Revista de Sistemas de Informação da FSMA
n. 22 (2018) pp. 14-25

SISTEMAS DE INFORMAÇÃO
http://www.fsma.edu.br/si/sistemas.html

TRILHA PRINCIPAL

JMatching: Correspondência entre esquemas através de amostras do usuário

Jacson Luiz Matte, *Universidade do Oeste de Santa Catarina (UNOESC) - Chapecó*
Denio Duarte, *Universidade Federal da Fronteira Sul (UFFS) - Chapecó*

***Resumo*—Correspondência entre esquemas é um problema de geração de correspondência entre elementos de dois ou mais esquemas que é aplicado, principalmente, na área de integração de dados. A tarefa de correspondência entre esquemas exige que os usuários tenham conhecimento dos esquemas dos bancos de dados envolvidos na correspondência. Porém, usuários interessados em construir uma base de dados de assuntos de seu interesse, geralmente têm conhecimento apenas de alguns exemplos (amostras) armazenados em um esquema no banco de dados ou em um resultado de uma consulta. Neste caso, eles acabam se deparando com a dificuldade de entender como armazenar tais dados ou como eles são representados. Neste contexto, este trabalho propõe uma ferramenta que, com base em um banco de dados relacional de origem e amostras, constrói uma correspondência entre esquema e as amostradas dadas. Foram conduzidos experimentos para identificar a coerência dos esquemas e a base de dados geradas, a usabilidade, o tempo de execução e o consumo de memória da ferramenta. Resultados indicam que *JMatching* pode ser utilizada por usuários não especialista em banco de dados mas conhecedores da aplicação para construírem seus próprios banco de dados.**

***Palavras-chave*—Correspondência entre esquemas, Amostras, Integração de dados, Banco de Dados.**

JMatching: A Sample-Driven Schema Matching.

***Abstract*—Schema mappings are high-level specifications that describe the relationship between schemas. In real-life applications schema mappings can be a quite complex task since users must know the semantic of the schemas to be matched. However, there exist applications that users know the domain but not the details of the schemas. Thus, tools that can build a target schema based on samples are demanded by users non-experts in computer science but experts in the application domain. This work proposes an approach for schema mapping based on samples to help non-experts users to build their own database. Some experiments were conducted to verify schemas and built database correcteness. Besides, we measured the amount of memory needed and execution time. In all experiments, our work showed good results.**

***Index Terms*—Schema Matching, Sample-Driven, Data Integration, Database.**

Autor correspondente: Jacson Luiz Matte, *jacson.matte@unoesc.edu.br*

I. Introdução

ESQUEMAS são representações formais das estruturas de um determinado objeto [1]. Um sistema gerenciador de banco de dados (SGBD) normalmente oferece três níveis de esquemas (ou visões) para os usuários, a chamada arquitetura de três níveis ANSI-SPARC [2]: visões externa, conceitual e física. A visão externa é aquela que abstrai os usuários dos detalhes de como os dados são manipulados pela aplicação, ela é representada pelos formulários e relatórios. Já a visão conceitual representa a estrutura lógica dos objetos armazenados no banco de dados, ela é aquela utilizada pelos engenheiros de software para construir as aplicações. Finalmente, a visão física representa como esquema do banco de dados é armazenado fisicamente no disco, geralmente apenas os responsáveis pela manutenção e *tuning* do banco de dados têm acesso a este nível. Dentro do contexto de correspondência entre esquemas, o nível conceitual é o mais utilizado pois descreve as tabelas armazenadas no banco de dados (*e.g.*, nome e tipo dos atributos) mas abstrai detalhes físicos do armazenamento.

A correspondência entre esquemas visa integrar dois ou mais esquemas em um único esquema baseada em um conjunto de regras de conversão [1]. Usuários não especialistas em banco de dados podem enfrentar dificuldades ao realizar a tarefa de correspondência, pois a construção das regras de conversão exige um conhecimento especializado dos esquemas envolvidos na correspondência. Enquanto usuários não especialistas conhecem o nível externo do banco de dados, é necessário conhecer o nível conceitual para entender a semântica dos esquemas envolvidos na correspondência [1]. Abordagens para automatizar a correspondência entre esquemas, abstraindo usuários do nível conceitual do banco de dados, vêm sendo propostas para atender a demanda de usuários não especialistas em banco de dados mas especialistas no domínio da aplicação (*e.g.*, [3]–[6]).

A correspondência entre esquemas pode ser definida como: dados um esquema de origem S e um esquema de destino T, construir um conjunto de regras Σ (também chamadas de expressões de mapeamento) que representam a correspondência dos elementos de S e T. Há basicamente duas técnicas para construir Σ [6]–[8]: (i) as expressões de correspondências são criadas a partir dos elementos dos

esquemas (correspondência baseada em elementos), e (*ii*) as expressões são criadas baseadas nos elementos e nos dados presentes nos esquemas (correspondência baseada em instâncias).

Geralmente, a correspondência entre esquemas é feita com ajuda de usuários conhecedores da aplicação e do banco de dados tanto no nível externo quanto no nível conceitual. Isso impede que usuários especialistas na aplicação e conhecedores apenas do nível externo construam suas próprias correspondências. A dificuldade está associada à compreensão do significado dos esquemas e suas correspondências [6]. Por exemplo, suponha que um usuário queira fazer a correspondência entre dois banco de dados acadêmicos. Caso este usuário seja o especialista do domínio, ele conhecerá apenas as instâncias das tabelas e não o esquema conceitual, ou seja, não é de interesse deste usuário saber que em um banco de dados o nome do estudante está armazenado no atributo `nome` e no outro `nestud`. O interesse é que um determinado nome represente o nome do estudante. Nesta situação hipotética, o usuário não teria a capacidade técnica de indicar na correspondência que `nome` e `nestud` representam o mesmo dado nos dois bancos de dados.

Assim, os usuários especialistas habitualmente conhecem algumas amostras que ou estão armazenadas em uma base de dados ou são resultados de uma consulta. Quando necessitam criar a sua própria base de dados podem encontrar dificuldades de entender como armazenar e representar os dados. Isto impossibilita que esses usuários, sem conhecimento da estrutura e semântica do banco de dados, possam construir seus mapeamentos entre os dados existentes e a nova visão desejada. Para isso, alguns trabalhos abordam o problema em que, dados uma base de dados de origem com o esquema provavelmente desconhecido pelo usuário e um conjunto de amostras, é gerado um esquema como saída baseado nas duas entradas.

Este artigo apresenta uma proposta de ferramenta, chamada *JMatching*, para correspondência entre esquemas através de amostras, destinada a usuários especialistas na aplicação, mas desconhecedores do nível conceitual do banco de dados. A ferramenta *JMatching* constrói um mapeamento derivando automaticamente os mapeamentos com base em um banco de dados de origem e amostras do usuário. A partir das amostras são apresentados resultados em formato de visões dos esquemas e das consultas. O diferencial da *JMatching* em relação a outras ferramentas com o mesmo objetivo (*e.g.*, [6], [9], [10]) é o não uso de índices auxiliares (*i.e.*, índices invertidos) para pesquisar as amostras no banco de dados, sendo uma ferramenta não invasiva/intrusiva, ou seja, o banco de dados de origem não necessita de nenhum novo componente para o *JMatching* funcionar. Perceba que a maioria dos Sistemas Gerenciadores de Banco de Dados (SGBD) oferecem recursos para criação de índices invertidos, porém, deve-se considerar o custo de manutenção de tais índices para operação do SGBD, além do usuário não se beneficiar da existência dos mesmos para executar as transações no banco de dados. O *JMatching* pode ser utilizado em qualquer SGBD sem a criação de estruturas auxiliares que muitas vezes não são utilizadas no uso diário de um SGBD.

