UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

LUCAS OLIVEIRA PACÍFICO

# Algoritmos de Junção por Similaridade Sobre Fluxo de Dados

**Dissertação**

Goiânia
2020

Processo: 23070.032308/2020-48 Documento: 1568680

UFG

UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

**TERMO DE CIÊNCIA E DE AUTORIZAÇÃO (TECA) PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TESES E DISSERTAÇÕES NA BIBLIOTECA DIGITAL DA UFG**

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFG), regulamentada pela Resolução CEPEC nº 832/2007, sem ressarcimento dos direitos autorais, de acordo com a Lei 9.610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo das Teses e Dissertações disponibilizado na BDTD/UFG é de responsabilidade exclusiva do autor. Ao encaminhar o produto final, o autor(a) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

**1. Identificação do material bibliográfico**

[ x ] Dissertação [ ] Tese

**2. Nome completo do autor**

Lucas Oliveira Pacífico

**3. Título do trabalho**

Algoritmos de Junção por Similaridade Sobre Fluxo de Dados

**4. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador)**

Concorda com a liberação total do documento [ X ] SIM [ ] NÃO[1]

**[1]** Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante:
**a)** consulta ao(à) autor(a) e ao(à) orientador(a);
**b)** novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo da tese ou dissertação.
O documento não será disponibilizado durante o período de embargo.
Casos de embargo:
- Solicitação de registro de patente;
- Submissão de artigo em revista científica;
- Publicação como capítulo de livro;
- Publicação da dissertação/tese em livro.

**Obs. Este termo deverá ser assinado no SEI pelo orientador e pelo autor.**

Documento assinado eletronicamente por **Leonardo Andrade Ribeiro**, **Professor do Magistério Superior**, em 22/09/2020, às 22:15, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **LUCAS OLIVEIRA PACÍFICO**, **Discente**, em 23/09/2020, às 09:10, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1568680** e o código CRC **256CD87E**.

LUCAS OLIVEIRA PACÍFICO

# Algoritmos de Junção por Similaridade Sobre Fluxo de Dados

## Dissertação

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

**Área de concentração:** Ciência da Computação.

**Orientador:** Prof. Leonardo Andrade Ribeiro

Goiânia
2020

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Oliveira Pacífico, Lucas
Algoritmos de Junção por Similaridade Sobre Fluxo de Dados [manuscrito] / Lucas Oliveira Pacífico. - 2020.
LI, 51 f.

Orientador: Prof. Dr. Leonardo Andrade Ribeiro.
Dissertação (Mestrado) - Universidade Federal de Goiás, Instituto de Informática (INF), Programa de Pós-Graduação em Ciência da Computação, Goiânia, 2020.
Bibliografia.
Inclui gráfico, tabelas, algoritmos, lista de figuras, lista de tabelas.

1. Similaridade. 2. Fluxo de Dados. 3. Auto-Junção. I. Andrade Ribeiro, Leonardo, orient. II. Título.

CDU 004

UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

**ATA DE DEFESA DE DISSERTAÇÃO**

Ata nº **20/2020** da sessão de Defesa de Dissertação de **Lucas Oliveira Pacífico**, que confere o título de Mestre em Ciência da Computação, na área de concentração em Ciência da Computação.

Aos vinte e um dias do mês de agosto de dois mil e vinte, a partir das dez horas, via sistema de weconferência da RNP, realizou-se a sessão pública de Defesa de Dissertação intitulada **"**Algoritmos de Junções por Similaridade Sobre Fluxo de Dados**"**. Os trabalhos foram instalados pelo Orientador, Professor Doutor Leonardo Andrade Ribeiro (INF/UFG) com a participação dos demais membros da Banca Examinadora: Professora Doutora Carina Friedrich Dorneles (INE/UFSC), membra titular externa; e Professor Doutor Plínio de Sá Leitão Júnior, membro titular interno. A realização da banca ocorreu per meio de videoconferência, em atendimento à recomendação de suspensão das atividades presenciais na UFG emitida pelo Comitê UFG para o Gerenciamento da Crise COVID-19, bem como à recomendação de isolamento social da Organização Mundial de Saúde e do Ministério da Saúde para enfrentamento da emergência de saúde pública decorrente do novo coronavírus. Durante a arguição os membros da banca **fizeram** sugestão de alteração do título do trabalho. A Banca Examinadora reuniu-se em sessão secreta a fim de concluir o julgamento da Dissertação, tendo sido o candidato **aprovado** pelos seus membros. Proclamados os resultados pelo Professor Doutor Leonardo Andrade Ribeiro, Presidente da Banca Examinadora, foram encerrados os trabalhos e, para constar, lavrou-se a presente ata que é assinada pelos Membros da Banca Examinadora, aos vinte e um dias do mês de agosto de dois mil e vinte.

TÍTULO SUGERIDO PELA BANCA

Algoritmos de Junção por Similaridade Sobre Fluxo de Dados

Documento assinado eletronicamente por **CARINA FRIEDRICH DORNELES**, **Usuário Externo**, em 21/08/2020, às 12:57, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Plinio De Sa Leitão Junior**, **Professor do Magistério Superior**, em 21/08/2020, às 12:57, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Leonardo Andrade Ribeiro**, **Professor do Magistério Superior**, em 21/08/2020, às 12:57, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **LUCAS OLIVEIRA PACÍFICO**, **Discente**, em 21/08/2020, às 18:15, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?

**Lucas Oliveira Pacífico**

Graduou–se em Ciências da Computação na UFG - Universidade Federal de Goiás. Durante sua graduação, foi monitor das disciplinas de Programação Orientada a Objetos, Análise e Projeto de Algoritmos e Linguagens Formais e Autômatos de Ciência da Computação da UFG e pesquisador em trabalho científico na área de Problemas de Tráfego Urbano. Atualmente,cursando o Mestrado , na UFG - Universidade Federal de Goiás, sendo bolsista do CAPES/CNPQ e desenvolvendo trabalhos na área de banco de dados.

Dedico este trabalho aos meus amigos do laboratório 254, minha família e namorada.

# Agradecimentos

Gostaria de agradecer primeiramente ao meu orientador Leonardo pelos ensinamentos, orientações e a sua grande ajuda na escrita dos artigos.

Agradeço também aos meus amigos do laboratório 254, Altino, Deuslirio, Diogo, Eduardo, Vilson, Felipe e a todos os outros que esqueci de escrever o nome. Sem vocês, esse trabalho não seria possível.

Gostaria de agradecer minha namorada Mychelle pela ajuda e suporte prestado durante todo o desenvolvimento do trabalho.

Por fim, agradeço minha família pela capacidade de acreditar em meu trabalho.

Não esquecendo, agradeço a CAPES. Pois, o presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior Brasil (CAPES).

The greatest enemy of knowledge is not ignorance, it is the illusion of knowledge

**Stephen Hawking**,

.

# Resumo

Oliveira Pacífico, Lucas. **Algoritmos de Junção por Similaridade Sobre Fluxo de Dados**. Goiânia, 2020. 52p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

Na atual era de Big Data, dados são gerados e coletados em grande velocidade, o que impõe requisitos severos de desempenho e memória para processamento desses dados. Além disso, a presença de heterogeneidade nos dados demanda o uso de operações de similaridade, que são mais onerosas computacionalmente. Neste contexto, o presente trabalho investiga o problema de realizar junção por similaridade sobre fluxo contínuo de dados representados como conjuntos. O conceito de similaridade temporal é empregado, onde a similaridade entre dois itens de dados é reduzida de acordo com a distância entre os tempos de chegada associados aos mesmos. Os algoritmos propostos incorporam diretamente esse conceito para redução do espaço de comparação e consumo de memória. Mais ainda, uma nova técnica baseada na frequência parcial dos elementos de dados é apresentada para reduzir substancialmente o custo de processamento. Resultados da avaliação experimental realizada demonstram que as técnicas apresentadas propiciam ganhos substanciais de desempenho e bom uso de memória.

**Palavras–chave**

Similaridade, Fluxo de Dados, Auto-Junção

# Abstract

Oliveira Pacífico, Lucas. . Goiânia, 2020. 52p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

In today's Big Data era, data is generated and collected at high speed, which imposes strict performance and memory requirements for processing this data. Also, the presence of heterogeneity data demands the use of similarity operations, which are computationally more expensive. In this context, the present work investigates the problem of performing similarity join over a continuous stream of data represented by sets. The concept of temporal similarity is employed, where the similarity between two data items decreases with the distance in their arrival time. The proposed algorithms directly incorporates this concept to reduce the comparison of space and memory consumption. Moreover, a new technique based on the partial frequency of the data elements is presented to substantially reduce processing cost. Results of the experimental evaluation performed demonstrate that the techniques presented provide substantial performance gains and good memory usage.

**Keywords**

Similarity, Streaming, Self-Join

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Lista de Algoritmos

CAPÍTULO 1

# Introdução

Todas as organizações, por menores que sejam, possuem quantidades cada vez maiores de informações a serem armazenadas. Todavia, a manipulação destes dados se tornou impossível de ser realizada manualmente, pois sua utilização, além de demorada, é passível de erros. Nesse sentido, tornou-se mais fácil encontrar a informação em um banco de dados, ou seja, as informações são salvas em meios digitais e recorrem à necessidade de um sistema gerenciador de banco de dados.

Os Sistemas Gerenciadores de Banco de Dados (SGBDs) relacionais foram inicialmente projetados para operar em dados do tipo numérico e cadeias de caracteres, tornando estes dados tangíveis à aplicação dos operadores de comparação por igualdade e relacional. Os operadores de comparação por igualdade ($=$ e $\neq$) em princípio são aplicáveis a qualquer tipo de dado e são responsáveis por definir a igualdade ou não dos elementos. Os operadores relacionais ($<, \leq, >$ e $\geq$) necessitam que os elementos sejam pertencentes a domínios que atendam à Relação de Ordem Total (ROT), propriedade que permite os elementos de um certo conjunto definir entre si quem procede ou sucede um ao outro.

A crescente complexidade das informações em diferentes áreas do conhecimento fez emergir a necessidade de novos tipos de dados, não convencionais ou complexos, como imagens, vídeos, sons, séries temporais e entre outros. Para dados complexos, resta apenas a realização de comparações por igualdade. Todavia, essa estratégia de comparação não possui muita utilidade para diversos domínios, pois a possibilidade de duas imagens serem exatamente iguais, por exemplo, é mínima.

Com a ascensão da sociedade moderna e o surgimento desses novos tipos de dados, a quantidade de informação trafegada na internet e armazenada em bancos de dados, saíram da ordem de *Megabytes*,*Gigabytes* e passaram a se tornar *Terabytes* ou *Petabytes*. A evolução dos dados e sua magnitude, contribuiu para a fomentação do termo *Big Data*, representação para uma enorme quantidade de dados.

