Inclusão de Operadores Físicos de Junção por Similaridade em um SGBD Comercial

Inclusion of Similarity Join Physical Operators on a Commercial DBMS

Guilherme Queiroz Vasconcelos
Universidade de São Paulo
Avenida do Trabalhador Sãocarlense,400
São Carlos – SP, Brasil
gui.queirozv@usp.br

Daniel dos Santos Kaster
Universidade Estadual de Londrina
PR 445 Km 380
Londrina – PR, Brasil
dskaster@uel.br

RESUMO

Dados complexos, como vídeo, imagem e áudio, requerem formas particularizadas de consulta, armazenamento e indexação que SGBDs comerciais ainda não provêm. Uma das formas dos SGBDs oferecerem suporte a dados complexos é estender operadores relacionais para representar consultas por similaridade. Consultas por similaridade recuperam dados baseando-se em relações de similaridade entre dados armazenados, que são derivadas do conteúdo intrínseco dos dados. Um tipo importante de consulta por similaridade é a junção por similaridade, que retorna pares de elementos de dois conjuntos de entrada que atendem à condição de junção definida, que pode ser, por exemplo, se são mais próximos do que um dado limiar (junção por abrangência) ou se um elemento é um dos k-vizinhos mais próximos do outro (junção k-NN). Algoritmos existentes para a execução de junções por similaridade essencialmente consideram que os conjuntos/relações de entrada são lidos diretamente do disco. Contudo, a junção é uma das operações mais custosas em uma consulta e por isso atrasá-la no plano de execução e efetuar a junção sobre dados filtrados em memória normalmente gera ganho de desempenho. Neste artigo são apresentados algoritmos de junção por abrangência desenvolvidos para operar sobre dados filtrados em um SGBD comercial. A proposta é permitir a execução de junções por similaridade em posições diferentes no plano de consulta e avaliar como algoritmos distintos comportam-se em situações variadas. Resultados apresentados mostram que as melhores opções desenvolvidas são o algoritmo estado-da-arte DBSimJoin para entradas com filtros altamente seletivos e um algoritmo de junção baseada em índice para consultas em que a seletividade de junção é alta e um índice apropriado está disponível.

Palavras-Chave

Consultas por similaridade, algoritmos de junção, SGBDs comerciais.

SBSI 2017 June 5^{th} – 8^{th}, 2017, Lavras, Minas Gerais, Brazil

ABSTRACT

Complex data, such as video, images and audio, require particular forms of querying, storing and indexing that commercial DBMSs still do not provide. One of the solutions for DBMS to offer support for complex data is to extend relational operators to represent similarity queries. Similarity queries retrieve data based on similarity relations among stored data, which are derived from the intrinsic data content. One important type of similarity query is the similarity join that returns pairs of elements from two input datasets that satisfy the stated join condition, which can be for instance if they are closer to each other than a given threshold (Range join) or if one element is among the k-nearest neighbors of the other (k-NN join). Existing algorithms to execute similarity joins essentially consider input datasets/relations are directly read from disk. However, join is one of the most time-consuming operations in a query, therefore delaying it in the execution plan and join filtered data in memory usually results in performance gain. In this paper, we present range join algorithms developed to perform in memory on filtered data on top of a commecial DBMS. Our proposal is to allow executing similarity joins in different positions in the query plan and evaluate how distinct algorithms behave according to varied situations. Presented results show that best developed options are the state-of-the-art DBSimJoin algorithm for high selective filtered inputs and an index-based similarity join for queries in which the join selectivity is high and a proper data index is available.

CCS Concepts

•**Information systems** → **Query planning;**

Keywords

Similarity queries, join algorithms, commercial DBMSs.

1. INTRODUÇÃO

Muitas novas aplicações têm exigido lidar com grandes volumes de dados e também manipular dados como imagens, vídeo, áudio, textos longos, sequência genéticas, etc. Esses dados são chamados de dados não escalares, ou complexos, pois não definem uma relação de ordem total entre seus elementos [11]. A forma mais comum de se manipular os dados complexos é representá-los por meio de vetores de características extraídas a partir do conteúdo intrínseco do dado, por exemplo, características que correspondam sob algum

aspecto o conteúdo visual de uma imagem ou de parte dela, e recuperá-los através de consultas por similaridade.

Consultas por similaridade retornam elementos de um conjunto de dados que satisfazem a um dado critério de similaridade. Em essência, essas consultas utilizam uma função de distância para calcular o quão similar um elemento (vetor de características) é em relação aos demais.

As consultas mais utilizadas são as seleções por similaridade e junções por similaridade, e são utilizadas em especial na área médica [21], onde a recuperação de imagens por similaridade auxiliam a detecção de anomalias por simetria, em situações emergenciais [3] como incêndio e desastres naturais, nos quais as imagens obtidas via *crowdsourcing* são processadas e agrupadas pela similaridade das mesmas e também em sistemas biométricos [9]. Apesar disto, Sistemas de Gerenciamento de Bancos de Dados (SGBDs) existentes ainda não possuem suporte adequado para esses consultas. Assim, ao incluir-se operações baseadas em similaridade nos SGBDs, além de aumentar o poder de expressividade das linguagens de consulta, torna-se possível explorar formas de otimizar essas operações em consultas complexas combinando-se expressões mais simples [11].

