse processo, é interessante entender melhor como é formado um *web form*.

Um *web form* possui diversos campos (campos de texto, *checkboxes*, *comboboxes* e etc.), que permitem ao usuário o seu preenchimento através da digitação ou seleção de um valor.

Como é possível observar na Figura 2, os formulários web possuem rótulos, que são utilizados para que o usuário saiba o que colocar em cada campo, e por isso é importante que esse rótulo seja entendível por seres humanos. Esses campos também podem conter os já citados valores para seleção, como é possível ver na Figura 2 para o campo ‘Select what you have:’, que possui duas possibilidades de preenchimento: ‘I have a bike’ e ‘I have a car’.

Assim, esses valores podem ser utilizados para a comparação de um *web form* com outro, tanto em termos do conteúdo do rótulo quanto dos valores para seleção.

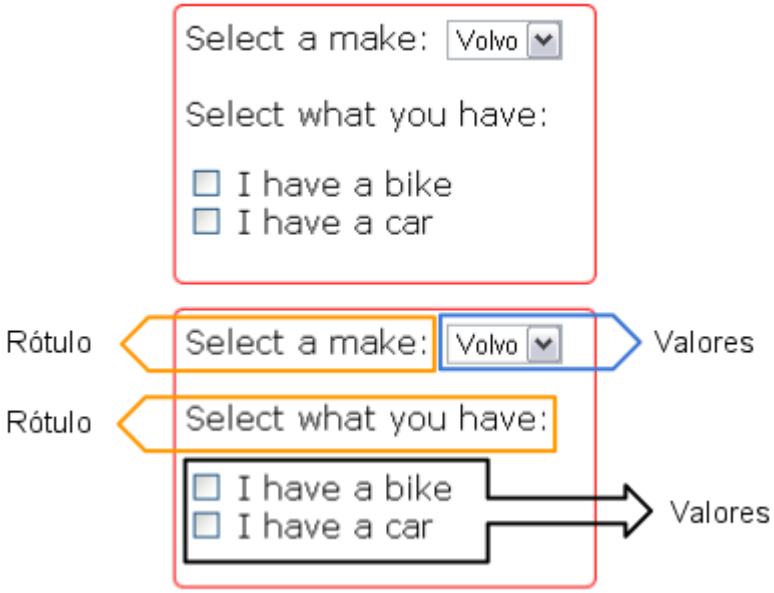

Figura 2 - Formulário web e seus elementos

3. WF-SIM

Encontrar similaridades entre formulários a partir de *strings* que representam valores em um campo é um grande desafio, já que muitas vezes duas palavras totalmente diferentes significam a mesma coisa, ou ainda duas palavras iguais em contextos diferentes possuem significados diferentes. Um exemplo disso pode ser visto na Figura 3. O formulário da esquerda possui os campos 'Manufactures' e 'Year of Manufacture'. Apesar dos nomes semelhantes, eles possuem significados totalmente diferentes. Os campos à direita, por sua vez, apesar de rótulos diferentes, significam a mesma coisa. Com isso, é difícil ensinar a um sistema informatizado quando dois elementos em formulários são semelhantes ou não.

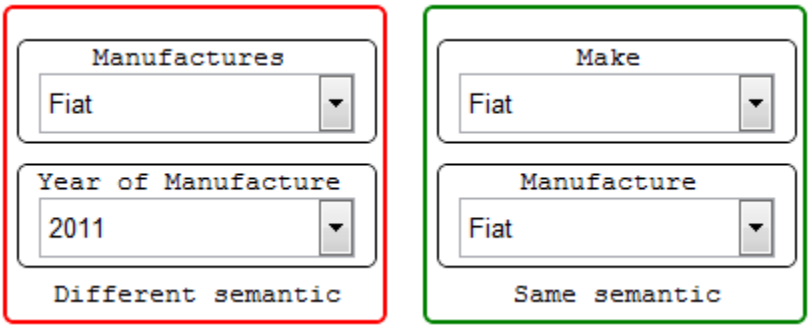

Figura 3 - Rótulos e suas semânticas

O trabalho em questão está inserido no projeto WF-Sim (*Web form Similarity*), um projeto desenvolvido na UFSC pelo grupo de BD, com financiamento do CNPq, que visa realizar buscas por formulários similares, tentando resolver parte desta problemática. As seções a seguir detalham o WF-Sim.

3.1 Funcionamento

O WF-Sim trata os *web forms* dividindo-os em elementos. Cada elemento representa um campo contendo um rótulo e opcionalmente um conjunto de valores. Essa divisão é representada pela Figura 4.

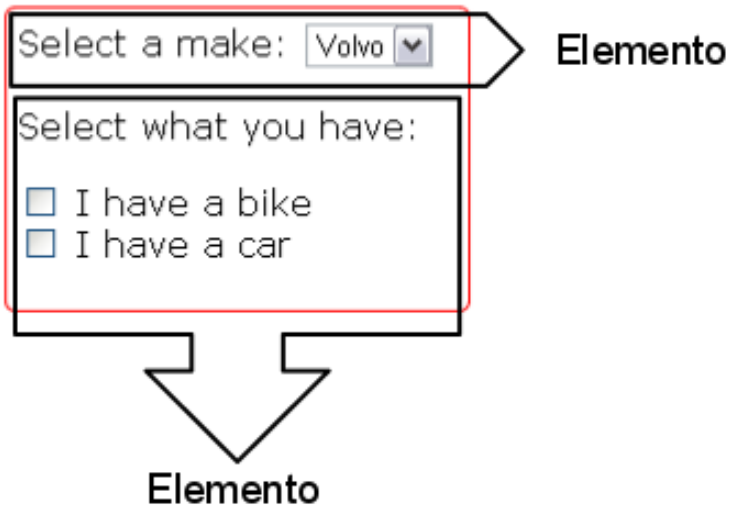

Figura 4 - Divisão de um formulário em elementos

O projeto visa desenvolver um sistema de busca por similaridade para *web forms.* Esse sistema, também chamado WF-Sim, é composto por três componentes principais ilustrados na Figura 5 (Gonçalves et. al. 2011). Estes componentes são: clusterização, indexação e busca. Neste sistema, os elementos são clusterizados de acordo com uma métrica de similaridade e indexados. A partir dessa indexação é possível realizar buscas, ou seja, a partir de um *web form* de entrada são encontrados os *web forms* semelhantes. Essa busca, ao invés de comparar elemento por elemento de *web form*, é feita buscando somente os centroides, que são elementos centrais de *clusters* que

representam um conjunto de elementos similares. Assim, a partir das comparações com os centroides, encontra-se os grupos de elementos semelhantes àquele *web form* de entrada.

Para entender melhor essas etapas é necessário entender como é calculada a similaridade entre os elementos.

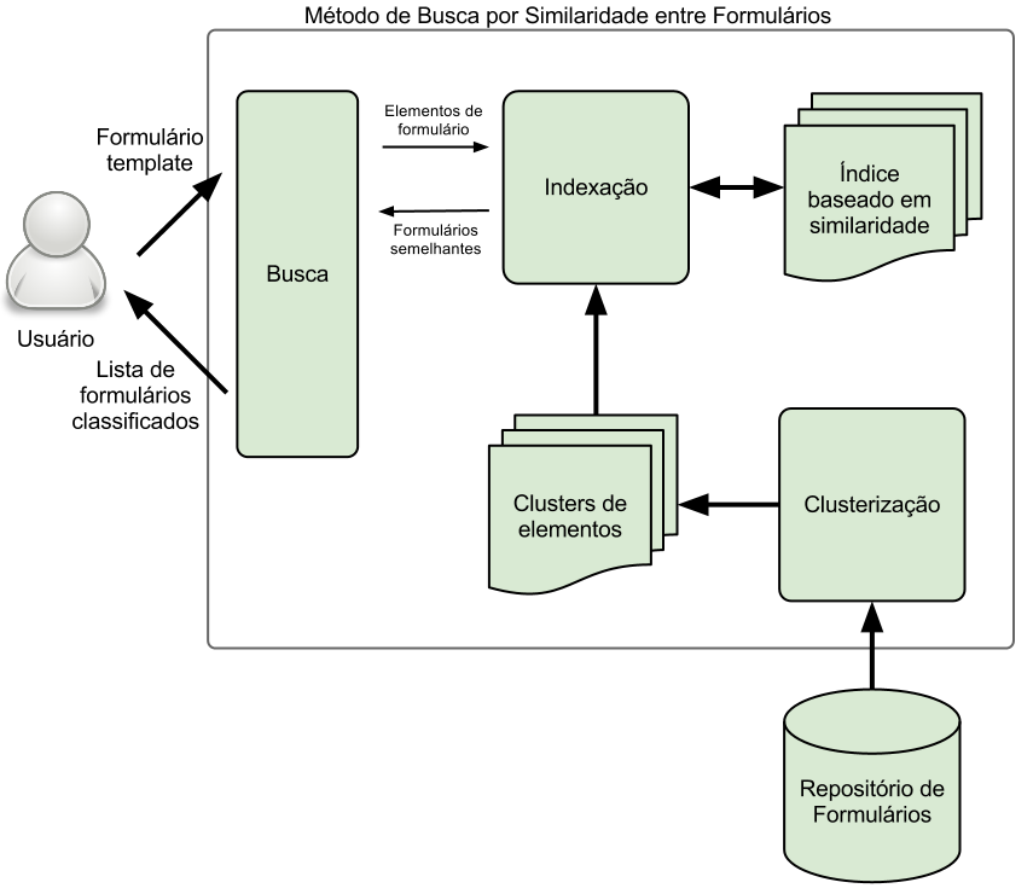

Figura 5 - Principais componentes do WF-Sim

3.2 Cálculo da Similaridade

Como visto anteriormente, os elementos possuem valores e rótulos. Esses atributos podem ser do tipo numérico ou *string*. A comparação na forma "valor igual a valor" não é adequada, pois como citado anteriormente, podem existir *strings* parecidas com significados diferentes e *strings* diferentes com significados iguais. Com isso é necessário utilizar um algoritmo de tratamento de textos em linguagem natural.

O WF-Sim utiliza, para o cálculo de similaridade dos rótulos, a métrica *TF-IDF* (Cohen, Ravikumar & Fienberg 2003) e para a similaridade dos valores, a métrica *SubSetSim* (Dorneles 2004), ambas escolhidas por possuírem os melhores resultados se comparados a outras métricas.

A partir do cálculo das similaridades dos rótulos e dos valores separadamente, é feita uma média ponderada desses valores, como é possível ver na Equação 1. Nesta equação, *labelSim* e *valuesSim* são os valores de similaridade de rótulos e valores respectivamente. Por ser uma média ponderada, é possível aplicar pesos diferentes para cada atributo do elemento, onde a soma dos pesos é igual a (1). Esses pesos são definidos manualmente no sistema pelo usuário e estão representados na equação pelas variáveis *labelWeight* e *valuesWeight*. Feito esse cálculo é obtida a similaridade entre os elementos.

$$ElementSim = labelSim(l_1, l_2) * labelWeight + valuesSim(vl_1, vl_2) * valuesWeight \quad (1)$$

3.3 Clusterização

A clusterização dos elementos é feita utilizando-se uma extensão do algoritmo Median Shift (Jouili, Tabbone & Lacroix 2010), onde os elementos medianos são definidos como os centroides dos *clusters*. Isso é feito da seguinte forma:

Primeiramente, é determinado pelo usuário um raio de clusterização. Assim, é verificado, em cada elemento, quais outros elementos estão contidos dentro desse raio de distância. Esse processo é chamado de cálculo de vizinhança.

Pelo fato dos elementos não estarem contidos em um espaço vetorial, é necessário calcular todas as similaridades entre os elementos e transformar esses valores em distâncias. Como os valores de similaridade estão no intervalo de ‘0’ a ‘1’, sendo que o valor ‘0’ significa ser diferente e o valor ‘1’ ser igual, a distância é calculada utilizando a Equação 2, sendo que para valores de similaridade igual a ‘1’ é aplicada diretamente a distância zero.

$$Distância = \frac{1}{Similaridade} \tag{2}$$

Com isso, são calculadas as distâncias entre os elementos, essas distâncias são armazenadas em uma estrutura de dados em memória e o algoritmo segue os seguintes passos:

1. Para cada elemento é criado um grupo contendo todos outros elementos cuja distância daquele é menor que o raio, aquele elemento central é considerado um possível elemento mediano;
2. As distâncias entre os elementos e o elemento mediano são considerados pontos em um espaço vetorial e é calculada a mediana desses pontos;

3. Pelo fato de não ser possível converter uma coordenada em um elemento de formulário, o elemento cuja distância é mais próxima da mediana é considerado o novo elemento mediano;
4. Volta-se ao passo 2 até que o elemento mediano retornado permaneça o mesmo.
5. No fim desse processo, o elemento mediano será considerado o centroide do grupo. Lembrando que neste algoritmo é calculado um *cluster* por vez, assim esse processo se repete do passo 1 ao 5 para cada elemento.

O algoritmo possui um valor de precisão definido pelo usuário que dita o quão preciso será o cálculo do centroide. Maiores detalhes sobre o procedimento de clusterização são apresentados em (Gonçalves et. al. 2011).

3.4 Indexação

A indexação no WF-Sim é feita diretamente a partir dos *clusters*. É criado um índice invertido (Baeza-Yates & Neto 2011) onde os centroides são as chaves dos índices e os outros elementos dos *clusters* ficam numa lista apontada pela chave, como é possível ver na Tabela 1. Isso facilita a busca, pois é necessário fazer comparações por similaridade apenas com os centroides e a partir deles obter o restante dos elementos semelhantes. Lembrando que os centroides também são elementos de formulário por isso existem elementos como chave e elementos como valor no índice invertido.

Tabela 1 - Estratégia de indexação dos *clusters*

CHAVE	VALOR
Elemento 1	ElementoX, ElementoY, ElementoZ ...
Elemento 2	ElementoW, ElementoV,...

3.5 Busca

As buscas no WF-Sim são feitas a partir de um *web form* de entrada, podendo ser um *web form* real ou criado pelo usuário com as características que ele deseja (também chamado de *template*). Como visto anteriormente, um formulário é dividido em elementos, e cada elemento possui uma URL que identifica de qual *web form* ele provém.

Cada elemento é comparado com os centroides dos *clusters* e são retornados os *clusters* mais similares àqueles elementos, ou seja, são retornados todos os elementos pertencentes àqueles *clusters*. A partir das URLs desses elementos é possível identificar os *web forms* semelhantes ao formulário de entrada. Porém não se tem uma noção de quais *web forms* possuem mais similaridades que outros.

Para resolver esse problema, foram feitos dois tipos de *ranking*. Um deles considera a contagem de elementos semelhantes entre dois *web forms*. Assim, é verificado nos elementos retornados quantos pertencem ao mesmo *web form*, e com isso é feito um *ranking* onde os *web forms* com mais elementos semelhantes são mais parecidos com o *web form* de entrada. Esse processo pode ser visto na Figura 6. Para esse *ranking*, só é necessário o cálculo de similaridade entre os elementos de entrada e os centroides.

A segunda forma de *ranking* proposta, representada pela Figura 7, é feita através da média das similaridades dos elementos. Diferente da primeira, nesta técnica é necessário os cálculos de similaridades com todos os elementos retornados. Esses valores são agrupados de acordo com o formulário que pertencem e é feita uma média das similaridades de cada formulário retornado. Assim sendo, é possível identificar os *web forms* mais semelhantes ao buscado.

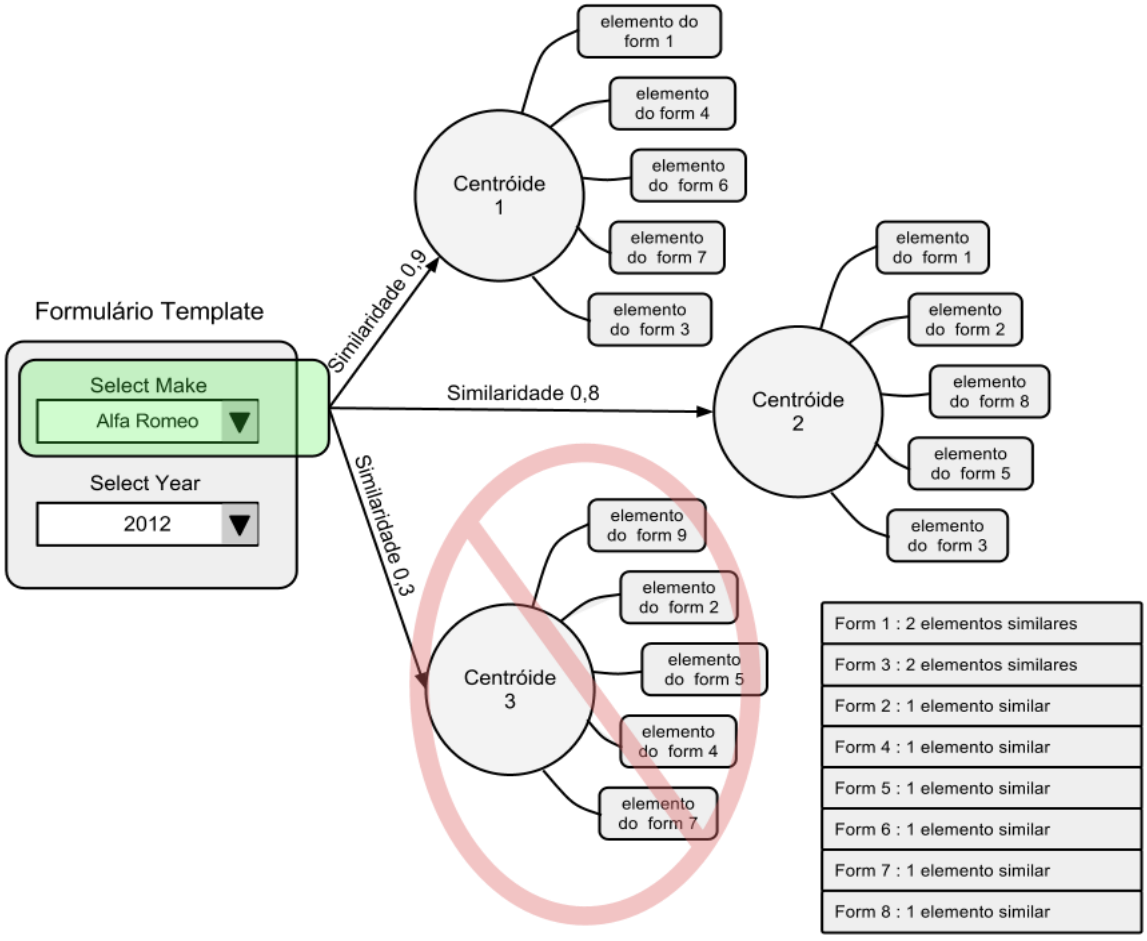

Figura 6 - Ranking pela quantidade de elementos semelhantes

Lembrando que, apesar das representações das figuras, o processo se aplica a todos elementos do formulário *template*, não somente a um deles como está representado nas figuras.

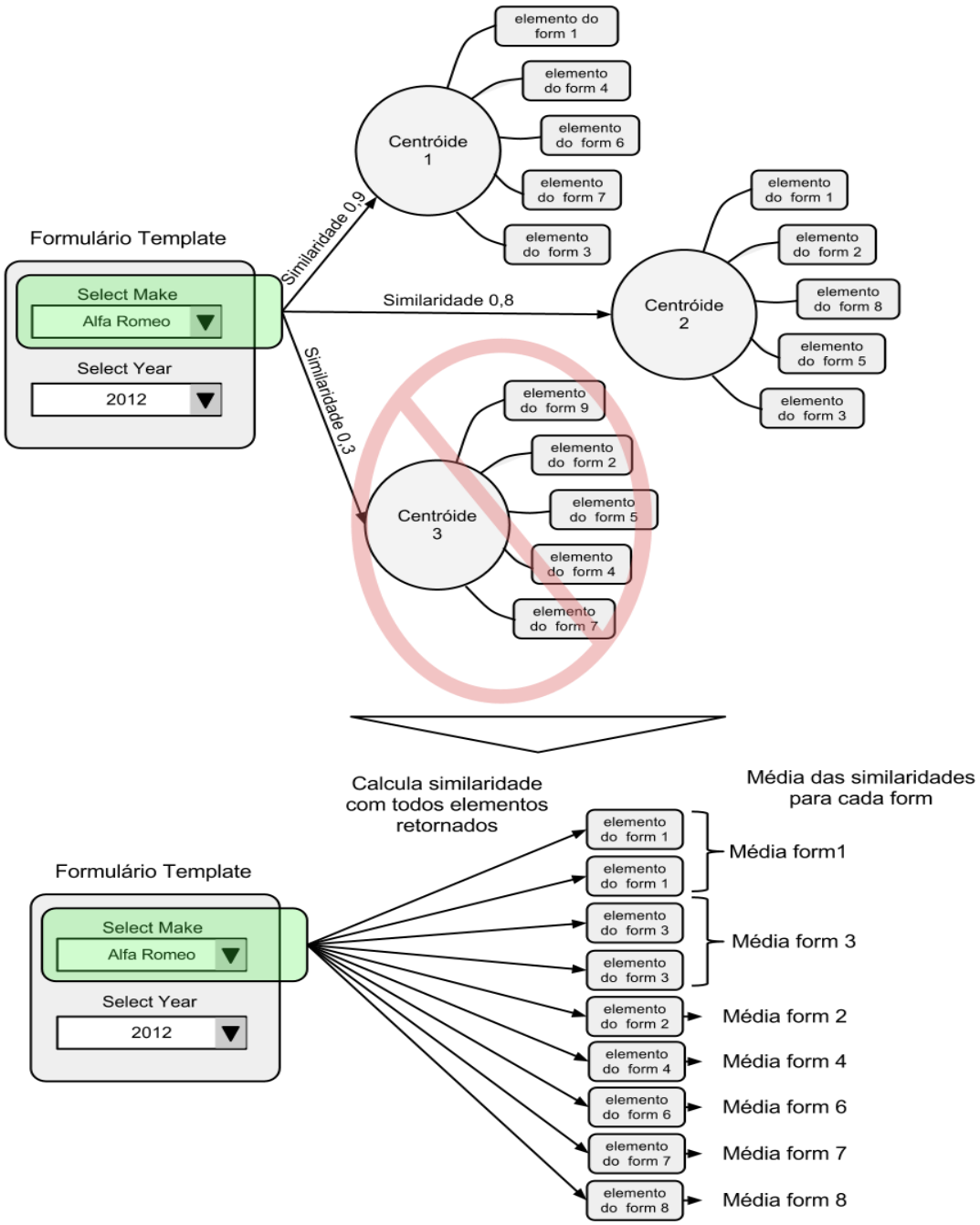

Figura 7 - Ranqueamento dos formulários pela média das similaridades

4. Persistência dos *Clusters*

Pelo fato do WF-Sim ainda estar em desenvolvimento, suas clusterizações não são persistidas, ou seja, a cada execução todos os formulários são novamente processados e clusterizados. Isso foi feito devido à necessidade de realizar experimentos preliminares sobre a implementação do módulo de clusterização, tendo mais relevância a qualidade dos resultados do que a velocidade para obtê-los.