Umas das principais características do fenômeno *Big Data* é a velocidade com que os dados são produzidos continuamente ao longo do tempo. Um cenário comum de aplicação que reproduz esse fluxo contínuo de dados, são as redes sociais, Internet das

Coisas, sensores e uma ampla variedade de registros de eventos, (*logs*) em sistemas. Essa demanda tem impulsionado intensos esforços de pequisa e desenvolvimento de sistemas para processamento de fluxo de dados [1, 10].

Os requisitos para diversos sistemas de processamento em fluxo de dados influenciam o modelo computacional para sua implementação. Por exemplo, muitas aplicações possuem a necessidade de se comparar dados atuais com dados históricos, também existe a necessidade de processar esses dados e produzir uma resposta instantânea (Regras 5 e 8 em [37]). Para produzir resultados em tempo real, é necessário manter os dados em memória principal e, com isso, evitar latências extremas causadas por acessos ao disco. Entretanto, este objetivo é inviável, considerando que o fluxo de dados pode ser contínuo e ilimitado.

O problema torna-se ainda mais desafiador na presença de imperfeições no fluxo de dados, que devem ser corrigidas sem causar atrasos nas operações (Regra 3 em [37]). No caso de múltiplos fluxos de dados que foram originados de diferentes fontes e concentrados para um fluxo único, essas imperfeições podem incluir a presença dados duplicados e difusos, isto é, dados representando uma mesma informação, mas que não são necessariamente cópias idênticas entre si. A identificação desse tipo de redundância demanda comparações de similaridade, que são mais onerosas computacionalmente em relação a simples comparações de igualdade.

Fluxos de dados possuem uma natureza temporal intrínseca, ou seja, um rótulo de tempo (*timestamp*) é tipicamente associado a cada item de dados e determina, por exemplo, o tempo de chegada desse item. Esta informação temporal (i.e. tempo de chegada) é pertinente à noção de similaridade, pois determina a distância temporal entre dois itens de dados. Desta maneira, a similaridade entre dois itens de dados pode ser reduzida de acordo com a distância temporal entre os mesmos.

**Tabela 1.1:** *Mensagens sobre um evento esportivo recebidas de diferentes fontes.*

| ID | Fonte | Tempo | Mensagem |
|---|---|---|---|
| 1 | X | 270 | Lance importante na entrada da grande área. |
| 2 | Y | 275 | Lance perigoso na entrada da grande área. |
| 3 | Z | 420 | Lance importante na entrada da grande área. |

Como um exemplo, considere um serviço que publica informações em tempo real sobre eventos esportivos. Este serviço agrega fluxos de outros provedores de serviços independentes com o intuito de fornecer ao usuário um conjunto mais rico de informações ("lance a lance"). Por outro lado, a publicação de múltiplas mensagens descrevendo o mesmo evento pode degradar a experiência do usuário. Uma solução para este problema é realizar uma *(auto)junção por similaridade* sobre o fluxo de mensagens — uma junção

por similaridade retorna todos os pares de dados de uma coleção cuja similaridade entre si seja maior que um *limiar de corte predefinido*. Uma nova mensagem somente será publicada caso não existam mensagens anteriores similares à mesma. Neste contexto, a informação temporal é fundamental para o cálculo de similaridade, porque duas mensagens textualmente similares podem ser consideradas distintas se a distância temporal entre as mesmas for *grande*. Por exemplo, a Tabela 1.1 mostra três mensagens, de três fontes diferentes, descrevendo lances de uma partida de futebol. As três mensagens são similares entre si. Entretanto, observando o tempo de chegada a cada mensagem, pode-se concluir: enquanto as duas primeiras mensagens descrevem um mesmo lance, a terceira mensagem descreve um lance distinto (mesmo sendo idêntica a primeira mensagem).

O conceito de similaridade temporal (ou similaridade dependente do tempo) foi formalmente definida em [26]. Esse conceito é explorado no projeto de algoritmos de junção por similaridade sobre fluxo de dados representados como vetores. O algoritmo proposto mais eficiente emprega diretamente a similaridade temporal para reduzir a quantidade de comparações. Além disso, a informação temporal estabelece um fator de "esquecimento"para itens de dados: após um determinado tempo, um item de dados não poderá ser similar a qualquer item futuro no fluxo de dados e poderá ser descartado para reduzir o consumo de memória.

O presente trabalho propõe algoritmos para junção por similaridade sobre fluxos de dados, ao qual, os objetos são representados como conjuntos. Existe uma vasta literatura abordando junções de conjuntos por similaridade sobre dados estáticos [12, 42, 31, 42, 33, 41]; entretanto, não se tem conhecimento de propostas anteriores para fluxo de dados. Os algoritmos propostos adotam a noção de similaridade temporal definida em [26] e também explora suas propriedades para reduzir o espaço de comparação e consumo de memória. Uma limitação do trabalho de [26] é que elementos de dados que possuem grande frequência podem prejudicar substancialmente a efetividade dos filtros empregados. Por fim, este trabalho propõe uma nova técnica para geração de conjuntos baseada em uma tabela de frequência parcial, que mitiga a limitação de objetos com alta frequência. Os resultados da avaliação experimental realizada demonstram que os algoritmos apresentados propiciam ganhos substanciais de desempenho,(*speedups* de até 10x), em relação às primeiras versões do algoritmo.

## 1.1 Objetivo Geral

O objetivo geral do presente trabalho é propor algoritmos para junção por similaridade em fluxos de dados, ao qual os objetos possuem uma dependência temporal e são representados como conjuntos. Para alcançar tal objetivo, foram definidos os cinco objetivos específicos, que são:

1. Extração e limpeza de dados reais coletados em redes sociais e a construção de um ambiente para geração artificial de tempo (*timestamp*) para os objetos, pois não são todos os conjuntos de dados que possuem um tempo vinculado ao objeto;
2. Adaptação, desenvolvimento e avaliação de um algoritmo baseado no PPJoin para ser utilizado como base de referência.
3. Projeto, desenvolvimento e avaliação de dois algoritmos para junção por similaridade sobre fluxos de dados;
4. Projeto, desenvolvimento e avaliação de uma técnica para ordenar os elementos dos conjuntos recebidos no fluxo de dados.;

## 1.2 Organização do Trabalho

O restante do trabalho é estruturado como segue. No Capítulo 2 é descrito conceitos teóricos importantes para o trabalho. Em sequência, no Capítulo 3 é discutido e detalhado os trabalhos relacionados. No Capítulo 4 é apresentado as técnicas e os algoritmos propostos no presente trabalho. Capítulo 5 descreve os experimentos realizados e analisa os resultados obtidos. Por fim, o Capítulo 6 apresenta as conclusões e propostas de trabalhos futuros.

CAPÍTULO 2

# Referencial Teórico

O presente capítulo apresenta as definições básicas que envolvem funções de similaridade e processamento em fluxo de dados usados neste trabalho, e está organizado da seguinte forma. A Seção 2.1 apresenta a definição e o conceito de similaridade. Em sequência, a Seção 2.2 apresenta como a tokenização de conjuntos é realizada. A Seção 2.3 apresenta as fórmulas utilizadas para se calcular a interseção. Brevemente, a Seção 2.4 apresenta a definição de junção por similaridade. Seção 2.5 apresenta as estratégias de processamento que foram consideradas para o desenvolvimento do trabalho. A Seção 2.6 apresenta os filtros utilizados durante as etapas de filtragem dos algoritmos. Seção 2.7 apresenta os conceitos de similaridade temporal e horizonte temporal. A Seção 2.8 define formalmente o problema apresentado no trabalho. Por fim, a Seção 2.9 apresenta um dos algoritmos usados como base de referência para o trabalho.

## 2.1 Similaridade

Diversas aplicações do mundo real requerem a resolução do problema de consulta por similaridade, ao qual se tem o objetivo de encontrar todos os pares de objetos, cuja semelhança ultrapasse um limite especificado [8]. Assim, uma maneira quantitativa para definir se dois objetos são similares ou duplicados é com a utilização das funções de similaridade que fornecem uma medida numérica para o grau em que duas entidades são iguais ou diferentes [31].

A escolha do tipo de função de similaridade depende da finalidade da tarefa. Trabalhos anteriores despenderam esforços na identificação de uma função que fosse a melhor para todos os domínios e nenhuma foi encontrada [43]. Duas classes de funções de similaridade que são bastante populares e amplamente utilizadas, são as funções de distância de edição e as baseadas em conjuntos. As funções de distância de edição são baseadas em caracteres, ao qual, a distância entre duas strings é dada pelo menor número de operações necessárias para transformar um objeto textual em outro, onde uma operação pode ser uma inserção, remoção ou substituição de um caractere [20].

O presente trabalho irá focar somente na classe de funções baseadas em conjuntos e técnicas de consulta por similaridade textual. No restante do documento, será utilizado apenas o termo função de similaridade para denotar a função de similaridade baseada em conjuntos, exceto quando dito explicitamente o contrário.

## 2.2 *Tokenização*

Um processo amplamente conhecido e bastante utilizado para a conversão de registros textuais em conjuntos é a utilização de processos de tokenização. Processo que possui como objetivo a divisão do documento em unidades indivisíveis, representações atômicas, e definidas como *tokens*. A forma mais simples para se tokenizar um registro textual é considerar que cada palavra é um *token*. Como por exemplo, o texto“Lucas Pacífico”, seria convertido no conjunto de tokens {“Lucas”,“Pacífico”}.

Entretanto, a tokenização de textos para palavras não é uma técnica interessante no contexto de identificação de duplicatas, pois impediria que palavras que tenham apenas um erro de digitação fossem contabilizadas em uma intersecção.

