Standard SQL Approaches for Similarity Searching

Pedro H. B. Siqueira[1], Paulo H. Oliveira[2], Marcos V. N. Bedo[3] and Daniel S. Kaster[1]
[1]*Universidade Estadual de Londrina (UEL) – Londrina/Brasil,* [2]*Universidade de São Paulo (USP) – São Carlos/Brasil,*
[3]*Universidade Federal Fluminense (UFF) – S. A. Pádua/Brasil*
pedro.braga.siqueira@gmail.com, pholiveira@usp.br, marcosbedo@id.uff.br, dskaster@uel.br

***Abstract*—This paper addresses complex data storage and retrieval in RDBMS, which depends on metric distance functions for the assessment of data dissimilarity. However, both the empirical analysis of strategies for complex data storage and the definition of a suitable representation for similarity query operators are still open issues in the literature. Here, we fulfill those gaps through the classification, implementation, and evaluation of existing approaches for complex data storage according to four structures found in standard SQL, namely *relational*, *object-relational*, *binary* and *semi-structured*. Moreover, we also discuss a comprehensive model for complex data retrieval, whose conception of similarity operators is consistent with standard SQL representations. Accordingly, a distance function representation is presented, which enables the RDBMS query processor to interpret and execute physical similarity operators. Experimental results indicate: *(i)* relational and object-relational structures outperform the other two competitors in the majority of scenarios, whereas *(ii)* object-relational strategy enables the use of a broader representation.**

***Index Terms*—Similarity Searching, SQL, Distance Functions, Metric Spaces, kNN**

I. Introdução

A quantidade e a diversidade de informações armazenadas digitalmente vêm crescendo exponencialmente nos últimos anos. Essas informações são representadas em diversos formatos e tipos de dados, o que requer que as técnicas computacionais de armazenamento e recuperação de dados se tornem ainda mais flexíveis e eficientes. Os Sistema de Gerenciamento de Bancos de Dados Relacionais (SGBDRs) comerciais existentes são capazes de tratar com eficiência os chamados dados tradicionais (como números ou pequenas cadeias de caracteres). Entretanto, outras categorias de dados requerem outras técnicas, usualmente muito distintas das empregadas por SGBDRs, para serem manipuladas.

Imagens, áudios, vídeos e séries temporais são alguns exemplos dessa crescente variedade de tipos de dados armazenados. A esses tipos de dados, não é possível aplicar operadores de comparação baseados na Relação de Ordem Total ($<$, $\leq$, $=$, $\neq$, $>$, $\geq$) com semântica adequada tal como no caso dos dados tradicionais, o que inicialmente inviabiliza seu tratamento em SGBDRs comerciais [1]. Uma alternativa semanticamente viável é comparar esses tipos de dados por meio do cálculo da distância entre eles seguindo a Teoria de Espaços Métricos [2], considerando o grau de suas respectivas dissimilaridades. No restante deste texto, refere-se aos dados comparados por similaridade como *dados complexos*.

Para representar o conteúdo dos dados complexos, funções de extração de características particulares de cada domínio são comumente empregadas [3]. Essas funções consistem em mapear um dado complexo para um segundo domínio $\mathbb{S}$, cuja representação é uma estrutura mais simples do que o dado complexo original, a qual denominamos vetor de características[1]. Se uma função de distância δ for capaz de mensurar a distância entre dois objetos no domínio $\mathbb{S}$, $\delta : \mathbb{S} \times \mathbb{S} \to \mathbb{R}_+$, então dizemos que δ é capaz de medir a dissimilaridade entre dados complexos. O par composto pelo domínio dos vetores de características e pela função de distância caracteriza um Espaço Métrico $\mathcal{M} = \langle \mathbb{S}, \delta \rangle$, o que permite modelar o comportamento de diversos critérios de consultas por similaridade, tais como operações sobre conjuntos [4], [5]. SGBDRs podem empregar essa modelagem para representar operadores de consulta por similaridade [6], e o desempenho dessas buscas pode ser aprimorado por meio de otimizações no plano de execução e em algoritmos para a execução dos respectivos operadores em estruturas de indexação especializadas, conhecidas como Métodos de Acesso Métricos (MAMs) [2].

Ferramentas existentes para a manipulação de dados complexos em SGBDRs comerciais são, tipicamente, soluções híbridas que usam o sistema operacional para gerenciar os arquivos que contêm os dados e os vetores de características associados [5]–[11]. No contexto de uma solução geral para consultas por similaridade em SGBDRs, deve-se tratar o armazenamento de dados complexos e suas representações considerando as estruturas e os tipos de dados oferecidos pelo SGBDR. Além disso, é importante que tal solução não altere a forma como os operadores de busca da SQL padrão são tratados pelos SGBDRs, de maneira que todo o processo de otimização de consultas não seja prejudicado pelo uso de implementações específicas de operadores por similaridade.

Neste trabalho, estendemos o módulo `FMI-SiR` [6] no *framework* `eFMI-SiR` com o objetivo de avaliar empiricamente nossa proposta. Considerando-se quaisquer dados complexos, o `eFMI-SiR` é capaz de armazenar seus vetores de características seguindo os diferentes quatro tipos de dados previstos na SQL padrão: *binário*, *relacional*, *objeto-relacional* e *semi-estruturado*. Nosso *framework* também apresenta uma modelagem flexível para funções de distância, o que permite representar um conjunto consistente de operadores por similaridade, associados a operadores físicos pelo processador de consultas do SGBDR. Os requisitos para inclusão da nossa proposta em SGBDRs comerciais, em termos de sintaxe, compreendem apenas a extensão do comando da *Data Defini-*

[1]Apesar do termo *vetor*, não há restrição para que $\mathbb{S} = \mathbb{R}^n$.

tion Language (DDL) `CREATE FUNCTION` para diferenciar funções de distância das demais funções, embora, naturalmente, exija a extensão do processador de consultas, com a implementação de operadores físicos, índices e estratégias de otimização envolvendo operadores por similaridade. Assim, as contribuições desse artigo são as seguintes:

1) **Uma taxonomia para vetores de características.** Os diferentes tipos de vetores de características da literatura são classificados de acordo com a sua estrutura, servindo como base para uma posterior classificação de funções de distância.
2) **Uma representação abrangente para operadores por similaridade.** Uma forma de representação em SQL padrão para funções de distância é proposta. Esse representação permite ao otimizador ter ciência da consulta por similaridade e empregar os operadores físicos adequados.
3) **Análise do impacto de diferentes tipos de armazenamento para vetores de características.** Quatro modelos de dados previstos na SQL padrão são analisados em termos de representação, armazenamento e recuperação.

O restante deste estudo está organizado conforme a seguir. A Seção II apresenta os principais operadores de consulta por similaridade. A Seção III apresenta propostas existentes de inclusão de suporte a similaridade em SGBDRs. A Seção IV apresenta a proposta de inclusão de similaridade em SGBDRs utilizando SQL padrão, incluindo uma taxonomia de vetores de características e funções de distância, representação de operadores por similaridade e análise de diferentes modelos de dados para implementação. A Seção V descreve a avaliação experimental da nossa proposta e, finalmente, a Seção VI apresenta as conclusões.