A inclusão dos operadores de similaridades nos SGBDs pode ser feita através de interceptadores que recebem a consulta canônica e realizam as operações de similaridades antes de entregá-las ao processador do banco ou incluindo-se operadores físicos no SGBD que são invocados diretamente pelo processador de consultas. Os algoritmos de junção por similaridade têm sido amplamente estudados e um número considerável de algoritmos foram propostos, variando desde implementação fora do banco, modificações nos operadores de junção clássicos, entre outros [2]. Entretanto, a avaliação desses algoritmos essencialmente considera que as relações são lidas diretamente do disco. Em contrapartida, em consultas complexas os otimizadores de consultas dos SGBDs tipicamente postergam as junções no plano de execução, fazendo com que operações de seleção sejam executadas primeiro e as junções sejam realizadas sobre relações temporárias, tipicamente em memória principal.

Neste contexto, este artigo apresenta o desenvolvimento de operadores físicos de junção por abrangência em um SGBD comercial, projetados para execução em memória principal sobre relações temporárias. Foram escolhidos alguns dos algoritmos mais relevantes para junção por abrangência para implementação e análise de desempenho. O artigo detalha o processo adotado para inclusão desses algoritmos em operadores físicos no SGBD Oracle, bem como apresenta uma análise comparativa entre os algoritmos de loops aninhados, de junção baseada em índice e o *DBSimJoin*, que é baseado em *hashing* por similaridade.

Experimentos apresentados mostraram que o algoritmo *DBSimJoin* foi o mais eficiente quando não há índice disponível para realizar a junção por similaridade, e também nos casos em que as relações de entrada são sujeitas a filtros altamente seletivos. Por outro lado, o algoritmo de junção baseada em índice foi o mais rápido para consultas em que a seletividade de junção é alta e um índice está disponível.

O restante deste artigo está organizado conforme segue. A Seção 2 apresenta os conceitos inerentes à recuperação de dados complexos por similaridade incluindo métodos de acesso e consultas por similaridade, e também os principais algoritmos de junção por similaridade. A Seção 3 detalha a proposta de inclusão de operadores de junção no SGBD Oracle, incluindo detalhes de implementação. A Seção 4 apresenta uma avaliação experimental dos algoritmos desenvolvidos e a Seção 5 traz as conclusões do trabalho e propostas de extensão.

2. FUNDAMENTAÇÃO TEÓRICA

Esta seção tem como objetivo introduzir os conceitos de recuperação de dados complexos, incluindo consultas por similaridade e estruturas de indexação para dados complexos, e apresentar os principais algoritmos de junção por similaridade.

2.1 Consultas por similaridade

A recuperação de dados complexos é tipicamente realizada através das consultas por similaridade. Essas consultas demandam de apenas duas informações: o conjunto de dados e a função de distância para analisar a dissimilaridade entre elementos do conjunto [16]. As consultas mais utilizadas são as seleções e as junções por similaridade.[22, 16].

As seleções por similaridade incluem os dois tipos básicos de consulta a seguir.

- Consulta por abrangência (*Range query*). Esta consulta tem por objetivo encontrar todos os elementos similares ao elemento da consulta até no máximo um limiar. Um exemplo é: *"selecione as imagens que são similares à imagem de consulta i_q em até cinco unidades de similaridade"*.

- Consulta aos k-vizinhos mais próximos (k-*NN query*). A consulta aos k-vizinhos mais próximos retorna os k elementos mais similares ao elemento de consulta. São exemplos dessa consulta: *"selecione as k imagens mais similares à imagem i_q"* e *"identifique os três clientes mais próximos de um dado cliente c_q"*.

As principais variações da operação de junção por similaridade são as que se seguem.

- Junção por abrangência (*Range join*). A junção por abrangência combina dois conjuntos ou relações através de uma função de distância e um limiar de abrangência, recuperando pares de elementos cuja distância é menor ou igual ao limiar fornecido. Um exemplo desta consulta é: *"obtenha uma listagem que identifique os restaurantes que estão a, no máximo, 1 km de cada hotel"*.

- Junção dos k-vizinhos mais próximos (k-*NN join*). Dados dois conjuntos ou relações, uma função de distância e um inteiro k maior ou igual a um, a junção dos k-vizinhos mais próximos retorna os pares de elementos dos conjuntos ou relações de maneira que cada elemento da primeira relação é concatenado com cada um dos seus k-vizinhos mais próximos na segunda relação. Um exemplo é: *"associe cada capital do sudeste às 6 cidades brasileiras mais próximas a ela"*.

- Junção dos k-pares de vizinhos mais próximos (k-*Closest Pairs join*). Dados dois conjuntos ou relações, uma função de distância e um inteiro k maior ou igual a um, esta consulta retorna os k pares de elementos de relações diferentes que sejam os mais próximos entre si. Um exemplo desta consulta é: *"retorne os 10 pares*

escola-foco de infestação que estão mais próximos entre si e precisam de atuação imediata da secretaria de saúde".

A Figura 1 ilustra esses três tipos de junção por similaridade. Nessa figura os círculos pretos representam elementos de um conjunto R e os de cor cinza representam elementos do outro conjunto. A Figura 1(a) mostra uma junção por abrangência. Cada elemento cinza formará uma tupla com o elemento preto no círculo em que se encontra. Na junção aos k-vizinhos mais próximos, ilustrada na Figura 1(b), cada um dos 2 elementos cinzas formarão uma tupla com o elemento preto associado (neste caso, $k = 2$). A Figura 1(c) mostra uma junção dos k-pares de vizinhos mais próximos com $k = 4$. Observe-se que, neste caso, um dos elementos pretos é associado com dois elementos cinza, pois a distância entre o elemento preto e cada elemento cinza é menor do que as distâncias entre os demais pares de elementos.