O cálculo dos *clusters* é algo que exige muito processamento e por isso é demorado. O fato de eles serem recalculados a cada execução torna o WF-Sim muito lento e inviável. Portanto, uma das contribuições deste trabalho é a extensão do WF-Sim para permitir o armazenamento dos *clusters* calculados e assim, executar o agrupamento somente quando necessário.

4.1 Persistência em Arquivos

Uma das estratégias de persistência dos *clusters* proposta foi na forma de arquivos. Num primeiro momento, optou-se por criar um grande arquivo serializado contendo todos os *clusters*. Pelo fato da existência de alguns parâmetros que podem variar na criação dos *clusters*, cada nova clusterização foi armazenada em pastas de acordo com estes parâmetros de entrada.

Estes parâmetros são:

- O raio de clusterização utilizado para agrupar elementos que estejam dentro de um raio de distância do centroide;
- A precisão com que o centroide é recalculado durante a clusterização;

- Os pesos aplicados a rótulos e valores no cálculo de similaridade;
- O número de elementos carregados da base de dados.

Assim, como mostra a Figura 8, várias clusterizações com configurações diferentes podem ser armazenadas. A figura mostra dois valores de raio armazenados, 1.5 e 1.0, ou seja, dentro de cada uma dessas pastas existe todo um conjunto de clusterizações para essas configurações. Também na figura mostram variações dos pesos de rótulos e valores, além de variações no número de elementos carregados da base.

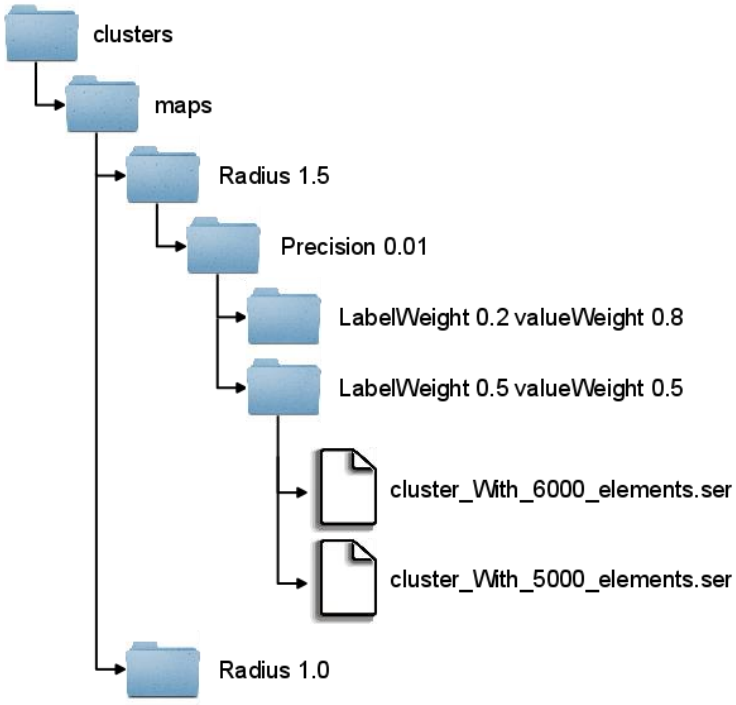

Figura 8 - Organização das pastas de arquivos com as clusterizações

Foram feitos alguns testes comparando o WF-Sim sem persistência e com persistência e a solução de armazenamento de *clusters* apresentou bons resultados de desempenho. Apesar disso, optou-se por outra forma de armazenamento que tornasse

mais fácil a indexação dos *clusters*, já que estando eles contidos em um único arquivo serializado, seria difícil acessar um *cluster* específico.

Assim sendo, a segunda alternativa para persistência dos *clusters* em arquivos foi através da separação de cada *cluster* em um arquivo diferente, armazenando-os em pastas nomeadas com o id do centroide, como é possível observar na Figura 9.

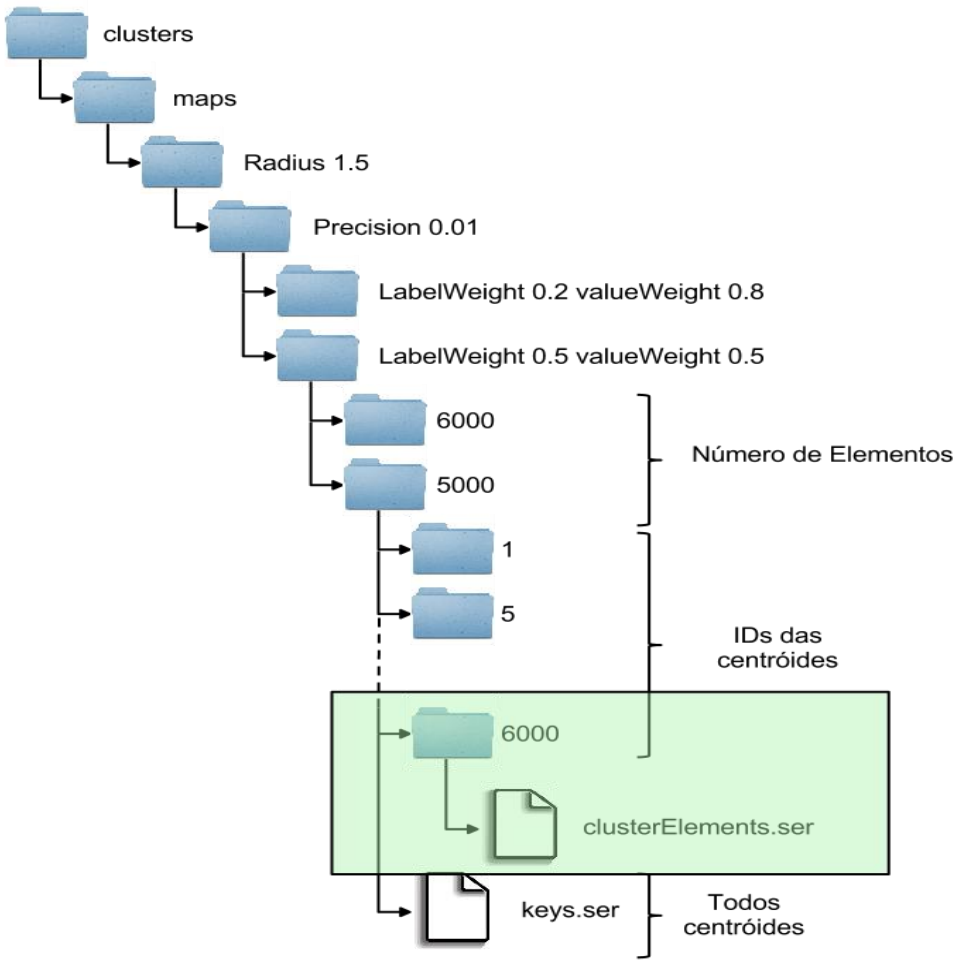

Figura 9 - Segunda organização dos *clusters*

A organização das clusterizações manteve-se a mesma, ou seja, dividida em pastas de acordo com os parâmetros de configuração. Porém, como é possível

observar na Figura 9, cada *cluster* está armazenado em uma pasta diferente, na figura, por exemplo, está destacado o *cluster* com centroide de id 6000, existem no exemplo também outras pastas com nomes 1 e 5, nessas pastas estão contidos os *clusters* cujos centroides possuem os ids 1 e 5 respectivamente.

Para facilitar o acesso aos centroides, criou-se um arquivo *keys.ser* contendo todos os centroides. Com isso, para acessar um *cluster* específico, basta carregar esse arquivo centroide, encontrar o desejado, verificar seu id e acessar a pasta com os elementos semelhantes.

Como é possível observar na Figura 9, essa solução ainda continha alguns problemas de acesso, já que para acessar um centroide específico seria necessário buscá-lo em um arquivo contendo todos os centroides. Assim, foi desenvolvida uma terceira solução. Esta solução, mostrada em destaque na Figura 10, tem por objetivo o acesso direto ao *cluster*. Com isso, além do arquivo contendo todos os centroides, também foram divididos os centroides em vários arquivos nomeados com o próprio rótulo, como no exemplo, é possível ver os arquivos com os nomes "select a make", "year" e "model", esses são os rótulos daqueles centroides armazenados. Portanto, para acessar um *cluster* específico, é necessário buscar somente o arquivo pelo nome do rótulo, tornando mais ágil o acesso direto ao mesmo.

Para possibilitar a criação de arquivos nomeados pelos rótulos dos centroides, foi necessário um tratamento retirando caracteres especiais que não são aceitos pelo sistema de arquivos. Também alguns rótulos possuíam muitos caracteres e por isso foram filtrados rótulos com mais de 250 caracteres, visto que o sistema de arquivos limita o tamanho dos nomes dos arquivos.

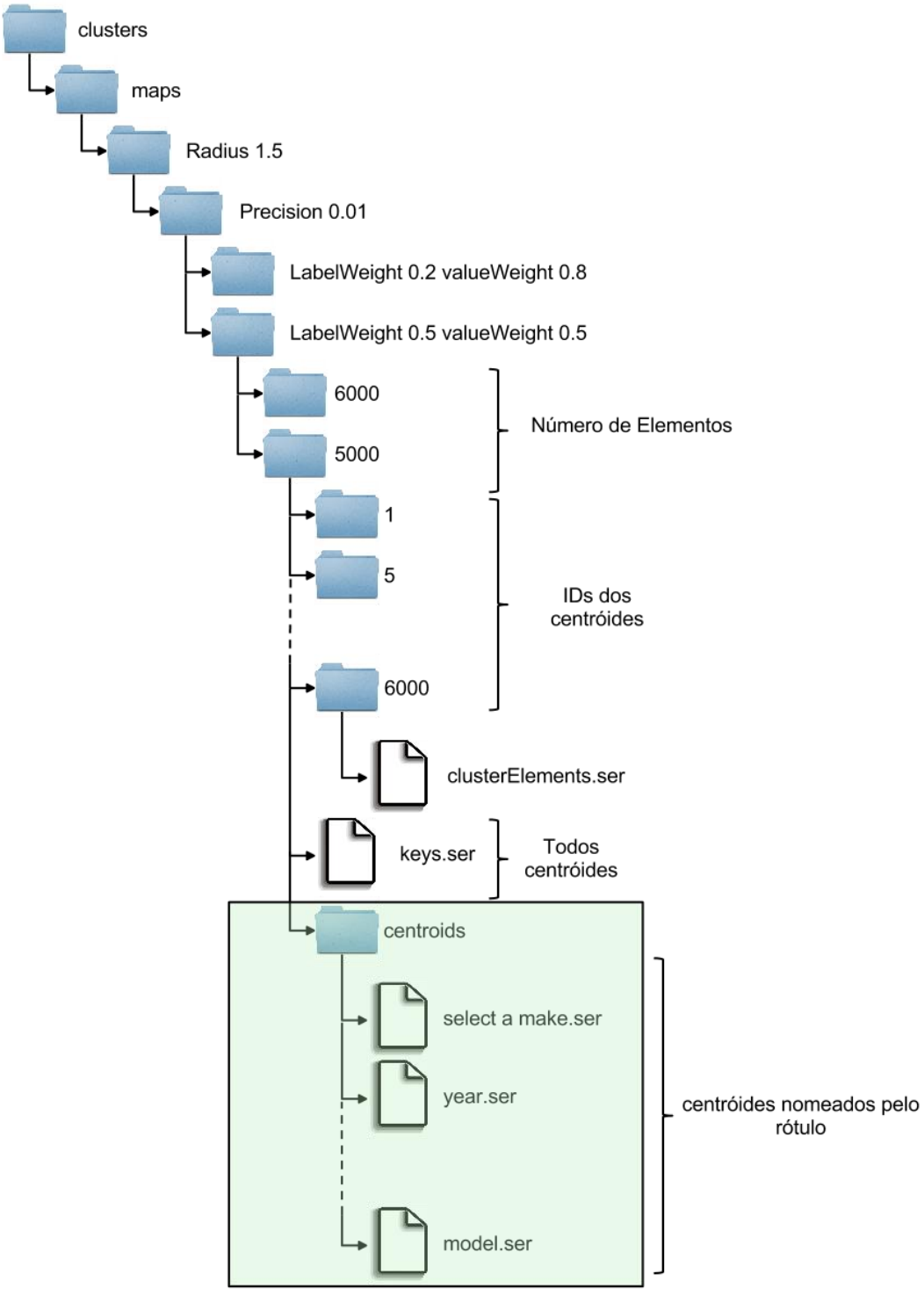

Figura 10 - Organização final das pastas de arquivos com os *clusters*

4.2 Persistência em Banco de Dados

Além da persistência dos *clusters* em arquivos organizados em pastas, também se decidiu testar o armazenamento dos *clusters* em banco de dados. Para tanto, foi necessário o projeto do banco de dados. A modelagem conceitual criada pode ser vista na Figura 11.

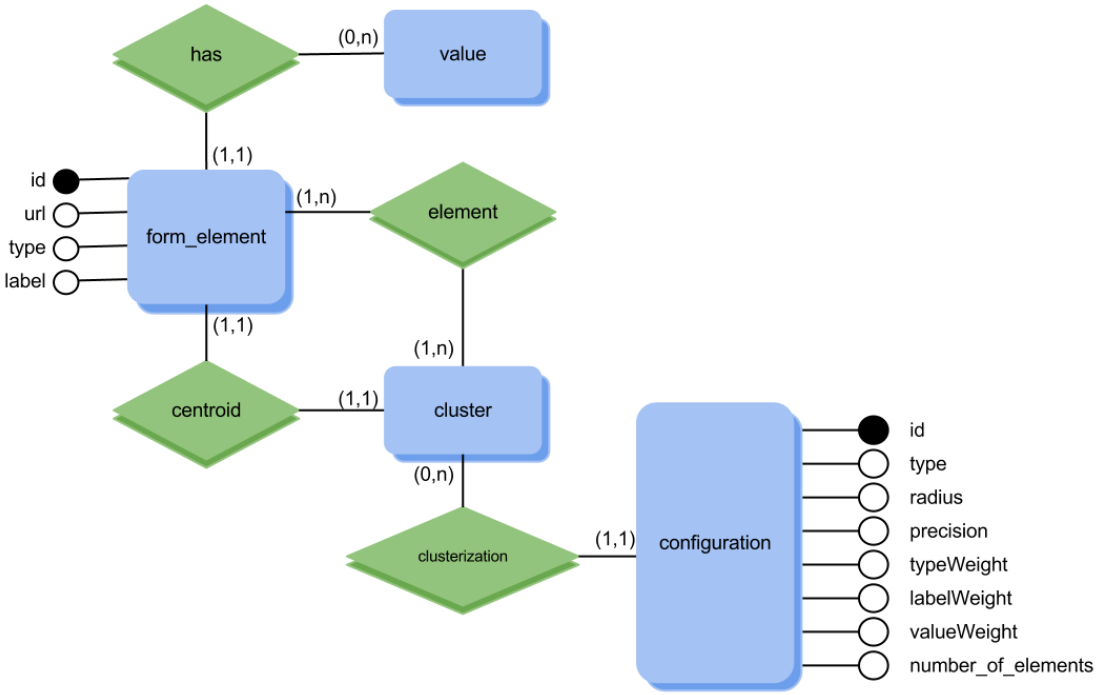

Figura 11 - Modelagem conceitual da persistência dos *clusters* em banco de dados

Assim como no armazenamento em arquivos, encontrou-se o problema de como separar as clusterizações de acordo com cada configuração. Para resolver isso foi criada a tabela ‘configuration’ que mantém essas configurações. Sendo assim é possível armazenar e identificar *clusters* com configurações diferentes.

Como os centroides também são elementos a tabela ‘*cluster*’ possui uma relação ‘centroid’ com ‘form_element’ e outra relação ‘element’ também com ‘form_element’. A

primeira relação indica qual é o centroide do *cluster* e a segunda indica os demais elementos do *cluster*.

Da mesma forma que na persistência em arquivos, para acessar os *clusters* desejados no banco de dados é feita uma busca pelos *clusters* cujo centroide possui determinado rótulo. A partir dos identificadores dos *clusters* retornados é possível acessar os demais elementos do *cluster*.

Sendo assim, com a adição das persistências das clusterizações, a nova arquitetura do WF-Sim pode ser vista na Figura 12. Os *clusters* são armazenados e podem ser acessados pelo sistema.

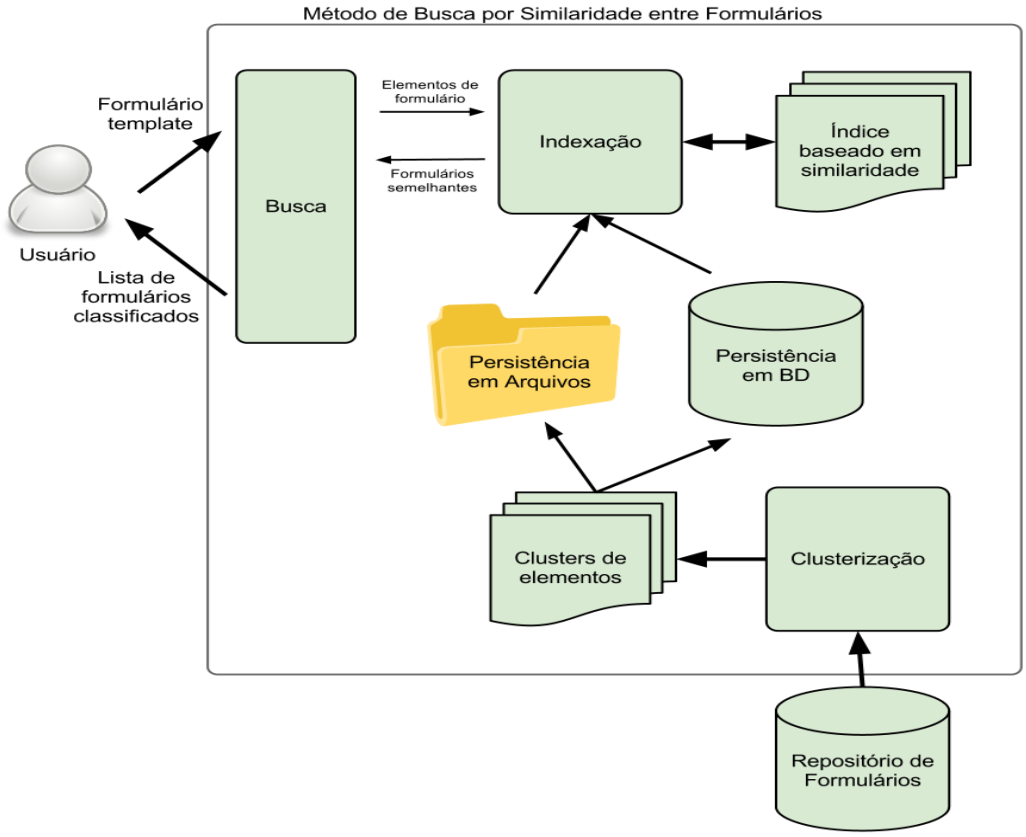

Figura 12 - Arquitetura do WF-Sim com persistências

5. K-Means

Como dito anteriormente, o algoritmo de clusterização utilizado originalmente pelo WF-Sim é o Median Shift. Este algoritmo foi adotado no projeto por ser adequado à clusterização de estruturas em grafo, considerando que cada formulário pode ser modelado como uma estrutura hierárquica onde o nodo raiz representa um formulário, que possui nodos filhos que representam rótulos e estes, por sua vez, podem ter nodos filhos que representam seus valores. Entretanto, notou-se que a clusterização é um processo muito demorado no WF-Sim. Com isso, optou-se por implementar e testar outro algoritmo de clusterização para fins de comparação.