II. Operadores de Consulta por Similaridade

Dados complexos são armazenados em SGBDs na sua forma original, tipicamente utilizando tipos de dados LOB (*Binary Large Object — BLOB —* ou *Character Large Objetct — CLOB*) e a sua recuperação é feita por meio de dados associados. Por exemplo, em um sistema de identificação, retornar a foto de uma pessoa a partir de credenciais, como login e senha, a partir de uma consulta ao SGBDR sobre atributos de tipos de dados convencionais. Outra forma seria retornar a foto a partir do reconhecimento de sua impressão digital, o que exige consultar o SGBDR utilizando uma operação por similaridade sobre o(s) atributo(s) que armazenam o(s) vetor(es) de caraterísticas que representa(m) o conteúdo das impressões digitais. Consultas como esta são conhecidas como consultas por similaridade.

Há uma infinidade de extratores de características na literatura. Tomando imagens como exemplo, a maior parte dos extratores gera vetores de características compostos por um número fixo de atributos numéricos [12]. Mas há outros casos que também utilizam vetores numéricos de tamanho fixo. Considerando características não numéricas, merecem destaque os atributos textuais, como sequências de caracteres (palavras, sequências genéticas, etc.) ou conjuntos de termos (*bag of words*) [13]. Por fim, muitas aplicações consideram composições de vetores de características, onde cada vetor representa uma faceta diferente do dado. Por exemplo, considerar simultaneamente diferentes vetores de características de uma imagem, baseados em cor, textura e forma, permite representar os dados complexos com maior riqueza de propriedades complementares do conteúdo intrínseco do dado [14].

Também existem vários tipos de operadores de consulta de similaridade que correspondem a operações para recuperar elementos de um ou dois conjuntos, em que os critérios de avaliação dependem da função de distância aplicada sobre um ou mais elementos de consulta [2]. Os operadores mais empregados são [15]: *(i)* a Seleção por Abrangência (*Range query* — Rq), que recupera dentre os elementos armazenados aqueles cuja distância até o elemento de consulta é menor ou igual a um determinado limite; e *(ii)* a Seleção por Vizinhança (k*-Nearest Neighbors query* — k-NNq), que recupera os k elementos mais próximos ao elemento de consulta. Esses operadores básicos podem ser usados para definir novos operadores sobre dados complexos. Por exemplo, as junções por similaridade unem pares de dados complexos de dois conjuntos de entrada de acordo com um critério de similaridade, seja ele baseado em abrangência (Junção por Abrangência — $\bowtie_{\mathrm{Rq}}$), por vizinhança (Junção por Vizinhança — $\bowtie_{k\mathrm{NN}}$) ou pares de vizinhança (Junção por Pares de Vizinhança — $\bowtie_{k\mathrm{CN}}$). Outro operador importante é o de Consulta por Vizinhança Reversa (*Reverse k-NN query* — Rk-NNq) [16].

Funções de distância capazes de tratar múltiplos centros de consulta são conhecidas como funções de distância *agregadas* d_p e constituem uma generalização de funções de distância que manipulam um único centro de consulta. De acordo com Razente et al. [17], dado um *fator de agregação* p, um grupo de centros Q e um conjunto de pesos w_q associados aos centros $s_q \in Q$, d_p é representada como $d_p(s_i, Q) = \sqrt[p]{\sum_{s_q \in Q} \delta(s_i, s_q)^p * w_q}$, onde δ é uma função de distância que opera sobre um centro único s_q, e $s_i \in \mathbb{S}$ são os elementos do conjunto consultado. A função d_p pode ser, então, utilizada na Seleção por Abrangência Agregada (ARq) e na Seleção por Vizinhança Agregada (k-ANNq).

Os operadores citados podem ser amplamente usados em aplicações baseadas em consultas por similaridade. Por exemplo, consultas por abrangência e vizinhança são aplicáveis na proteção à privacidade [18], ferramentas médicas de auxílio ao diagnóstico [19], enquanto o operador k-ACNq é utilizado no projeto de redes [20] e em visão computacional [21].

III. Abordages para Inclusão de Suporte a Similaridade em SGBDRs

Existem algumas propostas de inclusão de suporte a similaridade em SGBDRs. As próximas seções apresentam uma análise das principais abordagens, com foco na forma de armazenamento de dados complexos (Seção III-A), representação de operadores por similaridade (Seção III-B) e aderência ao padrão SQL (Seção III-C).

A. Armazenamento de Vetores de Características

Os trabalhos existentes dividem-se com relação ao armazenamento de dados complexos em três categorias: *(i)* os que propõem um tipo de armazenamento próprio, *(ii)* os que estendem o catálogo de um SGBDR e *(iii)* os que representam os dados por meio de *User-Defined Types* (UDTs).

Na primeira categoria, o MESSIF [22] armazena dados complexos como objetos semi-estruturados chamados *buckets*, os quais representam partições no espaço de busca. Na segunda categoria, o SIREN [23] consiste em um *middleware* que opera em uma camada acima de um SGBDR comercial. Ele trata dois domínios de dados usando tipos previstos no padrão SQL/MM, STILLIMAGE e PARTICULATE, definidos por meio de extensões nos comandos DDL do SGBDR utilizado. Um elemento STILLIMAGE é armazenado em uma coluna do tipo binário, e um elemento PARTICULATE é armazenado em um conjunto de colunas do tipo INTEGER, em uma tabela designada para armazenar os dados complexos.

Na terceira categoria, estão as soluções PostgreSQL-IE [9], FMI-SiR [6], SimDB [24] e MSQL [11]. As duas primeiras armazenam dados complexos e vetores de características como atributos binários. No caso do FMI-SiR, os vetores de características são inseridos/atualizados através de funções externas ao SGBDR. Nas duas últimas soluções, cada característica extraída é armazenada em um atributo de uma tabela relacional que contém a representação do dado complexo.

B. Representações de Operadores por Similaridade

As soluções apresentadas divergem na forma de representar consultas por similaridade. Em geral, existem duas vertentes: a que propõe extensões à linguagem SQL e a que utiliza recursos da SQL padrão. As abordagens SIREN, MESSIF e SimDB adotam a primeira vertente. Tanto o SIREN quanto o MESSIF processam os predicados por similaridade separadamente e repassam os predicados que envolvem apenas dados tradicionais para o SGBDR, que se encontra em uma camada abaixo. Entretanto, devido a essa estratégia, o SGBDR não é capaz de reescrever os planos de execução das consultas, o que prejudica possíveis otimizações. O SimDB modifica diretamente o código-fonte do PostgreSQL para tentar cobrir essa lacuna, mas a solução não é extensível para outros SGBDRs nem cobre os critérios de consultas por similaridade de forma genérica.

As abordagens PostgreSQL-IE, FMI-SiR e MSQL utilizam *User-Defined Functions* (UDFs) na representação dos operadores por similaridade através da SQL padrão. No caso do PostgreSQL-IE, o operador por similaridade é expresso na cláusula FROM, o que limita seu uso em representações de consultas complexas e não oferece suporte a otimização. O FMI-SiR implementa funções de distância usando o recurso *Oracle Data Cartridge*, que atua como uma interface para utilizar os métodos da biblioteca C++ externa Arboretum[2]. Adicionalmente, o FMI-SiR possibilita indexar os dados complexos por meio do comando CREATE OPERATOR do Oracle, além de permitir mesclar operadores por similaridade com operadores convencionais e manipular o plano de execução. A abordagem MSQL emprega uma estratégia parecida ao representar as funções de distância por meio de UDFs, que manipulam os vetores de características diretamente em colunas de tabelas. Assim, os elementos a serem buscados são filtrados por meio das UDFs LOCATE e DIST, chamadas em uma consulta por similaridade definida como outra UDF — SIMQ. Embora o MSQL seja uma solução abrangente, sua representação de consultas exige que o usuário monte consultas mais complexas (que incluem diversos operadores) por meio de várias chamadas SIMQ, o que impõe um obstáculo em termos de usabilidade.