2.2 Métodos de Acesso Métricos

Os dados complexos são representados por um conjunto de características que descrevem numericamente seu conteúdo, também conhecido como vetor de características. Um conjunto de vetores de características compõe o chamado espaço de similaridade, onde um determinado ponto nesse espaço representa as características de um objeto complexo [19]. Espaços de similaridade podem ser representados utilizando o conceito de espaço métrico. Formalmente, um espaço métrico representa um par $\mathbb{M} = \langle \mathbb{S}, \delta \rangle$, onde $\mathbb{S}$ é o conjunto de elementos e δ é a função de distância que expressa a relação de similaridade entre os elementos de $\mathbb{S}$ e satisfaz as propriedades de simetria, não negatividade e desigualdade triangular [22].

Devido ao alto custo computacional, as consultas por similaridade podem utilizar de estruturas que minimizem o numero de comparações e acessos ao disco ao reduzir o espaço de busca dos dados. Algumas das estruturas de indexação mais eficientes para a recuperação de dados complexos utilizam propriedades do espaço métrico, sendo conhecidas como métodos de acesso métricos (MAM). A eficiência dos métodos de acesso métricos está relacionada ao número de acessos ao disco e à quantidade de distâncias computadas [20].

Os MAMs baseados em árvore organizam a estrutura de forma hierárquica, gravando em seus nós internos a distância coberta e os pivôs que particionaram o conjunto de dados [12]. Em uma consulta, o elemento é comparado inicialmente com a raiz e em seguida com os pivôs, descartando porções de dados que não fazem parte do resultado da consulta, com isso reduz-se o número de cálculos de distância e acesso à disco [18]. Nessas estruturas a consulta por abrangência é realizada recursivamente para cada região intersectada pelo raio de abrangência cujo centro é o elemento de consulta. A consulta aos k-vizinhos mais próximos é tipicamente iniciada com raio de consulta infinito a partir da raiz e percorre a estrutura, procurando os elementos mais próximos e inserido-os ordenadamente em um conjunto de possíveis resultados e reduzindo gradativamente o raio de consulta, até que todas as regiões intersectadas tenham sido percorridas.

Na literatura encontram-se MAMs baseados em memória e em disco, sendo as baseadas em memória úteis para aplicações que requerem a construção do índice dinamicamente [18]. A M-tree [4] é o exemplo clássico de MAM dinâmico baseado em disco e muitas outras propostas são extensões à M-tree sob algum aspecto.

Uma dessas extensões é a Slim-tree [20], que é uma árvore dinâmica e balanceada que cresce de baixo para cima. A ideia principal é organizar os elementos em estruturas hierárquicas usando um representante como o centro de cada região da árvore. Os nós folha guardam todos os objetos da estrutura e os nós índice guardam os objetos que representam uma sub-árvore, ou uma região. Para inserir um elemento, o algoritmo tenta localizar um nó que cobre o elemento, isto é, o elemento pertence à região coberta pelo nó. Se não existir, seleciona-se um nó cujo representante é o mais próximo do elemento a ser inserido. Quando um nó atinge seu tamanho máximo, cria-se um outro nó para aquela região e os elementos são distribuídos entre os nós (*split*). Quando a raiz sofre uma nova raiz é alocada e a árvore cresce um nível.

2.3 Algoritmos de Junção

O operador relacional de junção toma um par de relações como argumento e uma condição de junção, retornando a combinação das relações onde os valores dos atributos em comum atendem às especificações da condição de junção. Quando a junção é com a mesma tabela chama-se de *self-join* e quando são relações diferentes é uma junção comum, ou *non-self-join.*Na literatura [8] encontram-se três principais algoritmos de junção para dados escalares, descritos a seguir.

- Junção por laços aninhados (*nested-loops join).* Este algoritmo toma duas relações R e S e para cada tupla em R, é testada a condição de junção para todos as tuplas de S, analisando uma única tupla de R por vez. Uma variação desse algoritmos é quando uma das relações possui um índice que permite satisfazer a condição de junção através de uma busca. Assim, para cada tupla da relação não indexada, por exemplo, R, as tuplas que satisfazem a condição de junção são selecionadas usando o índice da outra relação (S), ao invés de acessá-la por meio de uma varredura sequencial. Este algoritmo é conhecido como junção baseada em índice (*index-based join*).

- Junção por ordenação e intercalação (*merge join*). Neste algoritmo, ordena-se as relações de entrada para percorrê-las de forma linear. A cada iteração, um par de tupla é avaliado em relação à a condição de junção. Se satisfizer a condição, as tuplas serão concatenadas e avança-se um elemento em cada uma das relações. Caso contrário, avança-se uma unidade na relação cujo valor do atributo usado na condição de junção é o menor.

- Junção por *hash* (*hash join*). Neste algoritmo, cria-se uma tabela *hash* com a menor relação e agrupa-se os registros pelo valor da função de *hash* sobre o atributo utilizado na condição de junção. Em seguida, para cada elemento da segunda relação, aplica-se a mesma função de *hash*, gerando *buckets* correspondentes com tuplas de cada uma das relações. Por fim, a condição de junção é verificada considerando-se apenas elementos de *buckets* correspondentes, tipicamente usando um algoritmo de junção por laços aninhados.