Este trabalho está organizado da seguinte forma: a próxima seção apresenta os conceitos de correspondência entre esquemas. A Seção III apresenta os trabalhos similares a esta proposta. As Seções IV e V apresentam, respectivamente, a proposta deste trabalho e os experimentos realizados. Finalmente, a Seção VI conclui este trabalho e apresenta algumas direções futuras de pesquisa.

II. Correspondência entre Esquemas

SEGUNDO [6], uma correspondência entre esquemas é a transformação de instâncias (e esquemas) de bancos de dados de origem para uma nova instância única (esquema) que representa os banco de dados envolvidos. A correspondência entre esquemas pode ser feita manualmente, ou pode ser obtida por meio de um sistema de mapeamento que gera automaticamente a correspondência entre os esquemas a partir de uma representação visual [3].

A correspondência é uma operação principal na manipulação de informações de esquemas que é definida como: dados dois ou mais esquemas como entrada, o resultado irá produzir um mapeamento entre elementos dos esquemas que se correspondem semanticamente [7]. Correspondência de esquemas envolve também a descoberta de um conjunto de consultas (possivelmente unitário) que transformam os dados de origem em uma nova estrutura, correspondendo aos dados de destino [4].

A correspondência é estabelecida por um conjunto de elementos mapeados, como mostrado na Figura 1. Neste mapeamento, os elementos da tabela *Rios* e da tabela *Lagos* são correspondidos para a geração de um esquema destino representado pela tabela *Recursos de Água* (indicado por setas contínuas na figura). Perceba que os elementos de *Rios* e *Lagos* são mapeados, resultando em um novo esquema que representa a tabela *Recursos de Água* (indicado pela seta tracejada).

A Figura 1 ainda identifica atributos de uma correspondência individual, na qual um ou mais elementos do esquema de *Rios* podem se corresponder com um ou mais elementos do esquema de *Lagos*, indicando a representação de conceitos similares nos dois esquemas relacionais. O número de atributos de esquemas diferentes que se relacionam no mapeamento é chamado de grau. Os graus podem ser *1:1* (um para um) como no caso entre de *IdRio* (Tabela *Rios*) e *IDLago* (tabela *Lagos*) que se associam ao atributo *IDAgua* (tabela *Recursos de Água*); *N:1* ou *1:N* (muitos para um) que corresponde ao caso de *CidadeNascente* e *UFNascente* de *Rios* que se associam ao atributo *Localizacao* de *Lagos* gerando o mapeamento *Localizacao* em *Recursos de água*; e, finalmente, tem-se o grau *N:N* (muito para muitos) que envolvem vários atributos dos esquemas envolvidos que, por exemplo, devem ser concatenados para gerarem o atributo do esquema destino.

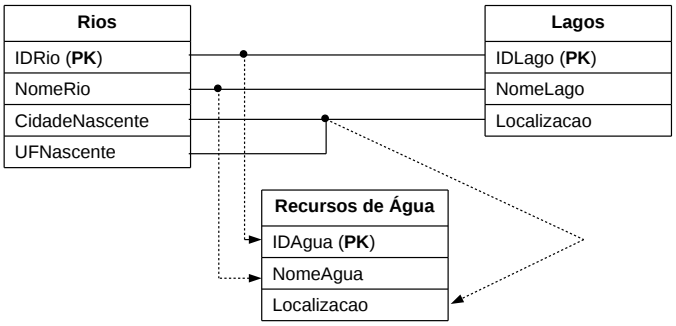

Fig. 1. Correspondência entre esquemas e o problema de geração de correspondências que identificam elementos relacionados em dois esquemas (atributos com nomes e cardinalidades diferentes).

A. A Operação de correspondência

A escolha de uma representação de correspondência é o passo inicial para implementar a correspondência entre esquemas. Existem diversas abordagens que realizam a correspondência, porém elas dependem muito das informações dos esquemas que são utilizados e de suas interpretações [7].

Uma representação de esquema que melhor se adapte a uma determinada abordagem deve ser escolhida. Por exemplo, pode-se utilizar a representação entidade-relacionamento (*ER*) ou orientada a objetos (*OO*) para construir uma determinada correspondência [7].

Algumas abordagens que implementam a correspondência entre esquemas são semelhantes a uma junção (*join*) em banco de dados relacional. Tanto a correspondência quanto a junção, que são operações binárias, determinam pares de elementos correspondentes a sua operação. Todavia, existem algumas diferenças: correspondência atua sobre metadados (elementos de esquemas) e junções em dados (tuplas de tabelas). Além disso, a correspondência é mais complexa que a junção. Cada elemento contido no resultado de uma junção combina apenas um elemento da primeira entrada com um elemento correspondente da segunda entrada. Já um elemento resultante de uma operação de correspondência pode relacionar vários elementos, de ambas as entradas [7].

Para tanto, a semântica que envolve uma junção é especificada por uma expressão de comparação simples, por exemplo, uma condição de igualdade para uma junção natural (*natural join*) que deve suportar todos os elementos de entrada correspondente. De outra maneira, cada elemento em um resultado de uma correspondência pode ter uma expressão de correspondência diferente, por exemplo, uma igualdade, adição ou concatenação. Desta forma, a semântica da operação de correspondência é menos restrita para realizar suas operações do que a junção [7].

A correspondência é composta por um conjunto de elementos mapeados, que é a união de todos os elementos correspondidos entre dois ou mais esquemas. Por exemplo, os elementos de *Rios* e *Lagos* (Figura 1) podem ser correspondidos entre si. Além disso, cada elemento do mapeamento pode ter uma expressão de correspondência que especifica seus relacionamentos.

Na Figura 1 são mostrados três esquemas: *Rios* e *Lagos* que correspondem aos esquemas de entrada e *Recursos de Água* que corresponde ao esquema destino. Uma correspondência entre *Rios* e *Lagos* pode conter um mapeamento simples, como *Rios.IDRio* para *Lagos.IDLago* com a expressão *Rios.IDRio* = *Lagos.IDLago*. Um elemento pode ser mapeado com uma expressão de concatenação "(*Rios.CidadeNascente, Rios.UFNascente*) = *Lagos.Localizacao*", que utiliza dois elementos vindos de atributos do esquema *Rios* para descrever a correspondência com um atributo do esquema *Lagos*.

Os critérios utilizados para combinar *Rios*, *Lagos* e *Recursos de Água* são baseados em heurísticas, uma vez que não é possível encontrar um modelo matemático para tal. Mesmo não sendo totalmente satisfatório produzir um mapeamento por heurística, por meio deste se obtém um resultado em que o usuário pode considerar uma boa correspondência.

B. Técnicas de correspondência entre esquemas

A Figura 2 apresenta uma classificação proposta por [7] das técnicas existentes na correspondência entre esquemas. As técnicas de mapeamento são divididas em dois grupos: abordagens de correspondências individuais e abordagens combinadas. No primeiro grupo cada ferramenta calcula uma correspondência baseada em um único critério de correspondência; no segundo, cada ferramenta utiliza mais de uma correspondência individual e vários critérios.

As abordagens de correspondências individuais se dividem em outros dois grupos, conforme apresenta a Figura 2. Um deles é baseado somente em esquemas e usa apenas as informações do esquema; e o outro é baseado em conteúdo/instância e considera também os dados das instâncias. Este último está relacionado em nível de elemento e possui as ferramentas de correspondência por linguística e por restrições.

Dados em nível de instância/conteúdo podem revelar importantes informações sobre o conteúdo e semântica dos elementos do esquema. Em um caso extremo, o esquema não é conhecido, mas pode ser construído a partir de dados de instância, criado manualmente ou automaticamente (e.g., um *guia de dados* [11], um grafo do esquema ou palavras-chave fornecidas por usuário). Mesmo quando se tem informações importantes sobre o esquema, o uso de correspondência em nível de instância pode ser útil para descobrir interpretações incorretas sobre tais informações.