O algoritmo escolhido foi o K-Means (MacQueen 1967). Essa escolha foi feita pois este é um algoritmo considerado eficiente e assim como o Median Shift, trabalha com noções de distância entre os elementos.

5.1 Algoritmo

O K-Means, da mesma forma que o Median Shift, agrupa elementos utilizando um centroide e os demais elementos são clusterizados baseados na semelhança com o centroide. A principal diferença entre o K-Means e o Median Shift é o fato de o primeiro realizar o cálculo de todos *clusters* de uma só vez, enquanto o segundo calcula um *cluster* por vez, ou seja, no primeiro os cálculos dos *clusters* são mais dependentes entre si.

O centroide no K-Means, a principio é escolhido aleatoriamente e conforme o algoritmo é executado, o centroide é reajustado para outros elementos conforme a

média da distância entre ele e os outros do grupo. Esse processo pode ser resumido nos seguintes passos[2]:

1. Definir os centroides iniciais;
2. Calcular a distância entre todos os elementos a serem clusterizados e os centroides;
3. Associar cada elemento com o *cluster* cujo centroide seja mais similar ao elemento (menor distância);
4. Recalcular os centroides;
5. Voltar ao passo 3 até que os centroides não mudem mais de posição.

5.2 Implementação

Para a implementação do algoritmo, primeiramente pensou-se no reaproveitamento de código de terceiros, como a implementação disponível na biblioteca de *Data Mining* Weka[3]. Porém, notou-se uma grande quantidade de dependências naquela implementação, o que dificultaria a integração com o WF-Sim. Também foram encontradas outras implementações mais simples do mesmo algoritmo. Entretanto, pelo fato do WF-Sim lidar com dados bem específicos, que são formulários *web*, decidiu-se criar a própria implementação do K-Means.

Diferente do Median Shift, o K-Means necessita que o usuário especifique a quantidade de *clusters* que devem ser gerados. Portanto, foi necessário adicionar este novo parâmetro de configuração ao cálculo dos *clusters*. Quanto aos outros parâmetros

[2] http://www.orhandemirel.com/blog/kmeans-*cluster*ing/

[3] http://www.cs.waikato.ac.nz/ml/weka/

utilizados na clusterização pelo Median Shift, foram todos utilizados no K-Means, exceto a precisão de cálculo dos centroides.

No K-Means também foi reutilizado o cálculo de distâncias entre elementos do outro algoritmo, pois este é um parâmetro necessário também para ele. Mesmo assim, foi modificada a estrutura de dados utilizada para armazenar os valores de distância em memória. A estrutura de dados utilizada anteriormente era uma tabela com colunas “CENTRO”, “VIZINHO” e “DISTÂNCIA” como é possível ver na Figura 13. Desta forma, eram armazenados dois elementos em cada tupla juntamente com a distância entre eles. A forma de acesso a esses dados era semelhante a um banco de dados, onde se filtrava os elementos desejados e se obtinha uma nova tabela mais específica. Porém, notou-se um grande consumo de memória por parte dessa estrutura. Por isso, ela foi substituída por uma estrutura do tipo *HashMaps*[4], como mostra a Figura 14.

CENTRO	VIZINHO	DISTÂNCIA
Elemento	Elemento	0,5
:	:	:
Elemento	Elemento	0,7

Figura 13 - Estrutura de dados antiga

A nova estrutura possui um *HashMap* numa hierarquia superior onde as chaves são elementos e os valores são outros *HashMaps*. Os *HashMap*s inferiores na hierarquia também possuem elementos como chaves e seus valores são as distâncias

[4] http://en.wikipedia.org/wiki/Hash_table

entre os elementos chave. Assim, caso se queira encontrar a distância entre um elemento e outro na estrutura, é possível acessá-la diretamente utilizando um dos elementos para obter o *HashMap* inferior e outro elemento para obter a distância entre os dois. O *HashMap* superior possui todos os elementos como chave. Os *HashMaps* inferiores, por sua vez, possuem apenas os elementos que estão a uma distância menor que o raio de clusterização. Também existe a possibilidade de alguns elementos estarem em mais de um *HashMap* inferior ao mesmo tempo. Isso ocorre caso o elemento esteja dentro do raio de clusterização de mais de um elemento. Com essa nova estrutura notou-se uma grande melhora quanto ao uso de memória durante o acesso aos dados.

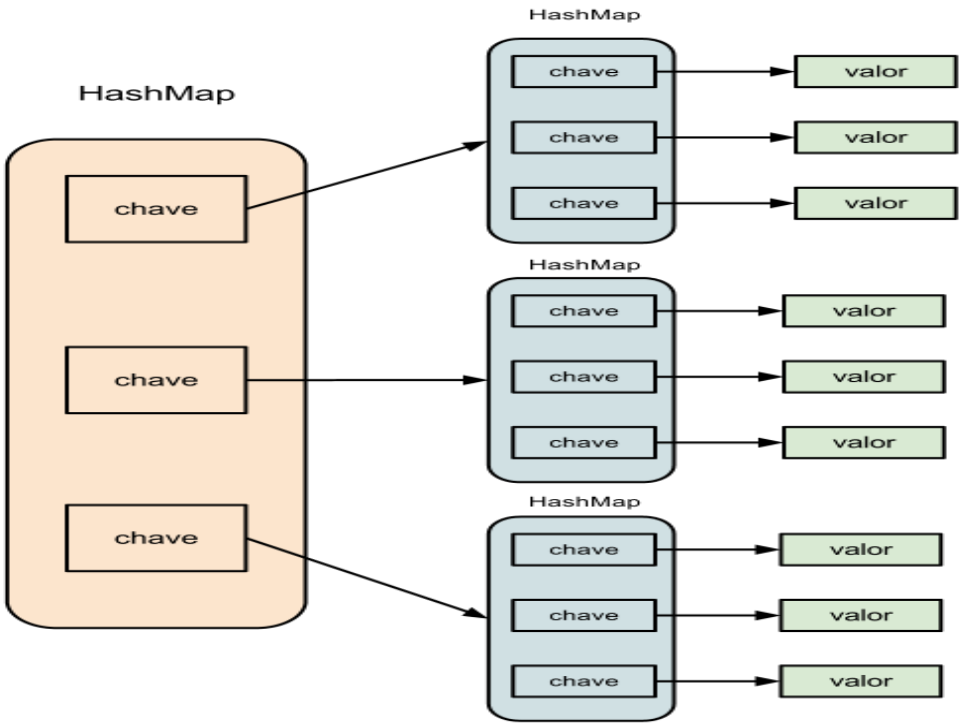

Figura 14 - Nova estrutura de dados para armazenar as distâncias entre elementos.

No K-Means, originalmente, depois de agrupados os dados, o centroide é recalculado para um ponto médio entre os elementos, sendo que esse ponto não necessariamente é um elemento. Ele pode ser um ponto no espaço vetorial. Assim sendo, para o WF-Sim foi necessário fazer uma adaptação nesse cálculo, visto que os elementos não estão contidos num espaço vetorial e não é possível calcular uma distância entre eles e um ponto no espaço. O cálculo foi adaptado para que o novo centroide sempre seja um elemento, sendo possível assim verificar a distância entre os demais elementos e o novo centroide. Essa adaptação é mostrada na Figura 15.

A adaptação feita leva em conta a distância média entre um elemento e os demais elementos do grupo, sendo que o elemento que possuir menor distância média entre os demais é definido como o novo centroide. Assim, esse processo de agrupamento e cálculo dos centroides se repete até que os centros se estabilizem e, com isso, tem-se os *clusters* definidos.

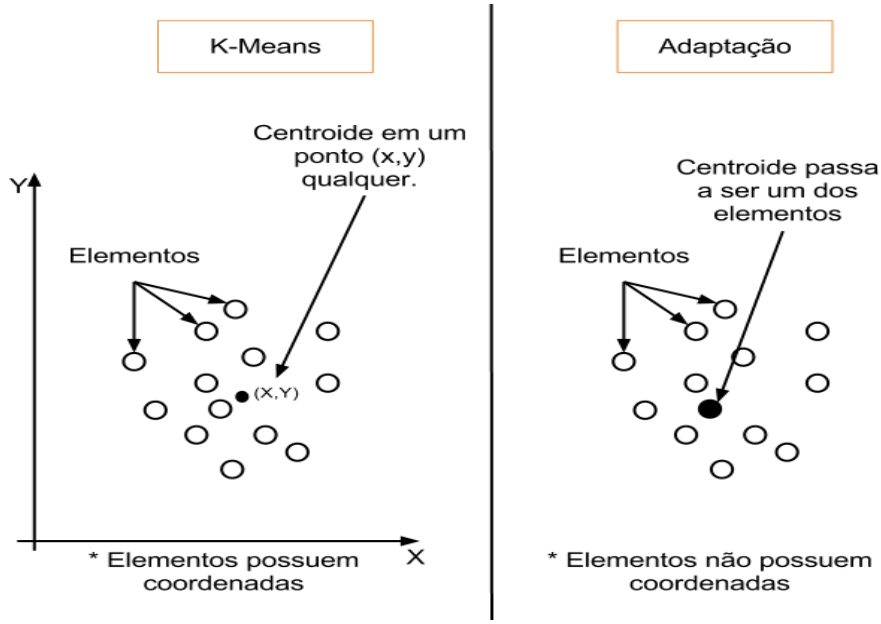

Figura 15 – Adaptação do K-Means para os elementos de formulário

6. Experimentos

Uma vez propostas e desenvolvidas essas otimizações no WF-Sim, foram realizados alguns experimentos sobre o sistema. Esses experimentos visam analisar a melhora de desempenho do WF-Sim tanto no processo de clusterização quanto no processo de busca. Também foram feitos alguns testes para verificar a qualidade dos resultados obtidos nas buscas e clusterizações, comparando os dois algoritmos de clusterização e também como as variações de pesos de rótulos e valores influenciam a qualidade dos *clusters*.

6.1 Experimentos com as Estratégias de Persistência de *Clusters*: Foco no Tempo de Processamento

Para confirmar a melhoria de desempenho das estratégias de persistência propostas e compará-las, foram realizados alguns experimentos preliminares de busca sobre as clusterizações persistidas. Esses testes foram realizados utilizando uma base de *Web form*s cedida pelo grupo de banco de dados da Universidade de Utah. Esta base de dados contém formulários dos domínios de automóveis, livros e filmes, dentre outros. A base foi carregada pelo WF-Sim e transformada em um arquivo serializado. Esse arquivo contém todos os formulários e elementos de formulário na forma de objetos de classes Java, como é possível ver na Figura 16. Assim, para os experimentos, os dados foram carregados para o sistema utilizando esse formato de arquivo serializado.

Para os experimentos foram carregados 6157 elementos da base (todos os elementos) e estes foram agrupados em 910 *clusters.* As configurações de hardware utilizadas para os testes podem ser vistas na Tabela 2.

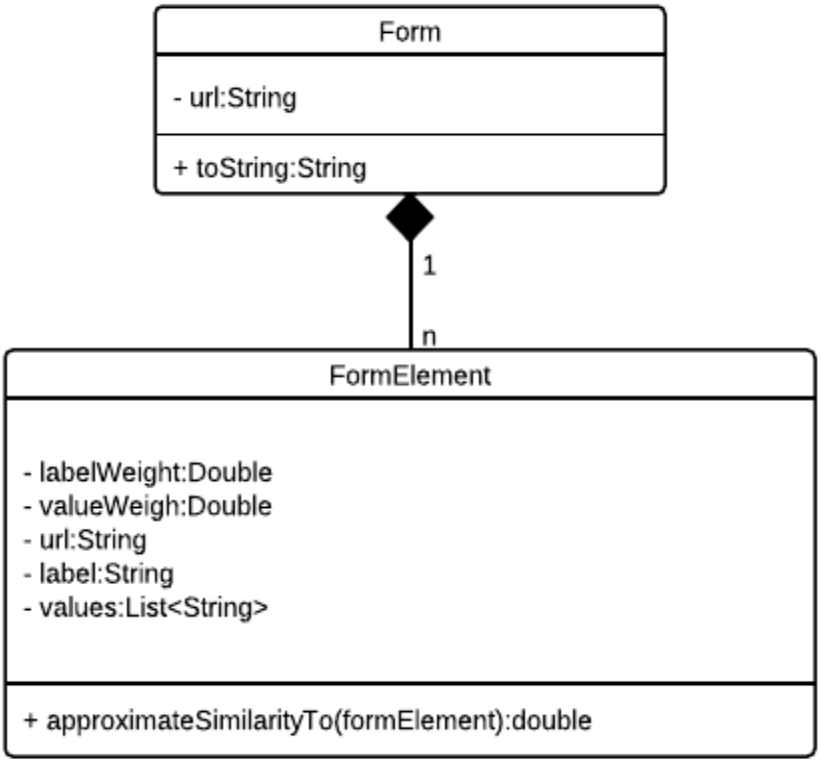

Figura 16 - Diagrama de classes dos dados utilizados

Para a realização dos testes foi criado um formulário base (*template*) no domínio de automóveis com 10 elementos. Este é um formulário típico para busca de automóveis para venda ou aluguel. Este domínio foi escolhido pois é o mais abundante na base de dados de formulários disponível. A Tabela 3 mostra os elementos considerados no *template* com suas restrições de valor, quando existentes.

Tabela 2 - Hardware utilizado nos testes de busca

Processador	Core 2 Duo CPU T5800, 2 GHz
Memória	3Gb
HD	160Gb SATA
Sistema Operacional	Windows XP SP3

Para variar a quantidade de *clusters* retornados nas buscas, em alguns testes foram retirados elementos do *template* sendo que os mantidos foram aqueles que retornavam mais dados. Assim, testes foram feitos com 1, 5 e 10 elementos, para fins de avaliação de desempenho de um número crescente de elementos a ser considerado nas buscas.

Para recuperar os centroides desejados, usando tanto o banco de dados quanto os arquivos, dividiu-se o *template* em seus elementos e, para cada elemento, foram encontrados os centroides cujos rótulos possuíssem algum termo em comum com os termos do elemento de entrada. Encontrados esses centroides, foi aplicado o cálculo de similaridade do WF-Sim sobre eles. Os centroides com similaridade abaixo de 0.7 foram desconsiderados para manter no resultado apenas dados com alta similaridade. Esse processo pode ser visto na Figura 17.

Tabela 3 – *Template* de formulário considerado nos testes

ID	RÓTULO	VALOR	ID	RÓTULO	VALOR
1	select make	ford hyundai nissan Toyota	6	model	any model blazer caravan century civic cobalt enclave
2	min price	20,000 30,000 40,000 50,000 60,000 70,000 80,000	7	miliage	20,000 30,000 40,000 50,000 60,000 70,000 80,000
3	choose year	2000 2001 2002 2003 2004 2005 2006	8	used car	acura alfa romeo audi bmw fiat chevrolet cadillac
4	max price	200,000 300,000 400,000 500,000 600,000 700,000 800,000	9	sort order	by price by make by date by year
5	include	all vehicles new vehicles used vehicles certified vehicles	10	modified	(Sem valores)

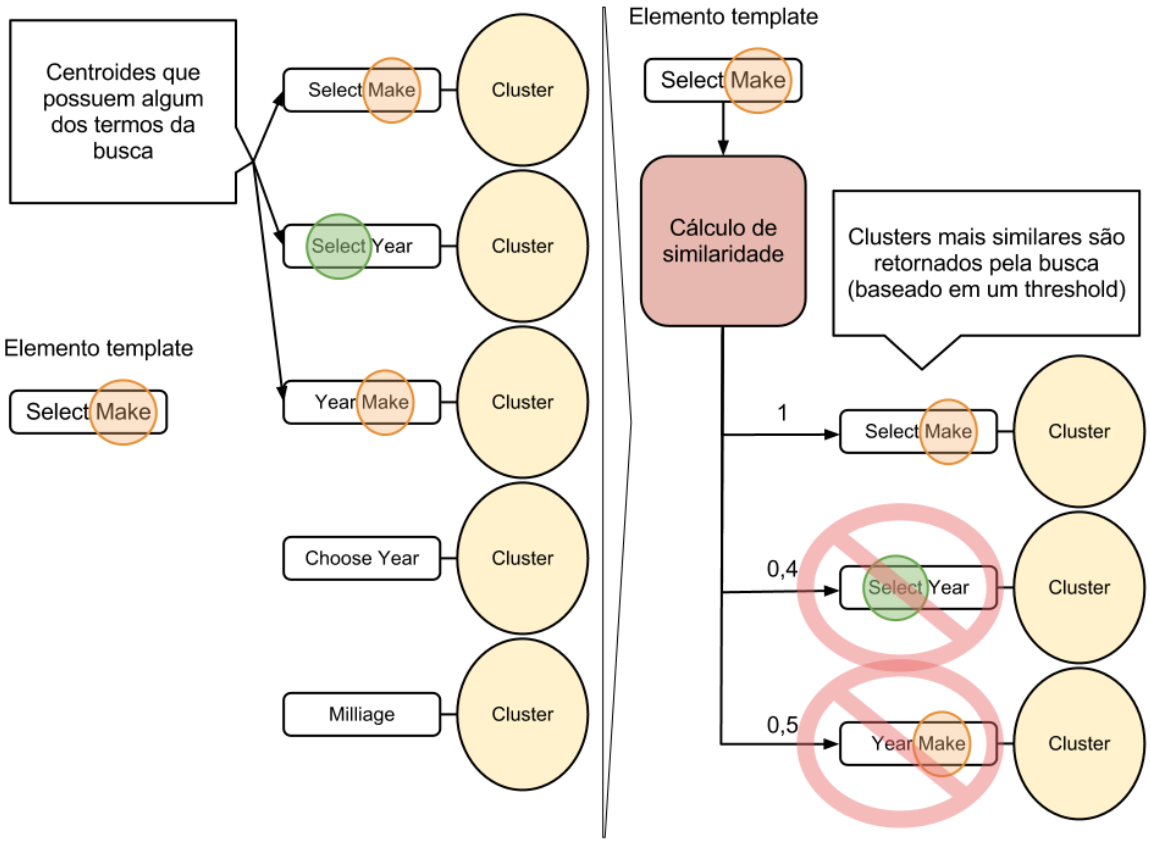

Figura 17 – Recuperação dos *clusters* pela busca

Além da busca retornando os *clusters* mais similares, também foram realizados testes utilizando o mecanismo de ranqueamento (*ranking*) do WF-Sim. Esse ranqueamento é feito durante as consultas para verificar, num *ranking*, quais *clusters* são mais semelhantes ao *template* de entrada fornecido pelo usuário. O WF-Sim originalmente utilizava todos os *clusters* calculados na clusterização para o ranqueamento, produzindo um *overhead* na geração do resultado da consulta. Para reduzir este *overhead*, o mecanismo de ranqueamento foi modificado neste trabalho para considerar somente os *clusters* recuperados das buscas em arquivos e no banco

de dados que superam um determinado limiar (*threshold*), como exemplificado anteriormente (0.7).

No experimento descrito a seguir, o foco foi avaliar o desempenho em termos de tempo das duas estratégias. A avaliação da relevância dos *clusters* retornados é abordada na seção 6.2.

A Figura 18 apresenta os resultados dos experimentos. Os tempos mostrados são a média de 3 execuções para cada busca. As consultas foram disjuntivas, ou seja, são retornados os *clusters* que casam com pelo menos um (1) dos elementos do *template*. Por isso, quanto mais elementos são considerados como entrada para a busca, maior é a quantidade de *clusters* recuperados.

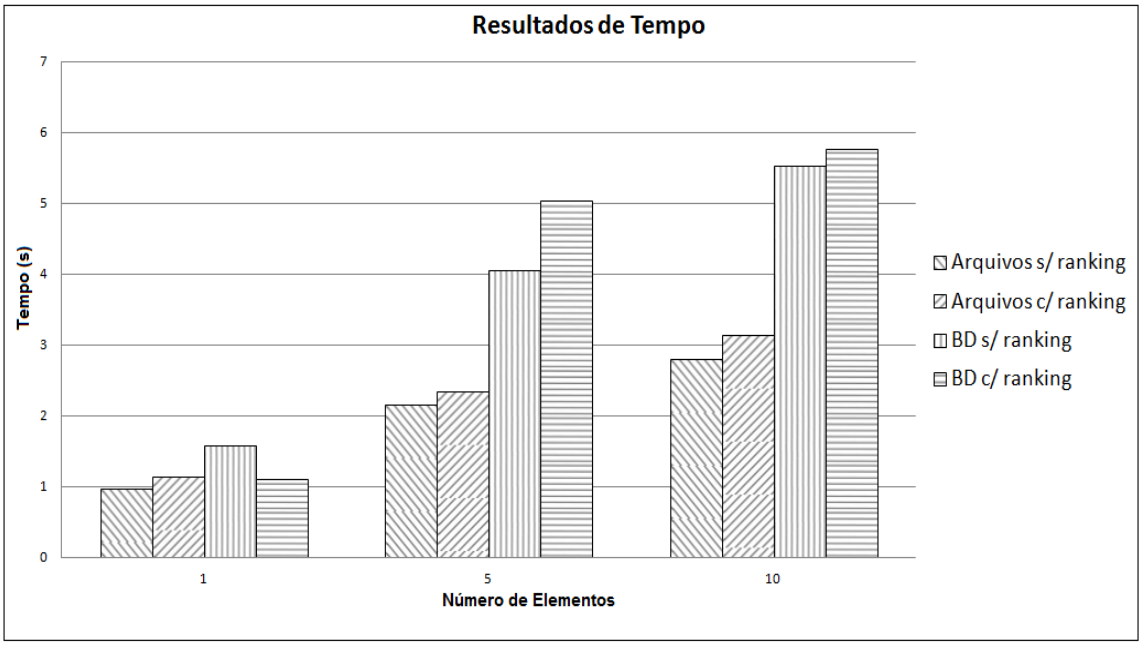

Figura 18 - Resultados dos Experimentos

Os testes com 1 elemento consideraram buscas envolvendo o elemento com ID 1 do *template* da Tabela 3. Os testes com 5 elementos consideraram os elementos com ID de 1 a 5 e os testes com 10 elementos consideraram todos os elementos do

template. Essa escolha foi feita baseada nos elementos que mais aparecem no domínio de automóveis, sendo que os cinco primeiros são os que mais aparecem e por isso foram escolhidos para os testes com 1 e 5 elementos.