Os algoritmos de junção tradicionais são incapazes de resolver operações com dados métricos sem modificações. Porém, há vários trabalhos que implementam o operador de

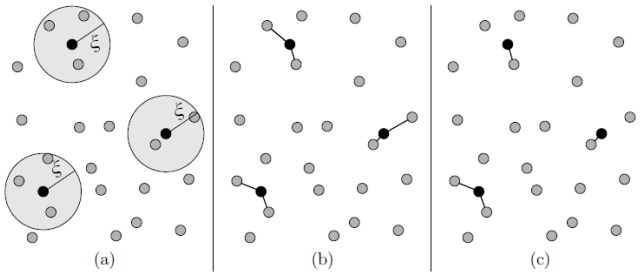


Figura 1: Tipos de junção por similaridade. a) Junção por abrangência. b) Junção dos k-vizinhos mais próximos, onde k é igual a 2. c) Junção dos k-pares de vizinhos mais próximos, com k = 4.

junção por similaridade e conseguem executar de forma eficiente essas consultas, propondo adaptações dos mesmos e em alguns casos utilizando uma estrutura de indexação apropriada [16].

Por exemplo, Silva et al. [15] propuseram um algoritmo chamado Join-Around, que, baseado no algoritmo de junção por ordernação e intercalação, combina as propriedades de abrangência e dos k-vizinhos mais próximos para avaliar dois conjuntos diferentes, concatenando elementos da primeira relação com os elementos mais próximos na segunda relação e retornando os pares distantes até um limiar fornecido na consulta. Já Olmes et al. [3] propuseram um algoritmo para juntar k pares elementos mais próximos. O algoritmo possui dois laços aninhados e uma fila de k-pares de elementos, ordenados pela distância de forma decrescente e separados em regiões utilizando pivôs. Quando dois objetos estão na mesma região em relação aos pivôs e suas distâncias são menores do que o raio corrente de consulta, eles são adicionados na fila. Quando k pares são obtidos, o par atual é comparado com o par do começo da fila e, se for o caso, substitui o mesmo. Dentre os tipos de junção, a junção por abrangência é a que têm sido mais amplamente estudada. Os algoritmos em destaque para essa operação são os descritos a seguir. O algoritmo EGO [1] impõe uma grade sobre o espaço métrico, separando em blocos de tamanho ϵ. Cada bloco é particionado até que nenhum elemento do bloco sobreponha outros blocos em uma distância ϵ e que seu tamanho seja suficientemente pequeno para ser resolvido pelo algoritmo de junção por laços aninhados. O GESS [6] divide o espaço em níveis, sendo que cada nível contém 2^{i+1} hiperquadrados, onde i é o índice do nível, iniciando em zero. Um hiperquadrado é uma região quadrada no espaço métrico, de lado epsilon, cujo elemento central é o pivô. O primeiro nível possui um hiperquadrado que abrange todo o conjunto de dados, o segundo nível possui 4 hiperquadrados e assim os níveis vão sendo divididos em hiperquadrados. O objeto pertencerá ao nível mais alto em que seu espaço não ultrapasse outro hiperquadrado. Para realizar a operação de junção, o algoritmo ordena os dados da célula onde está localizado o hiperquadrado e realiza a junção linearmente. O algoritmo *QuickJoin* [10] particiona os dados *in place* através da comparação entre dois pivôs, agrupando os dados mais próximos de um pivô em um lado e o dados mais próximos ao outro pivô em outro. Os dados são particionados recursivamente enquanto cabem na memória para ser resolvidos posteriormente pelo algoritmo de junção por laços aninhados. Silva et al. [17] propuseram uma variação iterativa deste algoritmo, chamada *DBSimJoin*, que prioriza a geração de resultados, resolvendo os conjuntos que não precisam ser reparticionados primeiro, liberando tuplas concatenadas para as demais operações no plano de consulta enquanto partições maiores são reparticionadas. Por fim, Dohnal et al. [7] propuseram um algoritmo de junção indexado que executa *self-join* de forma similar à junção por *hash*, utilizando a estrutura denominada *eD-Index*. O *eD-Index* particiona o conjunto de dados em níveis e para cada nível agrupa os elementos em regiões de separação e exclusão, através dos parâmetros ρ que é a distância máxima para o ϵ suportada pela estrutura, ϵ (raio de abrangência) e d (distância média entre os pivôs). O primeiro nível particiona todo o conjunto de dados e cada nível subsequente é criado particionando a região de exclusão do nível anterior. Cada região de separação possui um pivô e os elementos são inseridos na região do pivô mais próximo. Os elementos que se encontram a uma distância d + ρ e d - ρ de algum pivô são inseridos na região de exclusão, e os elementos que se encontram a distância d $\pm$ ($\rho \pm \epsilon$) são inseridos em sua respectiva região de separação e também na região de exclusão. Pearson et al. [14] estenderam esse trabalho para que a estrutura suportasse junção entre duas relações diferentes, propondo o algoritmo *I-SimJoin*.

Percebe-se que há várias propostas de algoritmos de junção na literatura. Algumas dessas propostas foram, inclusive, implementadas em um SGBD, como é o caso do *DBSimJoin*, que foi implementado no SGBD PostgreSQL. Contudo, essa implementação encontra-se disponível apenas em uma versão antiga do PostgreSQL, devido ao alto custo de manutenção da implementação para cada nova versão do SGBD. Neste contexto, esse trabalho propõe a inclusão de operadores físicos de junção que recebem como entrada relações filtradas, conforme apresentado na próxima seção.

3. INCLUSÃO DE JUNÇÃO POR SIMILARIDADE EM UM SGBD

Como mencionado anteriormente, os SGBDs ainda não possuem recursos nativos para manipulação de dados complexos, por isso é necessário o desenvolvimento de aplicações externas para realizar esse trabalho. Neste sentido, a proposta desse trabalho foi implementar algoritmos de junção em uma biblioteca externa que analisa a similaridade entre os dados complexos, tratando-os como objetos binários (*Binary Large Objects* – BLOBs).