No presente trabalho, adotou-se a utilização da técnica de tokenização conhecida como q-grams,*substrings* que podem ser obtidas através do deslocamento de uma janela de tamanho $q$ nos caracteres do registro textual original [38]. Da forma que, para um registro textual $s$, o conjunto de *q-grams* de $s$ é denotado por $Q_q(s)$. Exemplo do processo *q-grams* é demonstrado a seguir:

**Exemplo 1 (Conversão textual para Qgrams)**

*s:{“Lucas Pacífico”}*

*Adiciona-se Q-1 caracteres especiais no prefixo e no sufixo:*

*s:{“##Lucas Pacífico##”}*

*Em seguida os tokens são gerados com a utilização de uma notação numérica para identificar cada elemento de forma singular e distinguir tokens iguais:*

$Q_3(s)$*:{“##L1”,“#Lu1”,“Luc1”,“uca1”, “cas1”, “as 1”, “s P1”,“ Pa1”,“Pac1”, “ací1”,“cíf1”,“ífi1”, “fic1”, “ico1”, “co#1”,“o##1” }*

## 2.3 Cálculo de Interseção

As funções de similaridade transformam cada registro textual em um conjunto de tokens, em seguida é obtido a noção de similaridade através da sua intersecção. Da forma que, dois conjuntos $r$ e $s$, são similares a partir de uma função $sim(r,s)$, na qual

se retorna valores entre $[0,1]$ onde 0 significa totalmente diferente e 1 totalmente igual. Um popular exemplo de função é *Jaccard*, onde a similaridade é calculada através da razão da intersecção pela união de dois conjuntos. A Tabela 2.1 demonstra as fórmulas das principais funções de similaridade baseadas em conjuntos.

**Tabela 2.1:** *Funções de Similaridade Baseadas em Conjuntos*

| Função | Definição |
|---|---|
| *Jaccard* | $\frac{\lvert(r \cap s)\rvert}{\lvert(r \cup s)\rvert}$ |
| *Dice* | $\frac{2 * \lvert r \cap s\rvert}{\lvert r\rvert + \lvert s\rvert}$ |
| Cosseno | $\frac{\lvert r \cap s\rvert}{\sqrt{\lvert r\rvert * \lvert s\rvert}}$ |

Uma função de similaridade bastante utilizada no contexto de junção por similaridade textual, é a similaridade de Jaccard, cuja definição está na Tabela 2.1. Dados dois conjuntos de elementos $r$ e $s$, temos que, a similaridade Jaccard é definida como $J(r,s) = \frac{|r \cap s|}{|r \cup s|}$, ou seja, é a razão entre o número de *tokens* em comum (intersecção), dividido pela união dos elementos. O Exemplo 2, demonstra a utilização passo a passo da similaridade de Jaccard.

Trabalhos anteriores salientaram que a função de Jaccard apresenta uma eficácia comparável e competitiva em relação a outras funções de similaridade [11]. Portanto, o presente trabalho irá utilizar a função de Jaccard como função de similaridade.

**Exemplo 2** *Considere os conjuntos x e y abaixo, derivados das duas primeiras mensagens na Tabela 1.1 (fontes X e Y):* $x = \{$*'Lance', 'importante', 'na', 'entrada', 'da', 'grande', 'area'*$\}$ *e* $y = \{$*'Lance', 'perigoso', 'na', 'entrada', 'da', 'grande', 'area'*$\}$. *Tem-se que* $J(x,y) = \frac{6}{7+7-6} = 0.75$.

## 2.4 Junção por Similaridade

A junção por similaridade é amplamente explorada por diversos autores, também é conhecida como Busca por Similaridade de Todos os Pares [8]. Seu problema está focado em encontrar todos os pares em um conjunto de dados cuja similaridade é maior do que um determinado *threshold*. Uma definição formal sobre junção por similaridade feita por (Ribeiro et al., [31]), diz que:

**Definição 1** *Dado um universo de elementos U, uma coleção D, onde cada conjunto é composto por um número de elementos de U, uma função de similaridade Sim(r, s), que mapeia o par de conjuntos r e s para um número em* $[0,1]$ *e um threshold* $\gamma$*,* $0 \leq \gamma \leq 1$*, a*

*junção por similaridade tem como objetivo identificar todos os pares r, s, com r e s ∈ D, que satisfazem o predicado de similaridade Sim(r, s) ≥ γ.*

## 2.5 Estratégias de Processamento

Popularmente, existem duas estratégias de processamento para junção por similaridade. A primiera estratégia, apresentada na Figura 2.2, representa os conjuntos em relações satisfazendo a Primeira Forma Normal, na qual, as tuplas armazenam cada elemento do conjunto juntamente com o identificador único do conjunto. A junção por similaridade pode ser realizada usando tecnologia relacional e a linguagem SQL [12, 4].

**Tabela 2.2:** *Estratégia de processamento na 1º Forma Normal utilizando os valores da Tabela 1.1.*

| Conjunto | Token |
|---|---|
| $x_1$ | Lance |
| $x_1$ | importante |
| $x_1$ | na |
| $x_1$ | entrada |
| $x_1$ | da |
| $x_1$ | grande |
| $x_1$ | area |

| Conjunto | Token |
|---|---|
| $x_2$ | Lance |
| $x_1$ | perigoso |
| $x_2$ | na |
| $x_2$ | entrada |
| $x_2$ | da |
| $x_2$ | grande |
| $x_2$ | area |

O segunda estratégia de processamento é baseado em Índice Invertido e ilustrado na Figura 2.1, é construído para mapear os *tokens* para listas de conjuntos que os contêm [8]. Para cada conjunto, o índice é examinado para geração de candidatos que serão posteriormente comparados com este conjunto durante a etapa de verificação.

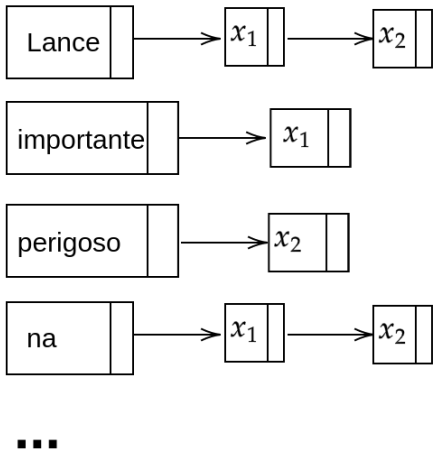


**Figura 2.1:** *Lista Invertida utilizando os valores da tabela 1.1.*

Trabalhos anteriores mostraram que as abordagens baseadas em índices invertidos superam as abordagens em tecnologia relacional em termos de desempenho [8].

Portanto, o presente trabalho irá utilizar a representação por índice invertido para conjuntos.

## 2.6 Filtragem de Candidatos

Devido ao fato que cálculos de similaridade são onerosos computacionalmente, existem diversas otimizações que têm o objetivo de diminuir a quantidade de cálculos realizados para a similaridade e o tempo gasto para estas operações e, com isso, construir soluções para que seja possível obter um desempenho satisfatório. A seguir, as principais otimizações utilizadas serão apresentadas.

### 2.6.1 Interseção Mínima

Dado um predicado de similaridade sobre dois conjuntos, é possível definir uma *interseção mínima* para que o predicado seja satisfeito.

**Lema 1 (Interseção Mínima [12])** *Dado um limiar de corte $\gamma$, para quaisquer dois conjuntos x e y, a interseção mínima é denotada por $I_{min}$:*

$$J(x,y) \geq \gamma \iff |x \cap y| \geq I_{min}(x,y,\gamma) = \frac{\gamma}{1+\gamma} \times (|x| + |y|).$$

### 2.6.2 Filtro por Tamanho

Intuitivamente, a diferença de tamanho entre dois conjuntos não pode ser muito grande. Assim, pode-se descartar rapidamente os pares de conjuntos, ao qual, os seus tamanhos possuem uma diferença substancial.

**Lema 2 (Filtro por Tamanho [34])** *Para dois conjuntos qualquer x e y, um limiar $\gamma$, temos que:*

$$J(x,y) \geq \gamma \implies \gamma \leq \frac{|x|}{|y|} \leq \frac{1}{\gamma}.$$

A partir do Lema 2 é possível definir tamanhos mínimos e máximos de acordo função de similaridade utilizada, melhor demonstradas na Tabela 2.3.

### 2.6.3 Filtro por Prefixo

A representação baseada em conjuntos permite a aplicação de diversas otimizações. Uma das técnicas mais populares e efetivas é o chamado *filtro por prefixo*, que permite descartar pares de conjuntos dissimilares verificando apenas uma fração dos mesmos. Para isso, assume-se que os tokens do universo $U$, de onde os elementos de todos os conjuntos são obtidos, estão ordenados de acordo uma determinada ordem global.

**Tabela 2.3:** *Definição de minsize e maxsize para filtragem por tamanho.*

| Função | $minsize(x)$ | $maxsize(x)$ |
|---|---|---|
| *Jaccard* | $\gamma\|x\|$ | $\frac{\|x\|}{\gamma}$ |
| *Dice* | $\gamma\|x\|$ | $(2-\gamma)\|x\|$ |
| *Cosseno* | $\gamma^2\|x\|$ | $\frac{\|x\|}{\gamma^2}$ |

**Lema 3 (Filtro por Prefixo [12])** *Dado um conjunto x e um threshold* $\gamma$*, seja* $pref(x,\gamma) \subseteq x$ *o subconjunto de x composto por seus primeiros* $|x| - \lceil |x| \times \gamma \rceil + 1$ *tokens. Então, para quaisquer dois conjuntos x e y, tem-se:*

$$J(x,y) \geq \gamma \implies pref(x,\gamma) \cap pref(y,\gamma) \neq \varnothing$$

### 2.6.4 Filtro por Posição

**Lema 4 (Filtro por Posição [42])** *Dado um conjunto x, seja* $w = x[i]$ *um token de x na posição i e que divide x em duas partições,* $x_e(w) = x[1,..,(i-1)]$ *e* $x_d(w) = x[i,..,|x|]$*. Então, para quaisquer dois conjuntos x e y, tem-se:*

$$J(x,y) \geq \gamma \implies |x_e \cap y_e| + min(|x_d|,|y_d|) \geq I_{min}(x,y,\gamma).$$

## 2.7 Similaridade Temporal

Este trabalho assume que a entrada de dados é um fluxo de conjuntos. A cada conjunto $x$ está associado um rótulo temporal, denotado por denotado por $t(x)$, indicando o tempo de chegada do mesmo. Formalmente, o fluxo de entrada é denotado por $\mathcal{F} = \langle ..., (x_i, t(x_i)), (x_i, t(x_{i+1}), ... \rangle$.

O conceito de similaridade temporal captura a intuição de que a similaridade entre dois conjuntos esmaece com a distância temporal entre os mesmos. Para isso, a diferença entre os tempos de chegada desses conjuntos é incorporada no cálculo de similaridade.