De acordo com a Figura 18, a estratégia de persistência em arquivos sem considerar ranqueamento do resultado foi a mais rápida, tornando-se 2.2 vezes mais lenta com 5 vezes mais elementos considerados na busca, e 2.8 vezes mais lenta com 10 vezes mais elementos considerados. Isso mostra uma tendência de redução do crescimento do *overhead* à medida que o número de elementos de entrada aumenta. A estratégia de arquivos com ranqueamento ficou, em média, 1.1 vez mais lenta, representando um *overhead* pouco significativo.

A estratégia de persistência em banco de dados sem ranqueamento tornou-se 2.5 vezes mais lenta com 5 vezes mais elementos considerados na busca e 3.4 vezes mais lenta com 10 vezes mais elementos considerados. Há uma tendência semelhante à da estratégia de persistência de arquivos, apesar de haver um *overhead* maior. A estratégia de banco de dados com ranqueamento ficou, em média, para 5 e 10 elementos, 1.1 vez mais lenta, representando também um *overhead* pouco significativo. Ela só obteve desempenho superior à estratégia sem ranqueamento para a busca com 1 elemento de entrada, sendo isso provavelmente uma pequena anomalia em termos de acesso ao banco de dados, tal como muitos dados em cache do SGBD.

Comparando as estratégias arquivo e banco de dados, ambas com ranqueamento, observa-se um desempenho praticamente equivalente para buscas com 1 elemento. Já para buscas com 5 e 10 elementos, a estratégia de banco de dados mostra-se, em média, 2 vezes mais lenta.

Esse resultado pode ser explicado pela quantidade de dados utilizados, sendo que para essa quantidade de dados é muito mais rápido para o sistema carregar os *clusters* em arquivos do que realizar os *joins* nas tabelas do banco de dados, visto que no sistema de arquivos cada *cluster* está serializado em seu próprio arquivo único, já no banco de dados os *clusters* estão divididos em várias tabelas.

Já com relação aos resultados de busca com e sem *ranking,* podem ser facilmente explicados pelo fato de utilizar *ranking* exigir mais processamento, e por isso, as buscas com *ranking* foram mais lentas.

6.2 Experimentos com as Estratégias de Persistência de *Clusters*: Foco na Relevância do Resultado

A partir dos *clusters* retornados dos testes de busca, também se verificou a qualidade dos resultados obtidos, ou seja, se os *clusters* retornados realmente possuíam semelhança com o formulário *template.*

Assim, foram coletados os *clusters* retornados das buscas realizadas na seção 6.1. Para cada *cluster* retornado, verificou-se se o centroide de cada grupo correspondia com o *template* de entrada busca e se os elementos do *cluster* correspondiam com o centroide.

Para facilitar essa verificação, que foi feita manualmente, foi criado um tipo de relatório em HTML que organizava em tabelas os *clusters* retornados pela busca. Também foram criadas funções *JavaScript* que, aplicadas ao arquivo HTML facilitavam a contagem e a geração do resultado final.

Feito isso, foram obtidos os resultados mostrados na Tabela 4. Essa tabela mostra os resultados quanto à precisão dos centroides retornados das buscas. Essa

precisão é calculada a partir da Equação 3, onde os valores verdadeiro positivos são os valores que foram retornados como positivos e verdadeiramente o são, e os valores falso positivos, são valores retornados como positivos porém são falsos.

$$Precisão = \frac{VerdadeiroPositivos}{VerdadeirosPositivos + FalsoPositivos} \quad (3)$$

Esses resultados foram separados da mesma forma que os testes das persistências, ou seja, testes variando o número de elementos do formulário *template* e executados sobre arquivos e sobre banco de dados. Esses resultados podem ser vistos na Tabela 4.

Tabela 4 – Resultados dos Testes dos Centroides Retornados

	TOTAL	VERD. POSITIVO	FALSO POSITIVO	PRECISÃO
Arquivo - 1 Elemento	10	6	4	0,60
Arquivo - 5 Elementos	37	33	4	0,89
Arquivo - 10 Elementos	62	49	13	0,79
BD – 1 Elemento	10	6	4	0,60
BD – 5 Elementos	39	35	4	0,90
BD – 10 Elementos	66	53	13	0,80

Como é possível observar na Tabela 4, os valores de precisão para arquivo e banco de dados mostraram-se bastante semelhantes, variando apenas com relação à quantidade de elementos utilizados como entrada para o sistema. Todos os valores de precisão mantiveram-se acima de 0.6 e os valores máximos de precisão foram obtidos pelas buscas utilizando 5 elementos de entrada.

Um fato curioso que se notou foi a diferença entre as quantidades totais de elementos retornados para arquivos e banco de dados. Para 5 elementos, por exemplo, a busca em arquivos retornou 37 elementos, enquanto que a busca em banco de dados retornou 39, ou seja, dois a mais. Isso ocorreu devido a uma limitação do sistema de arquivos, que não aceita nomes de arquivos muito grandes. Por isso, foi necessário, durante a persistência, descartar alguns elementos com rótulos muito grandes. Apesar deste fato ser uma limitação da busca, poucos elementos na nossa amostra de formulários possuem nomes muito grandes.

Essa diferença de quantidade de elementos em banco de dados e arquivos, como é possível verificar na Tabela 4, também interferiu nos valores de precisão, sendo que os valores de precisão para banco de dados ficaram ligeiramente maiores que os valores para arquivos.

Além dos centroides, também foram verificados os elementos contidos nos *clusters*, ou seja, se eles realmente correspondiam ao centroide. Assim, foi feito o mesmo cálculo de precisão para cada *cluster* retornado. Para facilitar a análise, foi feita a distribuição de frequência dos valores de precisão. Como esses valores são contínuos foi necessário dividi-los em classes.

Como os testes foram feitos sobre os *clusters* retornados das buscas, foi mantida a divisão dos *clusters* para as buscas por 10, 5 e 1 elementos, porém nesse caso foram comparados os elementos dos *clusters* com o centroide de cada *cluster*. Os resultados dessas comparações podem ser vistos nas figuras Figura 19, Figura 20 e Figura 21, respectivamente.

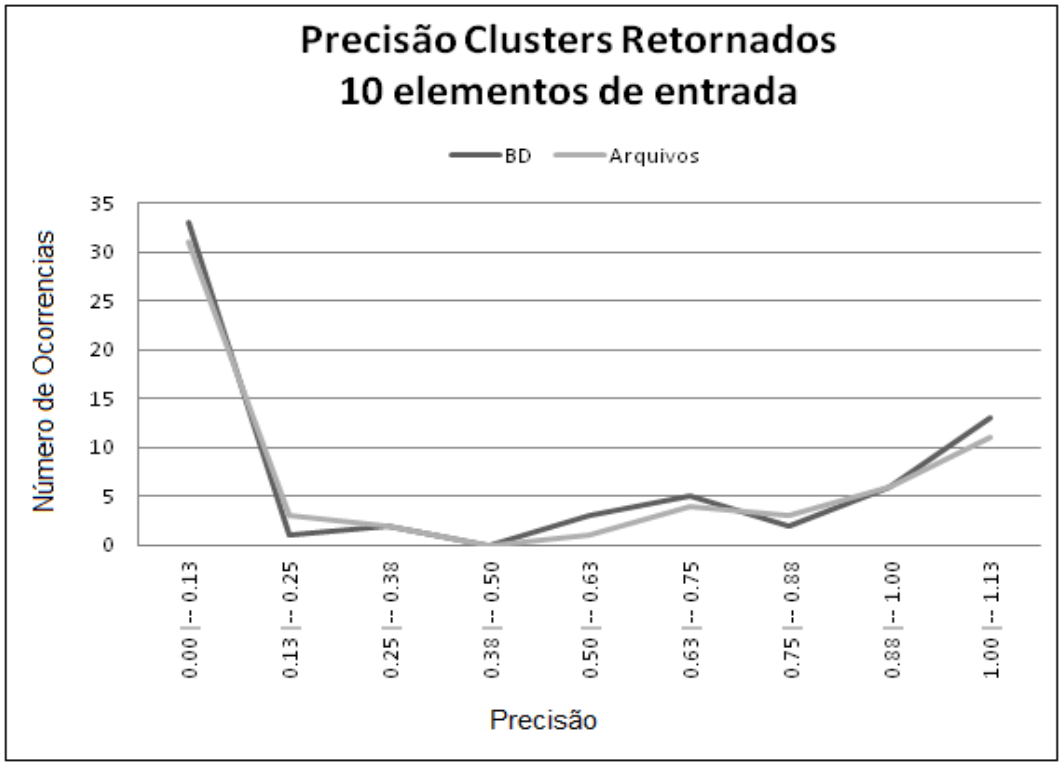

Figura 19 - Distribuição de frequência de precisão – 10 elementos de entrada

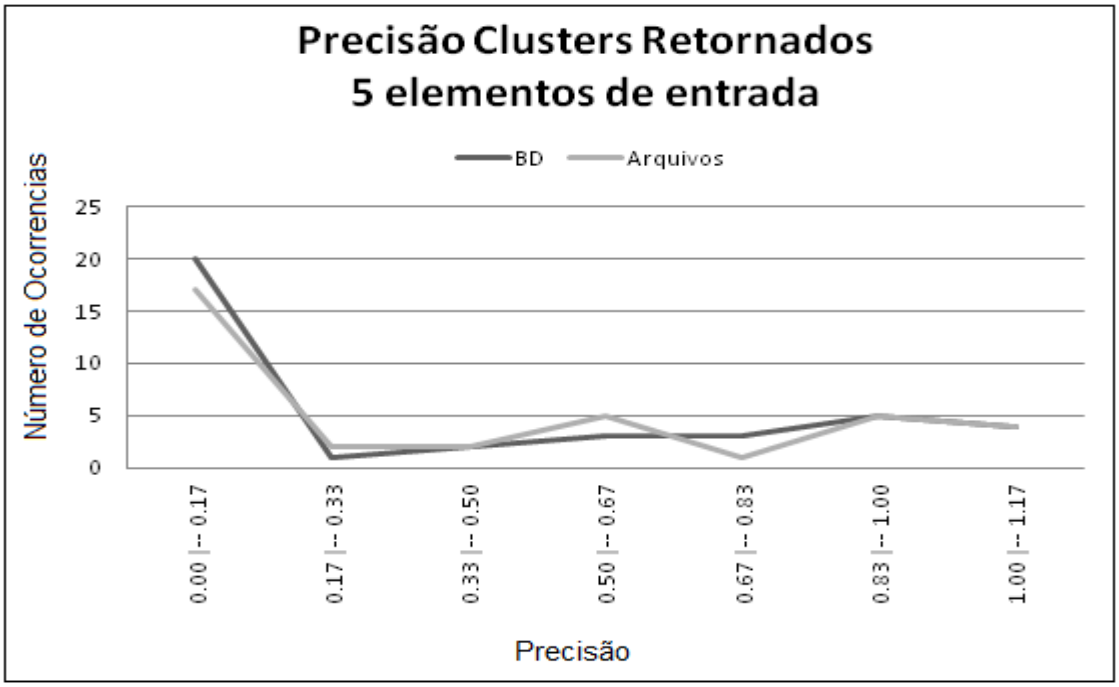

Figura 20 - Distribuição de frequência de precisão – 5 elementos de entrada

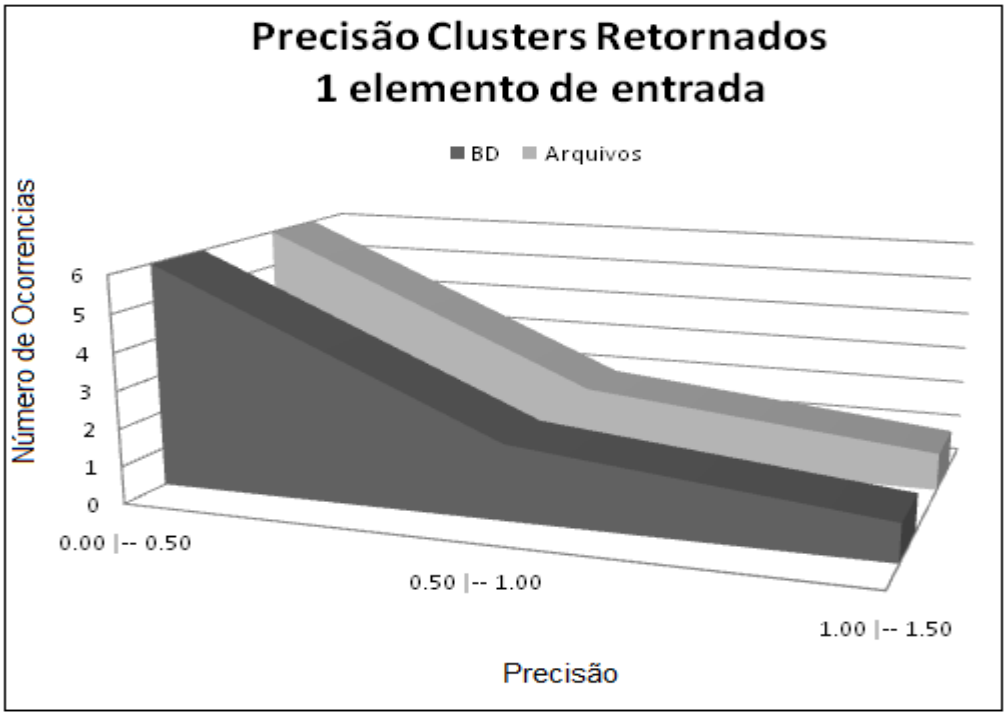

Figura 21 - Distribuição de frequência de precisão – 10 elementos de entrada

Como é possível observar nas figuras, existe uma maior frequência de valores de precisão menores que 0,4, ou seja, obtiveram valores inferiores se comparados com os valores de precisão para os centroides. Isso se deve ao fato de que muitos elementos possuem valores parecidos, porém semânticas diferentes. Por exemplo, elementos dos domínios de preço e de ano possuem ambos os valores numéricos e, verificou-se que em vários *clusters* onde o centroide se referia a preço, possuíam elementos de ano.

6.3 Experimentos de Qualidade de Clusterização

Além dos experimentos feitos sobre as buscas, também foram feitos alguns experimentos de clusterização variando os pesos aplicados aos valores e rótulos no cálculo de similaridade entre os elementos. Esses testes foram feitos com intuito de

verificar qual configuração de clusterização obteria melhores resultados com respeito à qualidade.

Para realizar os testes, primeiramente foi definido no WF-Sim o número de elementos carregados da base. Optou-se por utilizar todos elementos da base, ou seja, 6175. Os valores de precisão e raio utilizados foram 0.01 e 1.5 respectivamente. Também foram variados os valores de peso para verificar qual configuração obtém melhores resultados, os valores de peso utilizados são mostrados na Tabela 5.

Tabela 5 – Pesos utilizados para cada clusterização

CLUSTERIZAÇÃO	PESO RÓTULOS	PESO VALORES
1	0.7	0.3
2	0.5	0.5
3	0.3	0.7

Assim, foram feitas clusterizações utilizando primeiramente o algoritmo Median Shift, a base de formulários utilizada foi a mesma citada no Capítulo 6.1.

Da mesma forma que os *clusters* analisados nos testes de busca, também foram verificados, para cada *cluster* criado, qual a precisão dos valores. Também foram feitas distribuições de frequência de precisão para cada clusterização. Os resultados para os testes de clusterização com o Median Shift podem ser vistos na Figura 22.

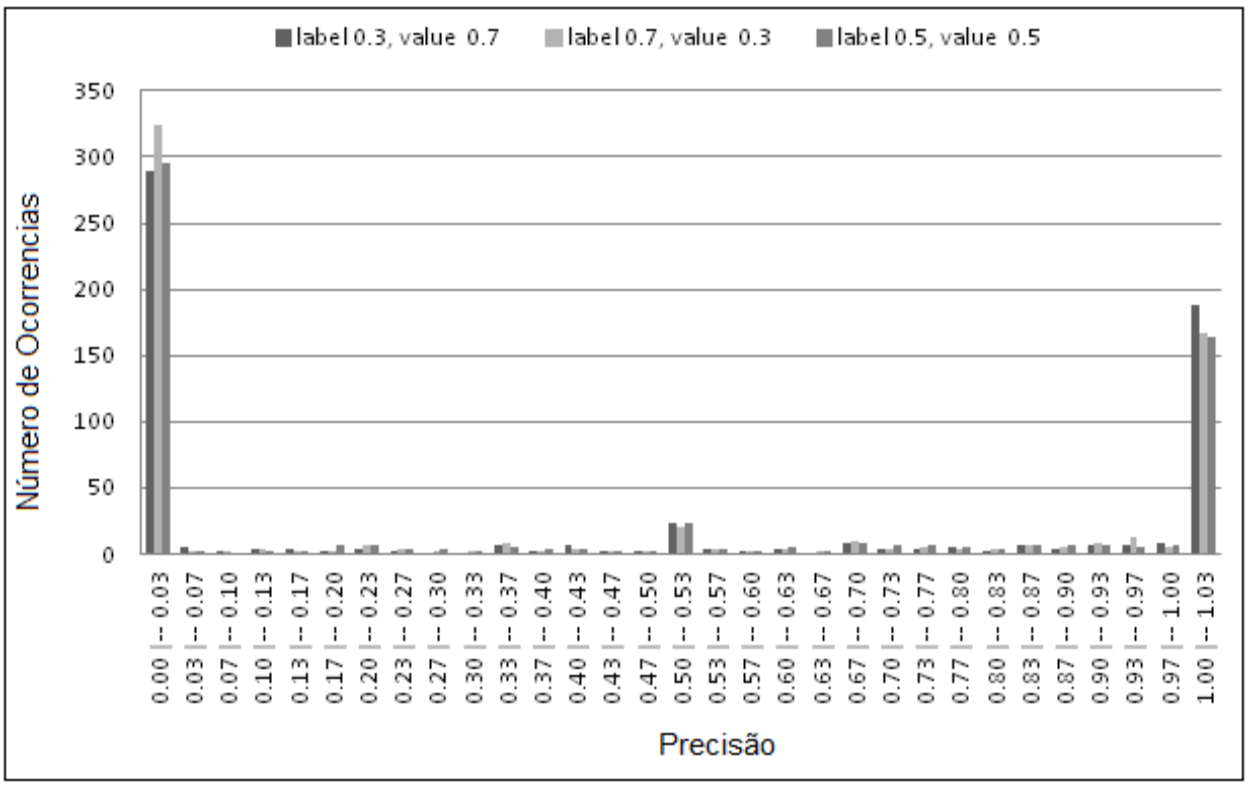

Figura 22 - Distribuição de frequência de precisão – Clusterização Median Shift

Como é possível observar, as frequências ficaram distribuídas em sua maior parte abaixo de 0.07 e outra porção mais significativa ficou acima de 0.97. Novamente acredita-se que isso se deve à quantidade de valores parecidos, porém semanticamente diferentes.

Também se notou que, para os valores de peso 0.3 para rótulos e 0.7 para valores, obteve-se uma frequência maior de *clusters* com precisão acima de 0.97, sendo 196 o número de *clusters* acima dessa precisão para essa configuração, comparado a 172 e 170 para as outras configurações. Essa configuração também obteve a menor frequência para valores menores que 0.03.

Para poder comparar melhor esses resultados, foram calculados os valores de média, mediana e moda das precisões de cada configuração. Esses valores podem ser vistos na Tabela 6.

Tabela 6 – Média, mediana e moda para cada configuração.

	MEDIA	MEDIANA	MODA
label 0.3, value 0.7	0.45	0.14	0.02
label 0.7, value 0.3	0.41	0.03	0.02
label 0.5, value 0.5	0.42	0.12	0.02

Como é possível observar, a configuração com os pesos 0.3 e 0.7 para rótulos e valores, respectivamente, obteve melhores valores de média e mediana. A configuração inversa (0.7 e 0.3 para rótulos e valores, respectivamente) por sua vez obteve os piores valores de média e mediana. Essa configuração foi a que obteve maior número de *clusters* criados (930 contra 912 e 910 das outras configurações), porém, foi a que obteve piores resultados de qualidade.