O Oracle foi o SGBD escolhido, pois permite a integração entre o SGBD e as aplicações externas através de interfaces como a *ODCITable*, responsável por manipular dados e retorná-lo em forma de tabelas. Os algoritmos foram imple-

mentados utilizando essa interface e também a bilioteca de manipulação de dados complexos. Nesta seção serão apresentados os detalhes da proposta, incluindo os recursos utilizados, o projeto e aspectos da implementação dos algoritmos.

3.1 O Módulo de Recuperação por Similaridade

A presente proposta é uma extensão a um módulo de recuperação por similaridade [11]. Este módulo oferece recursos para a manipulação de dados complexos, como funções de distância, extratores de características e métodos de acesso métrico para executar consultas indexadas. Sua implementação é em C++ como uma biblioteca compartilhada dinâmica. Durante a execução de uma consulta que requer a manipulação de dados compelxos, o processador de consultas do SGBD realiza chamadas à este módulo, possibilitando forte integração com outros operadores de consulta relacional ou por similaridade. Além disso, permite a extensão do código para utilização específicas ao domínio de cada aplicação através dos tipos de dados e conjuntos de funções SQL que invocam as funções da biblioteca compartilhada dinâmica.

Com este módulo, é possível executar junções por abrangência seguindo duas abordagens: junção por laços aninhados e junção baseada em índice. A manipulação dos dados inicia-se na extração de características que gera um vetor de características para cada elemento e salva em um arquivo que em seguida é invocado e os dados são inseridos em um BLOB. O cálculo de similaridade é feito por funções de distância, que recebem um par de BLOBs contendo vetores de características e retornam o quão dissimilares eles são.

Considere-se que existe uma relação que armazena imagens `ALOI(img_id, image, signature)`, onde `img_id` é um identificador único, `image` é um atributo que armazena uma imagem digital e `signature` é um BLOB que contém o vetor de característica utilizado para representar a imagem. Utilizando-se esta relação, uma junção por abrangência pode ser representada conforme segue.

```
SELECT * FROM ALOI A, ALOI B
WHERE
euclidean_dist(B.signature, A.signature) <= 0.3;
```

Esta junção é realizada utilizando-se o algoritmo de junção baseada em índice, caso exista um índice sobre o atributo complexo `signature` na relação `B`, ou o algoritmo de laços aninhados, caso contrário. O algoritmo de laços aninhados não é muito eficiente, pois realiza chamadas sucessivas à biblioteca externa para calcular a distância entre elementos em cada passo do algoritmo. Já o algoritmo baseado em índice sempre acessa a relação original, mesmo que exista uma seleção que poderia ser antecipada à junção, como na consulta a seguir.

Embora as opções presentes no módulo sejam eficientes para algumas situações, é interessante disponibilizar outros algoritmos de junção que sejam capazes de superá-las em termos de desempenho, em particular quando as relações de entrada são filtradas, como no segundo exemplo apresentado. A seguir é descrito como novos operadores físicos de junção foram incluídos no SGBD.

3.2 Operadores Físicos de Junção como Procedimentos Externos

O Oracle permite que o usuário invoque procedimentos externos ao banco para manipulação de dados customizados, permitindo estender as funcionalidades do SGBD, através dos chamados cartuchos de dados (*Data Cartridges*) [5]. Uma funcionalidade oferecida para a construção de *Data Cartridges* para manipulação de dados externos é a interface *ODCITable*. Esta interface permite a criação de funções externas que retornam dados em forma de tabela, chamadas de *Pipelined Table Functions*. Para utilizar a interface *ODCITable* é necessário definir, no esquema do banco de dados, um tipo de dado que implementa as funções dessa inteface.

As principais funções da interface *ODCITable* são: *ODCITableStart*, responsável por inicializar os contextos necessários para manipulação dos dados; *ODCITableFetch*, que é invocada repetidamente até retornar toda a coleção de dados; e *ODCITableClose*, que encerra a comunicação entre o sistema externo e o banco e também libera a memória. Assim, ao realizar uma chamada à uma *pipelined table function*, o processador de consulta procura no *cartridge* especificado as implementações das funções da interface e executa as funções externamente. A tabela a ser retornada também precisa ser especificada como um tipo no esquema do banco.

Utilizando este recurso do SGBD, desenvolveu-se os algoritmos de junção por laços aninhados, cuja função é chamada de `NestedLoopsJoin`, e `DBSimJoin`. A implementação compreende duas camadas. A primeira fica no esquema do banco, para fazer a invocação das funções da interface *ODCITable* e também para que o Oracle saiba o tipo de dado que será retornado. A segunda camada está na biblioteca externa, onde estão implementadas as funções da interface invocada. O diagrama da Figura 2 mostra a sequência de operações para invocar os algoritmos de junção implementados. Ao realizar a consulta, o processador de consulta relaciona os tipos definidos no schema com os tipos implementados na biblioteca e, em seguida, invoca as funções da interface para executar o algoritmo de junção.Ao término do processamento, uma tabela que é a junção das duas entradas será retornada ao processador de consultas. Os detalhes de projeto de cada camada são apresentados na próxima seção.

3.3 Decisões de projeto

Na camada de esquema definiu-se que as funções *DBSimJoin* e *NestedLoopJoin* receberiam dois cursores, para as relações temporárias de entrada, um número que representa o raio de abrangência da junção e um inteiro que é o identificador da função de distância que deseja-se utilizar na consulta. O retorno das funções é uma tabela cujas tuplas possuem um par de chaves que atendem ao critério da junção e distância entre os elementos.