**Definição 2 (Similaridade Temporal [26])** *Dados dois conjuntos x e y, seja* $\Delta t_{xy} = |t(x) - t(y)|$ *a diferença entre os tempos de chegadas dos mesmos. A similaridade temporal entre x e y é dada por*

$$J_{\Delta t}(x,y) = J(x,y) \times e^{-\lambda \times \Delta t_{xy}},$$

*onde* $\lambda$ *é um fator de decaimento temporal.*

**Exemplo 3** *Considere novamente os conjuntos x e y no Exemplo 2, derivadas das duas primeiras mensagens na Tabela 1.1. Seja* $\lambda = 0.01$. *Como* $\Delta t_{xy} = 5$, *então* $J_{\Delta t}(x,y) = 0.75 \times e^{-\lambda \times 5} \approx 0.71$. *Considere agora a terceira mensagem (fonte Z) representada por um conjunto z. Apesar dos conjuntos x e z serem idênticos, tem-se* $\Delta t_{xz} = 150$ *e, portanto,* $J_{\Delta t}(x,z) = 1 \times e^{-\lambda \times 150} \approx 0.22$.

É importante observar que $J_{\Delta t}(x,y) = J(x,y)$ para $\Delta t_{xy} = 0$ ou $\lambda = 0$ e tende para zero quando $\Delta t_{xy}$ se aproxima de infinito, a uma taxa exponencial modulada por $\lambda$. O fator de decaimento e juntamente com o limiar de similaridade permitem definir um *filtro por tempo*: dado um conjunto $x$, após um certo período chamado horizonte temporal, nenhum novo conjunto poderá ser similar a $x$. O horizonte temporal pode ser compreendido como uma janela deslizante em relação ao tempo.

**Lema 5 (Filtro por Tempo)** *Dado um fator de decaimento* $\lambda$, *seja* $\tau = \frac{1}{\lambda} \times log\frac{1}{\gamma}$ *o horizonte temporal. Então, para quaisquer dois conjuntos x e y tem-se:*

$$J_{\Delta t}(x,y) \geq \gamma \implies \Delta t_{xy} < \tau.$$

## 2.8 Definição do Problema

Após a introdução de todas as definições necessárias, o problema considerado no presente trabalho é formalmente definido a seguir.

**Definição 3 (Junção de Similaridade sobre Fluxo de Conjuntos)** *Seja* $\mathcal{F}$ *um fluxo de conjuntos rotulados pelo tempo de chegada,* $\gamma$ *um limiar de similaridade e* $\lambda$ *um fator de decaimento, uma junção de similaridade sobre* $\mathcal{F}$ *retorna todos os pares de conjuntos* $(x,y)$ *em* $\mathcal{F}$ *tal que* $J_{\Delta t}(x,y) \geq \gamma$.

## 2.9 PPJoin

O algoritmo PPJoin proposto por [42] é similar ao algoritmo proposto por [8], mas com a diferença que faz utilização de uma filtragem posicional. O *ppjoin* realiza um armazenamento da posição dos q-gramas para os dois conjuntos nas listas invertidas e no mapa de interseção parcial. Os pares de conjuntos que não possuem q-gramas suficientes para garantir uma similaridade maior ou igual ao limiar, são desconsiderados. É importante notar que mesmo os pares de conjuntos que já foram considerados candidatos podem ser descartados pela filtragem posicional.

Na linha 3, é definido o mapa $M$ para armazenar os valores parciais da interseção do conjunto de busca $x$ com os conjuntos das listas invertidas. Nas linhas 5-10 ocorre a aplicação dos filtros baseado em tamanho e interseção mínima em todos os conjuntos da lista. Após a fase de filtragem, é feita a indexação dos pares candidatos na linha 14. Antes da fase de verificação, os q-gramas restantes em x e y são conferidos para garantir que sejam suficientes para alcançar a similaridade escolhida. Após a verificação, o conjunto x será adicionado às listas invertidas dos n-gramas de seu prefixo.

**Algoritmo 1:** Algoritmo PPJoin.

**Entrada:** Uma coleção de conjuntos $C$ ordenada por tamanho; cada conjunto é ordenado de acordo com $O_{df}$; um limiar $\gamma$ e uma função de similaridade *sim*

**Saída:** Um conjunto $S$ contendo todos os pares,t.q., $sim(x,y) \geq \gamma$

```
1  I_1, I_2, ... I_|U| ← ∅; S ← ∅
2  para cada x ∈ C faça
3      M ← um mapa vazio associando id de conjunto a um inteiro
4      para cada q ∈ maxprefix(x) faça
5          para cada (y, j) ∈ I_q faça
6              se |y| < minsize(x) então
7                  I_q ← I_q − (y, j)
8              senão
9                  M(y) ← (M(y).os + 1, i, j)
10                 se M(y).os + min(rem(x,i), rem(y,j), ≤ I_min(x,y) então
11                     M(y).os ← −∞
12     S ← S U verificar(x, M, γ)
13     para cada q_i ∈ {|x| − I_min{x} + 1} faça
14         I_q ← I_q U(x, i)
15 retorna S
```

CAPÍTULO 3

# Trabalhos Relacionados

Existe uma grande variedade de pesquisas sobre a forma mais eficiente para se realizar a junção por similaridade entre conjuntos [34, 12, 42, 39, 31, 28, 33, 24, 41]. Otimizações populares, como a filtragem por tamanho, prefixo e por posição, foram incorporadas em nossos algoritmos e previamente explicadas no Capítulo 2. Recentemente, Wang et al. [41] explorou as relações entre os conjuntos para que se tornasse possível melhorar o desempenho – a principal percepção é que conjuntos semelhantes, produzem resultados semelhantes. No entanto, uma das técnicas subjacentes, chamada de *index-level skipping* depende da criação de todo o índice invertido de forma prévia ao início do processamento e, portanto, não pode ser utilizada em nosso contexto, pois novos conjuntos estão chegando continuamente.

Além disso, a junção de similaridade de conjuntos foi abordada em uma ampla variedade de configurações, incluindo: plataformas distribuídas [39, 16]; arquiteturas multi-core [28, 33]; SGBDs relacionais, declarativamente em SQL [32] ou dentro do mecanismo de consulta, como um operador físico [12]; ambientes em nuvem [36]; algoritmos integrados em *cluster* [29, 30]; e probabilístico, seja para aumentar o desempenho (à custa de perder alguns resultados válidos) [9, 13] ou modelando dados incertos [22]. No entanto, nenhum desses estudos anteriores considerou junção de similaridade sobre fluxos de conjuntos.

Existem trabalhos anteriores que pesquisaram à respeito da junção por similaridade sobre fluxo de dados, mas os objetos são representados como vetores, da forma que, a similaridade entre dois vetores é calculada com a utilização da distância euclidiana [21, 23] ou cosseno [26]. Lia e Chen [21] propuseram uma abordagem adaptativa baseada em um modelo formal de custos para a junção por similaridade de múltiplas vias de fluxos de dados. Posteriormente, os mesmos autores, abordam a junção por similaridade sobre fluxos incertos de dados [23].

Morales e Gionis [26] introduziram a noção da similaridade dependente do tempo. Os autores adaptaram os existentes algoritmos de junção por similaridade para objetos representados como vetores, AllPairs [8] e L2AP [3], e aplicaram incorporação da dependência temporal nas propriedades de similaridade, ou seja, exploraram o tempo

para reduzir a quantidade de pares candidatos a serem avaliados e dinamicamente remover entradas antigas que estavam presentes no índice invertido. O atual trabalho segue uma abordagem semelhante à apresentada pelos autores, mas os detalhes dessas otimizações não são diretamente aplicáveis ao contexto do trabalho, pois o mesmo, foca em um fluxo de objetos representados como conjuntos.

Realizar o processamento de todos os objetos para um certo fluxo de dados ilimitado é claramente inviável. Portanto, algum método deve ser utilizado para limitar a parte do fluxo a ser processada em cada avaliação de consulta. O modelo de janela deslizante é popularmente utilizado para processamento de similaridade em fluxos de dados [21, 23, 35]. Como já mencionado, apenas uma quantidade indeterminada de objetos recentes do fluxo são processados e a similaridade é calculada em cada avaliação de consulta [6]. Semelhantemente, a dependência temporal adotada no presente trabalho também induz uma janela deslizante e é chamado de horizonte temporal e também possui uma quantidade variável de dados de fluxo para o seu período temporal.

A seleção por similaridade em fluxo de dados encontra todos os objetos que são similares para uma determinada consulta [19]. Até certo ponto, a junção por similaridade pode ser vista como uma sequência de pesquisas usando cada objeto que chega do fluxo como um objeto de consulta. A diferença fundamental nesse contexto é que o *threshold* é fixo para todas as junções, ao ponto que, o mesmo pode variar junto com as consultas realizadas.

A consulta por k-pares similares também tem sido foco de estudos no ambiente de fluxo de dados [35, 2]. Focando em fluxo de vetores, Shen et al. [35] propuseram uma estrutura que suporta consultas com diferentes funções de similaridade e tamanhos de janela. Amagata et al. [2] apresentou um algoritmo para a auto-junção com $k$NN, uma consulta do tipo top-k que encontra os $k$ objetos mais semelhantes para cada objeto. Este trabalho assume objetos representados como conjuntos, no entanto, o cenário dinâmico considerado é muito diferente: em vez de um fluxo de conjuntos, o foco está em um fluxo de atualizações inserindo e excluindo continuamente elementos de conjuntos existentes.

Por fim, detecção de duplicatas em fluxos de dados é um problema bem estudado [25, 15, 17]. Uma abordagem comum para lidar com fluxos ilimitados é empregar estruturas de dados probabilísticas que utilizam o espaço de forma otimizada, como Bloom Filters e Quocient Filters, juntamente com modelos de janelas. No entanto, essas propostas visam detectar duplicatas exatas e, portanto, a correspondência de similaridade não é abordada.

CAPÍTULO 4

# Contribuições

No presente capítulo é apresentado nossa solução para a resolução do problema de junção por similaridade sobre fluxo de conjuntos, formalmente definido na Seção 2.8. A Seção 4.1 foca na discussão do algoritmo de base, SSPJ, e como será utilizado para base de criação e comparação os algoritmos seguintes. Em sequência, na Seção 4.2 é discutido o algoritmo SSTR, ao qual, é apresentado com detalhes o seu funcionamento. Na Seção 4.3 é apresentado uma técnica para a ordenação dos tokens do conjunto, ao qual, se é utilizado uma amostragem do fluxo de dados para a geração da frequência. Por fim, na Seção 4.4 é apresentado o principal algoritmo do presente trabalho e se baseia numa alteração da estrutura geral do algoritmo SSTR com o intuito de melhorar a eficácia do filtro por tempo e modificar como é realizado a indexação dos conjuntos candidatos.