A proposta deste trabalho se enquadra na correspondência individual baseada em instância/conteúdo em nível de elemento pois o interesse é oferecer a usuários não especialista em banco de dados uma ferramenta para gerar esquemas baseada em amostras e um banco de dados de origem. A próxima seção apresenta os principais trabalhos relacionados a esta proposta.

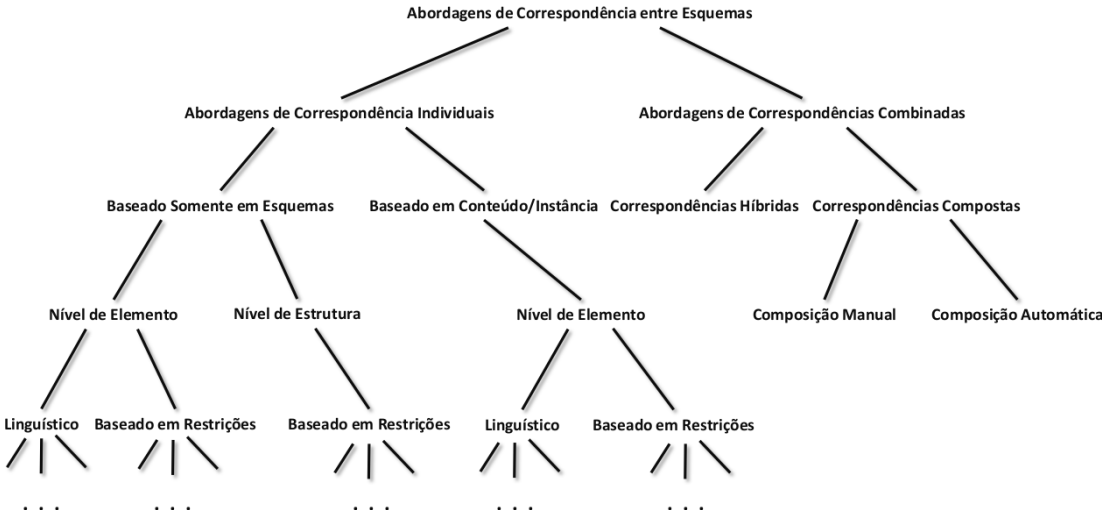

Fig. 2. Classificação das abordagens de correspondência entre esquemas proposta por [7]

.

III. Trabalhos Relacionados

NA literatura são encontradas diversas abordagens para implementar correspondência entre esquemas. Os leitores encontrarão em [7] e [1], revisões sistemáticas sobre o assunto. Porém, este trabalho foca em correspondência entre esquemas baseada em amostras. Dentre os trabalhos encontrados, quatro foram selecionados para discussão nesta seção, pois serviram de inspiração para a abordagem proposta.

Os trabalhos analisados [6], [9], [10], [12] realizam a correspondência entre esquemas utilizando amostras dos usuários e se baseiam no conceito de caminho para encontrar a correspondência. Em banco de dados relacional, caminho é a ligação entre tabelas através de atributos em comum, geralmente a relação entre chave primária e estrangeira. Por exemplo, o caminho para se encontrar se um estudante está cursando ou cursou um determinado componente curricular pode ser feito utilizando as tabelas Aluno x Matricula x CCR.

A abordagem proposta em [9] foi uma das primeiras a propor caminhos entre os esquemas para encontrar a correspondência baseada em amostras. A ferramenta proposta, chamada *DISCOVERY*, constrói vários caminhos no esquema que possam corresponder às amostras informadas. Os atributos das tabelas envolvidas são selecionados através de consultas ao banco de dados, baseado em um índice mestre implementado no Sistema Gerenciador de Banco de Dados (SGBD) Oracle. Após a criação dos vários possíveis caminhos, é feito um corte, eliminando os caminhos inválidos, *i.e.*, caminhos com atributos que não possuam correspondência no modelo de dados ou redundantes. Finalmente, os caminhos candidatos ao esquema correspondente são classificados por número de junções necessárias para criá-los e são apresentados ao usuário aqueles que atendem um número máximo de junções definidas.

As abordagens propostas em [6] e [10] (*MWEAVER* e *FILTER*, respectivamente) baseiam-se na mesma estratégia proposta em *DISCOVERY*, porém um grafo baseado no dicionário de dados é construído para identificar os caminhos de correspondência. Os atributos que correspondem às amostras são encontrados, em ambas as abordagens, através da construção de um índice invertido com as instâncias do banco de dados.

A ferramenta *MWEAVER* é composta pelo algoritmo de correspondência entre esquemas chamado *TPW* (*tuple path weaving*). Seu funcionamento está dividido em cinco etapas principais:

- Buscar ocorrências das amostras;
- Gerar caminho de mapeamentos pares;
- Gerar o caminho de tuplas pares;
- Construir o caminho de tuplas completo e;
- Classificar os mapeamentos;

A ideia é, basicamente, encontrar as tabelas com ocorrências das amostras, criar caminhos entre tais tabelas (através de chaves estrangeiras), fusionar os caminhos e classificá-los para o usuário final, apresentando os possíveis esquemas candidatos.

A abordagem *FILTER* tem duas melhorias em relação às anteriores. Primeiramente, ela é fortemente baseada na forma que a abordagem *DISCOVERY* encontra os candidatos, porém o corte dos candidatos é feito em tempo de construção, evitando que caminhos inválidos continuem sendo avaliados. O corte é feito através da utilização de filtros que verificam se o caminho candidato retorna um resultado não vazio. Em relação ao *MWEAVER*, *FILTER* não armazena os resultados das consultas intermediárias na memória, deixando a ferramenta mais escalável.

A proposta em [12], chamada de *Query from Examples (QFE)*, tenta encontrar um caminho entre os esquemas para determinar uma consulta. O *QFE* requer que o usuário determine se uma tabela de saída é o resultado que espera de uma consulta em um determinado banco de dados. Essa tabela de saída é um conjunto de amostras fornecidas pelo usuário. Seu funcionamento esta dividido

em dois módulos: Gerador de Consultas e Gerados de Banco de Dados.

A estratégia basicamente é gerar várias consultas equivalentes com base na tabela de saída e um banco de dados. Após isso, tendo como base as consultas geradas, é gerado um novo banco de dados que possa distinguir melhor as consultas. Por fim o usuário deve determinar quais daquelas consultas não representam a sua tabela de saída.

O processo anterior se repete até que seja encontrado uma consulta que o usuário julgue ser aquela que representa a tabela de saída ou o processo para de executar por ter um limite de iterações.

Uma das características utilizadas pelos algoritmos para a redução do acesso ao banco de dados relacional e das operações de junções é a utilização da estrutura de um grafo dirigido. Neste grafo, os nodos representam as tabelas do banco de dados e as arestas indicam as relações de chave-estrangeira e primária. A utilização de um grafo para navegar entre os metadados do banco de dados torna mais eficiente a busca dos caminhos candidatos que descrevam as amostras.

Outra característica importante é a forma de armazenamento dos resultados intermediários obtidos pelos algoritmos. *MWEAVER* delega a gestão de memória dos seus resultados intermediários para a memória principal do sistema operacional. Já *DISCOVER*, *FILTER* e *QFE* armazenam os seus resultados intermediários em tabelas temporárias no banco de dados, delegando assim a gestão de memória para o SGDB. Desta forma, *MWEAVER* pode sofrer com problemas de escalabilidade caso necessite de uma grande quantidade de memória, porém pode ganhar em desempenho de tempo caso os dados manipulados caibam na memória principal.