A camada externa é a biblioteca que implementa as junções em C++ através da *ODCITable*. Tanto o *DBSimJoin* quanto o *NestedLoopJoin* executam as mesmas funções para inicializar e terminar a operação. Porém, cada um possui sua implementação da função *ODCITableFetch*. A função *NestedLoopJoin* é uma implementação do algoritmo de junção por laços aninhados que traz as duas relações de entrada para a memória e aplica o algoritmo presente em [8]. E a função *DBSimJoin* é uma implementação do algoritmo *DBSimJoin* [17] que utiliza a função de partição de Jacox et al [10].

Para invocar os algoritmos de junção, utiliza-se uma sintaxe idêntica, alterando-se apenas o nome da função. Por

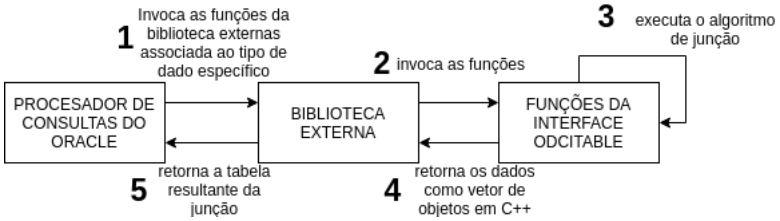


Figura 2: Sequência de operações para executar uma junção segundo a proposta do trabalho.

exemplo, para executar a consulta da Seção 3.1 utilizando-se o algoritmo DBSimJoin, a consulta deveria ser escrita como segue.

```
SELECT * FROM
TABLE( DBSIMJOIN(
CURSOR(SELECT img_id, signature FROM ALOI
WHERE img_id BETWEEN 15000 AND 20000),
CURSOR(SELECT img_id, signature FROM ALOI
WHERE img_id <= 13000) B, 0.3, 1) );
```

onde `1` é o identificador da função de distância.

Diferentemente da consulta da seção 3.1, a comparação é feita em memória sobre a relação filtrada. Trazer as relações para a memória pode resultar em grande aumento no desempenho quando a seletividade das condições de filtragens é alta, mesmo que para analisar a condição de junção seja feita leitura sequencial de todos os dados filtrados.

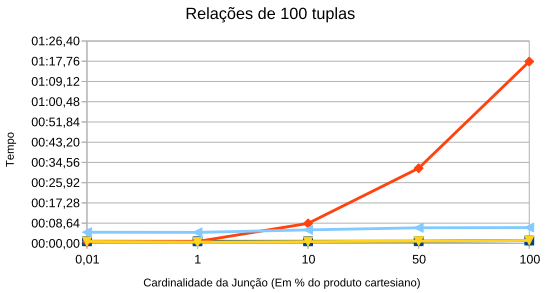
4. EXPERIMENTOS

Esta seção apresenta uma análise comparativa de desempenho dos algoritmos de junção por abrangência desenvolvidos. Os experimentos foram realizados em uma máquina Intel Core i7 5500U 2.4GHz com 8GB de RAM, com Ubuntu 64 bits como sistema operacional. A base de dados utilizada foi a ALOI (Amsterdam Library of Object Images) [13], que é uma coleção de fotos coloridas de objetos pequenos como brinquedos, alimentos, acessórios, etc. Foram selecionadas aleatoriamente 100 mil imagens deste conjunto. Este conjunto de dados possui um vetor de características de 256 dimensões, que representam um histograma de cores. Para cálculo de distância utilizou-se a distância Euclidiana.

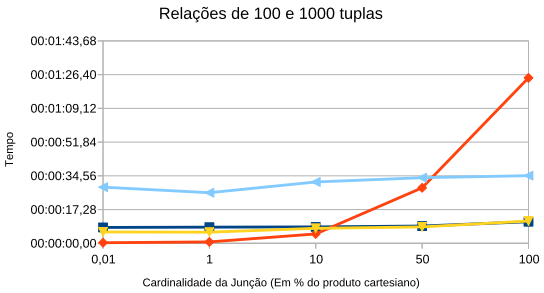
Foram realizados diversos testes, com consultas semelhantes à consulta apresentada na Seção 3.1, variando-se a seletividade das condições de filtragem das duas relações de entrada dos algoritmos e também a cardinalidade de junção (tamanho do resultado). Foram executados testes com a relação externa contendo 0,1% da relação de entrada (100 tuplas) ou 1% da entrada (1.000 tuplas). A relação interna, indexada no caso da junção baseada em índice, foi testada para os tamanhos 0,1%, 1% e 10% (10.000 tuplas) da entrada. A variação da cardinalidade é de 0,1%, 1%, 10%, 50% e 100% do tamanho do produto cartesiano das duas relações de entrada, que chega a 10 bilhões de tuplas para a combinação externa 1% e interna 10%. Os resultados apresentados correspondem sempre à média de 10 execuções de cada consulta.