Uma justificativa para esses resultados é o fato de a base de dados possuir muitos dados sujos e que são pouco significativos para uma busca do usuário. Esses elementos prejudicaram a qualidade das clusterizações, visto que existiam muitos *clusters* cujo centroide possuía valores muito particulares, sendo assim, nenhum elemento do grupo correspondia com o centroide, do ponto de vista semântico.

Também se notou uma grande quantidade de *clusters* que possuíam o próprio centroide também como elemento do *cluster*. Acredita-se que isso se deve ao fato de existirem valores repetidos na base de dados. Esses *clusters* foram computados

separadamente e não são mostrados na Figura 22. Porém, é possível vê-los na Tabela 7, que mostra o número total de grupos criados e o número de *clusters* com centroide repetido.

Tabela 7 – Número de *clusters* criados

	TOTAL	*CLUSTERS* COM CENTROIDES REPETIDOS
Label 0,3, Value 0,7	912	308
Label 0,7, Value 0,3	930	309
Label 0,5, Value 0,5	910	314

6.4 Experimentos com o algoritmo K-Means

Para validar a utilização do algoritmo K-Means na clusterização dos formulários, foram feitos experimentos de qualidade de *clusters* e tempo de clusterização. Novamente, os dados utilizados nesses experimentos são os mesmos descritos no capítulo 6.1.

Durante os testes foram variados alguns parâmetros de clusterização para comparar os resultados. Os parâmetros variados foram: o algoritmo de clusterização, pesos de rótulos e valores, número de elementos, bem como número de *clusters* gerados. Este último parâmetro é requerido pelo algoritmo K-Means. Os valores de precisão e raio utilizados nesses testes foram os mesmos dos testes com o Median Shift, ou seja, 0.01 e 1.5 respectivamente.

Os testes foram feitos sobre um hardware diferente do utilizado nos testes de busca, sendo este um pouco mais rápido que o utilizado anteriormente. Suas configurações são mostradas na Tabela 8.

Tabela 8 – Hardware utilizado nos testes de clusterização

Processador	2.66 GHz Intel Core 2 Duo
Memória	4 GB 1333 MHz DDR3
Vídeo	NVIDIA GeForce 320M 256 MB
Sistema Operacional	Mac OS X Lion 10.7.3

Primeiramente, foram verificados os tempos de execução do algoritmo e comparados com o Median Shift. Os testes foram feitos variando-se a quantidade de *clusters* gerados também para ver se a quantidade de *clusters* influencia no tempo de execução. Com relação ao algoritmo Median Shift, que não utiliza quantidade de elementos como parâmetro, variaram-se os pesos de rótulos e valores. Isso fez com que a quantidade de *clusters* gerados variasse também. Esses resultados podem ser vistos na Tabela 9.

Como é possível observar na tabela, é difícil distinguir alguma tendência de tempo com a variação da quantidade de *clusters* gerados pelo algoritmo K-Means, visto que alguns valores maiores obtiveram menor tempo e alguns valores menores obtiveram maior tempo. Isso se deve à aleatoriedade da escolha dos centroides. Como visto anteriormente, os primeiros centroides são escolhidos aleatoriamente e acredita-se

que, dependendo do centroide selecionado, será mais ou menos difícil para o algoritmo estabilizar o recálculo dos centroides.

Tabela 9 – Tempo de clusterização variando o algoritmo

ALGORITMO	ELEMENTOS CLUSTERIZADOS	*CLUSTERS* GERADOS	TEMPO
K-Means	1000	41	42s
K-Means	1000	72	64s
K-Means	1000	185	1051s (~17min)
K-Means	1000	191	54 s
Median Shift	1000	389	42s
Median Shift	1000	383	45s
Median Shift	1000	381	60s
K-Means	3000	20	614s (~10min)
K-Means	3000	28	732s (~12min)
K-Means	3000	46	1039s (~17min)
K-Means	3000	86	635s (~10min)
K-Means	3000	215	1644s (~27min)
K-Means	3000	322	2807s (~47min)
K-Means	3000	313	921s (~15min)
Median Shift	3000	580	870s (~14min)
Median Shift	3000	593	1166s (~19min)
Median Shift	3000	590	812s (~13 min)
MedimShift	3000	309	6140s (~1h 42min)
K-Means	6175	494	3742s (~62 min)
K-Means	6175	494	3712s (~62 min)
K-Means	6175	494	3803s (~63 min)
Median Shift	6175	914	16325s (~272 min)
Median Shift	6175	925	16407s (~273min)
Median Shift	6175	936	15755s (~262min)

Para melhor visualização dos dados, também foi feito um gráfico mostrando as diferenças de tempo entre os dois algoritmos, esse gráfico está na Figura 23.

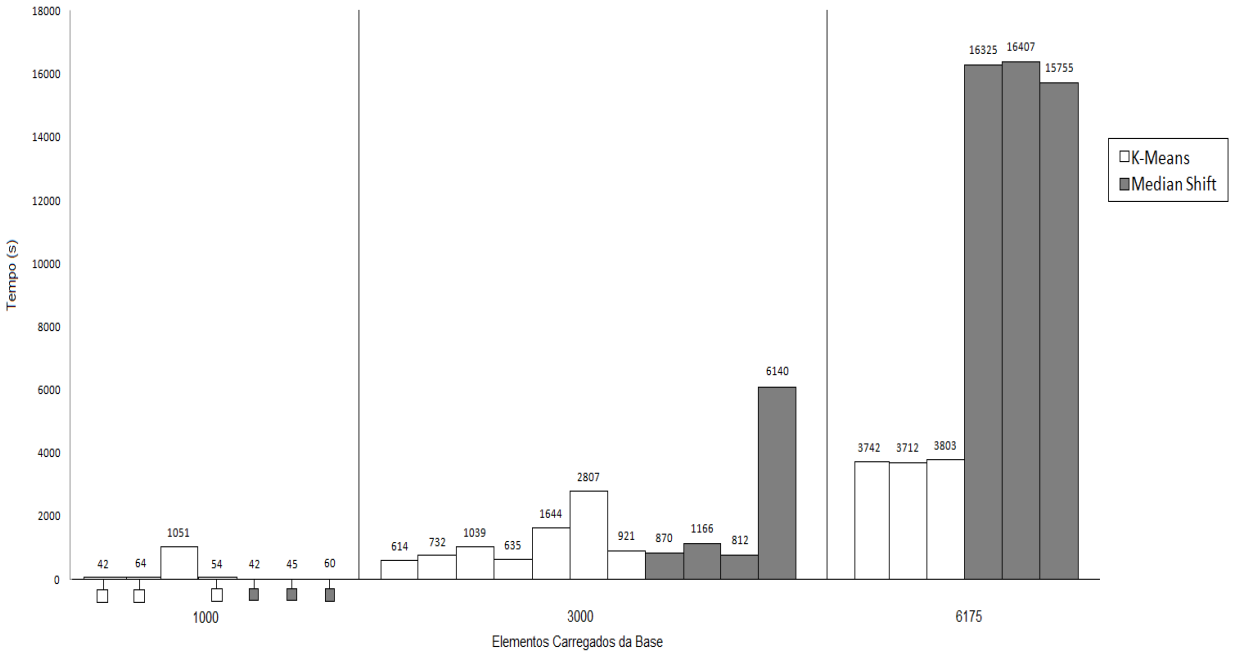

Figura 23 - Comparação de tempo de clusterização dos dois algoritmos

Comparando os tempos de execução dos dois algoritmos, para as execuções com 3000 elementos, verificou-se um melhor desempenho do algoritmo Median Shift. Este, além de gerar resultados com maior quantidade de *clusters*, também fez isso com um tempo muito próximo ou até menor que os resultados do K-Means, com exceção da primeira execução do Median Shift, que obteve um tempo bastante alto em comparação com as outras. Uma justificativa para isso é a carga de dados em memória, que geralmente na primeira vez que é feita leva mais tempo.

Quanto aos tempos de execução para 6175 elementos, o K-Means aparentemente obteve melhores resultados, porém durante os testes notou-se que o K-Means não conseguia gerar resultados para quantidades de elementos muito grandes,

por isso foi necessário adicionar um limitador ao algoritmo para que o mesmo não executasse eternamente o cálculo dos centróides. Esse limitador se baseia na quantidade de vezes que os centroides são recalculados. Com isso, caso a quantidade de vezes que o cálculo é executado passe de certo limite, o resultado é retornado mesmo que os centroides não tenham convergido. Esse limitador foi utilizado durante os testes de qualidade e tempo de clusterização, por isso o tempo de execução do K-Means acabou se tornando menor que o do Median Shift para as clusterizações com 6175 elementos.

Além dos testes de tempo de execução, foram feitos testes de qualidade de resultados. A Figura 24 mostra as distribuições de frequência de precisão para as execuções com 3000 elementos.

Como é possível observar na Figura 24, o algoritmo K-Means aparentemente obteve bons resultados quanto à qualidade das clusterizações. Novamente, notou-se uma concentração de valores abaixo de 0.12 e outra acima de 0.94, porém diferente do Median Shift. A maior parte dos *clusters* obteve precisão acima de 0.94, o que é um ótimo resultado.

Para poder visualizar melhor essa tendência, foram calculados os valores de moda, média e mediana para esses valores de precisão. Eles podem ser vistos na Tabela 10.

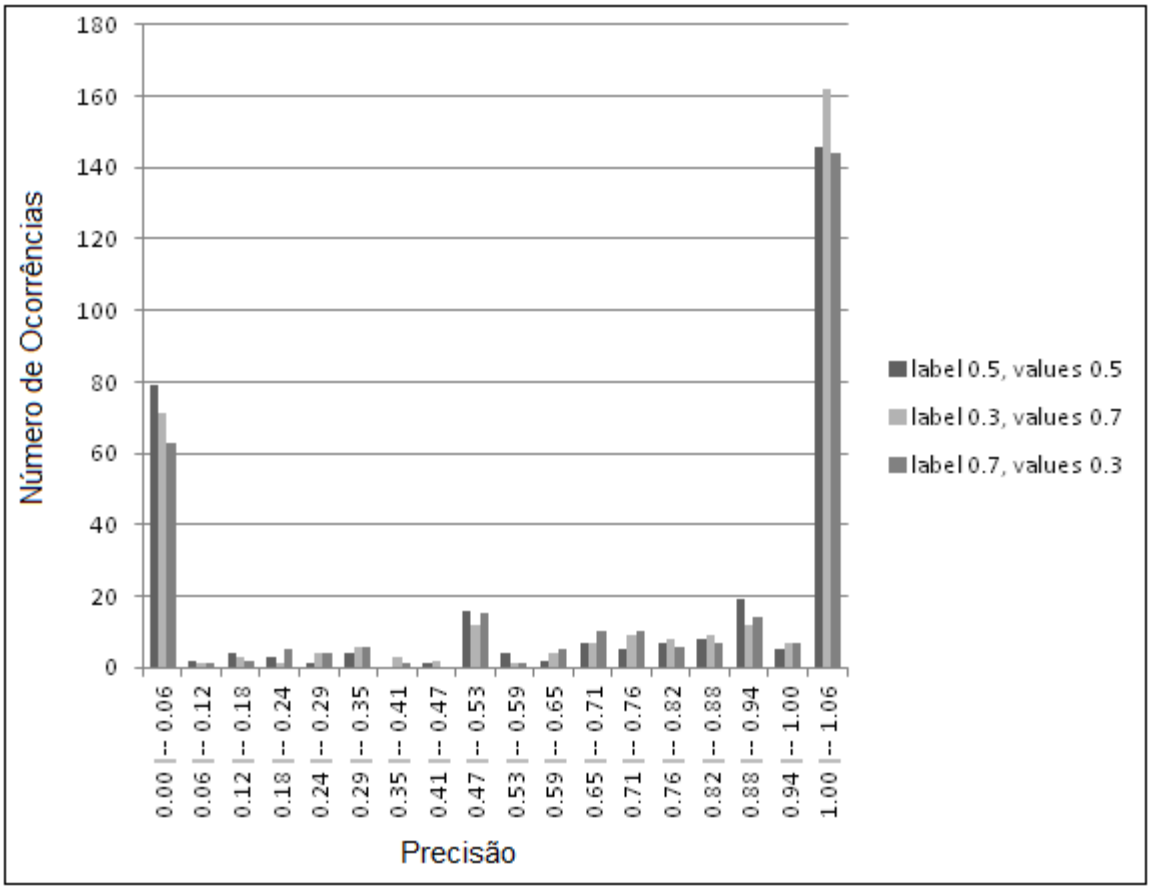

Figura 24 – Resultados de qualidade com K-Means – 3000 elementos

A análise da tabela confirma esse resultado satisfatório, visto que as três médias obtiveram valores acima de 0.67, as medianas acima de 0.92 e os valores de moda foram todos 1.03. Esses resultados indicam uma maior frequência de valores altos de precisão.

Tabela 10 – Valores de Média, Mediana e Moda – K-Means – 3000 elementos

	MEDIA	MEDIANA	MODA
label 0.3, value 0.7	0.7	1.00	1.03
label 0.7, value 0.3	0.68	0.95	1.03
label 0.5, value 0.5	0.67	0.92	1.03

Além dos testes com 3000 elementos, também foram verificados os resultados para 6175 elementos, e novamente foram feitas as distribuições de frequência para as precisões obtidas. Esses resultados estão na Figura 25.

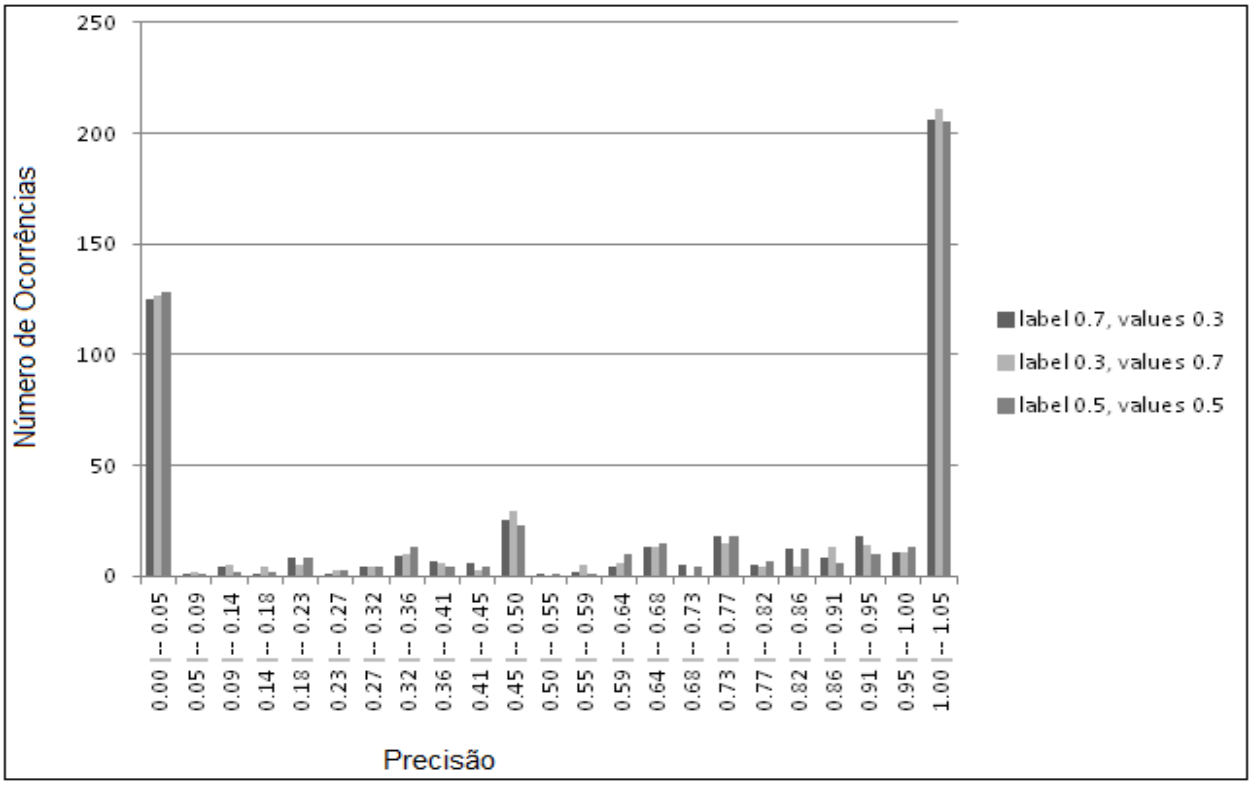

Figura 25 - Resultados de qualidade com K-Means – 6175 elementos

Analisando os dados para os testes com 6175 elementos, nota-se que os resultados obtidos anteriormente se repetem. Novamente obteve-se uma maior frequência de valores com alta precisão. Isso também pode ser verificado pela Tabela 11, que mostra os valores de moda, média e mediana.

Nesse caso, porém os valores de moda, média e mediana foram um pouco menores se comparados com os resultados para 3000 elementos. Apesar disso, permanece a tendência de altos valores de precisão.

Tabela 11 - Valores de Média, Mediana e Moda – K-Means – 6175 elementos

	MEDIA	MEDIANA	MODA
label 0.3, value 0.7	0,63	0,87	1,02
label 0.7, value 0.3	0,64	0,85	1,02
label 0.5, value 0.5	0,63	0,81	1,02

Comparando os resultados de 3000 e 6157 elementos, também se notou que a configuração de peso 0.3 para rótulos e 0.7 para valores obteve os maiores valores de mediana. Os menores valores de mediana, por sua vez, foram obtidos pela configuração de peso 0.5 para rótulos e 0.5 para valores.

Novamente comparando os resultados de qualidade de clusterização dos algoritmos Median Shift e K-Means, e analisando a forma com que eles calculam os cluster, nota-se que essa diferença de resultados entre o Median Shift e o K-Means se deve ao fato de o primeiro calcular cada *cluster* separadamente, enquanto o segundo calcula todos de uma vez. O fato de o Mediam Shift calcular cada *cluster* por vez, faz com que cada elemento seja associado a mais de um *cluster*, mesmo que sua similaridade com o cluster seja mínima dentro de um *threshold.*

No K-Means, por sua vez, cada elemento é associado a somente o *cluster* com o qual ele possua mais similaridade, isso faz com que os elementos incluídos nos *clusters* possuam valores maiores de similaridade, melhorando a qualidade dos *clusters*.

7. Conclusão

Este trabalho descreve algumas otimizações e experimentações realizadas sobre o projeto WF-Sim, que visa encontrar similaridades entre formulários na Web. Essas otimizações visam melhorar a eficácia e eficiência nas buscas e clusterizações do sistema. Também foram feitos testes para verificar quais melhores estratégias para obter melhores resultados com o WF-Sim. Dentre as modificações feitas estão as estratégias de persistência dos *clusters* e a implementação de outro algoritmo de clusterização, o K-Means.

Analisando os resultados dos experimentos quanto às estratégias de persistência, verificou-se um bom desempenho das duas estratégias, considerando a quantidade de *clusters* a ser acessada e filtrada. Entretanto, para uma maior quantidade de *clusters* retornados, a persistência em arquivos obteve melhor desempenho.

Quanto à utilização ou não de ranqueamento durante as buscas, não se notou grande diferença de desempenho, ou seja, os testes com e sem ranqueamento obtiveram tempos próximos. Conclui-se, assim, que organizar o resultado da busca não prejudica tanto o desempenho. Esse resultado era de certa forma esperado, pois o ranqueamento trabalha com um universo de *clusters* reduzido.

Pelo fato de não existir persistência anteriormente, a velocidade de busca do WF-Sim melhorou consideravelmente. Com a adição dessa melhoria, o tempo de busca chegou a ter redução de horas (pois a clusterização era reexecutada a cada sessão de utilização do sistema) para apenas alguns segundos, sendo tais resultados decisivos para tornar o WF-Sim um sistema de busca viável.

Quanto à qualidade das buscas, verificaram-se bons resultados, sendo que a precisão dos centroides ficou acima de 0.6. Também boa parte dos *clusters* retornados obtiveram bons resultados de precisão. Porém, apesar de uma boa quantidade de *clusters* possuírem alta precisão, notou-se também uma grande quantidade de *clusters* ruins. Acredita-se que isso se deve a uma grande quantidade de elementos com rótulos e valores semelhantes, porém com semântica diferente.

Quanto aos experimentos de variação dos pesos no algoritmo Median Shift, verificou-se que para os pesos 0.3 para rótulos e 0.7 para valores obteve-se melhores resultados de precisão. Também se verificou que boa parte dos *clusters* obteve boa precisão (acima de 0.9), porém, novamente notou-se outra boa parte dos *clusters* com baixa precisão. Com isso, conclui-se que talvez seja necessário incluir novos parâmetros de comparação entre os formulários já que em muitos casos, rótulos e valores não são suficientes.

Também de acordo com os resultados dos testes com o K-Means, a princípio ele não apresentou bons resultados quanto ao tempo de execução se comparado com o Median Shift, sendo que algumas vezes ele nem mesmo conseguiu gerar resultados. Porém, após a aplicação de um limitador de execuções foi possível obter bons resultados de tempo de execução para o K-Means.