## 4.1 Algoritmo Set Stream PPJoin (SSPJ)

A maioria dos algoritmos de junção por similaridade do estado da arte seguem a estrutura de filtragem e verificação [24]. Ou seja, a estrutura garante que a coleção de conjuntos de entradas é escaneada de forma sequencial e que cada conjunto passe pelas fases de filtragem e verificação. Durante a fase de filtragem, os tokens do atual conjunto em análise (daqui em diante é chamado de conjunto sondagem) são utilizados para encontrar pares potenciais de conjuntos semelhantes que já foram processados (a seguir, é chamado de conjunto candidatos). Os filtros discutidos no Capítulo 2 são aplicados para reduzir o número de candidatos, consequentemente, essa fase é apoiada pela utilização de um índice invertido, que é construído dinamicamente de acordo com o processamento dos conjuntos. Por fim, na fase de verificação, a similaridade entre o conjunto sondagem e cada um dos conjuntos candidatos é calculada completamente e os pares que satisfazem o *threshold* de similaridade são enviados para o resultado.

A forma exaustiva, para se realizar o cálculo da similaridade sobre fluxo de dados se baseia em executar as fases de filtragem, verificação e indexação para um algoritmo existente em cada conjunto recebido. O decaimento temporal pode ser aplicado

**Algoritmo 2:** O algoritmo Set Stream PPJoin para fluxo de conjuntos.

**Entrada:** Fluxo de Conjuntos $\mathcal{S}$, Threshold $\gamma$, Decaimento $\lambda$
**Saída:** Todos os pares $(x,y) \in \mathcal{S}$ s.t. $J_{\Delta t}(x,y) \geq \gamma$

```
1  I_i ← ∅ (1 ≤ i ≤ |U|)
2  enquanto true faça
3      x ← read(S)
4      M ← mapa vazio do ID do conjunto para int
5      para i ← 1 até |pref(x, γ)| faça
6          k ← x[i]
7          para cada (y, j) ∈ I_k faça
8              se |y| < |x| × γ então
9                  continue
10             ubound ← 1 + min(|x| − i, |y| − j)
11             se M[y] + ubound ≥ O(x, y, γ) então
12                 M[y] ← M[y] + 1
13             senão
14                 M[y] ← −∞
15         I_k ← I_k ∪ (x, i)
16     R' ← Verificar(x, M, γ)
17     R ← AplicarDecaimento(R', γ, λ)
18     Emit(R)
```

à similaridade dos pares retornados pela verificação em uma fase de pós-processamento, antes de se enviar os resultados para a saída.

O Algoritmo 2 refere-se à abordagem exaustiva para o algoritmo PPJoin [42], daqui em diante chamado de SSPJ (Set Stream PPJoin), um dos algoritmos com melhor desempenho em uma avaliação empírica recente [24]. O algoritmo processa continuamente conjuntos do fluxo de entrada à medida que chegam. A fase de filtragem usa os tokens do prefixo (Linha 5) para analisar o índice invertido (Linha 7). Cada conjunto encontrado na lista invertida é considerado candidato e verificado em relação às condições utilizadas no filtro baseado em tamanho (Linha 8) e o filtro posicional (Linhas 10 a 11). Uma referência ao conjunto sondagem é anexada à lista invertida e associada para cada token de prefixo (Linha 15). Não foi mostrado no algoritmo, mas a fase de verificação (Linha 16) pode ser altamente otimizada para explorar a ordem dos tokens de maneira semelhante à mesclagem e à sobreposição vinculada para definir as condições para uma parada antecipada [31]. Finalmente, o decaimento temporal é aplicado e a última verificação no *threshold* é executada para produzir uma saída (Linha 17).

Claramente, a abordagem acima tem duas desvantagens. Primeiro, o consumo de espaço do índice invertido pode ser exorbitante e rapidamente exceder a memória disponível. Pior ainda, uma grande parte do índice pode ser composta por entradas

obsoletas, ou seja, entradas que referenciam conjuntos que não serão semelhantes a qualquer conjunto que chegue no futuro. Segunda desvantagem, o decaimento temporal somente é aplicado após a fase de verificação. Como consequência, muita computação na fase de verificação é desperdiçado nos pares de conjuntos que não podem ser semelhantes devido à diferença nos horários de chegada.

## 4.2 Algoritmo Set Similarity Join over Stream (SSTR)

Agora, o próximo algoritmo proposto chamado de SSTR (Set Similarity Join over Stream), para junção por similaridade em fluxos de dados. O SSTR explora as propriedades de definição da similaridade temporal para evitar problemas existentes dentro da abordagem exaustiva. Primeiro, SSTR remove dinamicamente entradas antigas ainda existentes dentro do índice invertido e que estão fora da janela de tempo induzida pelo conjunto sondagem e pelo horizonte temporal. Segundo, é utilizado o decaimento temporal para formar-se um novo limite de similaridade entre o conjunto sondagem e cada conjunto candidato. O novo threshold é maior que o original, o que aumenta a eficácia dos filtros posicionais e baseados em tamanho.

As instruções para SSTR são formalizadas no Algoritmo 3. As referências para os conjuntos cuja a diferença no tempo de chegada em relação ao conjunto sondagem são maiores que o horizonte temporal são removidos à medida que as listas invertidas são processadas (Linha 8). Observe que, as entradas na lista invertida são classificadas em ordem crescente com base no tempo de chegada, logo, todas as entradas obsoletas são agrupadas no início das listas invertidas. Para cada conjunto candidato, um novo valor de *threshold* é calculado (Linha 10) que é usado no filtro baseado em tamanho e para calcular o limite de sobreposição (Linhas 11 e 15, respectivamente). Da mesma forma, o *threshold* aumentado e específico do candidato também é usado na fase de verificação para obter os limites de sobreposição maiores e melhorar a eficácia das condições de parada antecipada. Por esse motivo, o parâmetro de decaimento temporal é passado para o processo de *Verificação* (Linha 20), que agora produz diretamente os pares para o conjunto de saída[1].

Mesmo com a remoção das entradas obsoletas presente nas listas invertidas, o SSTR ainda pode gerar problemas de alto consumo de memória para as janelas temporais que contêm muitos conjuntos. O cenário descrito pode acontecer devido a parâmetros de decaimento de tempo muito pequenos, consequentemente, e cria existência de janelas grandes ou durante o processamento do fluxo se conduz a existência de janelas de alta

[1]Em nossas implementações, evitamos a realização de cálculos repetidos de limites específicos para alguns candidatos e sobrepomos esses limites armazenando-os no mapa $M$.

**Algoritmo 3:** O algoritmo SSTR.

**Entrada:** Fluxo de Conjuntos $\mathcal{S}$, Threshold $\gamma$, Decaimento $\lambda$
**Saída:** Todos os pares $(x,y) \in \mathcal{S}$ s.t. $J_{\Delta t}(x,y) \geq \gamma$

$\tau = \frac{1}{\lambda} \times ln\frac{1}{\gamma}$
$I_i \leftarrow \varnothing \ (1 \leq i \leq |\mathcal{U}|)$
**enquanto** *true* **faça**
  $x \leftarrow$ **read** $(\mathcal{S})$
  $M \leftarrow$ mapa vazio do ID do conjunto para `int`
  **para** $i \leftarrow 1$ **até** $|pref(x,\gamma)|$ **faça**
    $k \leftarrow x[i]$
    Remova todos $(y,j)$ de $I_k$ t.q. $\Delta t_{xy} > \tau$
    **para cada** $(y,j) \in I_k$ **faça**
      $\gamma' \leftarrow \frac{\gamma}{e^{-\lambda \times \Delta t_{xy}}}$
      **se** $|y| < |x| \times \gamma'$ **então**
        $M[y] \leftarrow -\infty$
        **continue**
      $ubound \leftarrow 1 + min(|x|-i, |y|-j)$
      **se** $M[y].s + ubound \geq O(x,y,\gamma')$ **então**
        $M[y].s \leftarrow M[y].s + 1$
      **senão**
        $M[y] \leftarrow -\infty$
    $I_k \leftarrow I_k \cup (x,i)$
  $R \leftarrow$ **Verify** $(x, M, \gamma, \lambda)$
  **Emit** $(R)$

densidade. Nesses casos, se faz necessário sacrificar a pontualidade do processamento e recorrer a algum método de aproximação, como processamento em lote [6]. No entanto, considerando um cenário prático em que um limite máximo de memória é definido, o algoritmo SSTR pode reduzir drasticamente a frequência do processamento em lote, se comparado com a abordagem base (SSPJ), como demonstrado empiricamente no Capítulo 5.

## 4.3 Ordenação por Tabela de Frequência Parcial

Recapitulando a partir do Capítulo 2, a filtragem por prefixo exige que todos os conjuntos sejam classificados de acordo com uma ordem total do universo de tokens, o filtro posicional também requer esse ordenamento global. Por mais que, qualquer ordenação dos tokens garanta a correção dos filtros, a escolha de uma ordem específica tem implicações no desempenho [12]. A abordagem padrão se baseia em ordenar os conjuntos em relação frequência dos tokens no conjunto de dados, pois dessa forma, os tokens raros

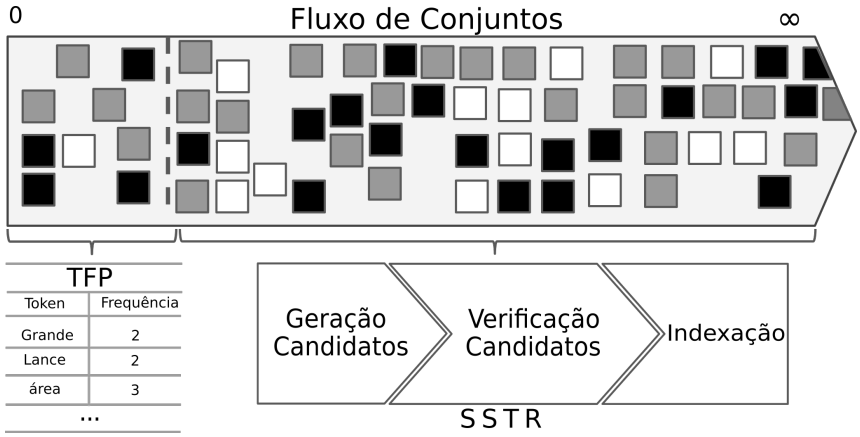


**Figura 4.1:** *Esquema da Tabela de Frequência Parcial.*

são posicionados no prefixo do conjunto. Como resultado desse posicionamento, as listas invertidas são menores e há menos sobreposição de prefixo entre conjuntos diferentes, diminuindo assim o número de candidatos gerados.

Obviamente, a ordenação baseada por frequência não pode ser utilizada em fluxos de dados, pois a frequência dos tokens é continuamente atualizada em relação à chegada de novos tokens. Portanto, se faz necessário escolher uma ordem que não dependa da frequência, como a ordenação lexicográfica. O problema para a ordenação léxica, é que tokens de alta frequência podem se posicionar na dentro do prefixo, consequentemente, incrementando a quantidade de candidatos gerados.