A Figura 3 mostra o desempenho de quatro algoritmos para cada combinação de parâmetros: junção por laços aninhados com chamadas sucessivas à biblioteca externa (NL), junção por laços aninhados via cartucho de dados (NL(TABLE)), junção baseada em índice (INDEXED), e *DBSimJoin* (DBSIM). A Figura 3(a) mostra o desempenho dos algoritmos quando as duas relações de entrada têm tamanho igual a 0,1% da relação original após a execução das seleções baseadas no atributo `img_id`. Neste caso, os algoritmos mais eficientes são o NL(TABLE) e o DBSIM. O NL tem um desempenho inferior a esses algoritmos, o que é explicado pelo *overhead* inerente às sucessivas chamadas à biblioteca externa para realizar cálculos de distância. Comparado ao NL(TABLE), nesta configuração o NL teve desempenho em média 6 vezes mais lento. Já o algoritmo de junção baseada em índice tem desempenho comparável ao NL(TABLE) e ao DBSIM para cardinalidades pequenas de junção e degrada rapidamente com o aumento da cardinalidade, chegando a ser mais de 60 vezes mais lento para cardinalidade de 100%. Quando o tamanho da relação externa é mantido e o tamanho da relação interna cresce, o desempenho do INDEXED melhora significativamente. Na Figura 3(b), onde a relação interna tem tamanho 1% da original, esse algoritmo supera os demais para cardinalidades de junção até 10% do produto cartesiano, embora degrade rapidamente acima dessa cardinalidade. Já na Figura 3(c), que representa a configuração relação interna 0,1% e relação externa 10%, o algoritmo INDEXED supera os demais para toda cardinalidade de junção. Por exemplo, para a cardinalidade de junção 1%, o INDEXED é 18 vezes mais rápido que o DBSim e 90 vezes mais rápido que o NL(TABLE). Nesta configuração, o DBSIM é consideravelmente mais rápido que o NL(TABLE) para cardinalidades de junção menores, quase 5 vezes mais rápido para cardinalidade 1%, mas torna-se mais lento para altas cardinalidades. Já o NL é em média 3 vezes mais lento que o NL(TABLE).

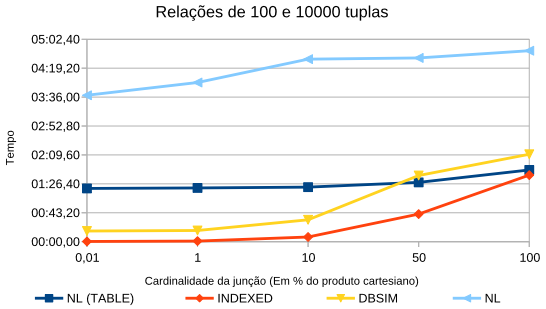
Quando o tamanho da relação externa é maior, o padrão de comportamento relativo entre os algoritmos é semelhante, porém com um distanciamento maior entre os algoritmos baseados em laços aninhados e os demais, pois o número de comparações aumenta multiplicativamente. A Figura 3(d) mostra o desempenho dos algoritmos quando a relação externa é 1% da relação original e o tamanho da relação interna também é de 1%. Nesse caso, as melhores opções são INDEXED para baixas cardinalidades de junção e DBSIM ou NL(TABLE) para altas cardinalidades. Já quando a relação externa é de 1% e a interna de 10% da relação original, o desempenho do INDEXED e do DBSIM são próximos, com vantagem para o INDEXED. Quando comparados ao NL(TABLE), o INDEXED é 50 vezes mais rápido e o DBSIM 10 vezes mais rápido para cardinalidade de 1% e, quando comparados ao NL, o INDEXED é 150 vezes mais rápido e o DBSIM 30 vezes mais rápido para a mesma cardinalidade. Portanto, sempre que as relações de entrada forem de pequeno e médio tamanho e a cardinalidade for



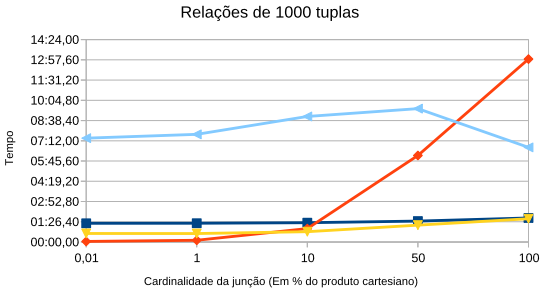
(a) Externa 0,1% e Interna 0,1%



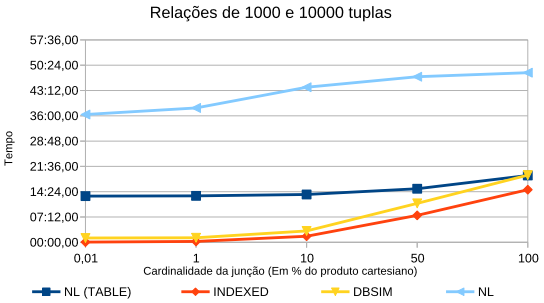
(b) Externa 0,1% e Interna 1%



(c) Externa 0,1% e Interna 10%



(d) Externa 1% e Interna 1%



(e) Externa 1% e Interna 10%

Figura 3: Tempo médio de execução dos algoritmos de junção variando-se o tamanho das relações de entrada e a cardinalidade de junção.

alta, recomenda-se utilizar a junção em memória, em especial o *DBSimJoin* cujo desempenho foi superior ou igual à junção por laços aninhados. Nos casos em que a relação interna for grande em comparação com a externa e houver um índice apropriado, a junção baseada em índice tende a ser mais eficiente, particularmente para cardinalidades de junção baixas. A tabela 1 apresenta o melhor caso para os algoritmos, com base nos experimentos realizados.

Tabela 1: Melhor desempenho de cada algoritmo.