Por fim, as ferramentas se diferenciam quanto à forma de busca das ocorrências de amostras na base de dados. *MWEAVER* usa uma busca pelo texto completo, o usuário precisa informar o valor exato a ser encontrado na base de dados. As outras três abordagens relacionadas fazem a busca pelo texto aproximado, permitindo que o usuário saiba apenas parte do texto para realizar a correspondência. Um levantamento teórico sobre os problemas envolvendo a correspondência entre esquemas por amostra pode ser encontrado em [3].

A Figura 3 apresenta a estrutura de como os caminhos são criados a partir de uma amostra. O banco de dados fonte é aquele apresentado na Figura 4.

Na parte de baixo da figura encontram-se as amostras informadas e, em seguida, as amostras são confrontadas com o esquema. Assim, a amostra *Carlos* foi encontrada no atributo *nome* da Tabela *Aluno*. O mesmo ocorre com as outras amostras. Em seguida, são verificados os relacionamentos entre as tabelas envolvidas nas amostras e um esquema $\mathcal{E}$ é retornado para o usuário em uma forma de visão. Uma segunda visão é apresentada com as consultas *Q1* e *Q2* geradas em relação a $\mathcal{E}$ e às entradas do usuário. As consultas são extraídas das tabelas que possuem chave-estrangeira *Matriculas* e *AlunoDisciplina*;

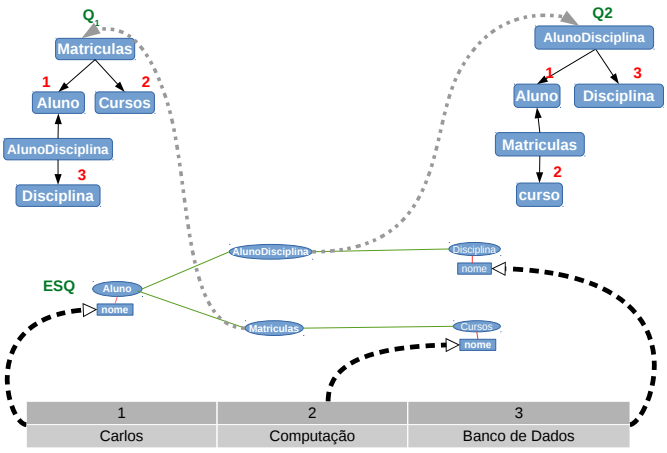

Fig. 3. Correspondência através de amostras na geração de visões.

a seta tracejada indica a relação entre as tabelas e as consultas. Os números *1*, *2* e *3* mostrados na visão das consultas indicam em quais colunas da tabelas as amostras estão relacionadas. Essas abordagens se apoiam na eliminação de candidatos inválidos para gerar as visões corretas, sendo este o principal desafio. Observe que as consultas *Q1* e *Q2* são oriundas do mesmo esquema, porém utilizando dois caminhos diferentes para apresentar o resultado: um pela tabela *AlunoDisciplina* e outro pela tabela *Matriculas*, respectivamente. As visões, extraídas através destas abordagens, podem auxiliar os usuários leigos a construir sua própria base de dados e obter suas consultas.

Os algoritmos propostos em [6] (*TPW*) e em [9] (*VERIFYALL*) serviram de referência para a proposta da *JMatching* e os mesmos são discutidos na próxima seção. A Tabela I na Seção IV apresenta um resumo comparativo entre as abordagens relacionadas à proposta deste trabalho.

IV. JMatching

Este trabalho é baseado nas abordagens propostas em [6], [10] e [9] e visa a criação de uma ferramenta que, através de dados (que podem corresponder a uma frase, data ou valor numérico) digitados pelo usuário (amostras) e uma base de dados relacional de origem, cria automaticamente uma base de dados de destino e suas consultas correspondentes. A amostra informada pelo usuário é uma n-tupla (onde n é o número de elementos na amostra) com os dados que o usuário julga importante para a criação do novo esquema. A Figura 3 apresenta, na parte de baixo, o formato de uma amostra, sendo n=3.

Como citado anteriormente, o diferencial do *JMatching* em relação aos trabalhos relacionados é o fato de sua implementação não ser invasiva, ou seja, a ferramenta não necessita de criação de índices invertidos ou outras formas de preparação dos dados para realizar a correspondência. O não uso dessas estruturas permite ao *JMatching* realizar

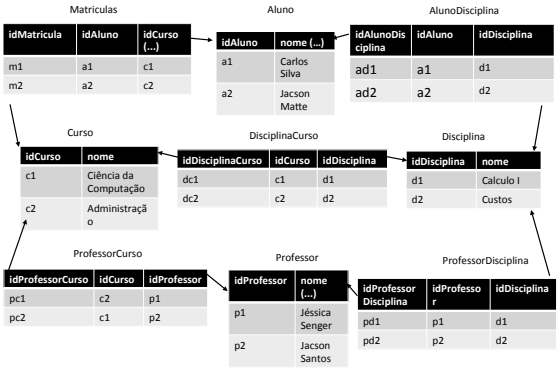

Fig. 4. Modelo lógico simplificado e instância de um sistema acadêmico.

mapeamento em nível de instância em qualquer esquema relacional, apoiado apenas no dicionário de dados.

A busca das amostras no banco de dados alvo baseia-se na criação de uma classificação da importância das tabelas para a busca das tuplas que correspondam com as amostras. A classificação, bem como outros detalhes de implementação, são apresentados nesta seção.

A correspondência entre as amostras do usuário e o esquema do banco de dados é realizada em sete passos:

I Acesso ao banco de dados e consulta no dicionário de dados para gerar o grafo;
II O uso das amostras informadas pelo usuário;
III Elaboração de critérios de seleção das amostras para minimizar o espaço de busca na base de dados;
IV Criação de esquemas candidatos baseados na entrada;
V Critérios de eliminação dos mapeamentos inválidos;
VI Geração das consultas, e;
VII Classificação e exibição dos resultados.

O Algoritmo 1 apresenta os passos para criação dos mapeamentos que são considerados candidatos[1]. A linha 22 corresponde a implementação do *VERIFYALL* e das linhas 11 a 21 ao algoritmo *TPW*. Como entrada, o algoritmo recebe dois parâmetros: D (base de dados) e T (amostras informadas pelo usuário) com o objetivo de criar um mapeamento, e retorna como saída os mapeamentos e consultas. O extrato de um banco de dados acadêmico apresentado na Figura 4 é utilizado como exemplo para descrever o Algoritmo 1.

A. Consulta ao Dicionário de Dados e Geração do Grafo

No primeiro passo é extraído o esquema E de D a partir do dicionário de dados. Em seguida, o grafo dirigido G que descreve E é construído (Função `GeraGrafo` - Linha 4 do Algoritmo 1). $G = (V, E)$ é definido como: V à um conjunto de nodos rotulados com a tripla $(N, \{S(t,r)_1, \ldots, S(t,r)_n\}, n_r)$, sendo N o rótulo do nodo que corresponde ao nome de uma das tabelas de D, $S(t,r)_i$ $(1 \leq i \leq n)$ é o nome de um dos atributos de N, t corresponde ao tipo do atributo de S e r indica se o atributo é chave-estrangeira (FK), chave-primária (PK) ou não chave $(NULL)$ e nr indica a cardinalidade de N (número de tuplas). E é um par ordenado que corresponde a uma aresta de origem no nodo V_1 apontada para um nodo V_2, sendo rotulado pela tupla (fk, pk), onde fk é o nome do atributo que possui restrição de chave-estrangeira de V_2 e pk é o nome do atributo que possui restrição de chave-primária de V_1. A Figura 5 apresenta o grafo gerado a partir do banco de dados da Figura 4. Por exemplo, a tripla $(Aluno, \{idAluno(int, PK), nome(varchar, NULL)\}, 563)$ é a representação da tabela *Aluno* em um nodo de G. A aresta que faz a ligação entre os nodo *Aluno* e *Matriculas* contém a tupla $(idAluno(PK), idAluno(FK))$, indicando que *Matriculas* está associada a *Aluno* pelo atributo *idAluno*. A construção de G é realizada através de consultas *SQL* ao dicionário de dados do *SGBD* que armazena D.