[2]https://bitbucket.org/gbdi/arboretum

C. Aderência à SQL Padrão

Como visto, a literatura emprega ora propostas de extensão à SQL padrão para representar consultas por similaridade, ora recursos da SQL para representar cada operador por similaridade individualmente. A SQL padrão oferece vantagens inatas em comparação às extensões, uma vez que é um padrão amplamente conhecido e utilizado pelos usuários. Além disso, os SGBDRs comerciais dão suporte à SQL sem a necessidade de quaisquer alterações no código-fonte das ferramentas, o que possibilita aproveitar o aprendizado obtido no desenvolvimento de recursos especializados para recuperação de dados tradicionais. Por fim, o uso de SGBDRs e da SQL padrão garante a integridade de transações, o que pode ser diretamente estendido aos dados complexos [11].

Não obstante, mesmo as abordagens que empregam a SQL padrão: *(i)* consideram apenas tipos específicos de vetores de características e/ou operadores de consulta por similaridade; e *(ii)* não o fazem de forma transparente o suficiente para que seja possível ao processador de consultas identificar os operadores por similaridade físicos a serem executados, independentemente da forma de armazenamento do vetor de características. A proposta deste trabalho é abordar de forma abrangente como é possível oferecer, utilizando a SQL padrão, os recursos necessários para a execução de uma grande variedade de consultas por similaridade, considerando diferentes tipos e combinações de vetores de características.

IV. Proposta de Inclusão de Similaridade em SGBDRs com SQL Padrão

Para incluir recursos de similaridade em SGBDRs, este trabalho aborda a representação de vetores de características, funções de distância e operadores de consulta utilizando quatro modelos de dados suportados atualmente pela SQL padrão: binário, relacional, objeto-relacional e semi-estruturado. Contudo, independentemente do modelo, é preciso identificar um conjunto reduzido de tipos de dados, funções e operadores que permitam representar a maior parte do universo de recuperação de dados complexos por similaridade.

Esta seção descreve a proposta do trabalho, incluindo uma classificação dos vetores de características e das funções de distância (Seção IV-A), representação de operadores

por similaridade em SQL padrão (Seção IV-B), análise de implementação nos modelos diferentes modelos (binário, relacional, objeto-relacional e semi-estruturado (Seção IV-C) e uma discussão das vantagens e limitações dos modelos (Seção IV-D). Todas essas variações foram implementadas no `eFMI-SiR`, que é uma extensão do módulo `FMI-SiR` [6], acoplado ao SGBDR Oracle, com o objetivo de verificar a aplicabilidade de cada modelo, identificar suas vantagens e limitações para a representação dos diferentes recursos, e avaliar empiricamente o desempenho de cada um deles.

A. Classificação de Vetores de Características e de Funções de Distância

Um dado complexo pode ter inúmeros atributos que o descrevem de alguma forma. Definimos o tipo genérico *vetor de características* (*feature vector* — FV) a todo atributo, ou conjunto de atributos, que é comparado utilizando-se uma função de distância. As operações definidas para cada tipo distinto de FV é conjunto de funções distância aplicáveis a ele.

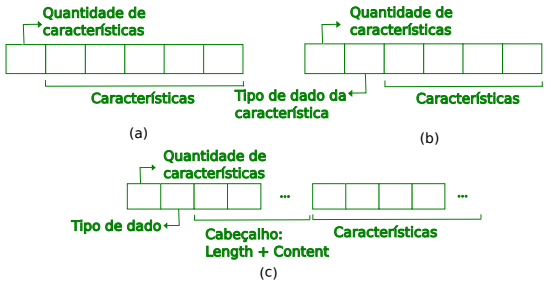

Fig. 1. Vetores de características. (a) Características com tipo de dados de tamanho fixo. (b) Características de tamanho fixo incluindo com o metadado indicando o tipo de dados. (c) Características com tamanho variável.

Embora um vetor de características seja um tipo de dado abrangente, uma vez que correspondem a instâncias em qualquer domínio $\mathbb{S}$, a grande maioria dos FVs encontrados na literatura podem ser classificados em três dimensões: *(i) dimensionais* ou *adimensionais*; *(ii) simples* ou *compostos*; e *(iii) homogêneos* ou *heterogêneos*.

Um *FV Dimensional* tem um número fixo de características numéricas. Este é o tipo de FV mais comum para dados complexos. Já os FVs adimensionais são conjuntos de características numéricas em que o número de elementos não é fixo, denominados *FVs Numéricos Adimensionais*, ou conjuntos de características não numéricas. FVs Numéricos Adimensionais são usados, por exemplo, para representar características extraídas de regiões de interesse de imagens, cujo número pode variar de imagem para imagem. FVs adimensionais não numéricos podem ser divididos em *FVs de Caracteres*, usados, por exemplo, para representar sequências genéticas, e *FVs de Textos*, que permite representar, por exemplo, *tags* descritivas associadas a uma imagem, no estilo *bag of words*, ou características correspondentes a atributos categóricos.

Um *FV Simples* tem características pertencentes ao mesmo domínio de dados tradicionais. Os metadados necessários a um FV Simples são o número de características e o tipo de dados das características. A Figura 1(a) ilustra o caso em que o tipo de dados é de tamanho fixo e é armazenado de alguma forma no catálogo do banco de dados, mas o número de características é associado ao FV, o que permite representar vetores de diferentes tamanhos. Esse tipo de estrutura permite armazenar FVs Dimensionais, FVs de Caracteres e FVs Numéricos Adimensionais. Note-se que o número de características poderia ser armazenado no catálogo do banco para os dois primeiros casos, mas FVs numéricos adimensionais exigem que o número de características esteja no vetor, por isso essa estrutura é a que engloba essas três possibilidades. Já no FV da Figura 1(b) o tipo de dados das características também está embutido. Isso mantém o FV autocontido, entretanto, esse dado é repetido para todos os valores de atributo desse tipo em uma tabela, o que ocasiona disperdício de espaço. Por fim, a Figura 1(c) ilustra o caso em que o tipo de dados das características é de tamanho variante, como os FVs de Texto, onde a estratégia de representação usada contém um cabeçalho com um conjunto de entradas no estilo `[tamanho,referência para o conteúdo]` seguido do conteúdo de cada característica.

FVs Compostos consistem em um conjunto de FVs Simples, em que cada FV Simples também é chamado de *componente*. FVs Compostos podem representar simultaneamente diversos aspectos de um dado complexo, tais como, para uma imagem, FVs Simples baseados em cor, textura, formas, rótulos, regiões de interesse e anotações. Nossa análise é que os FVs compostos podem ser subdivididos entre predefinidos e dinâmicos. *FVs Compostos Predefinidos* (Figura 2(a)) têm a estrutura definida a priori, enquanto *FVs Compostos Dinâmicos* (Figura 2(b)) possibilitam que componentes sejam adicionados e/ou removidos de acordo com a necessidade, definindo a estrutura a posteriori, de acordo com o uso.