Algoritmo	Tamanho das relações de entrada	Cardinalidade da Junção
Nested Loops	Muito pequeno	Baixa
DBSimJoin	Pequeno e médio	Alta
Junção Indexada	Médio e grande	Média

5. CONCLUSÃO

O crescimento da disponibilidade de dados complexos tem demandado a extensão dos SGBDs para atender as necessidades de gerenciamento desses dados. Dentre os operadores para execução de consultas por similaridade, um operador que tem recebido bastante atenção em termos de pesquisa é a junção por similaridade. Neste artigo foram apresentada uma proposta de inclusão de operadores físicos para execução de junção por abrangência desenvolvidos para execução em memória. Esta abordagem visa possibilitar a avaliação do comportamento de diferentes algoritmos para que possam ser movimentados no plano de execução, pois as relações de entrada podem ser produzidas por operações anteriores, tipicamente seleções.

Após um estudo dos algoritmos existentes na literatura, foram implementados uma versão do algoritmo de junção baseado em laços aninhados e o *DBSimJoin*, utilizando a

interface *ODCITable* do Oracle. Experimentos compararam os dois algoritmos com outros dois algoritmos de junção, outra versão do algoritmo de laços aninhados e um algoritmo de junção baseada em índice. Os resultados apresentados indicaram que, ao particionar os dados, como faz o *DBSimJoin*, obtém-se um ganho de execução em relação ao algoritmo de junção por laços aninhados, que realiza leitura sequencial. Esse algoritmo também mostrou-se mais eficiente que o algoritmo indexado para relaçes de entrada pequenas (i.e., com filtros bastante seletivos) e com alta cardinalidade de junção. Em outras situações, o algoritmo indexado mostrou a melhor alternativa.

Como trabalhos futuros, propõe-se utilizar algoritmos que utilizem estruturas de indexação baseadas em memória principal e analisar o comportamento dos algoritmos em planos de execução mais complexos.

6. REFERÊNCIAS

[1] C. Böhm, B. Braunmüller, F. Krebs, and H.-P. Kriegel. Epsilon grid order: An algorithm for the similarity join on massive high-dimensional data. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, pages 379–388, 2001.

[2] C. Böhm and H. P. Kriegel. A cost model and index architecture for the similarity join. In *Proceedings of the 17th International Conference on Data Engineering*, pages 411–420, 2001.

[3] L. O. Carvalho, L. F. D. Santos, W. D. Oliveira, A. J. M. Traina, and C. T. Jr. Efficient self-similarity range wide-joins fostering near-duplicate image detection in emergency scenarios. In *Proceedings of the 18th International Conference on Enterprise Information Systems*, pages 81–91, 2016.

[4] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 426–435, 1997.

[5] O. Database. Data cartridge's developer's guide, 2010. 11g Release 2 (11.2).

[6] J. P. Dittrich and B. Seeger. Gess: A scalable similarity-join algorithm for mining large data sets in high dimensional spaces. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 47–56, 2001.

[7] V. Dohnal, C. Gennaro, and P. Zezula. Similarity join in metric spaces using ed-index. In *Database and Expert Systems Applications*, pages 484–493, 2003.

[8] R. Elmasri and S. B. Navathe. *Sistemas de Banco de Dados*. Pearson Education Inc, Boston, MA, USA, 6th edition, 2010.

[9] K. Iqbal, M. O. Odetayo, and A. James. Content-based image retrieval approach for biometric security using colour, texture and shape features controlled by fuzzy heuristics. *Journal of Computer and System Sciences*, 78(4):1258–1277, 2012.

[10] E. H. Jacox and H. Samet. Metric space similarity joins. *ACM Trans. Database Syst.*, 33(2), 2008.

[11] D. d. S. Kaster. *Tratamento de Condições Especiais para Busca por Similaridade em Bancos de Dados Complexos*. PhD thesis, Instituto de Ciências Matemáticas e de Computação, 2012.

[12] T. Liu, A. W. Moore, and A. Gray. New algorithms for efficient high-dimensional nonparametric classification. *J. Mach. Learn. Res.*, 7:1135–1158, 2006.

[13] J. M., G. J. Burghouts, and A. W. M. Smeulders. The amsterdam library of object images. *International Journal of Computer Vision*, 61(1):103–112, 2005.

[14] S. S. Pearson and Y. N. Silva. *Index-Based R-S Similarity Joins*, pages 106–112. Springer International Publishing, Cham, 2014.

[15] Y. N. Silva, W. G. Aref, and M. H. Ali. The similarity join database operator. In *26th International Conference on Data Engineering (ICDE)*, pages 892–903, 2010.

[16] Y. N. Silva, W. G. Aref, P.-A. Larson, S. S. Pearson, and M. H. Ali. Similarity queries: Their conceptual evaluation, transformations and processing. *The VLDB Journal*, 22(3), 2013.

[17] Y. N. Silva, S. S. Pearson, J. Chon, and R. Roberts. Similarity joins: Their implementation and interactions with other database operators. *Information Systems*, 52:149–162, 2015.

[18] T. Skopal. *On Fast Non-metric Similarity Search by Metric Access Methods*, pages 718–736. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[19] A. J. M. Traina, C. Traina, J. M. Bueno, F. J. T. Chino, and P. Azevedo-Marques. Efficient content-based image retrieval through metric histograms. *World Wide Web*, 6(2):157–185, 2003.

[20] C. Traina Júnior and A. Juci Machado Traina. Fast indexing and visualization of metric data sets using slim-trees. *IEEE Transaction On Knowledge And Data Engeneering*, 14(2):244–260, 2002.

[21] K. Trojacanec, I. Dimitrovski, and S. Loskovska. Content based image retrieval in medical applications: An improvement of the two-level architecture. In *IEEE EUROCON 2009*, pages 118–121, 2009.

[22] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach*. Springer Publishing Company, Incorporated, United States, 1st edition, 2010.