É proposto uma nova técnica para evitar a presença de tokens quem possuem alta frequência dentro do prefixo. Durante a fase de pré-processamento – tokenização, ordenação dos conjuntos e cálculo da frequência – é construído uma tabela e associado cada token com a sua devida frequência, em sequência, a tabela é consultada para se obter a frequência dos tokens durante o processamento da junção por similaridade. Claramente, o objetivo da tabela é capturar uma frequência parcial dos tokens. A TFP (Tabela de Frequência Parcial) é ilustrada na Figura 4.1. Algoritmos de amostragem aleatória podem ser utilizados para coletar os tokens do fluxo na fase de pré-processamento, como por exemplo, o algoritmo *reservoir sampling* [40].

Claramente, a relação token e frequência é parcial, isto é, as frequências coletadas irão mudar posteriormente e parte do universo dos tokens estará ausente na TFP. Uma pergunta que pode emergir, é como se ordenará os tokens sem entrada existente dentro da tabela. Para tal cenário, é estabelecido a frequência mínima entre todos os tokens presente na tabela. A explicação racional para a escolha do valor mínimo se baseia que as distribuições de frequência das palavras tendem a obedecer à lei de potência (*Power law*) e, portanto, são caracterizadas por um grande número de palavras raras [5]. No Capítulo 5 essa decisão será melhor avaliada.

Finalmente, observamos que, alternativamente, outras técnicas de amostragem

de fluxos de dados podem ser utilizadas para identificar tokens de alta frequência. Em particular, há muitos algoritmos disponíveis para encontrar itens frequentes no fluxo [14].

## 4.4 Algoritmo Set Similarity Join over Stream – Late Indexing (SSTR-LI)

Por fim, é apresentado o último algoritmo proposto chamado SSTR-LI (Set Similarity Join over Stream –Late Indexing), para junção por similaridade em fluxos de conjuntos e com a aplicação da indexação tardia nos tokens candidatos. Primeiro, o algoritmo explora as lacunas existentes dentro de um fluxos de dados, ao fato que, o tempo de chegada dos elementos do mundo real não obedece um padrão para o surgimento dos elementos. Devido a essa imperfeição presente no fluxo, pode ser aplicado uma inversão nas etapas do algoritmo, promovendo uma indexação do conjunto de entrada antes da geração dos candidatos, como visto na Figura 4.2.

Ressaltando que, a existência de lacunas dentro do fluxo de dados não se remetem a uma falha gerada pelo emissor, pois em fluxos de dados reais é comum existirem alternâncias entre períodos de maior e menor frequência dos elementos, inclusive a ausência. Uma característica presente nos algoritmos anteriores é que os objetos recebidos após o tempo decorrido da lacuna ainda eram processados de alguma forma. Para a resolução desse problema é adotado a utilização da indexação tardia, ou seja, o conjunto recente consumido pelo algoritmo será processado se somente se a diferença de tempo entre os mesmos for menor que o horizonte temporal, consequentemente, evitando a geração desnecessária de candidatos e descartando imediatamente os conjuntos após o período de lacuna no fluxo.

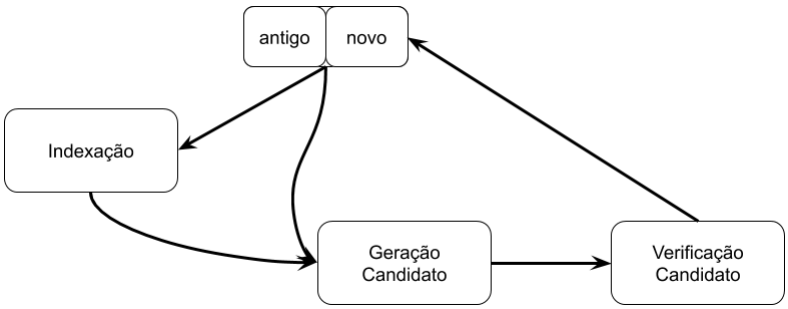


**Figura 4.2:** *Esquema do algoritmo SSTR–LI*

Observando-se a Figura 4.2 é perceptível a existência de duas novas variáveis ambientes dentro do algoritmo. O campo *novo* possui a missão de receber e armazenar o elemento mais recente, ou seja , durante a execução irá armazenar o conjunto mais recente consumido do fluxo. Em sequência, o campo *antigo* é responsável por armazenar o antigo valor do campo *novo*, da forma que, a medida que os conjuntos são consumidos a partir

do fluxo, é realizado uma troca do mais novo para o mais antigo. Essa cópia entre os dois campos é o interruptor do algoritmo SSTR-LI, pois é o mecanismo que aponta a presença de lacunas dentro do fluxo e permite que uma limpeza completa dos conjuntos existentes dentro do índice invertido seja passível de ser realizada.

As instruções para o SSTR-LI são formalizadas no Algoritmo 4. Para a primeira iteração, é necessário substituir o valor inicialmente atribuído para a variável responsável pelo armazenamento do valor temporal antigo do conjunto (Linhas 7 e 8). Para os próximos conjuntos consumidos pelo algoritmo, é realizado uma comparação entre o tempo atual do novo elemento em relação ao antigo, caso a diferença de tempo entre os dois elementos seja maior que o horizonte temporal $\tau$, é realizado uma limpeza dos elementos do índice (Linha 11), caso contrário, o novo elemento será indexado (Linha 13). Para o restante do algoritmo (Linhas 14 até 28) a sua execução se mantém como explicado anteriormente, ou seja, um novo valor de *threshold* é calculado (Linha 18) que é utilizado para o filtro baseado em tamanho e para calcular o limite da sobreposição (Linhas 19 e 23, respectivamente).

O algoritmo mantém todas as melhorias previamente apresentadas para os algoritmos anteriores. A herança dos melhoramentos acontece pois as otimizações anteriormente apresentadas se baseavam nas etapas de geração e verificação de candidatos. Consequentemente, o SSTR-LI remove dinamicamente as entradas antigas pertencentes do índice invertida e aplica o decaimento temporal para formar um novo limite de similaridade para o conjunto sondagem em relação ao conjunto de candidatos. A permanência das melhorias será melhor explorada no Capítulo 5.

**Algoritmo 4:** O algoritmo SSTR–LI.

**Entrada:** Fluxo de Conjuntos $\mathcal{S}$, Threshold $\gamma$, Decaimento $\lambda$
**Saída:** Todos os pares $(x,y) \in \mathcal{S}$ s.t. $J_{\Delta t}(x,y) \geq \gamma$

$\tau = \frac{1}{\lambda} \times ln\frac{1}{\gamma}$
$I_i \leftarrow \varnothing \ (1 \leq i \leq |\mathcal{U}|)$
$x\_time_{old} = 0$
**enquanto** *true* **faça**
  $M \leftarrow$ mapa vazio do ID do conjunto para `int`
  $x \leftarrow$ **read** $(\mathcal{S})$
  **se** $x\_time_{old} == 0$ **então**
    $x\_time_{old} = x$
  **senão**
    **se** $(x\_time_{old} - x.time) > \tau$ **então**
      $I.clear()$
    **senão**
      $\gamma' \leftarrow \frac{\gamma}{e^{-\lambda \times (x\_time_{old} - x_{time})}}$
      **para** $i \leftarrow 1$ **até** $|pref(x,\gamma')|$ **faça**
        $I_{x[i]} \leftarrow I_{x[i]} \cup (x,i)$
  **para** $i \leftarrow 1$ **até** $|pref(x,\gamma)|$ **faça**
    $k \leftarrow x[i]$
    **para cada** $(y,j) \in I_k$ **faça**
      $\gamma' \leftarrow \frac{\gamma}{e^{-\lambda \times \Delta t_{xy}}}$
      **se** $|y| < |x| \times \gamma'$ **então**
        $M[y] \leftarrow -\infty$
        **continue**
      $ubound \leftarrow 1 + min(|x| - i, |y| - j)$
      **se** $M[y].s + ubound \geq O(x,y,\gamma')$ **então**
        $M[y].s \leftarrow M[y].s + 1$
      **senão**
        $M[y] \leftarrow -\infty$
    $R \leftarrow$ **Verify** $(x, M, \gamma, \lambda)$
    **Emit** $(R)$

CAPÍTULO 5

# Experimentos e Resultados

O presente capítulo descreve as configurações utilizadas para os experimentos que foram realizados para avaliar os algoritmos e otimizações descritas no Capítulo 4 e os resultados obtidos, e está organizado da seguinte forma. Na Seção 5.1 será descrito os *datasets* utilizados e porcentagem de sucesso para as múltiplas configurações de execução. Por fim, na Seção 5.2 será apresentado e interpretado os gráficos da execução dos experimentos.

## 5.1 Configuração dos Experimentos

Para a realização dos experimentos, quatro *datasets* foram escolhidos para serem utilizados e podem ser divididos em 2 grupos, artificiais e reais, melhor explicados a seguir. Os *datasets* artificiais são de domínio público, oriundos de diferentes contextos e possuem as seguintes características: DBLP [1] (*Digital Bibliography & Library Project*), consiste de um repositório de informações sobre publicações científicas em Ciência da Computação. Por fim, WIKI[2] (*Wikipedia*) contêm informações generalizadas para os mais diversos assuntos.

Os *datasets* reais consistem de dados coletados em redes sociais, melhor explicados a seguir. Twitter[3], representado por *tweets* coletados de forma geo codificada para a região do Brasil durante os debates políticos de 2018 para a eleição presidencial. Por fim, Reddit, uma rede social para agregação de notícias sociais, classificação de conteúdo da web e discussão [7].

Para os *datasets* artificiais supracitados, são selecionados aleatoriamente registros contendo atributos textuais. Em sequência, uma determinada quantidade de duplicatas, cópias sujas, são geradas contendo modificações aleatórias sobre o texto. Com base

[1] DBLP - https://dblp.uni-trier.de
[2] WIKI - https://www.wikipedia.com
[3] TWITTER - https://developer.twitter.com/en/products/tweets

em [18], os valores textuais são mapeados para conjuntos de 3-*qgrams*. Na Tabela 5.1 é possível observar os valores utilizados para a geração das duplicatas.

**Tabela 5.1:** *Duplicatas geradas por tupla para cada dataset.*

| Dataset | População Inicial | Duplicatas |
|---|---|---|
| DBLP | 70 000 | 5 |
| WIKI | 200 000 | 5 |

Para os dados reais cópias sujas não foram geradas, pois os *datasets* possuíam uma população de tamanho satisfatório. No entanto, um pré-processamento foi realizado para que se tornasse possível a aplicação dos algoritmos, o pré-processamento se baseou na remoção de *emojis* (ideogramas utilizados em mensagens eletrônicas), links e referências para imagens.