Algoritmo 1: *Mapeamento por amostra*

```
1  Entrada : Base de dados D, amostras do usuário (T1,...,Tn) ;
2  Saída : Mapeamento candidato MAP e consultas QRY;
3  Grafo G, TipoEntrada TE, respConsulta OCR, CaminhoPar
     MPAR, CaminhoCompleto MAP ;
4  G ← GeraGrafo(D);
5  TE ← ClassEntrada(T);
6  foreach i = 0; i > TE.length ; i++ do
7  |   TC ← ClassTabela(TE_i, G);
8  |   OCR += ConsultaAmostra(TC, TE_i);
9  end
10 AMT ← OCR_1;
11 foreach i = 1; i > OCR.length ; i++ do
12 |   if ! CriaMapPar(AMT, OCR_i, G, MPAR) then
13 |   |   continue;
14 |   end
15 end
16 MAP ← MPAR_1;
17 foreach i = 1; i > MPAR.length ; i++ do
18 |   if ! weave(MAP, MPAR_i) then
19 |   |   continue;
20 |   end
21 end
22 QRY ← GeraConsula(MAP, OCR, G);
23 ranking(MAP, C);
```

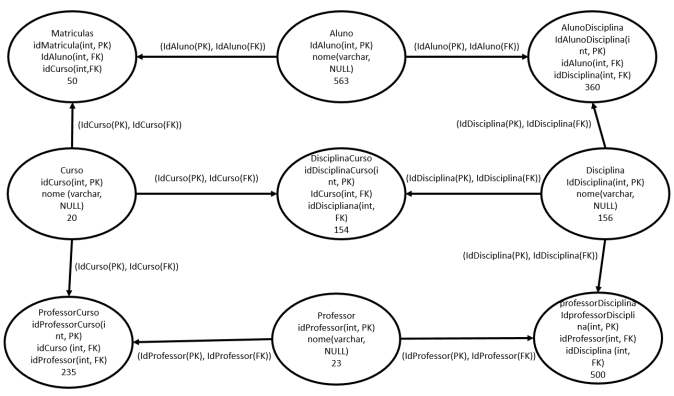

Fig. 5. Grafo orientado G gerado a partir da Figura 4.

[1]O código fonte do JMatching pode ser encontrado em https://github.com/jacsonmatte/JMatching

B. Uso das amostras informadas pelo usuário

O segundo passo é a criação de uma tabela T a partir das amostras de entrada. T é composta por (row, col), onde row é o número de linhas e col é o número de colunas. Inicialmente T = (1, 2), ou seja, são necessários dois valores de atributos e uma tupla para iniciar o processo de correspondência de esquema. Por exemplo, o usuário pode informar os valores ("Jacson" e "Cálculo") no intuito de montar uma base de dados com alunos e disciplinas. Cada entrada em T possui um tipo, sendo S, I ou D, onde S indica o tipo *string*, I o tipo *integer* e D o tipo *date*. Essa distinção é realizada pela função `ClassEntrada` (linha 5) no Algoritmo 1, e tem como saída a classificação em uma estrutura TE =(amt, tpe), onde tpe indica o tipo da amostra e amt é a amostra. Assim, "Jacson" e "Cálculo" gerariam $TE = \{("Jacson", S), ("Calculo", S)\}$.

C. Critérios de Seleção das Amostras e Espaço de Busca

No terceiro passo, as tuplas em D são recuperadas e combinadas com as amostras de T. Neste passo, a busca é dividida em duas etapas: na primeira, são classificadas as tabelas e os atributos e na segunda etapa são realizadas as consultas SQLs (linha 6 a 9 do Algoritmo 1).

A primeira etapa do terceiro passo é importante, pois reduz o espaço de busca dos caminhos no esquema. Essa classificação visa diminuir o número de acessos ao banco de dados e, consequentemente, aumentar a eficiência da abordagem. Os critérios de classificação são: a quantidade de chaves estrangeiras das tabelas (C_1), o número de atributos da tabela que possui tipos compatíveis com a amostra em T (C_2), e o número de tuplas da tabela - cardinalidade (C_3).

O critério C_1 possui o maior peso para classificação. É construído da seguinte forma: percorrendo o grafo G é verificado a quantidade de atributos com chaves estrangeiras existente em cada nodo. A partir disso, as tabelas são classificadas do nodo que possui o menor número de chaves estrangeiras para o nodo que possui o maior número de chaves estrangeiras. Aplicando esta prioridade, as tabelas com o maior número de chaves estrangeiras serão as últimas a serem consultadas. Esse critério foi criado pois acredita-se que as tabelas com menos chaves estrangeiras são aquelas que armazenam dados mais relevantes da aplicação e tabelas com muitas chaves estrangeiras têm o papel de fazer a junção entre outras tabelas. Sendo assim, o conteúdo mais relevante vai estar nas tabelas com o menor número de chaves estrangeiras. Para as tabelas que possuam o mesmo número de chaves estrangeiras é aplicado o segundo critério, reclassificando essas tabelas.

O critério C_2 tem um peso menor que C_1 para a classificação. Neste critério, é realizada uma busca em cada nodo do grafo G para verificar a quantidade de atributos que são do mesmo tipo da amostra em T. Desta forma, as tabelas são classificadas a partir daquela que contém o maior número de atributos para a que contém o menor número de atributos. Essas tabelas têm maior probabilidade de armazenar um dado correspondente à amostra. Para as tabelas que possuam o mesmo número de atributos é aplicada a terceira restrição, reclassificando essas tabelas.

O critério C_3 tem peso menor que o critério anterior para a classificação. Neste critério, é realizada uma busca em cada nodo de G e a classificação é feita através do valor nr de V. As tabelas com maior valor em nr são as primeiras a serem consultadas.

Todas as tabelas de D são classificadas em ordem crescente por C_1, C_2 e C_3 (em caso de empate, as tabelas são escolhidas aleatoriamente). No Algoritmo 1, as tabelas são classificadas pela função `ClassTabela` (linha 7) que recebe uma amostra TE_i e o grafo G, retornando o resultado em TC.

Dado o grafo G representado pela Figura 5 e a amostra *"Jessica"* como entrada, a classificação das tabelas aplicando os critérios é: 1. *Aluno*, 2. *Disciplinas*, 3. *Professor*, 4. *Curso*, 5. *ProfessorDisciplina*, 6. *AlunoDisciplina*, 7. *ProfessorCurso*, 8. *DisciplinaCurso* e 9. *Matriculas*.

As tabelas classificadas de 1-4 empatam no número de chaves estrangeiras (que é zero) e no número de atributos do mesmo tipo (*varchar*) (que é um). O último critério faz com que a tabela *Aluno* seja a primeira da lista, pois tem o maior número de tuplas. Neste exemplo, na primeira tabela consultada *Aluno*, a amostra *"Jessica"* seria encontrada no atributo *nome*. Nenhuma outra tabela possui em sua instância o valor *"Jessica"* em seus atributos, assim apenas a tabela *Aluno* seria utilizada para compor o espaço de busca. Perceba que outras tabelas poderiam fazer parte do espaço de busca pois a amostra não possui valor semântico para a ferramenta.