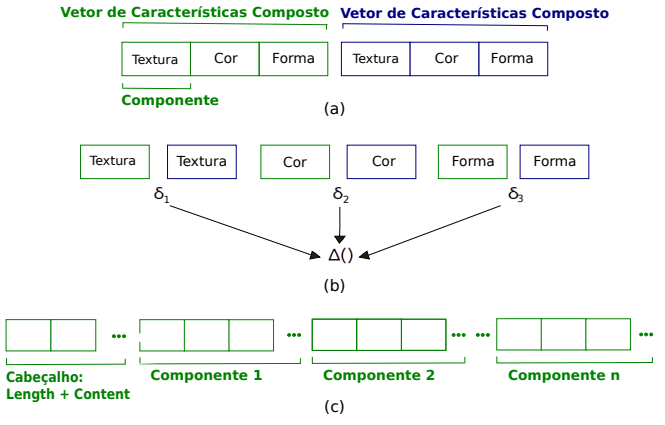

Fig. 2. (a) FVs Compostos Predefinidos. (b) Aplicação de Métrica Produto em um par de FVs Compostos Predefinidos. (c) FV Composto Dinâmico.

Por fim, os *FVs Homogêneos* constituem uma categoria de FVs simples cujas características são representadas por dados pertencentes a um mesmo domínio (Figura 1) e os *FVs Heterogêneos* possuem características que pertencem a

diferentes domínios (e.g., números, textos/palavras e atributos categóricos). Um FV Heterogêneo pode ser considerado um caso particular de FV Composto, cujos componentes são FVs Homogêneos de forma a representar todos os tipos diferentes de características do vetor. Note-se que um FV Composto Homogêneo pode ser convertido em um FV Simples Homogêneo, sendo que, nesse caso, não haveria mais distinção entre os subgrupos de características.

Considerando as funções de distância (também referenciada como FD neste estudo), a classificação proposta é: *(i)* simples ou composta; e *(ii)* de um ou múltiplos centros de consulta.

FDs Simples podem ser aplicadas a FVs Simples Homogêneos. Exemplos de FDs Simples são as funções da família Minkowski, que podem ser aplicada a FVs (Simples) Dimensionais cujas características estejam no domínio $\mathbb{R}$, a FD de Edição (Levenshtein ou *Edit*), aplicada a FVs Adimennsionais de Caracteres, e a FD de Jaccard, aplicável a qualquer domínio de conjuntos, tipicamente FVs de Textos [2], [25]. *FDs Compostas* são aplicadas a FVs compostos (o que engloba os FVs Heterogêneos). FDs Compostas também são conhecidas na literatura como FDs de múltiplos descritores [26]. Como mostra a Figura 2(b), uma FD Composta agrega em um único valor os resultados de FDs Simples calculadas sobre os componentes de um FV Composto, sendo que a forma mais comum de agregação é a utilização de uma Métrica Produto. Um exemplo típico de Métrica Produto é a combinação linear dos resultados das FDs Simples.

A proposta desse trabalho é consolidar todas essas variações de FVs nos tipos de dados indicados na hierarquia de classes da Figura 3. Na figura, FV_NUMBER permite representar FVs Dimensionais e FVs Numéricos Adimensionais, ao passo que FV_TEXT e FV_CHAR permitem representar, respectivamente, FVs de Textos e FVs de Caracteres, e o tipo FV_COMPOSITE é para FVs Compostos. A figura também indica exemplos de FDs associadas a cada tipo de dados.

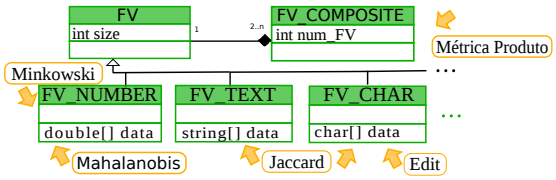

Fig. 3. Hierarquia de classes de vetores de características e exemplos de funções de distâncias definidas sobre os tipos.

B. Representação de Operadores por Similaridade

No que tange à representação de operadores por similaridade, nota-se que os trabalhos existentes: *(i)* abordam um conjunto reduzido de operadores; ou *(ii)* propõem extensões significativas à SQL de forma a compreender uma gama maior de operadores. Nossa proposta é que é possível representar a maioria dos operadores por similaridade em SQL padrão apenas definindo funções de distância de forma adequada.

A forma trivial de representar uma função de distância é por meio de uma função SQL que recebe como parâmetros dois FVs de um mesmo tipo e retorna a distância entre eles. O exemplo a seguir mostra como representar uma Seleção por Abrangência utilizando uma função desse tipo (`manhattan_distance`) sobre a tabela `cophir`, cujo atributo `image` é de um tipo imagem, `fv` é de um tipo FV e o elemento de consulta é o que tem `id=2`.

```
-- Seleção por abrangência
SELECT image
FROM cophir WHERE manhattan_distance(fv,
 (SELECT fv FROM cophir WHERE id=2)) <= 10;
```

Embora essa representação seja executável, o SGBDR só poderá realizar adequadamente um processo de otimização caso consiga mapear univocamente essa sintaxe para o operador físico por similaridade de seleção por abrangência. Caso esse mapeamento seja factível, as regras específicas de reescrita de consulta aplicáveis ao operador físico por similaridade utilizado podem ser aplicadas, bem como a escolha de algoritmos e índices específicos a esse operador. Para possibilitar isso, nossa proposta é funções de distância sejam identificadas de forma diferenciada de outras funções. Neste tralho, propomos estender o comando `DDL CREATE FUNCTION` e seus comandos complementares (`ALTER`, `DROP`, etc.) com a palavra reservada `DISTANCE` para indicar que trata-se de uma função de distância. Desta forma, ao detectar uma função de distância em uma consulta, o processador de consultas pode fazer sem ambiguidade o processo de tradução para o operador por similaridade correspondente.

Para que algoritmos e índices específicos sejam utilizados para agilizar a execução de operadores por similaridade, novamente é preciso identificar uma correspondência unívoca. Essa correspondência depende: *(i)* do domínio do vetor de características, *(ii)* da função de distância e *(iii)* dos parâmetros da função de distância, pois diferenças que afetam a distribuição de distâncias no espaço de busca inviabilizam o uso de índices. Por exemplo, se existe um índice para um atributo de um determinado tipo de FV usando a função de distância Euclidiana, esse índice não pode ser utilizado em consultas envolvendo outras distâncias ou mesmo em consultas que utilizam a distância Euclidiana para o mesmo atributo, mas com diferenças nos pesos das características.

O domínio do vetor de características pode ser definido por meio do tipo de FV correspondente ao atributo e por restrições de integridade eventualmente associadas[3]. As funções de distância podem ser escolhidas a partir de um conjunto básico oferecido pelo SGBDR ou definidas pelo usuário. Já os parâmetros de funções de distância precisam ser associados a cada instanciação diferente de função de distância. A parametrização de uma função de distância pode ser feita a priori ou na invocação da função. Como o número de parâmetros pode ser grande, fazê-lo na invocação resulta em uma sintaxe carregada em consultas. Portanto, nossa proposta é parametrizar a priori, associando um rótulo a cada

[3]Restrições de integridade não são objeto desse trabalho, mas, por exemplo, a um atributo do tipo FV_NUMBER de uma tabela poderia ser adicionada uma restrição de integridade que verifique o tamanho (`size`) do FV, garantindo que todos os FVs armazenados tenham a mesma dimensionalidade.

parametrização. A criação de índices sobre atributos de tipos FV também deve indicar a parametrização utilizada para que o processador de consulta possa identificar sem ambiguidade os índices elegíveis para otimização, bem como a estrutura de dados utilizada, quando for o caso. Com isso, no exemplo a seguir, o otimizador poderia traduzir a instrução SQL em uma Junção por Abrangência que retorna os pares de imagens da tabela `cophir` e da tabela `london` (imagens de Londres) cuja distância entre os seus FVs (de tipos compatíveis) é limitada a `10` unidades, utilizando a função de distância `my_weighted_manhattan` e, eventualmente, utilizar o índice `cophir_my_weighted_manhattan_ix`, cuja estrutura é uma Slim-tree[4].

```
-- Criação de uma distância parametrizada
CREATE DISTANCE FUNCTION my_weighted_manhattan
  ...;
-- Criação de um índice por similaridade
CREATE INDEX cophir_my_weighted_manhattan_ix
 ON cophir(fv)
 USING slimtree my_weighted_manhattan;
-- Junção por abrangência
SELECT l.image AS limg, c.image AS cimg
FROM london l, cophir c
WHERE my_weighted_manhattan(l.fv, c.fv) <= 10;
```

Analogamente, operadores que envolvam a manipulação de múltiplos centros de consulta podem ser representados por meio de funções de distâncias específicas definidas pelos usuários e devidamente parametrizadas a priori. O otimizador de consultas pode identificar que trata-se de um operador de múltiplos centros a partir do segundo parâmetro da função de distância, que é um conjunto de elementos.