Para DBLP e WIKI, respectivamente, atribuímos timestamp de data/hora artificiais a cada sequência de dado nesses *datasets*, que foram amostradas das funções de distribuição Poisson (DBLP) e Uniforme (WIKI). Por esse motivo, já definido anteriormente, chamamos os conjuntos de dados DBLP e WIKI de artificiais. Em contrapartida, para TWITTER e REDDIT, o tempo de publicação disponível para cada dado foi utilizado como *timestamp*. Por esse motivo, são chamados conjuntos de dados reais. Todos os datasets são heterogêneos, exibem diferentes características e conforme resumido na Tabela 5.2 .

**Tabela 5.2:** *Estatísticas dos Datasets.*

| Nome | População | Tamanho Médio Conj. | Distribuição Temporal |
|---|---|---|---|
| DBLP | 350 000 | 76 | Poisson |
| WIKI | 1 000 000 | 53 | Uniforme |
| TWITTER | 2 824 998 | 90 | Data de Publicação |
| REDDIT | 19 456 493 | 53 | Data de Publicação |

Os experimentos foram conduzidos em uma máquina com um processador Intel Xeon E5-2620 com 2.10 GHz, 16 MB de cache de CPU, 16GB de memória principal e com sistema operacional *Ubuntu 16.04 LTS*. Todos os experimentos são executados de forma sequencial sobre um único *core* da CPU. Os tempos obtidos são a média de 5 execuções, variações significativas na quantidade de execuções foram testadas, mas não foi observado diferença significativa no tempo de processamento. Para o valor de similaridade $\gamma$, é utilizado valores dentro do intervalo $[0.5, 0.95]$. Por fim, para o valor de lambda $\lambda$ é utilizado valores exponencialmente crescentes no intervalo $[10^{-3}, 10^{-1}]$.

Para que houvesse consistência na execução dos algoritmos, é definido um limite máximo de 3 horas para a execução de cada experimento. Os experimentos foram abortados ao atingir o tempo limite ou por falta de memória na JVM. A fracão de sucesso

**Tabela 5.3:** *Fração das execuções que obtiveram sucesso.*

| Dataset | SSPJ | | SSTR | | SSTR-LI | |
|---|---|---|---|---|---|---|
| | Lex | TFP | Lex | TFP | Lex | TFP |
| DBLP | 0,33 | 1,0 | 0,44 | 1.0 | 0,44 | 1.0 |
| WIKI | 0,33 | 1,0 | 1,0 | 1,0 | 1,0 | 1,0 |
| TWITTER | 0 | 0 | 1,0 | 1,0 | 1,0 | 1,0 |
| REDDIT | 0 | 0 | 1,0 | 1,0 | 1,0 | 1,0 |

na execução para todos os algoritmos e pelas ordenações de tokens utilizadas, Lex(léxica) e TFP(Tabela de Frequência Parcial), podem ser vistas na Tabela 5.3.

## 5.2 Resultados

A presente seção tem como objetivo apresentar, comparar e analisar os diversos resultados obtidos com os experimentos realizados. Da forma que, serão realizadas comparações individuais entre os múltiplos algoritmos propostos e será seguido a ordem cronológica de criação e evolução dos mesmos.

### 5.2.1 SSPJ vs SSTR|Lex

Primeiro iremos analisar os resultados obtidos para os *datasets* artificiais. A Figura 5.1 demonstra os tempos de execução obtidos para os algoritmos SSPJ e SSTR|Lex (ordenação léxica) nos *datasets* DBLP e WIKI. Como esperado, SSPJ conseguiu concluir sua execução somente para valores muito altos de *threshold*. Em contraste, SSTR|Lex terminou todas as execuções de experimentos para o WIKI. Altos valores de *threshold* aumentam a eficácia do filtro por prefixo, e beneficia os dois algoritmos. No entanto, na maioria dos casos em que SSPJ conseguiu terminar a execução, STTR|Lex se mostrou mais rápido em até três ordens de magnitude. Esses resultados destacam a eficácia das nossas técnicas aplicadas na redução drástica do número de comparações para o cálculo similaridade e para a utilização de memória.

No conjunto de dados DBLP, o SSTR terminou dentro do prazo de tempo estipulado para todos os valores de *threshold* para $\lambda = 0,1$. A razão para os resultados apresentados é que a distribuição de Poisson gera algumas janelas temporais muito densas, com objetos definidos temporalmente muito próximos um do outro. Para valores pequenos de decaimento temporal $\lambda$, as janelas temporais são grandes e mais conjuntos precisam ser mantidos nas listas invertidas e comparados na fase de verificação. Por outro lado, valores maiores de decaimento temporal se traduzem em um horizonte temporal menor e, portanto, em janelas temporais mais estreitas. Como resultado, o filtro de tempo é mais eficaz para remover entradas obsoletas das listas invertidas. Além disso, os valores de

decaimento do tempo levam a maiores thresholds específicos do candidato, que, por sua vez, melhoram o poder de poda dos filtros posicionais e por tamanho.

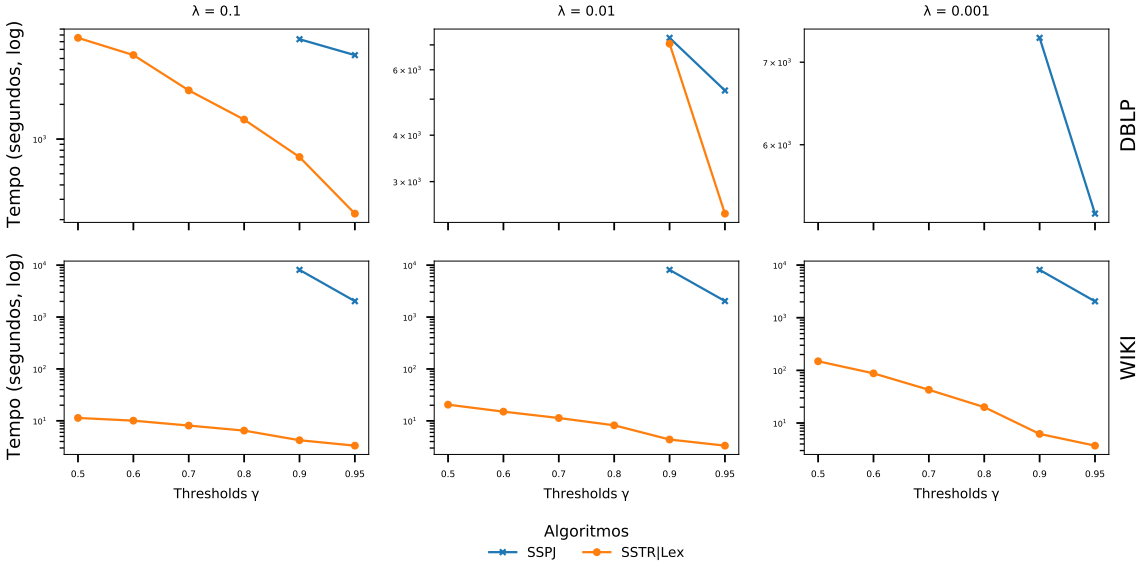


**Figura 5.1:** *Tempo de execução para os datasets artificiais.*

Por fim, analisamos os resultados obtidos para os *datasets* reais. A Figura 5.2 plota os tempos de execução do SSTR|Lex no TWITTER e REDDIT. Os resultados do SSPJ não são exibidos, pois como demonstrado na Tabela 5.3 não se obteve sucesso nas execuções, pois falhas aconteceram devido à falta de memória nesses *datasets* em todas as configurações. Obviamente, como o SSPJ não remove as entradas obsoletas do índice, ele não pode lidar diretamente com os maiores conjuntos de dados em nossa configuração experimental. Observe que, sempre podemos reconstruir o índice invertido, por exemplo, depois de atingir algum limite de espaço. No entanto, essa estratégia sacrifica tempo de resposta, precisão ou ambas. Embora o recurso a esse modo de processamento em lote seja inevitável em cenários estressantes, os resultados mostram que o algoritmo SSTR|Lex pode, no entanto, sustentar o processamento contínuo de fluxos por muito mais tempo que o SSPJ.

É importante observar que o SSTR|Lex termina com êxito em todas as configurações em conjuntos de dados do mundo real. Além disso, mesmo que esses conjuntos de dados sejam maiores que DBLP e WIKI, o SSTR|Lex é até duas ordens de magnitude mais rápido. A explicação está na distribuição de data e hora dos conjuntos de dados do mundo real, que exibem mais "lacunas"em comparação com as sintéticas. Portanto, as janelas temporais induzidas são mais "esparsas"nesses conjuntos de dados que é efetivamente explorado pelo filtro de tempo para manter dinamicamente o comprimento das listas invertidas e mantidas a um valor mínimo. As outras tendências permanecem as mesmas: os tempos de execução aumentam e diminuem à medida que os limites de similaridade e os parâmetros de redução de tempo diminuem e aumentam, respectivamente.

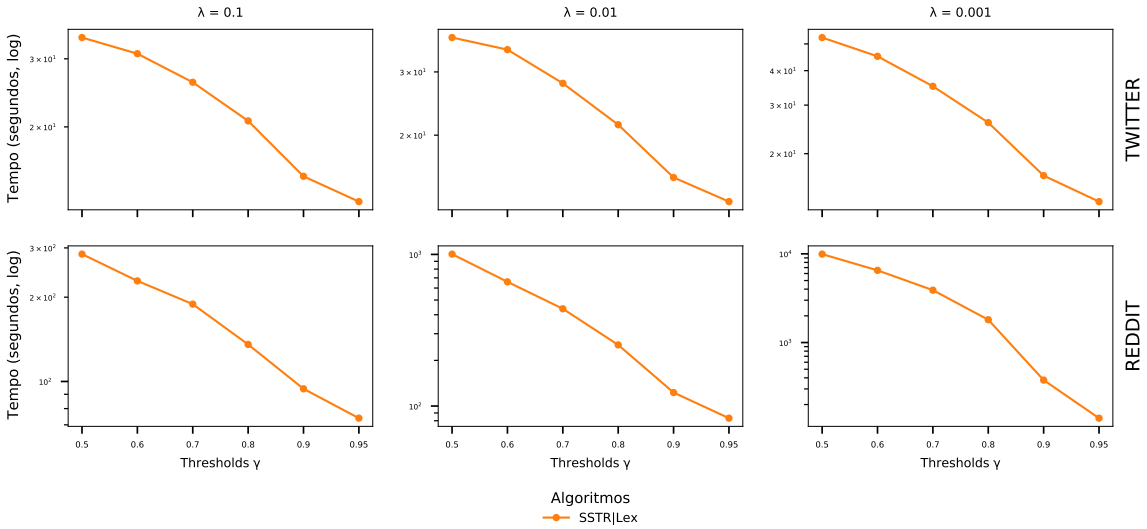


**Figura 5.2:** *Tempo de execução para os datasets reais.*

### 5.2.2 Tabela de Frequência Parcial

A Tabela de Frequência Parcial (TFP), explicada anteriormente no Capítulo 4, necessita que o algoritmo de ordenação aplique um atribuição para valores de frequência aos tokens não existentes durante o processo de amostragem utilizado para o cálculo da frequência parcial. Os valores atribuídos para os tokens desconhecidos foram inicialmente designados com base no cálculo da média, mediana, valor máximo e mínimo do tokens presentes na amostragem.