Ainda no terceiro passo (linha 8 do Algoritmo 1 - função `ConsultaAmostra`) são geradas as consultas SQLs. Como característica principal, esta consulta utiliza o operador *LIKE* e *TOP* 1, pois uma única tupla é necessária para verificar se a amostra pertence a tabela e ao atributo sendo analisados. As consultas não vazias são armazenadas em uma estrutura chamada OCR = (tab, atr, tpl), onde tab é o nome da tabela, atr é o nome do atributo e tpl é o valor do atributo retornado pela consulta. Por exemplo, para uma consulta com a amostra *Ciência* em T, a estrutura resultante é *(Curso, nome, Ciência da Computação)*. Todas as amostras de uma mesma coluna são consultadas na tabela selecionada e todas devem ser não vazias para a estrutura resultante ser considerada na próxima etapa do algoritmo. É importante lembrar que uma mesma amostra pode ser aplicada a todas as tabelas de D respeitando os critérios definidos.

D. Criação de Esquemas Candidatos

A quarta etapa possui como entrada o grafo G e a estrutura OCR gerada na etapa anterior e, a partir dessas entradas, cria os mapeamentos que são as visões da base de dados de destino. Com base no algoritmo TPW proposto em [6], mapeamentos intermediários são

gerados em pares (linha 10 a 15 do Algoritmo 1), ou seja, as ocorrências contidas em OCR são correspondidas em pares. As amostras fornecidas na primeira coluna são usadas como base para combinar com as amostras das próximas colunas. Isso é realizado para evitar a geração de caminhos pares duplicados e que o número de combinações cresça exponencialmente. Assim, a ordem em que são inseridas as amostras pode gerar mapeamentos diferentes. Desta forma, uma nova estrutura do tipo $MPAR = (t_1, t_2, \ldots, t_n)$ é gerada para armazenar os resultados intermediários. $MPAR$ é um mapeamento par, onde $t_1, t_2, \ldots, t_n$ correspondem aos nomes das tabelas que formam um caminho intermediário através de uma busca em G.

Por exemplo, se T contém duas amostras *"Carlos"* e *"Computação"*, baseado no banco de dados da Figura 4, dois resultados serão gerados para OCR = *(<Aluno, nome, Carlos Silva> , <Curso, nome, Ciência da Computação>)*, pois as tabelas *Aluno* e *Curso* contém as entradas de T no atributo *nome*. Com isso, é gerado um caminho intermediário *MPAR = (Aluno, Matriculas, Curso)*, pois através de uma busca no grafo G foi encontrado um caminho entre *Aluno* e *Curso* por meio do nodo *Matriculas*, ou seja, essas tabelas possuem uma ligação por chave estrangeira. Naturalmente, algumas amostras podem ser encontradas em mais de uma tabela, por exemplo, caso a amostra *"Carlos"* seja também encontrada na tabela professor, é verificado se existe um caminho no grafo entre o nodo professor e curso, caso exista é criado um novo caminho intermediário, caso contrário, a ligação será ignorada. Após gerar todos os caminhos intermediários pares, é realizada uma fusão entre os caminhos pares. Para realizar a fusão, são combinados os nodos que possuem os mesmos atributos de N. Como resultado, é criado o mapeamento de destino MAP que corresponde à fusão de todos os caminhos pares. A Figura 6 apresenta a geração de um MAP a partir das amostras de T=(*"Carlos", "Computação" e "Cálculo"*), ou seja, os caminhos pares *MPAR(1) e MPAR(2)*. Com isso, é realizada a fusão entre os nodos iguais dos caminhos pares, gerando o mapeamento final MAP.

Perceba que os caminhos pares são fusionados apenas se compartilharem um mesmo nodo. Caso algum caminho não compartilhe nodos com outros existente, este será ignorado na fusão. Na Figura 6, o nodo *Aluno* é compartilhado por ambos caminhos pares intermediários, gerando o MAP final. Caminhos pares com nodos distintos não são fusionados. No Algoritmo 1, a função chamada de *weave* (linha 18) realiza a fusão e tem dois parâmetros: o mapeamento par $MPAR$ e o mapeamento final MAP.

E. Eliminação dos Mapeamentos Inválidos e Geração das Consultas

No quinto passo são eliminados mapeamentos que atendam as duas condições:

I Inicialmente é consultado se existe um caminho no grafo G para um par de ocorrências em OCR (função `CriaMapPar` (linha 12)), ou seja, é verificado se existe um caminho em G entre os nodos que estão relacionados ao par de ocorrências. Tal função recebe como parâmetro uma ocorrência inicial AMT e uma segunda ocorrência OCR_i, o grafo G e $MPAR$. Caso a consulta em G resulte em um possível caminho, ela é adicionada em $MPAR$ (linha 13); e

II Por fim, para validar os mapeamentos pares é executado uma consulta na base de dados, tal consulta é realizada com base nos nodos do caminho par. A partir disso, caso a consulta não retorne vazia os caminhos pares são fusionados (linha 18).

Sendo finalizado este passo, no próximo passo são geradas as consultas com base no algoritmo *VERIFYALL* tendo como entrada G, OCR e MAPs, ou seja, os caminhos finais após a fusão, assim gerando uma visão da consulta.

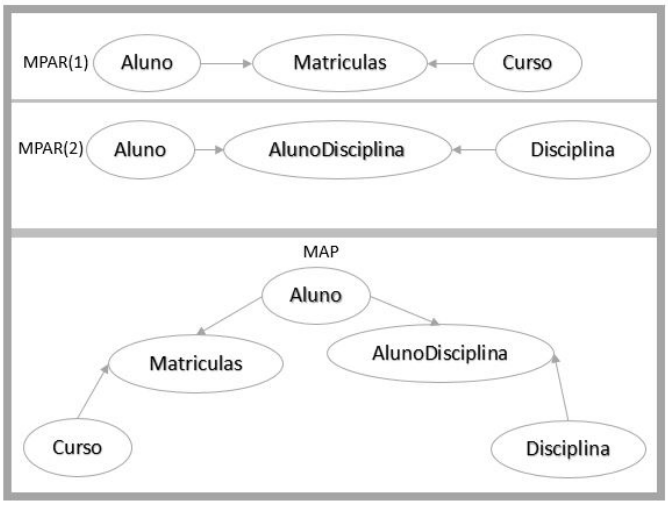

Fig. 6. Geração de caminhos intermediários pares e caminho final.

F. Classificação e Exibição dos Resultados

Nesta última etapa, as visões são apresentadas com algumas amostras de tuplas para o usuário bem como o código SQL correspondente às visões criadas. Os resultados de mapeamentos são mostrados em ordem crescente ao número de junções (linha 23).

JMatching trata apenas os tipos de dados tradicionais para as amostras (*i.e.*, sequência de caracteres, números e datas). Os dados das amostras são convertidos para uns desses tipos através de expressões regulares pré-definidas. Acredita-se que os tipos suportados pelo *JMatching* sejam suficientes para o bom funcionamento da ferramenta.

A Tabela I apresenta algumas características de quatro abordagens do estado da arte e o *JMatching*. As colunas *Classificação*, *Estrutura*, *Estratégia*, *Entradas mínimas* e *Memória* indicam, respectivamente:

I Formas que os esquemas são classificados para serem apresentados para o usuário;

II Estrutura auxiliar utilizada para gerar os esquemas;

III Estratégia para recuperar as amostras no banco de dados;

TABELA I
Comparativo das abordagens analisadas.

Método	Classificação	Estrutura	Estratégia	Entradas mínimas	Memória
MWEAVER	Pontuação	Grafo	Índice Invertido	1^a linha completa	SO
DISCOVER	#Junções	Grafo	Prop. (Oracle)	Única amostra	SGBD
FILTER	Probabilístico	Grafo	Índice Invertido	Única amostra	SGBD
QFE	#Probabilístico	Grafo	Construção de Índices	Par de amostras	SGBD
JMatching	#Junções	Grafo	Dicionário de Dados	Duas amostras	SGBD

IV Número de entradas mínimas para iniciar a correspondência; e
V Gerenciamento da memória.