Consultas por vizinhança podem ser representadas utilizando opções baseadas em ranqueamento documentadas no padrão SQL. O comando `FETCH FIRST k ROWS ONLY` é capaz de representar a Seleção por Vizinhança, enquanto as funções de janela (*window functions*) baseadas em ranqueamento `ROW_NUMBER` e `RANK`[5] podem ser usadas para representar tanto a Seleção quanto a Junção por Vizinhança. Os dois exemplos a seguir se beneficiam desses comandos para representar uma Seleção por Vizinhança (as `5` imagens mais similares à imagem com `id=2`, considerando os vetores em `fv`) e uma Junção por Vizinhança (para cada imagem da tabela `london`, concatene-a com cada uma das `5` imagens da tabela `cophir` mais similares a ela).

```
-- Seleção kNN
SELECT image FROM cophir
ORDER BY my_weighted_manhattan(fv,
 (SELECT fv FROM cophir WHERE id=2))
FETCH FIRST 5 ROWS ONLY;
```

```
-- Junção kNN
SELECT l_img, c_img FROM (
 SELECT l.image AS l_img, c.image AS c_img
  ROW_NUMBER() OVER (PARTITION BY l.fv ORDER BY
  my_weighted_manhattan(l.fv, c.fv)) AS rn
 FROM london l, cophir c)
WHERE rn <= 5;
```

Na execução convencional dessas expressões, os elementos são ordenados e depois os k primeiros elementos são recuperados[6]. Esta sequência de operações corresponde à execução sequencial das consultas por similaridade indicadas. Contudo, como as ordenações são baseadas em uma função de distância, o otimizador pode traduzir as instruções para os operadores por similaridade correspondentes (Seleção k-NN e Junção k-NN) e utilizar algoritmos e estruturas de dados que resultem em uma execução mais eficiente. Observe-se que, no exemplo da Junção por Vizinhança, se houvesse outras condições de filtragem na subconsulta seriam aplicadas antes da Junção k-NN, ao passo que condições de filtragem na consulta externa seriam aplicadas depois da junção.

Para ilustrar o poder de representação destas construções sintáticas da SQL padrão, a consulta a seguir é uma Consulta por Vizinhança Reversa.

```
-- kNN Reversos (RkNNq)
SELECT c_image FROM (
 SELECT c.image AS c_image,
  center.id AS center_id,
  c.fv AS c_fv, center.fv as center_fv,
  ROW_NUMBER() OVER (PARTITION BY c.fv
  ORDER BY my_weighted_manhattan(c.fv,
   center.fv)) AS rn
 FROM cophir c, cophir center
 WHERE c.id != center.id)
WHERE rn <= 1 AND center_id=2
ORDER BY my_weighted_manhattan(c_fv, center_fv)
FETCH FIRST 10 ROWS ONLY;
```

Nesta consulta, primeiro identifica-se o vizinho mais próximo de cada elemento da tabela com *alias* `c` (subconsulta + `rn <= 1`). Em seguida, identifica-se quais elementos da tabela com *alias* `c` têm como seu vizinho mais próximo o elemento de consulta (`center_id=2`). Por fim, o resultado atual é ordenado e retorna-se as `10` primeiras imagens, que são dos vizinhos mais próximos reversos do elemento de consulta.

C. Armazenamento de Vetores de Características em SGBDR

Esta seção apresenta uma análise de um conjunto consistente de tipos de dados previstos na SQL padrão para representar e armazenar os vetores de características. Em particular, inspecionamos os seguintes tipos: *binário*, *relacional*, *objeto-relacional* (OR) e *semi-estruturado*.

1) Binário: O armazenamento binário consiste em utilizar atributos do tipo BLOB para armazenar FVs e funções de distância implementadas como funções externas. Dessa forma, o SGBDR não tem ciência das informações armazenadas no FV e, portanto, não oferece um suporte mais elaborado, atuando quase como como um repositório de arquivos. Os metadados do vetor de características devem ser inclusos no próprio FV (Figura 1), para que as funções externas saibam como tratá-lo. A representação de consultas é idêntica ao apresentado na

[4]A Slim-tree é um método de acesso métrico hierárquico e dinâmico [27].

[5]A função `RANK` produz o mesmo efeito do modificador `WITH TIES` da construção sintática `FETCH FIRST`.

[6]Normalmente, não é uma ordenação completa, pois só é preciso manter os k primeiros elementos ordenados a cada iteração.

Seção IV-B. Não obstante, o armazenamento e a recuperação desses vetores de características podem ser conduzidos por linguagens externas, inicialmente mais eficientes do que as baseadas no padrão SQL/PSM, embora exista uma sobrecarga ao invocá-las.

2) Relacional: O armazenamento relacional consiste em criar uma tabela para cada FV do banco de dados, referenciando a tabela que armazena o respectivo dado complexo. Essa abordagem permite que o usuário manipule as características armazenadas por meio de comandos DML que apresentam grande flexibilidade. Contudo, a passagem de parâmetros para as funções de distância exige informar cada atributo de cada FV individualmente, o que, além da sintaxe carregada, exige criar FDs específicas para cada dimensionalidade, como ilustrado a seguir.

```
my_weighted_manhattan(a1, ..., an, b1, ..., bn)
```

Uma alternativa poderia ser indicar apenas a relação corresponde a cada FV, mas esta alternativa não é genérica pois mantém as funções de distância amarradas ao esquema do banco. Outra opção seria passar cursores como parâmetros, mas essa opção perde totalmente a verificação de tipos, que é um recurso nativo e importante oferecido pelos SGBDRs.

Outra limitação da abordagem relacional é para armazenar FVs Adimensionais, pois exige que a tabela tenha o mesmo número de atributos do maior FV armazenado, o que pode gerar muitos nulos. Além disso, o esquema da tabela tem que ser atualizado a cada elemento com mais características que o maior armazenado até então.