Quanto à qualidade dos *clusters* gerados, o K-Means apresentou melhores resultados de precisão se comparados com o Median Shift. Cada *cluster* gerado foi analisado um por um, comparando os elementos do *cluster* com seu centroide e calculando a precisão. Sendo assim, a frequência de valores de precisão mais altos foi obtida pelo algoritmo K-Means. Devido a esses resultados, conclui-se que o algoritmo

K-Means pode sim ser utilizado como um bom substituto para o Median Shift no WF-Sim.

Por fim, conclui-se que a busca por similaridade em formulários é uma linha de pesquisa promissora e que ainda tem muito a ser explorada, sendo que este trabalho de conclusão de curso contribuiu para a produção de dois artigos: um deles foi aceito no evento IADIS WWW/Internet, 2011, Rio de Janeiro, tratando sobre a qualidade de busca do projeto WF-Sim, e o outro foi aceito no evento ERBD 2012 em Curitiba, e tratou das persistências de *clusters* criadas para o projeto.

7.1 Trabalhos Futuros

Como trabalho futuro sugere-se avaliar melhor a escalabilidade das estratégias com um volume maior de dados, dado que existem milhões de *web forms* disponíveis atualmente. Assim seria interessante realizar testes com uma base maior e verificar se os desempenhos, tanto das estratégias de persistência quanto das clusterizações, permanecem aceitáveis.

Como trabalho futuro também seria interessante melhorar o limitador de execuções do K-Means de forma a garantir que a execução pare ao encontrar algum padrão de execuções repetidas.

Algo que prejudicou os resultados dos testes foi a grande quantidade de sujeira na base de dados. Sendo assim, sugere-se também como trabalho futuro a criação de algoritmos de limpeza dos dados que possam remover heterogeneidades na grafia dos rótulos e valores, como por exemplo, remoção de sufixos e prefixos, *stop words* e etc.

Outra sugestão é que futuramente seja adotado um dicionário semântico como o Wordnet que permita a descoberta de similaridades entre dados com grafias diferentes, mas que são semanticamente iguais.

Referências

BERGMAN. K. Michael. The *Deep web*: Surfacing Hidden Value. Disponível em: http://brightplanet.com/images/uploads/12550176481-*Deep web*whitepaper.pdf Data de acesso: 27/11/2011.

GONÇALVES, Rodrigo ; D'Agostini, C. S. ; Silva, F. R. ; DORNELES, C. F. ; MELLO, R. S. .A Similarity Search Approach for *Web forms*. In: IADIS International Conference IADIS WWW/Internet, 2011, Rio de Janeiro. Proceedings of the IADIS International Conference IADIS WWW/Internet, 2011

JUANICÓ – Environmental Consultants Ltd. What is an information system on environmental information? Disponível em: http://www.juanico.co.il/main%20frame%20-%20english/issues/information%20systems.htm Data de acesso: 06/12/2011

Cohen, W., Ravikumar, P. & Fienberg, S. (2003), A comparison of string distance metrics for name-*matching* tasks, in `Proc. of the IJCAI-2003 Workshop on Information Integration on the Web (IIWeb-03)', Citeseer, pp. 73-78.

Dorneles, C. F. e. a. (2004), `Measuring similarity between collection of values', Proc. of the ACM WIDM p. 56.

Jouili, S., Tabbone, S. & Lacroix, V. (2010), Median graph shift: A new *cluster*ing algorithm for graph domain, in `Proc. of the 20th International Conference on Pattern Recognition', Washington, DC, USA, pp. 950-953.

MacQueen, J. B. (1967). "Some Methods for classification and Analysis of Multivariate Observations". In: Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. p. 281–297

Ricardo A. Baeza-Yates,Berthier A. Ribeiro-Neto: Modern Information Retrieval - the concepts and technology behind search, 2nd edition Pearson Education Ltd., Harlow, England 2011

Halevy, A., Madhavan, J., Afanasiev, L., Antova, L. (2009). “Harnessing the *Deep web*: Present and Future”, In: Conference on Innovative Data Systems Research (CIDR).

DEMIREL, Orhan. Software Development Blog, Kmeans *cluster*ing. Disponível em: http://www.orhandemirel.com/blog/kmeans-*cluster*ing/ Data de acesso: 21/05/2012

WordNet, A lexical database for English. Disponível em: http://wordnet.princeton.edu/ Data de acesso: 02/06/2012

Anexo 1 - Artigo

Estratégias de Persistência de *Clusters* em uma Técnica de Casamento por Similaridade para *Web Forms*[5]

Filipe Roberto Silva, Ronaldo dos Santos Mello

Departamento de Informática e Estatística (INE) – Centro Tecnológico (CTC)
Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brasil

{filipesilva.sc,ronaldo}@inf.ufsc.br

Abstract. *Most data available currently in Web are just accessible by queries in Web forms. These hidden data bases that are just revealed in response to those queries are called Deep Web. One way to search contents in the Deep Web is to provide to the user relevant Web forms to his/her interests (car dealers, airfare, etc) and the own user can formulate his/her queries. This paper presents and evaluates strategies for clusters' persistence in a Web forms' similarity search system called WF-Sim with the purpose of identifying similar Web forms efficiently and effectively.*

Resumo. *Grande parte dos dados disponíveis atualmente na Web só é acessível através de consultas formuladas sobre Web forms. Estes bancos de dados "escondidos" e revelados apenas como resposta a estas consultas são denominados de Deep Web. Uma das formas de busca a conteúdos na Deep Web é fornecer ao usuário Web forms relevantes para os seus interesses (revendas de automóveis, passagens aéreas, etc) e o próprio usuário realiza suas buscas. Este trabalho apresenta e avalia estratégias de persistência de clusters de afinidade em um mecanismo de casamento (matching) de Web forms por similaridade chamado WF-Sim, de forma que seja possível identificar Web forms semelhantes entre si de forma eficiente e eficaz.*

1. Introdução

A Internet está presente na vida de muitas pessoas. Qualquer dúvida, pesquisa, notícia e etc, pode ser alcançada através da Internet utilizando as populares máquinas de busca, como Google ou Bing. Porém, o que poucos sabem é que essas buscas atingem apenas a superfície da Web. Grande parte do conteúdo na *Web* encontra-se escondido em bancos de dados que não são acessados por tais máquinas de busca. A

[5] Este trabalho é parcialmente financiado pelo CNPq através do projeto WF-Sim (Nro. processo: 481569/2010-3) e de uma bolsa de produtividade (Nro. processo: 307992/2010-1).

esse conteúdo é dado o nome de *Deep Web* ou Web oculta (*Hidden Web*)[6]. O conteúdo da *Deep Web* é revelado em páginas dinâmicas, geradas automaticamente conforme requisições do usuário realizadas sobre *Web form*s (*Web forms*). Assim sendo, tais páginas não podem ser indexadas pelas máquinas de busca.

Dada a dificuldade para a descoberta e indexação de todo o conteúdo da *Deep Web*, uma abordagem possível de busca e que é o foco deste trabalho, é pesquisar somente por campos de *Web form*s, porém indexando os campos destes *Web form*s e permitindo a busca por outros *Web form*s similares, ou seja, considerando um processo de *matching* de *Web forms*. Para entender melhor esse processo, é interessante entender melhor como é formado um *Web form*.

Um *Web form* possui diversos campos, são campos de texto, *check boxes*, *combo boxes* e etc, que permitem ao usuário preencher digitando ou selecionando um valor. Como é possível observar na Figura 2, os *Web form*s possuem rótulos, que são utilizados para que o usuário saiba o que colocar em cada campo. Esses campos podem conter os já citados valores para seleção, como é possível ver na Figura 2 para o campo '*Select what you have*:', que possui duas possibilidades de preenchimento: '*I have a bike* e *'I have a car*'.

Assim, esses valores podem ser utilizados para a comparação de um *Web form* com outro, tanto os valores de rótulo, quando os valores de preenchimento.

[6] http://brightplanet.com/images/uploads/12550176481-deepWebwhitepaper.pdf

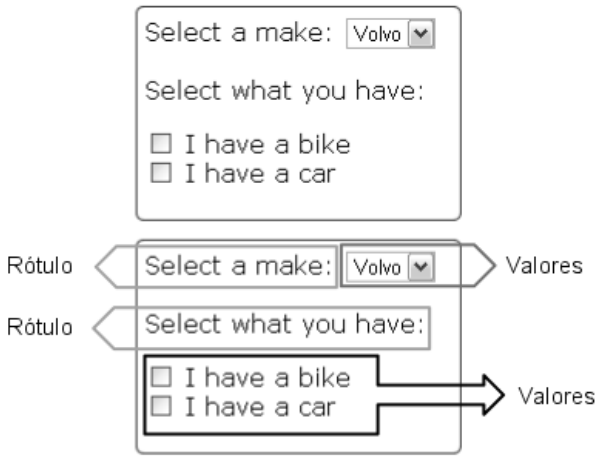

Figura 26. *Web form* e suas características

Este trabalho é uma continuação dos resultados obtidos com o projeto *WF-Sim*, projeto este que está em andamento e visa tornar possíveis buscas por similaridade sobre *Web form*s. Com o projeto já é possível agrupar por semelhança *Web form*s previamente extraídos da Web, indexar e a partir de um *Web form* de entrada, encontrar outros semelhantes. Experimentos foram realizados sobre o *WF-Sim* e o mesmo já tem resultados satisfatórios de busca em termos de acurácia (Gonçalves et. al. 2011). Entretanto, existem algumas melhorias a serem feitas e este trabalho visa implementar algumas dessas melhorias.

Especificamente, o objetivo desse trabalho é apresentar otimizações no algoritmo de *clusterização* de dados de *Web form*s utilizado pelo *WF-Sim* em termos de persistência desses *clusters*. O projeto, como foi implementado, realiza agrupamentos dos *Web form*s a cada vez que é executado. Assim, ele somente guarda os *clusters* em memória, não persistindo os mesmos. A contribuição deste trabalho é a implementação e teste de duas soluções para a persistência desses dados: uma em bancos de dados e outra em arquivos. Cada alternativa foi testada visando verificar a melhor solução

para o problema. Resultados de experimentos preliminares são relatados, demonstrando qual das duas estratégias se mostrou mais promissora.

Os demais capítulos detalham o desenvolvimento deste trabalho. O capítulo 2 aborda o projeto *WF-Sim*, com foco na atividade de *clusterização*. O capítulo 3 mostra como foram desenvolvidas as persistências dos *clusters* e o capítulo 4 descreve os experimentos preliminares. Por fim, no capítulo 5 são apresentadas as conclusões e atividades futuras.

2. WF-Sim

A descoberta de similaridades entre *Web form*s a partir de *strings* que representam valores em um campo é um grande desafio, já que muitas vezes duas palavras totalmente diferentes significam a mesma coisa, ou ainda duas palavras iguais em contextos diferentes possuem significados diferentes. Um exemplo disso seriam 2 *Web form*s com campos '*Manufactures*' e '*Year of Manufacture*'. Apesar dos nomes semelhantes, eles possuem significados totalmente diferentes.

O trabalho em questão é uma extensão do projeto *WF-Sim* (*Web form Similarity*), um projeto em desenvolvimento pelo Grupo de Banco de Dados da UFSC[7] que visa realizar buscas por *Web form*s similares, tentando resolver parte desta problemática. A seguir são abordados alguns detalhes sobre o funcionamento do *WF-Sim*.

[7] http://www.gbd.inf.ufsc.br

2.1 Funcionamento

O *WF-Sim* trata *Web form*s dividindo-os em elementos. Cada elemento representa um campo contendo um rótulo e opcionalmente um conjunto de valores. Essa divisão é exemplificada pela Figura 4 onde, no primeiro elemento, temos o rótulo "*Select a make*" e uma relação de valores iniciada por "Volvo".

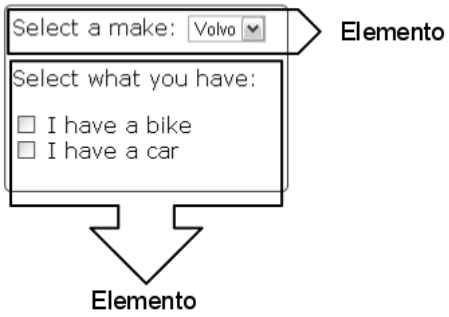

Figura 27. Divisão de um *Web form* em elementos

O projeto visa desenvolver um sistema de busca por similaridade para *Web form*s que é composto por 3 módulos principais. Estes módulos são: *clusterização*, a *indexação* e *busca*. Neste sistema, os elementos são clusterizados de acordo com uma métrica de similaridade (ver Seção 2.2) e então indexados. A partir dessa indexação é possível realizar buscas, ou seja, a partir de um *Web form* de entrada são encontrados outros semelhantes. Essa busca, ao invés de comparar elemento por elemento de todos os *Web form*, compara somente os centróides, que são os elementos com maior afinidade com todos os demais elementos no *clusters* e que representam o *cluster* como um todo. Assim, a partir das comparações com os centróides, encontra-se os grupos de elementos semelhantes àquele *Web form* de entrada.

2.2 Cálculo da Similaridade

O *WF-Sim* utiliza para o cálculo de similaridade dos rótulos, a métrica *TF-IDF* (Cohen et. al. 2003) e para a similaridade dos valores, a métrica *SubSetSim* (Dorneles 2004), ambas escolhidas por possuírem os melhores resultados se comparados a outras métricas. A partir do cálculo das similaridades dos rótulos e dos valores separadamente, é feita uma média ponderada desses valores, como é possível ver na Equação 1, onde l_1 e l_2 são os rótulos, vl_1e vl_2 os valores, *labelSim* e *valuesSim* são os cálculos de similaridade dos rótulos e valores respectivamente. Por ser uma média ponderada, é possível aplicar pesos diferentes para cada atributo do elemento (*labelWeight* e *valuesWeight*), sendo a soma dos pesos igual a um (1). Vale lembrar que para elementos pertencentes a um mesmo *web form* não é calculada a similaridade, visto que o objetivo é encontrar similaridades entre *web forms* diferentes.

$$ElementSim = labelSim(l_1, l_2) * labelWeight + valuesSim(vl_1, vl_2) * valuesWeight \quad (1)$$

2.3 Clusterização

A *clusterização* dos elementos é feita utilizando uma extensão do algoritmo Median Shift (Jouili, Tabbone & Lacroix 2010), onde os elementos medianos são definidos como centróides dos *clusters*. Primeiramente, um raio de *clusterização* é determinado pelo usuário. Assim, é verificado em cada elemento, quais outros elementos estão contidos dentro desse raio. Esse processo é chamado de cálculo de vizinhança.

Pelo fato dos elementos não estarem contidos em um espaço vetorial, é necessário calcular todas as similaridades entre os elementos e transformar estes valores em distâncias. Como os valores de similaridade estão no intervalo de ‘0’ a ‘1’,

sendo que o valor ‘0’ significa ser diferente e o valor ‘1’ ser igual, a distância é calculada utilizando a Equação 2, sendo que para valores de similaridade igual a ‘1’ é aplicada diretamente a distância zero.

$$Distância = \frac{1}{Similaridade} \quad (2)$$

Com isso, são calculadas as distâncias entre os elementos, estes são agrupados e os centróides definidos.

3. Persistência dos Clusters

Pelo fato do *WF-Sim* ainda estar em desenvolvimento, suas *clusterizações* não são persistidas, ou seja, a cada execução, todos os formulários são novamente processados e clusterizados. Isso foi feito devido à necessidade inicial de analisar os resultados obtidos, tendo mais relevância a qualidade dos resultados do que a velocidade para obtê-los.

O cálculo dos *clusters* é algo que demanda muito processamento e por isso é demorado. Portanto, uma das contribuições deste trabalho é a extensão do *WF-Sim* para permitir o armazenamento dos *clusters* calculados e, assim, executar o agrupamento somente quando novas *Web forms* forem consideradas.

Duas estratégias de persistência dos *clusters* foram implementadas: a persistência em diretórios de arquivos e a persistência em um banco de dados. As próximas seções detalham estas estratégias.

3.1 Persistência em arquivos

Uma das estratégias para guardar os dados dos *clusters* foi através de arquivos gerenciados diretamente pelo WF-Sim. Como existem alguns parâmetros que podem variar na criação dos *clusters*, cada nova *clusterização* foi armazenada em pastas de acordo com os parâmetros de entrada definidos. Com isso é possível armazenar várias *clusterizações* com diferentes configurações.

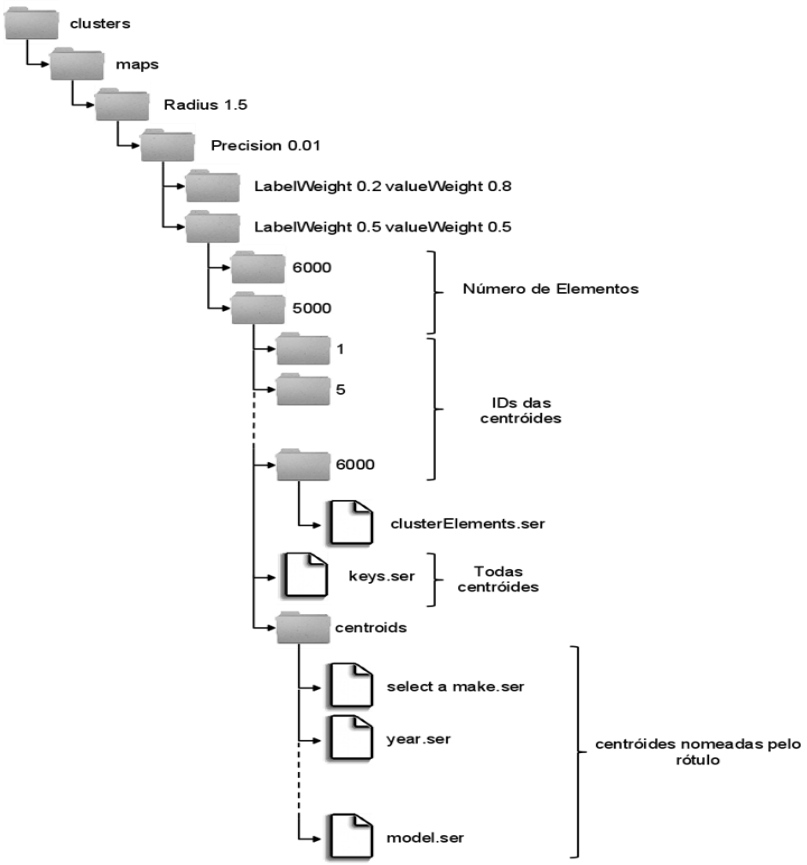

Figura 3. Organização dos *clusters* em pastas e arquivos

Como é possível ver na Figura 10, existem configurações de peso para rótulos e valores, precisão e raio de *clusterização*. Além disso, as *clusterizações* estão divididas pelo número de elementos utilizados na *clusterização*. Esta divisão considera outro

parâmetro de configuração do WF-sim, que é o número de elementos a ser carregado da base de dados de *Web forms*.

Cada *clusterização* gera vários *clusters* e, como visto anteriormente, cada *cluster* possui um centróide. Assim, os elementos de cada *cluster* foram armazenados em pastas nomeadas pelo identificador do centróide e os centróides foram armazenados separadamente em duas formas. A primeira forma foi um arquivo contendo todos os centróides identificados com um ID gerado pelo sistema. Desta forma, mantêm-se em memória somente as informações dos centróides e se carrega os *clusters* desejados (mantidos em pastas específicas identificadas pelos IDs dos centróides), conforme a busca do usuário. A segunda forma foi armazenar cada centróide em um arquivo separado nomeado pelo próprio rótulo. Com isso, é possível fazer buscas acessando diretamente as pastas dos *clusters* desejados.

Para possibilitar a criação de arquivos nomeados pelos rótulos dos centróides, foi necessário um pré-processamento que retira caracteres especiais que não são aceitos pelo sistema de arquivos. Ainda, alguns rótulos possuíam muitos caracteres e por isso foram filtrados rótulos com mais de 250 caracteres.

3.2 Persistência em BD

Além da persistência em arquivos, também se decidiu testar o armazenamento dos *clusters* em um banco de dados relacional. Para isso, foi necessário o projeto do banco de dados. A modelagem conceitual definida pode ser vista na Figura 114.