Observando a Tabela I, percebe-se que todas as abordagens se apoiam em grafos para construir o mapeamento. Os grafos são estruturas que representam bem esquemas de banco de dados relacional, utilizando as arestas para representar as chaves estrangeiras. Salienta-se aqui, conforme já mencionado, o diferencial do *JMatching* é a estratégia para buscar as tuplas no banco de dados. A estratégia permite que *JMatching* possa ser utilizado em qualquer SGBD relacional, sendo necessário apenas configurar o método de acesso ao dicionário de dados. A estratégia de acesso ao banco de dados sem o uso de índices invertidos pode perder um pouco em eficiência de tempo, porém, conforme os experimentos realizados, não é proibitiva.

Em termos de complexidade, encontrar o conjunto de caminhos candidatos é NP-Completo [9], porém o Algorítimo 1 não realiza todas as combinações possíveis para gerar os candidatos. Muitos candidatos são descartados durante o processo de construção. Assim, analisando os três laços principais do algoritmo, tem-se os seguintes tempos:
(*i*) A complexidade de tempo para seleção das amostras é $O(|T|\times|G|)$, onde $|T|$ corresponde ao tamanho das amostras e $|G|$ o número de vértices do grafo de entrada (*i.e.*, o número de tabelas do banco de dados).
(*ii*) A complexidade de tempo para a criação de esquemas candidatos é $O(|OCR|\times|AMT|\times|G|)$, onde $|OCR|$ corresponde ao número de tabelas envolvidas na criação dos caminhos e $|AMT|$ é o tamanho da amostra inicial, e
(*iii*) A complexidade de tempo para eliminar e fusionar os caminhos é $O(|MPAR|^2)$, onde $|MPAR|$ é o número de caminhos pares candidatos gerados.
As consultas às tabelas do SGBD foram ignoradas nos cálculos acima pois são gerenciadas pelo otimizador de consultas que procura criar o plano mais eficiente de acesso [13].

V. Experimentos

NESTA seção são apresentados experimentos realizados para avaliar a usabilidade e o desempenho do *JMatching*. A usabilidade foi avaliada através de número de teclas digitadas e cliques de mouse. O desempenho foi avaliado através do tempo de resposta necessário para propor o conjunto de esquemas candidatos (possivelmente unitário) a partir das amostras informadas pelo usuário. O teste de usabilidade foi realizado com usuários que conhecem apenas a aplicação sem conhecimento da estrutura do banco de dados. O resultado da avaliação da usabilidade foi comparada com os resultados obtidos pela ferramenta *MWEAVER* [6], pois foi a abordagem base para a proposta aqui apresentada. Infelizmente, não foi possível comparar os desempenhos computacionais da nossa proposta e o *MWEAVER* pois não foram encontrados os códigos fontes do mesmo. O resultado da usabilidade do *MWAEVER* foi extraído de [6].

A interação do usuário com JMatching se dá por meio de uma interface web gerada a partir do *framework Bootstrap*. Os dados informados são enviados via *AJAX* e processados em *PHP*. O *IMDb* foi a base de dados utilizada para os ambos experimentos. O *IMDb* ocupa 12GB de armazenamento e foi gerado no *SGBD MYSQL 5*. Ele é composto por 21 relações com 108 atributos no total. As duas maiores (número de tuplas) tabelas são: `title` com 25 milhões e `akaname` com 3 bilhões. Todos os experimentos foram executados em uma máquina Intel Core i3-4005U CPU 1.70GHz e 4GB RAM.

A. Usabilidade

A usabilidade de um programa computacional pode ser medida de várias formas (vide [14]). Na avaliação aqui proposta, pretende-se medir o esforço de um usuário para finalizar uma determinada tarefa. O esforço é medido através do número de teclas pressionadas e cliques de mouse.

Foram convidados três usuários conhecedores do domínio cinema. Todos acompanham filmes e utilizam sítios de filmes[2] para saberem detalhes sobre determinados filmes. A tarefa a ser realizada, a mesma proposta no experimento de usabilidade envolvendo os mesmos casos de teste propostos da abordagem *MWEAVER*, era criar uma nova base de dados com os seguintes dados: título, diretor, data de lançamento e produtora do filme.

Os usuários tiveram uma breve explicação sobre a interface da ferramenta e em seguida foram instruídos a criar o banco de dados descrito anteriormente.

Conforme apresentado anteriormente, a usabilidade foi medida através da quantidade de teclas pressionadas e

[2]IMDb (http://www.imdb.com/) e
Yahoo Movies (www.yahoo.com/movies)

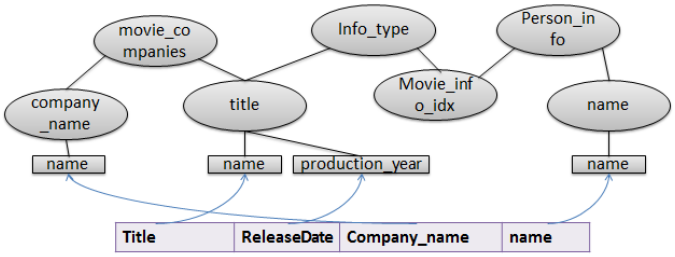

Fig. 7. Mapeamento resultante do experimento.

a quantidade de cliques do mouse. Como a repetição do experimento poderia levar a um treinamento dos usuários, o mesmo foi executado apenas uma vez. Durante o teste, também foi levantado o tempo de execução das respostas da ferramenta às solicitações dos usuários.

A Figura 7 apresenta o resultado do mapeamento após a digitação das amostras dos usuários. Na parte de baixo da figura são apresentados os atributos resultantes do mapeamento, enquanto na parte superior, as tabelas envolvidas do banco de dados IMDb. A Tabela II apresenta os resultados dos itens medidos: cliques de ambas as abordagens, uso de teclas de ambas as abordagens e o tempo total que o usuário levou para realizar a atividade para o *JMatching*. Em [6], oito usuários participaram do experimento. Para fins de comparação com o *JMatching*, foram colocados o melhor, o mediano e o pior desempenhos. Os resultados da eficiência computacional não foram comparados pois foram executados em ambientes diferentes.

Percebe-se que as abordagens, em termos de usabilidade, são bastante similares apesar dos melhores casos do MWEAVER serem superiores. Os usuários do *JMatching* mantiveram um padrão de cliques e pressão de teclas. O mesmo não ocorreu com o MWEAVER já que, por exemplo, o melhor usuário realizou a atividade com apenas 6 cliques de mouse e o pior com 43.

Alguns trabalhos, *e.g.* [15] e [16], mostram que a avaliação apenas pelas teclas pressionadas e pelos cliques de mouse não é o suficiente para se afirmar que uma *interface* é ou não mais fácil de utilizar que outra. Porém, pode ser considerada como um guia para medir a satisfação da interação com a ferramenta.

B. Desempenho Computacional

O teste de desempenho foi realizado no mesmo ambiente do teste de usabilidade. Porém, foram medidos o tempo de processamento para retornar as repostas dos cenários configurados, além do consumo de memória.

Foram propostos 5 cenários para o teste de desempenho:

C1: Foram digitadas duas amostras correspondentes a um nome de diretor e um título de filme.

C2: Foram digitadas três amostras correspondentes a um nome de diretor, um título de filme e o país onde foi produzido um filme.

C3: Foram digitadas quatro amostras correspondentes a dois nomes de diretores e dois títulos de filmes.

C4: Foram digitadas cinco amostras correspondentes a dois nomes de diretores, dois títulos de filmes e um país onde foi produzido um filme.