3) Objeto-relacional: O tipo de armazenamento Objeto-relacional (OR) é implementado por meio de UDTs, o que permite definir hierarquias de tipo e representar funções de distância como apresentado na Seção IV-B. No armazenamento OR, há três opções para a representação dos vetores de características, a saber: *(i)* cada característica pode ser armazenada em uma coluna, *(ii)* armazenamento em tabelas aninhadas e *(iii)* armazenamento em *arrays*. Como as duas primeiras opções compartilham limitações com a alternativa relacional, a terceira opção foi a selecionada nesse estudo para representar o armazenamento de vetores de características como OR. *Arrays* permitem armazenar todos os tipos de FV, incluindo FVs Adimensionais, e não requer implementar funções de distâncias para diferentes dimensionalidades. Além disso, o modelo OR permite representar FV Compostos como objetos complexos, o que facilita a interpretação. A seguir, são ilustrados exemplos de declaração de dois FVs Simples e de um FV Composto Heterogêneo[7].

```
-- FV Dimensional do tipo INTEGER
CREATE TYPE INT_FV AS VARRAY(64) OF INTEGER;
-- FV Adimensional de Texto
CREATE TYPE TEXT_FV AS
  VARRAY(10) OF VARCHAR(45);
--FV Composto Heterogêneo
CREATE TYPE COPHIR_FV AS OBJECT
( scalableColor       INT_FV,
  colorStructure      INT_FV,
  tags                TEXT_FV );
```

[7]No Oracle define-se o número máximo de características presentes no *array*, mas só as posições efetivamente ocupadas são armazenadas. A FD deve validar se o mesmo é dimensional, se necessário.

4) Semi-estruturado: O modelo semi-estruturado é o semanticamente mais próximo a natureza diversificada dos dados complexos e permite a representação de consultas como apresentado na Seção IV-B. Nesse estudo adotamos a representação `XML` para o tipo semi-estruturado, pois a implementação dessa estrutura se encontra madura em SGBDR comerciais, em particular com amplo suporte no Oracle. Assim, torna-se possível representar todos os tipos de vetores de características apresentados por meio de *XML Schemas* e manipular as características armazenadas através dos comandos `DML`, embora isso seja feito via um *parser*, o que introduz um limitador de desempenho. Adicionalmente, os SGBDR permitem que os dados sejam sejam armazenados como estruturas binárias (que chamaremos de tipo XML binário), objeto-relacionais (que chamaremos de XML e XML OR, alternadamente). A seguir é apresentado um exemplo de *XML Schema* para um FV Composto Heterogêneo.

```
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="FeatureVector">
 <xs:complexType><xs:sequence>
  <xs:element name="Component_Integer">
   <xs:complexType><xs:sequence><xs:element
    name="feature" type="xs:integer"
    minOccurs="282" maxOccurs="282"/>
   </xs:sequence></xs:complexType>
  </xs:element>
  <xs:element name="Component_String">
   <xs:complexType><xs:sequence><xs:element
    name="feature" type="xs:string"
    minOccurs="1" maxOccurs="32"/>
   </xs:sequence></xs:complexType>
  </xs:element>
 </xs:sequence></xs:complexType>
</xs:element></xs:schema>
```

D. Discussão

Todos os tipos de dados analisados para o armazenamento de vetores de características possuem suas respectivas vantagens e desvantagens. O modelo de armazenamento binário apresenta flexibilidade na implementação de todos os tipos de vetores de características, porém a manipulação é via funções externas ao SGBDR. O modelo de armazenamento relacional é o mais conhecido, mas requer a criação de novas FD para cada dimensionalidade e possui uma sintaxe carregada para passagem de elementos para funções de distância. O modelo de armazenamento semi-estruturado é o mais flexível na representação dos diversos tipos de vetores de características, contudo, exige *parsers* e usa um único tipo para todos os FVs, o que limita a verificação de tipos das funções de distância. Por fim, o modelo de armazenamento objeto-relacional é o melhor modelo quando considera-se flexibilidade e usabilidade. Além de ter suporte amplo em SGBDs comerciais, possui sintaxe simples para passagem de elementos para funções de distância, permite a melhor verificação de tipos e é capaz de representar todos os tipos de vetores de características de forma elegante.

V. EXPERIMENTOS

Os experimentos dessa seção avaliam o armazenamento e a recuperação de dados complexos com relação à sua representação em vetores de características por meio das quatro formas de armazenamento apresentadas na Seção IV-C. A implementação foi realizada no eFMI−SiR sobre o SGBDR Oracle, utilizando rotinas em SQL, PL/SQL, SQL/XML e C++. Foram utilizados um milhão de vetores de características de 282 dimensões provenientes do conjunto de dados *Content-based Photo Image Retrieval* (CoPhIR), que é uma coleção de fotos provenientes da rede social Flickr.

Para a recuperação por similaridade foram consideradas as funções de distância L_1 e L_2, que são "baratas", pois sua complexidade de cálculo é linear no número de dimensões, e a distância Mahalanobis, que é bem mais "cara", pois envolve diversos cálculos matriciais. O objetivo dos três experimentos é avaliar o desempenho dos tipos de representação de vetores de características considerando os aspectos de recuperação, inserção e armazenamento.

O SGBDR utilizado nos experimentos foi o Oracle 12c, que possui os recursos necessários para a implementação de todos os tipos de dados avaliados. A execução das avaliações se deu em um computador com as seguintes configurações: processador Intel(R) Xeon(R) CPU E5-2420 0 @ 1.90GHz, cache de 15MB, 8GB de memória RAM e S.O. Ubuntu 14.04 LTS. O disco usado nos experimentos foi um Seagate Constellation SATA 3.0 de 1.0TB e vazão 6.0 de Gb/s.

A. Avaliação da recuperação por similaridade

Nesse experimento foram utilizados 100 centros de buscas em todos os conjuntos de teste que não foram inseridos nas tabelas pesquisadas. Em todos os conjuntos de experimentos foi executada uma busca por abrangência para cada centro de busca, cujo limite permite filtrar entre 0,3% e 1% dos elementos consultados.

A Figura 4 mostra que o desempenho da recuperação de dados complexos armazenados no modelo relacional foi superior ao dos competidores, porém acompanhado de perto do armazenamento Objeto-Relacional em VArray. Um comportamento que foge a esse padrão é o desempenho do armazenamento binário em BLOB para o caso de vetores de características de alta dimensionalidade que são recuperados pela função de distância Mahalanobis. Nesse cenário, a recuperação de vetores de características de 282 dimensões pela Mahalanobis, considerando o armazenamento em BLOB, foi até 7% mais eficiente na comparação com o armazenamento relacional. De forma complementar, o armazenamento XML (OR e binário) apresentou o pior desempenho em todas as situações. Por exemplo, a recuperação de dados complexos no modelo relacional necessitou de, no máximo, 3%, 11% e 9% do tempo gasto pelo competidor XML OR considerando as funções L_1, L_2 e Mahalanobis, respectivamente.

B. Avaliação da inserção de dados complexos

Nesse experimento foram feitas vinte inserções de 100, 1.000 e 10.000 tuplas considerando as colunas dos tipos de dados analisados. As médias do tempo necessário para as inserções dessas tuplas é mostrada na Figura 5.

O desempenho da inserção de dados foi, em geral, mais eficiente no armazenamento VArray (OR) e menos eficiente no BLOB (Binário), como detalhado na Figura 5. A inserção de vetores de características no VArray gastou apenas 41% a 42% do tempo necessário para a inserção de dados como BLOBs, sendo que o tempo de inserção em ambos casos foi estável se comparado aos demais competidores. O armazenamento no modelo relacional e o armazenamento em XML (OR e binário) obtiveram o segundo e terceiros melhores resultados, respectivamente. Na comparação com o armazenamento relacional (2^o colocado), o VArray gastou apenas 68% e 54% do tempo para inserir vetores de características de 12 e 282 dimensões, respectivamente. Como última observação, o armazenamento em XML (OR ou Binário) teve um desempenho que se degenerou com o aumento da dimensionalidade.