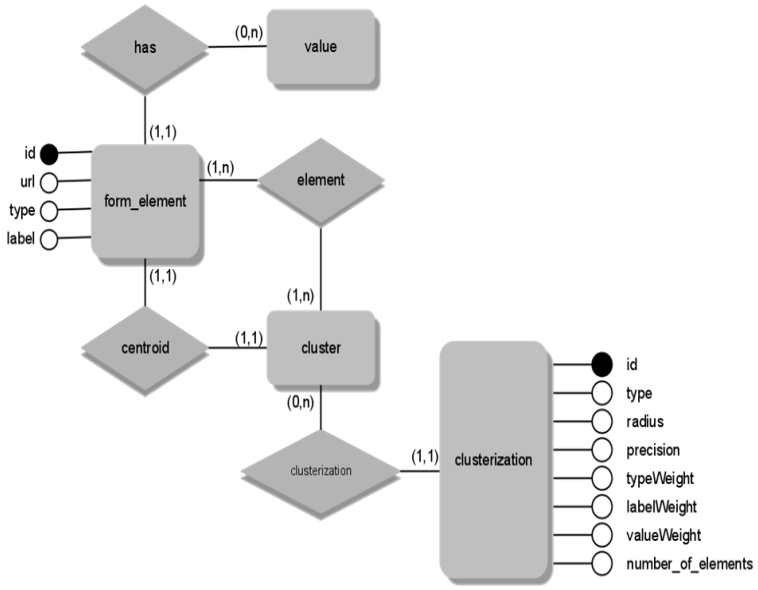

Figura 4. Modelagem conceitual da persistência dos *clusters*

Assim como no armazenamento em arquivos, encontrou-se o problema de como separar as *clusterizações* de acordo com cada configuração. Para resolver isso, uma entidade "*clusterization*" foi criada para manter estas configurações. Assim, cada *cluster* está relacionado a uma *clusterização*, podendo haver *clusters* iguais com configurações diferentes.

Como os centróides também são elementos, optou-se por não criar uma entidade centróides. Ao invés disso, a entidade '*cluster*' possui uma relação '*centroid*' com '*form_element*' e outra relação '***element'*** também com '*form_element*'.

4. Experimentos

Para confirmar a melhoria de desempenho das estratégias propostas e compará-las, foram relizados alguns experimentos preliminares de busca sobre as *clusterizações* persistidas. Esses testes foram realizados em um computador com Intel Core2 Duo

CPU T5800, 2 GHz, 3GB RAM, Windows XP SP3, sendo 6157 o número de elementos clusterizados, divididos em 910 *clusters*.

Para a realização dos testes foi criado um formulário base (*template*) no domínio de automóveis com 10 elementos. Este é um formulário típico para busca de automóveis para venda ou aluguel. Este domínio foi escolhido pois é o que existe em maior quantidade na base de dados de *Web forms*. A Tabela 1 mostra os elementos considerados no *template* com suas restrições de valor, quando existentes.

Tabela 1. Web Form template considerado nos testes

ID	Rótulo	Valor	ID	Rótulo	Valor
1	select make	ford hyundai nissan Toyota	6	model	any model blazer caravan century civic cobalt enclave
2	min price	20,000 30,000 40,000 50,000 60,000 70,000 80,000	7	miliage	20,000 30,000 40,000 50,000 60,000 70,000 80,000
3	choose	2000	8	used car	acura

	year	2001 2002 2003 2004 2005 2006			alfa romeo audi bmw fiat chevrolet cadillac
4	max price	200,000 300,000 400,000 500,000 600,000 700,000 800,000	9	sort order	by price by make by date by year
5	include	all vehicles new vehicles used vehicles certified vehicles	10	modified	

Para variar a quantidade de *clusters* retornados na busca, em alguns testes foram retirados elementos do *template* sendo que os mantidos foram aqueles que retornavam mais dados. Assim, testes foram feitos com 1, 5 e 10 elementos, para fins de avaliação de desempenho de um número crescente de elementos a ser considerado nas buscas.

Para encontrar os centróides desejados, tanto no banco de dados quanto nos arquivos, dividiu-se o *template* em seus elementos e, para cada elemento, foram encontrados os centróides cujos rótulos possuíssem os mesmos termos do elemento de entrada. Encontrados esses centróides, foi aplicado o cálculo de similaridade do WF-Sim sobre eles. Os centróides com similaridade abaixo de 0.7 foram desconsiderados para manter no resultado apenas dados com alta similaridade.

Além da busca retornando os *clusters* mais similares, também foram realizados testes utilizando o mecanismo de ranqueamento (*ranking*) do WF-Sim. Esse ranqueamento é feito durante as consultas para verificar, num *ranking*, quais *clusters* são mais semelhantes à consulta. O WF-Sim originalmente utilizava todos os *clusters* calculados na *clusterização* para o ranqueamento, produzindo um *overhead* na geração do resultado da consulta. Para reduzir este *overhead*, o mecanismo de ranqueamento foi modificado neste trabalho para considerar somente os *clusters* recuperados das buscas em arquivos e no banco de dados.

Nos experimentos o foco foi avaliar o desempenho em termos de tempo das duas estratégias. A avaliação da relevância dos *clusters* retornados não foi tratada nesse trabalho, visto que os resultados das *clusterizações* já foram validados pelo WF-sim.

A Tabela 2 apresenta os resultados dos experimentos. Os tempos mostrados são a média de 3 execuções para cada busca. As consultas foram *disjuntivas*, ou seja, são retornados os *clusters* que casam com pelo menos um (1) dos elementos do *template*. Por isso, quanto mais elementos são considerados como entrada para a busca, maior é a quantidade de *clusters* recuperados.

Tabela 2. Resultados dos Experimentos

Tipo de teste	**#Elementos**	***#Clusters* retornados**	**Tempo (seg.)**
Arquivos s/ ranking	1	9	0,968
Arquivos c/ ranking	1	9	1,135
BD s/ ranking	1	9	1,588
BD c/ ranking	1	9	1,098
Arquivos s/ ranking	5	38	2,162
Arquivos c/ ranking	5	38	2,338
BD s/ ranking	5	38	4,052
BD c/ ranking	5	38	5,031
Arquivos s/ ranking	10	65	2,807
Arquivos c/ ranking	10	65	3,135
BD s/ ranking	10	65	5,531
BD c/ ranking	10	65	5,766

Os testes com 1 elemento consideraram buscas envolvendo o elemento com ID 1 do *template* na Tabela 1. Os testes com 5 elementos consideraram os elementos com ID de 1 a 5 e os testes com 10 elementos consideraram todos os elementos do *template*.

De acordo com a Tabela 2, a estratégia de persistência em arquivos sem considerar ranqueamento do resultado foi a mais rápida, tornando-se 2,2 vezes mais lenta com 5 vezes mais elementos considerados na busca e 2,8 vezes mais lenta com 10 vezes mais elementos considerados. Isso mostra uma tendência de redução do

crescimento do *overhead* à medida que o número de elementos de entrada aumenta. A estratégia de arquivos com ranqueamento ficou, em média, 1,1 vez mais lenta, representando um *overhead* pouco significativo.

A estratégia de persistência em banco de dados sem ranqueamento tornou-se 2,5 vezes mais lenta com 5 vezes mais elementos considerados na busca e 3,4 vezes mais lenta com 10 vezes mais elementos considerados. Há uma tendência semelhante à da estratégia de persistência de arquivos, apesar de haver um *overhead* maior. A estratégia de banco de dados com ranqueamento ficou, em média, para 5 e 10 elementos, 1,1 vez mais lenta, representando também um *overhead* pouco significativo. Ela só obteve desempenho superior à estratégia sem ranqueamento para a busca com 1 elemento de entrada, sendo isso provavelmente uma pequena anomalia em termos de acesso ao banco de dados.

Comparando as estratégias arquivo e banco de dados, ambas com ranqueamento, observa-se um desempenho praticamente equivalente para buscas com 1 elemento. Já para buscas com 5 e 10 elementos, a estratégia de banco de dados mostra-se, em média, 2 vezes mais lenta.

5. Conclusão

Este trabalho descreve duas estratégias para armazenamento de *clusters* de dados de *Web forms*, bem como uma avaliação de desempenho de acesso para ambas no contexto do projeto WF-Sim, que realiza buscas por similaridade em *Web forms*. Comparado com trabalhos relacionados (Chang and Cheng 2007; Silva 2007; Hao et. al. 2010; Hong et. al. 2010; Nguyen et. al. 2010), verifica-se que os mesmos não

apresentam detalhes sobre a persistência da *clusterização* ou utilizam apenas bancos de dados para armazenar os *clusters* gerados. Além disso, nenhum desses trabalhos utiliza o algoritmo de *clusterização* proposto pelo WF-Sim para o contexto de *Web forms*. Este algoritmo foi escolhido por gerar automaticamente centróides de cada *cluster*. Essa facilidade se mostrou útil para fins de indexação, pois apenas os centróides são indexados e servem como base para buscas, ao invés de se indexar todos os elementos do *cluster*.

Analisando os resultados dos experimentos, verificou-se um bom desempenho das duas estratégias, considerando a quantidade de *clusters* a ser acessada e filtrada. Entretanto, para uma maior quantidade *clusters* retornados, a persistência em arquivos obteve melhor desempenho. Mesmo assim, é necessário avaliar melhor a escalabilidade das estratégias com um volume maior de dados, dado que existem milhões de *Web forms* disponíveis atualmente.

Quanto à utilização ou não de ranqueamento, não se notou grande diferença de desempenho, ou seja, os testes com e sem ranqueamento obtiveram tempos próximos. Conclui-se, assim, que organizar o resultado da busca não prejudica tanto o desempenho. Esse resultado era de certa forma esperado, pois o ranqueamento trabalha com um universo de *clusters* reduzido. Como trabalho futuro seria interessante realizar esses testes com uma base maior e verificar se esse desempenho permanece aceitável.

Pelo fato de não existir persistência anteriormente, a velocidade de busca do WF-Sim melhorou consideravelmente. Com a adição dessa melhoria, o tempo de busca chegou a ter redução de horas (pois a *clusterização* era re-executada a cada sessão de

utilização do sistema) para apenas alguns segundos, sendo tais resultados decisivos para tornar o WF-Sim um sistema de busca viável.

Vale lembrar que os testes realizados se referem apenas a atividades de busca. A *cluster*ização no WF-Sim ainda é um processo demorado, porém, não é executado frequentemente. Mesmo assim, a persistência do resultado desse processamento melhorou a desempenho das buscas no WF-Sim. Conforme mencionado anteriormente, pretende-se melhorar o desempenho da *cluster*ização e também avaliar o desempenho com uma implementação alternativa utilizando o K-Means (MacQueen 1967), um algoritmo popular para *cluster*ização de dados.

Referências

Chang, K. C.-C and Cheng, T. (2007) "Entity Search Engine: Towards Agile Best-Effort Information Integration over the Web". In: Proceedings of the 2nd Conference on Innovative Data Systems Research (CIDR). p.108-113.

Cohen, W., Ravikumar, P. and Fienberg, S. (2003) "A Comparison of String Distance Metrics for Name-Matching Tasks". In: Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web (IIWeb-03). p.73-78.

Dorneles, C. F. et. al. (2004) "Measuring Similarity between Collection of Values". In: Proceedings of 6th ACM CIKM International Workshop on Web Information and Data Management (WIDM 2004). p.56-63.

Gonçalves, R.; D'Agostini, C. S.; Silva, F. R.; Dorneles, C. F.; Mello, R. S. (2011). "A Similarity Search Approach for *Web forms*". In: Proceedings of the IADIS International Conference IADIS WWW/Internet.

Hao, L.; Wan-Li,Z. and Fei, R., (2010) "Describing the Semantic Relation of the Deep Web Query Interfaces Using Ontology Extended LAV". Journal of Software, v. 5, n. 1. p.89-98.

Hong, J., He, Z., and Bell, D. A. (2010). "An evidential approach to query interface matching on the Deep Web". Information Systems, v.35, n.2. p.140-148.

Jouili, S., Tabbone, S., and Lacroix, V. (2010). "Median graph shift: A new clustering algorithm for graph domain". In: Proceedings of the 20th International Conference on Pattern Recognition. p. 950-953.

MacQueen, J. B. (1967). "Some Methods for classification and Analysis of Multivariate Observations". In: Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. p. 281–297.

Nguyen, T., Nguyen, H. and Freire, J. (2010). "PruSM: A Prudent Schema Matching Approach for Web Forms". In: Proceedings of 19th ACM Conference on Information and Knowledge Management (CIKM). p.1385-1388.

Silva, A., Freire, J., and Barbosa, L. (2007). "Organizing Hidden-Web Databases by Clustering Visible Web Documents". In: Proceedings of the International Conference on Data Enginering (ICDE). p.326-335.