C5: Foram digitadas seis amostras correspondentes a dois nomes de diretores, dois títulos de filmes e dois países onde foram gravados filmes.

Cada cenário foi executado cinco vezes, seus tempos em segundos foram computados e, em seguida, a média e o desvio padrão foram calculados. A Tabela III apresenta os resultados. Perceba que os tempos das cinco execuções foram bem similares, pois o valor do desvio padrão é bastante baixo. Para os cenários C1 e C3 os tempos de execução são equivalentes, pois o desvio padrão é zero até a quarta casa após a vírgula.

Baseado nos tempos apresentados na Tabela III, pode-se considerar que o tempo de execução e o consumo de memória do *JMatching* não são proibitivos. Isso indica que é possível implementar uma ferramenta de correspondência entre esquemas baseada em amostra, sem utilizar estruturas auxiliares no banco de dados (*e.g.*, construção de índices invertidos).

VI. Conclusão

NESTE trabalho foi apresentada uma abordagem para correspondência entre esquemas em nível de instância para auxiliar usuários especialistas na aplicação mas sem conhecimento do nível conceitual do banco de dados. A criação do novo banco de dados é intuitiva (como mostrou os experimentos) e com a digitação de poucas amostras, o novo banco de dados já é apresentado na forma de interface gráfica. Além da intuitividade, *JMatching* obteve uma eficiência não proibitiva, mesmo em banco de dados de grande volume, conforme experimentos apontaram.

Apesar dos índices invertidos otimizarem as buscas das amostras no banco de dados, eles também provocam um *overhead* para o SGBD pois é necessário mantê-los atualizados. *JMatching* apoia-se no próprio dicionário de dados para identificar uma forma de otimizar o espaço de busca das amostras no BD, conforme apresentado na Seção IV-C, evitando o uso de estruturas auxiliares associadas ao banco de dados.

O correto resultado do mapeamento depende, como esperado, da qualidade das amostras informadas pelo usuário. Assim, a corretude do resultado tem como dependência principal as amostras do usuário especialista, pois quanto mais completa for a amostra em relação a instância do banco de dados, mais correta será a sua representação.

A geração de caminhos intermediários são reduzidos, pois são gerados pela combinação da primeira coluna de amostras com o restante das colunas. Desta forma, não são realizadas combinações de todas as amostras individualmente, pois, além de obter resultados possivelmente ambíguos, o número de combinações cresceria exponencialmente.

Como trabalhos futuros, pode-se citar: (*i*) utilização de mais de uma base de dados do mesmo domínio para gerar o

TABELA II
RESULTADOS DOS EXPERIMENTOS.

	JMatching			MWEAVER	
Usuário	**Cliques**	**Teclas**	**Tempo (s)**	**Cliques**	**Teclas**
U1	20	65	900	6	17
U2	25	53	1.200	17	58
U3	33	72	1.600	43	82

TABELA III
RESULTADOS DO EXPERIMENTO DO DESEMPENHO COMPUTACIONAL (EM SEGUNDOS).

Cenário	**Média**	**Desvio Padrão**	**Consumo Memória (Mb)**
C1	18.43	0.0000	3.5
C2	25.24	0.0010	3.5
C3	33.43	0.0000	3.7
C4	35.00	0.0011	4.1
C5	35.83	0.0001	4.3

esquema desejado (*e.g.*, utilizar Yahoo Movies e ImoDB), (*ii*) adequação para que a busca na base de dados seja por texto completo ou partes do texto, (*iii*) elaborar um aprimoramento do número de combinações dos resultados intermediários, e (*iv*) adaptar a ferramenta proposta para mais sistemas gerenciadores de banco de dados (*JMatching* depende da estrutura do dicionário de dados do SGBD fonte, atualmente, conecta-se apenas com o MySQL).

REFERÊNCIAS

[1] P. A. Bernstein, J. Madhavan, and E. Rahm, "Generic schema matching, ten years later.," *PVLDB*, vol. 4, no. 11, pp. 695–701, 2011.

[2] D. Tsichritzis and A. Klug, "The ansi/x3/sparc dbms framework report of the study group on database management systems," *Information Systems*, vol. 3, no. 3, pp. 173–191, 1978.

[3] B. Alexe, B. T. Cate, P. G. Kolaitis, and W.-C. Tan, "Characterizing schema mappings via data examples," *ACM Trans. Database Syst.*, vol. 36, no. 4, 23:1–23:48, Dec. 2011.

[4] R. J. Miller, L. M. Haas, and M. A. Hernández, "Schema mapping as query discovery.," in *VLDB*, vol. 2000, 2000, pp. 77–88.

[5] J. Madhavan, P. A. Bernstein, and E. Rahm, "Generic schema matching with cupid," in *VLDB*, vol. 1, 2001, pp. 49–58.

[6] L. Qian, M. J. Cafarella, and H. V. Jagadish, "Sample-driven schema mapping," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '12, Scottsdale, Arizona, USA: ACM, 2012, pp. 73–84, ISBN: 978-1-4503-1247-9.

[7] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *The VLDB Journal*, vol. 10, no. 4, pp. 334–350, Dec. 2001, ISSN: 1066-8888.

[8] P. Shvaiko and J. Euzenat, "A survey of schema-based matching approaches," in *Journal on Data Semantics IV*, S. Spaccapietra, Ed., vol. 3730, Springer Berlin Heidelberg, 2005, pp. 146–171.

[9] V. Hristidis and Y. Papakonstantinou, "Discover: Keyword search in relational databases," in *Proceedings of the 28th International Conference on Very Large Data Bases*, ser. VLDB '02, Hong Kong, China: VLDB Endowment, 2002, pp. 670–681.

[10] Y. Shen, K. Chakrabarti, S. Chaudhuri, B. Ding, and L. Novik, "Discovering queries based on example tuples," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '14, Snowbird, Utah, USA: ACM, 2014, pp. 493–504, ISBN: 978-1-4503-2376-5.

[11] R. Goldman and J. Widom, "Dataguides: Enabling query formulation and optimization in semistructured databases," in *Proceedings of the 23rd International Conference on Very Large Data Bases*, ser. VLDB '97, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997.

[12] H. Li, C.-Y. Chan, and D. Maier, "Query from examples: An iterative, data-driven approach to query construction," *Proc. VLDB Endow.*, vol. 8, no. 13, pp. 2158–2169, Sep. 2015, ISSN: 2150-8097.

[13] H. Garcia-Molina, J. D. Ullman, and J. Widom, *Database Systems: The Complete Book*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2008, ISBN: 9780131873254.

[14] X. Ferré, N. Juristo, H. Windl, and L. Constantine, "Usability basics for software developers," *IEEE software*, vol. 18, no. 1, pp. 22–29, 2001.

[15] D. Tamir, O. V. Komogortsev, and C. J. Mueller, "An effort and time based measure of usability," in *Proceedings of the 6th international workshop on Software quality*, ACM, 2008, pp. 47–52.

[16] P. Kortum and C. Z. Acemyan, "The relationship between user mouse-based performance and

subjective usability assessments," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, SAGE Publications Sage CA: Los Angeles, CA, vol. 60, 2016, pp. 1174–1178.

Jacson Luiz Matte é professor vinculado a Universidade do Oeste de Santa Catarina, graduado pela Universidade Federal da Fronteira Sul e possui especialização pela Universidade Norte do Paraná. As áreas de interesses são em pesquisas em banco de dados e sistemas embarcados.

Denio Duarte é professor associado na Universidade Federal da Fronteira Sul e obteve seu título de doutro pela Université François-Rabelais de Tours - France. Seus interesses em pesquisa são banco de dados nas nuvens e aprendizado de máquina.