C. Avaliação do espaço gasto para armazenar FVs

Nesse experimento foram realizadas medidas sobre o dicionário de dados do Oracle para obter os valores de espaço em disco ocupados pelas tabelas que armazenam efetivamente os vetores de características como colunas. Nós variamos essa análise para tabelas que incluem os vetores de características armazenados como diferentes tipos e considerando variações amplas nas quantidades de tuplas armazenadas (de dez mil até um milhão de entradas) e na dimensionalidade (de 12 até 282 dimensões) dos FVS em si. A Figura 6) detalha o espaço em disco gasto para cada uma das configurações avaliadas.

O armazenamento relacional apresentou o melhor desempenho dentre todos os competidores analisados. Na sequência, as representações em BLOB, XML Binário e VArray apresentaram um desempenho parecido, visto que os dois primeiros são armazenados da mesma forma – em arquivos binários e o XML tem uma pequena sobrecarga por seus metadados. Como contraponto ao tempo de recuperação, o armazenamento diretamente em tabelas com o tipo relacional requereu até 57% menos espaço do que o armazenamento em VArrays. Como última observação, o tipo de armazenamento XML OR foi o que demandou a maior quantidade de disco – entre $16\times$ e $28\times$ o espaço necessário no armazenamento relacional.

D. Discussão

Em resumo, os resultados indicam que o desempenho do armazenamento *relacional* superou os outros competidores. No entanto, o modelo de armazenamento *binário* (BLOB) apresentou o melhor desempenho para consultas que empregam funções de distância computacionalmente dispendiosas, enquanto que o modelo de armazenamento *objeto-relacional* (VArray) apresentou melhor desempenho para os casos de funções de distância com complexidade linear. Portanto, se considerados ambos os desempenhos (armazenamento e consulta), o modelo *objeto-relacional* (VArray) foi a estratégia que apresentou um melhor compromisso entre recuperação e gasto de memória secundária, no caso de funções de distância não-custosas. Para o caso complementar, o armazenamento

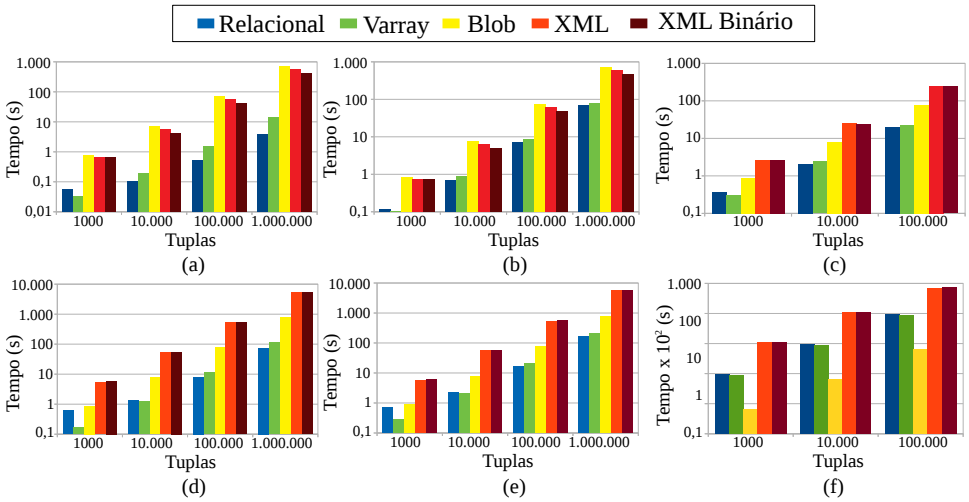

Fig. 4. Consulta por abrangência para 1% das tuplas contidas nas tabelas considerando as funções de distância L_1, L_2 e Mahalanobis, respectivamente. (a-c) Vetores de características de 12 dimensões e (d-f) vetor de características de 282 dimensões.

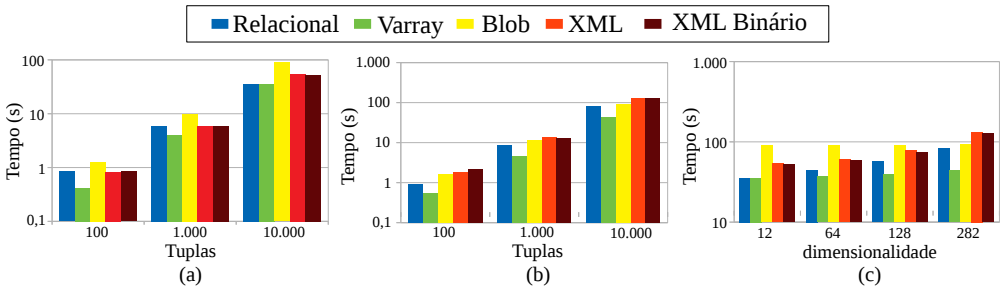

Fig. 5. Inserção de vetores de características de (a) 12 e (b) 282 dimensões. (c) Inserção de 10.000 vetores de características.

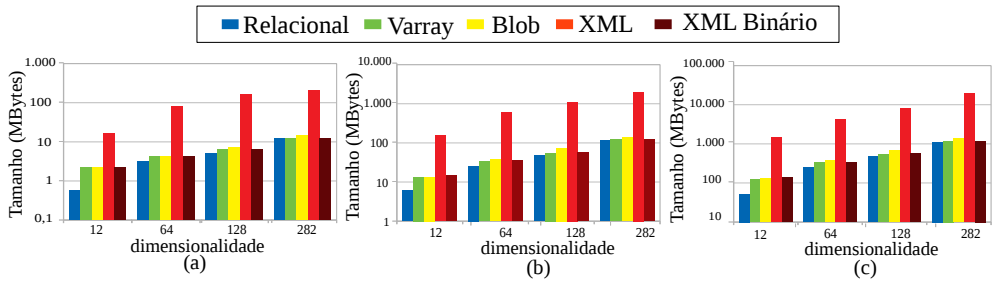

Fig. 6. Espaço em disco ocupado por vetores de características. Tabelas de (a) 10.000, (b) 100.000 e (c) 1.000.000 tuplas.

binário (BLOB) foi a solução mais adequada. Nesse sentido, os resultados mostram que é importante levar em consideração quais formas de comparação serão usadas para recuperar dados complexos com a finalidade de se otimizar o armazenamento e recuperação dessa categoria de dados em SGBDRs. A Tabela I apresenta um resumo comparativo dos pontos positivos e negativos de cada uma dessas quatro abordagens de armazenamento para vetores de características.

VI. Conclusões

Até onde sabemos, nenhuma avaliação da forma de representar e armazenar dados complexos utilizando SQL padrão, assim como uma categorização das abordagens existentes, foi realizada anteriomente. Trabalhos anteriores não são focados na discussão de padrões existentes em linguagens de alto nível que permitem expressar consultas por similaridade com transparência para o processador de consultas de SGBDRs.

Nesse estudo, preenchemos estas lacunas fornecendo: *(i)* uma avaliação dos modelos de armazenamento existentes em SGBDRs em termos de flexibilidade, eficiência e usabilidade e *(ii)* uma representação para as funções de distâncias que permite ao processador de consultas reconhecê-las e associá-las à operadores físicos. Quatro tipos de dados previstos na

TABLE I
ARMAZENAMENTO E RECUPERAÇÃO POR SIMILARIDADE - MODELOS DE DADOS SUPORTADOS PELA SQL PADRÃO.