Anexo 2 - Fonte

```
package net.wfsim.clusterizer;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Random;
import java.util.Set;

import net.wfsim.log.StatusDisplay;
import net.wfsim.matching.MatchException;
import net.wfsim.matching.Matcher;
import net.wfsim.structures.graph.Graph;
import net.wfsim.structures.graph.table.InvalidFieldException;
import net.wfsim.structures.graph.table.InvalidTupleException;
import net.wfsim.structures.graph.table.Schema;
import net.wfsim.structures.graph.table.Table;

/**
 * Clusterizador
 * @author Filipe
 * @param <T>
 * @version 03/05/2012 - Criacao do Algoritmo
 */

public class KMeansClusterer<T> implements Clusterer<T>{

    static String CENTER = "center";
    static String NEIGHBOR = "neighbor";
    static String DISTANCE = "distance";
    static String KEY = "key";
    protected Matcher<T> m;
    private Map<T, Map<T, Double>> neighbors;

    public KMeansClusterer(Matcher<T> m) {
       System.out.println("Usando KMeans");
              this.m = m;
       }

       @Override
       public Map<T, List<T>> clusterize(T[] objects, double radius, float precision, int
numberOfClusters) {

              System.out.println("Quantidade "+objects.length);

              Map<T, List<T>> gms = new HashMap<T, List<T>>();
              if (objects.length > 0) {

                     neighbors = this.calcularVizinhos(objects, radius);

                     Object[] centroids = this.getRandomCentroids(objects, numberOfClusters);
                     Object[] oldCentroids = centroids;

                     boolean notConverged = true;
                     int tries = 0;

                     while(notConverged && tries<5000)
                     {
                            for(T element : objects)
                            {
                                   Map<T, Double> nearBy;
```

```
                                        nearBy = neighbors.get(element);

                                        int nearestCentroidIndex = -1;
                                        Double nearCentroidDistance = Double.MAX_VALUE;
                                        for(int i=0;i<centroids.length;i++)
                                        {
                                                Double distance = nearBy.get(centroids[i]);
                                                if(distance != null)
                                                {
                                                        if(nearCentroidDistance > distance)
                                                        {
                                                                nearCentroidDistance = distance;

                                                                nearestCentroidIndex = i;
                                                        }
                                                }
                                        }

                                        if(nearestCentroidIndex > -1)
                                        {

        if(!element.equals(centroids[nearestCentroidIndex]))
                                                {

        if(!gms.containsKey(centroids[nearestCentroidIndex]))
                                                        {
                                                                List<T> elements = new
ArrayList<T>();

                                                                elements.add(element);
                                                                gms.put((T)
centroids[nearestCentroidIndex], elements);
                                                        }
                                                        else

        gms.get(centroids[nearestCentroidIndex]).add(element);
                                                }
                                        }
                                }

                                Object[] newCentroids = this.getCentroids(centroids, gms);
                                if(this.equals(centroids,newCentroids) ||
this.equals(oldCentroids,newCentroids))
                                {
                                        notConverged = false;
                                }
                                else
                                {
                                        oldCentroids = centroids;
                                        centroids = newCentroids;
                                        tries++;
                                        if(tries < 5000)
                                                gms.clear();

                                }
                        }
                        if(tries >= 5000)
                                System.out.println("Out of tries");

                }
```

```
		return gms;
	}

	private boolean equals(Object[] anObject,Object[] anotherObject)
	{
//		System.out.println("_____________________");
		if(anObject.length != anotherObject.length)
			return false;
		for(int i=0;i<anObject.length;i++)
		{
//			System.out.println(anObject[i].toString());
//			System.out.println(anotherObject[i].toString());
			if(anObject[i].toString().compareTo(anotherObject[i].toString()) != 0)
				return false;
		}
		return true;
	}

	private Object[] getCentroids(Object[] actualCentroids, Map<T, List<T>> actualClusters)
	{
		Object[] newCentroids = new Object[actualCentroids.length];



		for(int i=0;i<actualCentroids.length;i++)
		{
			if(actualClusters.containsKey(actualCentroids[i]))
			{
				List<T> elements = actualClusters.get(actualCentroids[i]);
				Double minorDistance = Double.MAX_VALUE;
				int minorDistanceIndex = -1;
				for(int j=0;j<elements.size();j++)
				{
					Double distanceSum = 0d;
					for(T elementComparing : elements)
					{
						Double distance =
this.getDistanceBetween(elements.get(j), elementComparing);
						if(distance!=null)
							distanceSum += distance;
					}
					distanceSum = distanceSum/elements.size();
					if(minorDistance > distanceSum)
					{
						minorDistance = distanceSum;
						minorDistanceIndex = j;
					}
				}
				if(minorDistance!=-1)
				{
					newCentroids[i] = elements.get(minorDistanceIndex);
				}
				else
				{
					newCentroids[i] = actualCentroids[i];
				}
			}
			else
			{
```

```
                        newCentroids[i] = actualCentroids[i];
                }
        }

        return newCentroids;
}

private Double getDistanceBetween(T anElement,T otherElement)
{
        Map<T, Double> nearBy;

        nearBy = neighbors.get(anElement);
        Double centroidNearBy = nearBy.get(otherElement);
        if(centroidNearBy != null)
        {
                return centroidNearBy;
        }
        return null;
}

@Override
public Map<T, List<T>> clusterize(T[] objects, double radius, float precision) {

        int numberOfClusters = (int) Math.sqrt(objects.length);
        return this.clusterize(objects, radius, numberOfClusters);
}

@Override
public Graph clusterizeAsGraph(T[] objects, double radius, float precision) {
        return null;
}

private Object[] getRandomCentroids(T[] objects, int numberOfClusters)
{
 Random r = new Random();
 int[] check = new int[numberOfClusters];
        Object[] centroids = new Object[numberOfClusters];
 for (int i = 0; i < numberOfClusters; i++) {
     int rand = r.nextInt(objects.length);
     if (check[i] == 0) {
        centroids[i] = objects[rand];
          check[i] = 1;
     } else {
         i--;
     }
 }
        return centroids;
}

private Map<T,Map<T,Double>> calcularVizinhos(T[] objects, double radius) {

 Map<T,Map<T,Double>> neighbors = new HashMap<T,Map<T,Double>>();

 long antes = System.currentTimeMillis();
 System.out.println(" ComeÃ§ando a calcular vizinhancas ");

 T object, neighbor;
 double distance;
 for (int i = 0; i < objects.length; i++) {
```

```
            object = objects[i];
            try {

               if(neighbors.containsKey(object))
               {
                      neighbors.get(object).put(object, 0d);
               }
               else
               {
                      HashMap<T, Double> newCenter = new HashMap<T, Double>();
                      newCenter.put(object, 0d);
                      neighbors.put(object, newCenter);
               }

                for (int j = i + 1; j < objects.length; j++) {
                    neighbor = objects[j];
                    distance = 0;

                    distance = m.distance(object, neighbor);
                    if (distance < radius) {

                      neighbors.get(object).put(neighbor, distance);

                      if(neighbors.containsKey(neighbor))
                      {
                             neighbors.get(neighbor).put(object, distance);
                      }
                      else
                      {
                             HashMap<T, Double> newNeigbor = new HashMap<T, Double>();
                             newNeigbor.put(object,distance);
                             neighbors.put(neighbor, newNeigbor);
                      }
                    }
                }

            } catch (MatchException e) {
                e.printStackTrace();
            }
        }
        System.out.println(" Terminou de calcular vizinhancas = " +
(System.currentTimeMillis() - antes) / 1000.0);
        return neighbors;
    }

}
```

```
package net.wfsim.clusterizer;

import net.wfsim.matching.FormElementMatching;
import net.wfsim.matching.Matcher;
import net.wfsim.structures.form.FormElement;

public class FactoryOfClusterers {

	public static final int MEDIAM_SHIFT = 0;
	public static final int KMEANS = 1;
	private static Matcher<FormElement> matcher = new FormElementMatching();

	public static <T> Clusterer<FormElement> getClusterer(int clusterer)
	{
		Clusterer<FormElement> clustererObject;
		switch (clusterer) {
		case MEDIAM_SHIFT:
			clustererObject = (Clusterer<FormElement>) new
MediumShiftClusterer<T>((Matcher<T>) matcher);
			break;
		case KMEANS:
			clustererObject = (Clusterer<FormElement>) new
KMeansClusterer<T>((Matcher<T>) matcher);
			break;
		default:
			clustererObject = (Clusterer<FormElement>) new
MediumShiftClusterer<T>((Matcher<T>) matcher);
			break;
		}
		return clustererObject;
	}
}
```

```
package net.wfsim.loader;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import com.mysql.jdbc.exceptions.jdbc4.MySQLIntegrityConstraintViolationException;

import net.wfsim.conf.ConfigLoader;
import net.wfsim.structures.form.Form;
import net.wfsim.structures.form.FormElement;


public class BancoDeDados {

	private Statement statement;
	private int actualConfiguration;

	public BancoDeDados()
	{
		Connection connection = ConexaoMySQL.getConexaoMySQL();
		try {
			statement = connection.createStatement();
			actualConfiguration = this.getIdOfActualConfigurationon();
		} catch (SQLException e) {
			e.printStackTrace();
		}
	}

	private int getIdOfActualConfigurationon()
	{
		int id = -1;
		String query = "SELECT id FROM configuration WHERE" +
				" number_of_elements="+ConfigLoader.numberOfElements()+
				" AND type = 'maps'"+
				" AND radius = "+ConfigLoader.radius()+
				" AND typeWeight = "+ConfigLoader.typeWeight()+
				" AND labelWeight = "+ConfigLoader.labelWeight()+
				" AND valueWeight = "+ConfigLoader.valueWeight()/*+
				" AND precision = "+ConfigLoader.precition()*/;
		try {
			ResultSet result = statement.executeQuery(query);
			while(result.next())
				id = Integer.parseInt(result.getString("id"));

		} catch (SQLException e) {
			e.printStackTrace();
		}
		return id;
	}

	private void insertFormElement(FormElement element,String table)
	{
		String queryVerify = "SELECT * FROM "+table+" WHERE id="+element.getId()+";";
```

```
		try {

			ResultSet result = statement.executeQuery(queryVerify);
			if(!result.next())
			{
				String query = "INSERT INTO "+table+" VALUES
("+element.getId()+",'"+element.getUrl()+"','"+element.getType()+"','"+element.getLabel()+"')"
;
				statement.executeUpdate(query);
			}
		} catch (SQLException ex) {

			ex.printStackTrace();
		}
	}

	public void insertFormElement(FormElement element)
	{
		this.insertFormElement(element,"form_element");
	}

	public void insertValuesToElement(List<String> values,int id)
	{
		for(String value : values)
		{
			value = value.replaceAll("'", "Â´");
			value = value.replaceAll("\"\\\"", "|");
			try {
				statement.executeUpdate("INSERT INTO value VALUES
('"+value+"',"+id+")");
			} catch (SQLException e) {
			}

		}
	}

	public void removeFormElement(FormElement element)
	{
		String query = "DELETE FROM form_element WHERE id="+element.getId()+";";
		try {
			statement.executeUpdate(query);
		} catch (SQLException e) {
			e.printStackTrace();
		}
	}

	public void insertClusterization()
	{
		String queryVerify = "SELECT * FROM configuration WHERE
id="+actualConfiguration+";";
		String query = "INSERT INTO configuration VALUES
(1,"+ConfigLoader.numberOfElements()+",'maps',"+ConfigLoader.radius()+","+ConfigLoader.typeWei
ght()+","+ConfigLoader.labelWeight()+","+ConfigLoader.valueWeight()+","+ConfigLoader.precition
()+")";
		try {
			ResultSet result = statement.executeQuery(queryVerify);
			if(!result.next())
			{
				statement.executeUpdate(query);
```

```
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        public void removeActualClusterization()
        {
            String query = "DELETE FROM configuration WHERE id="+actualConfiguration;
            try {
                statement.executeUpdate(query);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        public void insertClusterWithCentroid(FormElement centroid)
        {
            String queryVerify = "SELECT * FROM cluster WHERE
clusterization="+actualConfiguration+" AND centroid="+centroid.getId();
            String query = "INSERT INTO cluster VALUES
("+actualConfiguration+","+centroid.getId()+")";
            try {
                ResultSet result = statement.executeQuery(queryVerify);
                if(!result.next())
                    statement.executeUpdate(query);

            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        public void removeClusterWithCentroid(FormElement centroid)
        {
            String query = "DELETE FROM cluster WHERE clusterization="+actualConfiguration+"
AND centroid="+centroid.getId();
            try {
                statement.executeUpdate(query);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        public void insertClusterElementToCluster(FormElement element, FormElement centroid)
        {
            String query = "INSERT INTO cluster_to_element VALUES
("+centroid.getId()+","+element.getId()+")";
            String queryVerify = "SELECT * FROM cluster_to_element WHERE
cluster="+centroid.getId()+" AND element="+element.getId();
            try {
                ResultSet result = statement.executeQuery(queryVerify);
                if(!result.next())
                    statement.executeUpdate(query);

            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
```

```
	public void removeClusterElementFormCluster(FormElement element, FormElement centroid)
	{
		String query = "DELETE FROM cluster_to_element WHERE
cluster="+centroid.getId()+" AND element="+element.getId();
		try {
			statement.executeUpdate(query);
		} catch (SQLException e) {
			e.printStackTrace();
		}
	}

	public List<FormElement> selectElementsFromCluster(FormElement centroid)
	{
		String query =	"SELECT id,url,type,label FROM form_element AS f INNER
JOIN cluster_to_element AS c ON c.element = f.id WHERE c.cluster = "+centroid.getId();
		List<FormElement> elements = new ArrayList<FormElement>();
		try
		{
			ResultSet result = statement.executeQuery(query);
			while(result.next())
			{
				FormElement element = new FormElement();
				element.setId(Integer.parseInt(result.getString("id")));
				element.setUrl(result.getString("url"));
				element.setType(result.getString("type"));
				element.setLabel(result.getString("label"));

				elements.add(element);
			}

			for(FormElement element : elements)
			{
				List<String> values = new ArrayList<String>();
				String valuesQuery = "SELECT value FROM value WHERE element =
"+element.getId();

				ResultSet valuesResult = statement.executeQuery(valuesQuery);
				while(valuesResult.next())
				{
					values.add(valuesResult.getString("value"));
				}
				element.setValues(values);
			}
		}
		catch (SQLException e) {
			e.printStackTrace();
		}
		return elements;
	}

	public List<FormElement> selectCentroidsLikeTemplate(FormElement template)
	{
		String regex = "";
		String[] partes = template.getLabel().split(" ");
		for(int i=0;i<partes.length;i++)
		{
			if(partes[i].length()>2)
			{
				regex += partes[i]+".*|";
```

```
                        }
                }
                regex = regex.substring(0, regex.length()-1);
                String query;
                if(regex != "")
                        query = "SELECT id,url,type,label FROM form_element AS f INNER JOIN
cluster AS c ON c.centroid = f.id WHERE c.clusterization = "+actualConfiguration+" AND f.label
REGEXP '"+regex+"'";
                else
                        query = "SELECT id,url,type,label FROM form_element AS f INNER JOIN
cluster AS c ON c.centroid = f.id WHERE c.clusterization = "+actualConfiguration;

                List<FormElement> elements = new ArrayList<FormElement>();
                try
                {
                        ResultSet result = statement.executeQuery(query);
                        while(result.next())
                        {
                                FormElement element = new FormElement();
                                element.setId(Integer.parseInt(result.getString("id")));
                                element.setUrl(result.getString("url"));
                                element.setType(result.getString("type"));
                                element.setLabel(result.getString("label"));

                                if(!elements.contains(element) &&
template.approximateSimilarityTo(element) >= 0.7)
                                        elements.add(element);
                        }

                        for(FormElement element : elements)
                        {
                                List<String> values = new ArrayList<String>();
                                String valuesQuery = "SELECT value FROM value WHERE element = "+
element.getId();
                                ResultSet valuesResult = statement.executeQuery(valuesQuery);
                                while (valuesResult.next()) {
                                        values.add(valuesResult.getString("value"));
                                }
                                element.setValues(values);
                        }
                }
                catch (SQLException e) {
                        e.printStackTrace();
                }
                return elements;
        }

        public Map<FormElement, List<FormElement>> selectSimilarClusters(FormElement template)
        {
                Map<FormElement, List<FormElement>> clusters = new HashMap<FormElement,
List<FormElement>>();

                List<FormElement> centroids = this.selectCentroidsLikeTemplate(template);

                for(FormElement centroid : centroids)
                {
                        //System.out.println(centroid.getLabel());
                        clusters.put(centroid, selectElementsFromCluster(centroid));
                }
```

```
                return clusters;
        }

        public Map<FormElement, List<FormElement>> selectSimilarClusters(Form form)
        {
                Map<FormElement, List<FormElement>> result = new HashMap<FormElement,
List<FormElement>>();

                for(FormElement element : form.getElements())
                {
                        result.putAll(selectSimilarClusters(element));
                }
                return result;
        }
}
```

```
package net.wfsim.loader;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Set;

import net.wfsim.conf.ConfigLoader;
import net.wfsim.log.Log;
import net.wfsim.log.StatusDisplay;
import net.wfsim.structures.form.FormElement;
import net.wfsim.structures.graph.Graph;

public class ClusterSerializer {

	public static void serializeCluster(Map<FormElement, List<FormElement>> cluster, int
id) {
		try {
			String dirName = "clusters/maps"+ConfigLoader.clusterer()+"/radius_" +
ConfigLoader.radius() + "/precision_" + ConfigLoader.precition() + "/labelWeight_" +
ConfigLoader.labelWeight() + "_valueWeight_" + ConfigLoader.valueWeight() + "/";
			File dir = new File(dirName);
			if (!dir.exists())
				dir.mkdirs();
			FileOutputStream fos = new FileOutputStream(dirName + "cluster_with_" +
id + "_elements.ser");
			ObjectOutputStream oos = new ObjectOutputStream(fos);
			oos.writeObject(cluster);
			oos.close();
		} catch (FileNotFoundException e) {

		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	public static void serializeCluster(Graph cluster, int id) {
		try {
			String dirName = "clusters/graphs/radius_" + ConfigLoader.radius() +
"/precision_" + ConfigLoader.precition() + "/labelWeight_" + ConfigLoader.labelWeight() +
"_valueWeight_" + ConfigLoader.valueWeight() + "/";
			File dir = new File(dirName);
			if (!dir.exists())
				dir.mkdirs();
			FileOutputStream fos = new FileOutputStream(dirName + "cluster_with_" +
id + "_elements.ser");
			ObjectOutputStream oos = new ObjectOutputStream(fos);
			oos.writeObject(cluster);
			oos.close();
		} catch (FileNotFoundException e) {
```

```
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	public static Map<FormElement, List<FormElement>> deserializeCluster(int id) {
		HashMap<FormElement, List<FormElement>> cluster = null;
		String dirName = "clusters/maps"+ConfigLoader.clusterer()+"/radius_" +
ConfigLoader.radius() + "/precision_" + ConfigLoader.precition() + "/labelWeight_" +
ConfigLoader.labelWeight() + "_valueWeight_" + ConfigLoader.valueWeight() + "/";
		String clusterName = "cluster_with_" + id + "_elements.ser";
		try {
			FileInputStream fis = new FileInputStream(dirName + clusterName);
			ObjectInputStream ois = new ObjectInputStream(fis);
			cluster = (HashMap<FormElement, List<FormElement>>) ois.readObject();
			ois.close();
		} catch (FileNotFoundException e) {
			System.err.print("Cluster: '" + clusterName + "' nao encontrado\nO
cluster sera calculado\n");
		} catch (IOException e) {
			e.printStackTrace();
		} catch (ClassNotFoundException e) {
			e.printStackTrace();
		}
		return cluster;
	}

	public static Graph deserializeClusterAsGraph(int id) {
		Graph cluster = null;
		String dirName = "clusters/graphs/radius_" + ConfigLoader.radius() +
"/precision_" + ConfigLoader.precition() + "/labelWeight_" + ConfigLoader.labelWeight() +
"_valueWeight_" + ConfigLoader.valueWeight() + "/";
		String clusterName = "cluster_with_" + id + "_elements.ser";
		try {
			FileInputStream fis = new FileInputStream(dirName + clusterName);
			ObjectInputStream ois = new ObjectInputStream(fis);
			cluster = (Graph) ois.readObject();
			ois.close();
		} catch (FileNotFoundException e) {
			System.err.print("Cluster: '" + clusterName + "' nao encontrado\nO
cluster sera calculado\n");
		} catch (IOException e) {
			e.printStackTrace();
		} catch (ClassNotFoundException e) {
			e.printStackTrace();
		}
		return cluster;
	}

	public static void serializeSeparetedClustersAsMap(Map<FormElement, List<FormElement>>
cluster, int id) {

		try {
			String dirName = "separetedClusters/maps/radius_" +
ConfigLoader.radius() + "/precision_" + ConfigLoader.precition() + "/labelWeight_" +
ConfigLoader.labelWeight() + "_valueWeight_" + ConfigLoader.valueWeight() + "/" + id +"/";
			File dir = new File(dirName);
			if (!dir.exists())
```

```
                        dir.mkdirs();
                    File centroidsDir = new File(dirName + "/centroids");
                    if(!centroidsDir.exists())
                        centroidsDir.mkdir();
                    ArrayList<FormElement> keys = new
ArrayList<FormElement>(cluster.keySet());
                    File aCluster;
                    for(FormElement centroid : keys) {
                        aCluster = new File(dirName + centroid.getId());
                        aCluster.mkdir();
                        FileOutputStream fos = new FileOutputStream(dirName +
centroid.getId() + "/clusterElements.ser");
                        ObjectOutputStream oos = new ObjectOutputStream(fos);
                        List<FormElement> forms = cluster.get(centroid);
                        oos.writeObject(forms);
                        oos.close();

                        String name = centroid.getLabel();
                        System.out.println(name);
                        name = name.replaceAll(":", "_");
                        name = name.replaceAll("\\\\", "_");
                        name = name.replaceAll("/", "_");
                        name = name.replaceAll("\"", "_");
                        name = name.replaceAll("\\?", "_");
                        //name = name.replaceAll("\\.", "_");
                        name = name.replaceAll(">", "_");
                        name = name.replaceAll("<", "_");
                        name = name.replaceAll("\\*", "_");
                        name = name.replaceAll("\\|", "_");
                        name = name.replaceAll("'", "_");
                        if(name.length() < 250)
                        {
                            int i = 0;
                            while(new File(dirName +
"/centroids/"+name+(i==0?"":"("+i+")")+".centroid").exists())
                            {
                                i++;
                            }
                            FileOutputStream foscentroid = new
FileOutputStream(dirName + "/centroids/"+name+(i==0?"":"("+i+")")+".centroid");
                            ObjectOutputStream ooscentroid = new
ObjectOutputStream(foscentroid);
                            ooscentroid.writeObject(centroid);
                            ooscentroid.close();
                        }
                    }

                    FileOutputStream fos = new FileOutputStream(dirName+"/keys.ser");
                    ObjectOutputStream oos = new ObjectOutputStream(fos);
                    oos.writeObject(keys);
                    oos.close();

            } catch (Exception e) {
                    e.printStackTrace();
            }
        }

        public static Map<FormElement, List<FormElement>> deserializeSeparetedCluster(int id)
        {
```

```
		HashMap<FormElement, List<FormElement>> cluster = new HashMap<FormElement,
List<FormElement>>();
		List<FormElement> keys = null;
		try{
			String dirName = "separetedClusters/maps/radius_" +
ConfigLoader.radius() + "/precision_" + ConfigLoader.precition() + "/labelWeight_" +
ConfigLoader.labelWeight() + "_valueWeight_" + ConfigLoader.valueWeight() + "/" + id +"/";

			FileInputStream fis = new FileInputStream(dirName + "/keys.ser");
			ObjectInputStream ois = new ObjectInputStream(fis);
			keys =  (List<FormElement>) ois.readObject();
			ois.close();



			File aCluster;
			for(FormElement key : keys)
			{
				aCluster = new File(dirName + key.getId());
				FileInputStream fiss = new FileInputStream(dirName + key.getId()
+ "/clusterElements.ser");
				ObjectInputStream oiss = new ObjectInputStream(fiss);
				cluster.put(key,(List<FormElement>) oiss.readObject());
				oiss.close();

			}

		} catch (Exception e) {
		e.printStackTrace();
		cluster = null;
		}

		return cluster;
	}

	public static void saveToBD(Map<FormElement, List<FormElement>> cluster, int id)
	{
		ArrayList<FormElement> keys = new ArrayList<FormElement>(cluster.keySet());
		BancoDeDados bd = new BancoDeDados();
		bd.insertClusterization();

		List<FormElement> allForms = new ArrayList<FormElement>();

		for(FormElement centroid : keys) {

			if(!allForms.contains(centroid))
				allForms.add(centroid);
			List<FormElement> forms = cluster.get(centroid);
			for(FormElement element : forms)
			{
				if(!allForms.contains(element))
					allForms.add(element);
			}
		}

		System.out.println("Number of elements: "+allForms.size());

		/*for(FormElement element : allForms)
		{
```

```
                        bd.insertFormElement(element);
                        StatusDisplay.showStatusLoading("Inserting elements");
                }*/

                for(FormElement centroid : keys) {

                        bd.insertClusterWithCentroid(centroid);
                        if(!allForms.contains(centroid))
                                allForms.add(centroid);

                        List<FormElement> forms = cluster.get(centroid);
                        for(FormElement element : forms)
                        {
                                bd.insertClusterElementToCluster(element, centroid);
                                if(!allForms.contains(element))
                                        allForms.add(element);
                        }
                        StatusDisplay.showStatusLoading("Inserting clusters");
                }
                /*
                for(FormElement element : allForms)
                {
                        bd.insertValuesToElement(element.getValues(), element.getId());
                        StatusDisplay.showStatusLoading("Inserting values");

                }*/
        }
}
```

```
package net.wfsim.loader;/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */



//Classes necessÃ¡rias para uso de Banco de dados //

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;


/**
 *
 * @author Filipe
 */
public class ConexaoMySQL {

        public static String status = "NÃ£o conectou...";
        //MÃ©todo de ConexÃ£o//

public static java.sql.Connection getConexaoMySQL() {

        Connection connection = null;          //atributo do tipo Connection



try {

// Carregando o JDBC Driver padrÃ£o

String driverName = "com.mysql.jdbc.Driver";

Class.forName(driverName);



// Configurando a nossa conexÃ£o com um banco de dados//

            String serverName = "localhost";    //caminho do servidor do BD

            String mydatabase ="wfsim_clusters";        //nome do seu banco de dados

            String url = "jdbc:mysql://" + serverName + "/" + mydatabase;

            String username = "root";        //nome de um usuÃ¡rio de seu BD

            String password = "";      //sua senha de acesso

            connection = DriverManager.getConnection(url, username, password);


            //Testa sua conexÃ£o//
```

```
            if (connection != null) {
                status = ("STATUS--->Conectado com sucesso!");
            } else {
                status = ("STATUS--->NÃ£o foi possivel realizar conexÃ£o");
            }

            return connection;

        } catch (ClassNotFoundException e) {  //Driver nÃ£o encontrado

            System.out.println("O driver expecificado nao foi encontrado.");
            return null;
        } catch (SQLException e) {
//NÃ£o conseguindo se conectar ao banco
            System.out.println("Nao foi possivel conectar ao Banco de Dados.");
            return null;
        }

    }

    //MÃ©todo que retorna o status da sua conexÃ£o//
    public static String statusConection() {
        return status;
    }

   //MÃ©todo que fecha sua conexÃ£o//
    public static boolean FecharConexao() {
        try {
            ConexaoMySQL.getConexaoMySQL().close();
            return true;
```

```
            } catch (SQLException e) {
                return false;
            }

        }

       //MÃ©todo que reinicia sua conexÃ£o//
        public static java.sql.Connection ReiniciarConexao() {
            FecharConexao();

            return ConexaoMySQL.getConexaoMySQL();
        }
    }
```

```
package net.wfsim.searcher;

import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.apache.tools.ant.DirectoryScanner;

import net.wfsim.conf.ConfigLoader;
import net.wfsim.structures.form.Form;
import net.wfsim.structures.form.FormElement;
import net.wfsim.structures.graph.Graph;


public class FileSearcher<E> {

	public Map<FormElement, List<FormElement>> searchSimilarClusters(FormElement template)
	{
		DirectoryScanner scanner = new DirectoryScanner();
		String[] partes = template.getLabel().split(" ");
		for(int i=0;i<partes.length;i++)
		{
			if(partes[i].length()>2)
				partes[i] = "*"+partes[i]+"*.centroid";
		}

		scanner.setIncludes(partes);
		String dirName = "separetedClusters/maps/radius_" + ConfigLoader.radius() +
"/precision_" + ConfigLoader.precition() + "/labelWeight_" + ConfigLoader.labelWeight() +
"_valueWeight_" + ConfigLoader.valueWeight() + "/" + 6157 +"/";
		scanner.setBasedir(dirName + "centroids");
		scanner.setCaseSensitive(false);
		scanner.scan();
		String[] files = scanner.getIncludedFiles();

		List<FormElement> centroids = new ArrayList<FormElement>();

		for(String clusterName : files)
		{
			try
			{
				FileInputStream fis = new FileInputStream(dirName + "/centroids/"
	+ clusterName);
				ObjectInputStream ois = new ObjectInputStream(fis);
				FormElement centroid = (FormElement) ois.readObject();
				if(!centroids.contains(centroid) &&
template.approximateSimilarityTo(centroid) >= 0.7)
					centroids.add(centroid);
				ois.close();

			} catch (Exception e) {
				e.printStackTrace();
				//System.err.println(e.getMessage());
			}
		}
```

```
		Map<FormElement, List<FormElement>> clusters = new HashMap<FormElement,
List<FormElement>>();

		for(FormElement c: centroids)
		{
			//ve similaridade
			//System.out.println(c.getLabel());
			try
			{
				FileInputStream fis = new FileInputStream(dirName + "/"
+c.getId()+"/clusterElements.ser");
				ObjectInputStream ois = new ObjectInputStream(fis);
				clusters.put(c, (List<FormElement>) ois.readObject());
				ois.close();


			}
			catch (Exception e) {
				System.err.println(e.getMessage());
			}
		}

		return clusters;
	}

	public Map<FormElement, List<FormElement>> searchSimilarClusters(Form form)
	{
		Map<FormElement, List<FormElement>> result = new HashMap<FormElement,
List<FormElement>>();

		for(FormElement element : form.getElements())
		{
			result.putAll(searchSimilarClusters(element));
		}
		return result;
	}

}
```