Tipo do dado	Desempenho da recuperação (FD de baixo custo)	Desempenho da recuperação (FD de alto custo)	Sintaxe carregada para funções de distância	Desempenho da inserção	Desempenho do armazenamento	Manipulação dos dados
Relacional	Alto	Médio	Sim	Médio	Alto	DML
VArray	Alto	Médio	Não	Alto	Alto	DML e linguagem procedural
XML	Baixo	Baixo	Não	Baixo	Alto (binário) Baixo (OR)	DML e SQL/XML
BLOB	Baixo	Alto	Não	Médio	Alto	Funções externas

SQL padrão foram analisados experimentalmente: *binário*, *relacional*, *objeto-relacional* e *semi-estruturado*.

Os resultados experimentais mostram que o desempenho dos modelos de armazenamento *relacional* e *objeto-relacional* superaram os outros modelos para a maioria dos cenários, enquanto o modelo de armazenamento *binário* obteve o melhor desempenho para consultas que empregam comparações dispendiosas de dados. Como resultado geral, o modelo de armazenamento *objeto-relacional* apresentou um melhor compromisso quando considerados ambos os desempenhos (representação e armazenamento).

AGRADECIMENTOS

Este trabalho foi financiado pela CAPES, Fundação Araucária, FAPESP e CNPq.

REFERENCES

[1] C. Faloutsos, *Searching multimedia databases by content*. Springer Science & Business Media, 1996, vol. 3.

[2] P. Zezula, G. Amato, V. Dohnal, and M. Batko, *Similarity Search: The Metric Space Approach*, 1st ed. Springer, 2010.

[3] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[4] R. da Silva Torres and A. X. Falcao, "Content-based image retrieval: theory and applications." *Revista de Informática Teórica e Aplicada*, vol. 13, no. 2, pp. 161–185, 2006.

[5] M. C. N. Barioni, H. L. Razente, A. J. M. Traina, and C. Traina-Jr., "Seamlessly Integrating Similarity Queries in SQL," *Software: Practice and Experience*, vol. 39, no. 4, pp. 355–384, 2009.

[6] D. S. Kaster, P. H. Bugatti, A. J. M. Traina, and C. Traina-Jr., "FMI-SiR: A flexible and efficient module for similarity searching on Oracle database," *Journal of Information and Data Management*, vol. 1, no. 2, p. 229, 2010.

[7] M. Batko, D. Novak, and P. Zezula, "MESSIF: Metric similarity search implementation framework," in *Digital Libraries: Res. and Development*. Springer, 2007, pp. 1–10.

[8] L. R. Long, S. Antani, T. M. Deserno, and G. R. Thoma, "Content-based image retrieval in medicine: Retrospective assessment, state of the art, and future directions," *International Journal of Healthcare Information Systems and Informatics*, vol. 4, no. 1, pp. 1–16, 2009.

[9] D. Guliato, E. V. Melo, R. M. Rangayyan, and R. C. Soares, "PostgreSQL-IE: An image-handling extension for PostgreSQL," *Journal of Digital Imaging*, vol. 22, no. 2, pp. 149–165, 2009.

[10] Y. N. Silva, S. S. Pearson, and J. A. Cheney, "Database similarity join for metric spaces," in *International Conference on Similarity Search and Applications*. Springer, 2013, pp. 266–279.

[11] W. Lu, J. Hou, Y. Yan, M. Zhang, X. Du, and T. M., "MSQL: efficient similarity search in metric spaces using SQL," *The VLDB Journal*, vol. 26, no. 6, pp. 829–854, 2017.

[12] C. Vertan, M.-S. Badea, C. Florea, L. Florea, and S. Bădoiu, "Mpeg-7 visual descriptors selection for burn characterization by multidimensional scaling match," in *E-Health and Bioengineering Conference*. IEEE, 2017, pp. 253–256.

[13] H. M. Wallach, "Topic modeling: Beyond bag-of-words," in *International Conference on Machine Learning*. New York, NY, USA: ACM, 2006, pp. 977–984.

[14] R. Bueno, M. X. Ribeiro, A. J. M. Traina, and C. Traina-Jr., "Improving medical image retrieval through multi-descriptor similarity functions and association rules," in *International Symposium on Computer-Based Medical Systems*. IEEE, 2010, pp. 309–314.

[15] M. C. N. Barioni, D. S. Kaster, H. L. Razente, A. J. M. Traina, and C. Traina-Jr., "Querying multimedia data by similarity in relational DBMS," in *Advanced Database Query Systems: Techniques, Applications and Technologies*, 1st ed., L. Yan and Z. Ma, Eds. Hershey, PA, USA: IGI Global, 2011, ch. 14, pp. 323–359.

[16] E. Achtert, C. Böhm, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz, "Efficient reverse k-nearest neighbor search in arbitrary metric spaces," in *ACM SIGMOD International Conference on Management of Data*, 2006, pp. 515–526. [Online]. Available: http://doi.acm.org/10.1145/1142473.1142531

[17] H. L. Razente, M. C. N. Barioni, A. J. M. Traina, C. Faloutsos, and C. Traina-Jr, "A novel optimization approach to efficiently process aggregate similarity queries in metric access methods," in *Conference on Information and Knowledge Management*. ACM, 2008, pp. 193–202.

[18] K. P. Puttaswamy, S. Wang, T. Steinbauer, D. Agrawal, A. El Abbadi, C. Kruegel, and B. Y. Zhao, "Preserving location privacy in geosocial applications," *IEEE Transactions on Mobile Computing*, vol. 13, no. 1, pp. 159–173, 2014.

[19] L. O. Carvalho, E. Seraphim, T. F. Seraphim, A. J. M. Traina, and C. Traina-Jr., "Medinject: A general-purpose information retrieval framework applied in a medical context," in *International Symposium on Computer-Based Medical Systems*. IEEE, 2014, pp. 308–313.

[20] H. Cambazard, D. Mehta, B. O'Sullivan, L. Quesada, M. Ruffini, D. Payne, and L. Doyle, "A combinatorial optimisation approach to the design of dual parented long-reach passive optical networks," in *International Conference on Tools with Artificial Intelligence*. IEEE, 2011, pp. 785–792.

[21] R. F. Barroso, M. Ponciano-Silva, A. J. M. Traina, and R. Bueno, "Using boundary conditions for combining multiple descriptors in similarity based queries," in *Iberoamerican Congress on Pattern Recognition*. Springer, 2013, pp. 375–382.

[22] P. Budikova, M. Batko, and P. Zezula, "Query language for complex similarity queries," in *Advances in Databases and Information Systems*. Springer, 2012, pp. 85–98.

[23] M. Barioni, H. Razente, A. Traina, and C. Traina-Jr., "SIREN: A Similarity Retrieval Engine for Complex Data," in *PVLDB*. VLDB Endowment, 2006, pp. 1155–1158.

[24] Y. N. Silva, A. M. Aly, W. G. Aref, and P.-A. Larson, "SimDB: a similarity-aware database system," in *ACM SIGMOD International Conference on Management of Data*. ACM, 2010, pp. 1243–1246.

[25] S.-H. Cha, "Comprehensive Survey on Distance/Similarity Measures Between Probability Density Functions," *International Journal of Mathematical Models and Methods in Applied Sciences*, vol. 1, no. 4, pp. 300–307, 2007.

[26] R. Bueno, D. S. Kaster, A. A. Paterlini, A. J. M. Traina, and C. Traina-Jr., "Unsupervised scaling of multi-descriptor similarity functions for medical image datasets," in *International Symposium on Computer-Based Medical Systems*. IEEE, 2009, pp. 1–8.

[27] C. Traina-Jr., A. J. M. Traina, C. Faloutsos, and B. Seeger, "Fast indexing and visualization of metric datasets using slim-trees," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 14, no. 2, pp. 244–260, 2002.