Experimentos empíricos foram realizados para definir qual o melhor valor possível dentre as quatro funções anteriormente supracitadas para atribuição do novo valor de frequência para o token, melhor explicadas a seguir. Média, a soma da frequência de todos os tokens divido pelo total de tokens, assume que o novo token possui característica de não raridade, ou seja, um token comum. A mediana, assume papel similar a média e considera o novo token como algo próximo ao comum. A máxima, destaca o token como um elemento comum. Por fim, a atribuição da frequência mínima ao token, define que ele possui um grau de raridade em relação aos outros.

Surpreendentemente, por mais que as quatro funções exibissem características diferentes dentro do espectro de comum e raro, os resultados para uma comparação por tempo de execução permanacem bastante similares, como pode ser visto na figura 5.3. Para os menores valores de *threshold*, o cálculo da máxima demonstrou ser relativamente pior que as outras funções. Devido ao resultado obtido, o valor de frequência mínima foi adotado para todos os novos tokens encontrados durante o processamento dos algoritmos.

Para que fosse possível a replicabilidade e validação dos experimentos realizados sobre a TFP, porcentagens de utilização do *dataset* foram definidas como amostras na construção da frequência parcial. Definido o conjunto de porcentagens $\{1\%, 5\%, 10\%\}$,

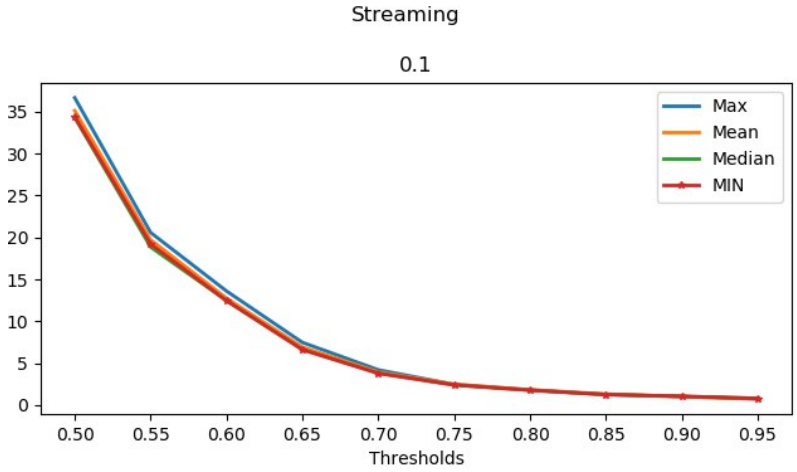


**Figura 5.3:** *Projeção das funções matemáticas utilizadas na TFP.*

os resultados podem ser acompanhados na Tabela 5.4. É mostrado um *dataset* para cada conjunto de *datasets* reais ou artificiais, pois os resultados se mostraram similares para todos os *datasets*.

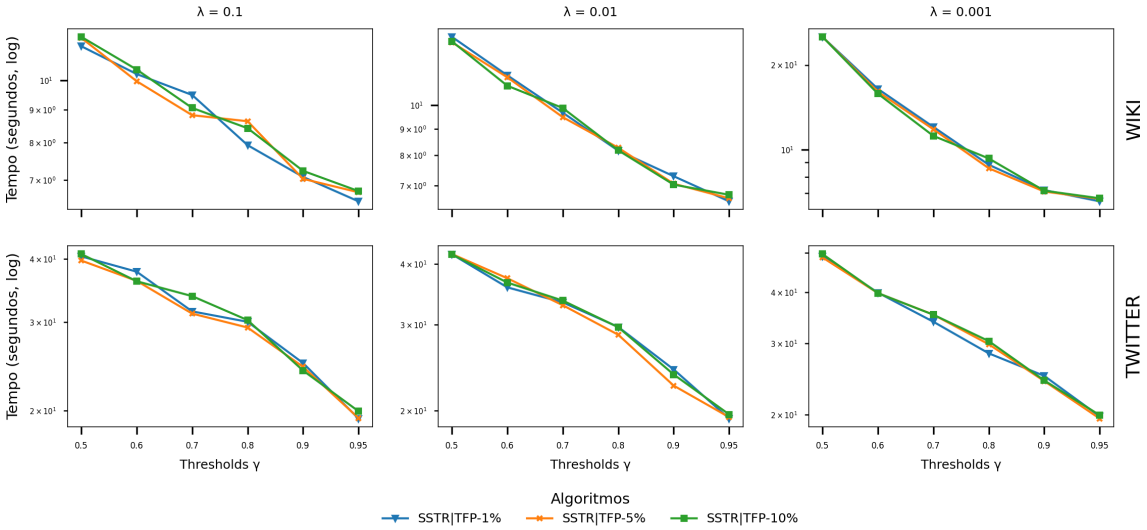


**Figura 5.4:** *Comparação entre as diferentes porcentagens para a TFP.*

Os resultados se demonstraram interessantes, pois não era esperado que todas as porcentagens utilizadas mostrassem resultados semelhantes. Aguardava-se que porcentagens maiores da amostragem do *dataset* promovesse um maior ganho de tempo na execução, ao fato que, a tabela de frequência estaria representando maior quantidade de tokens do universo e se aproximaria da frequência absoluta. Os resultados se provaram contrários à essa expectativa, pois 1% de amostragem demonstrou o mesmo resultado com a utilização de 10% e comprovou que uma baixa porcentagem de uso do *dataset* já é o suficiente para prover uma amostra que seja capaz de gerar resultados no tempo de processamento dos algoritmos.

### 5.2.3 SSTR|Lex vs SSTR|TFP

Analisando os resultados obtidos para os *datasets* artificiais, a Figura 5.5 demonstra os tempos de execução obtidos para o algoritmo SSTR para as ordenações léxica e por frequência se utilizando da TFP, devidamente legendadas como SSTR|Lex e SSTR|TFP, para os *datasets* DBLP e WIKI. Como demonstrado anteriormente, o SSTR|Lex não se mostrou apto a terminar todos os casos de testes propostos para o *dataset* do DBLP e demonstrou uma baixa porcentagem de completude. No entanto, o SSTR|TFP conseguiu concluir a execução para todos os casos de testes e ainda demonstrou um desempenho superior em relação a ordenação léxica. Contudo, o SSTR|TFP não se mostrou superior para todos os casos do *dataset* WIKI e para pesquenos valores de $\lambda$ apresentou resultados abaixo do esperado.

Para o conjunto de dados WIKI é possível ver um padrão de desempenho para o SSTR|TFP em relação a ordenação léxica. O padrão se demonstra uma curva que desliza sobre a constante curva apresentada pela ordenação léxica. Comparando se os valores para $\lambda = 0.1$ e $\lambda = 0.001$ é possível notar que a curva de desempenho do SSTR|TFP decai a medida que o valor de $\lambda$ fica menor, demonstrando uma melhor eficácia para horizontes temporais com um maior intervalo de tempo, ou seja, uma quantidade maior de conjuntos presentes dentro da janela temporal.

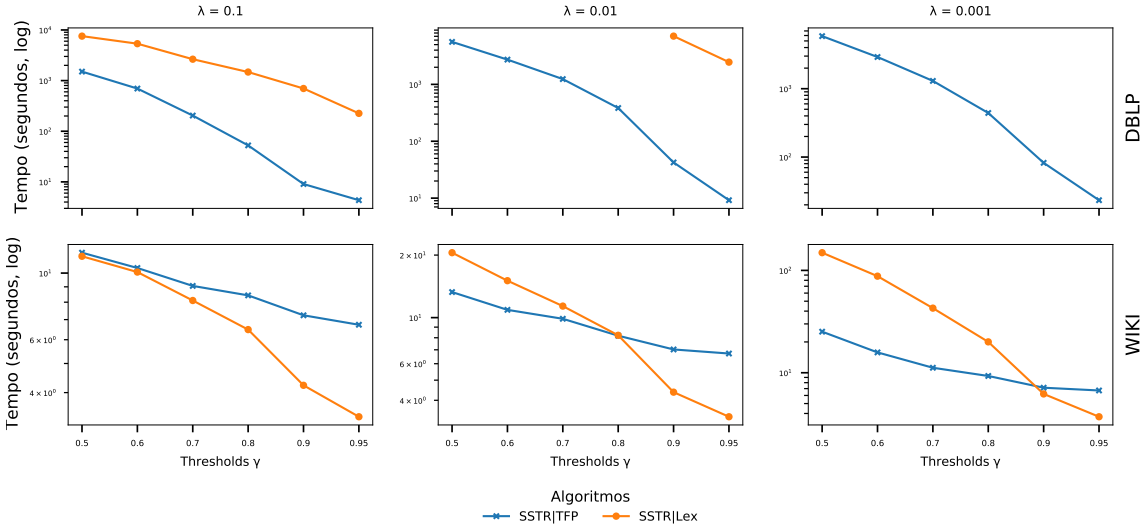


**Figura 5.5:** *Tempo de execução para datasets artificiais.*

Analisando os resultados para os *datasets* reais. Na Figura 5.6 é possível a plotagem para os *datasets* do TWITTER e REDDIT. Os resultados do SSTR|TFP para o *dataset* do TWITTER não foi como o esperado, pois para a maioria dos casos a ordenação léxica mostrou possuir um melhor desempenho. Não foram encontrados motivos para justificar a baixa performance do SSTR|TFP, diversas análises foram realizadas para se encontrar uma explicação do resultado, como a exploração da frequência dos tokens

e tamanho médio do conjunto, mas nenhuma resposta satisfatória foi encontrada. No entanto, é importante salientar que os resultados demonstrados acontecem unicamente e exclusivamente para o *dataset* do TWITTER e não se repete em outros casos, e para o SSTR-LI temos um cenário totalmente diferente para o TFP, em relação ao apresentado na Figura 5.6.

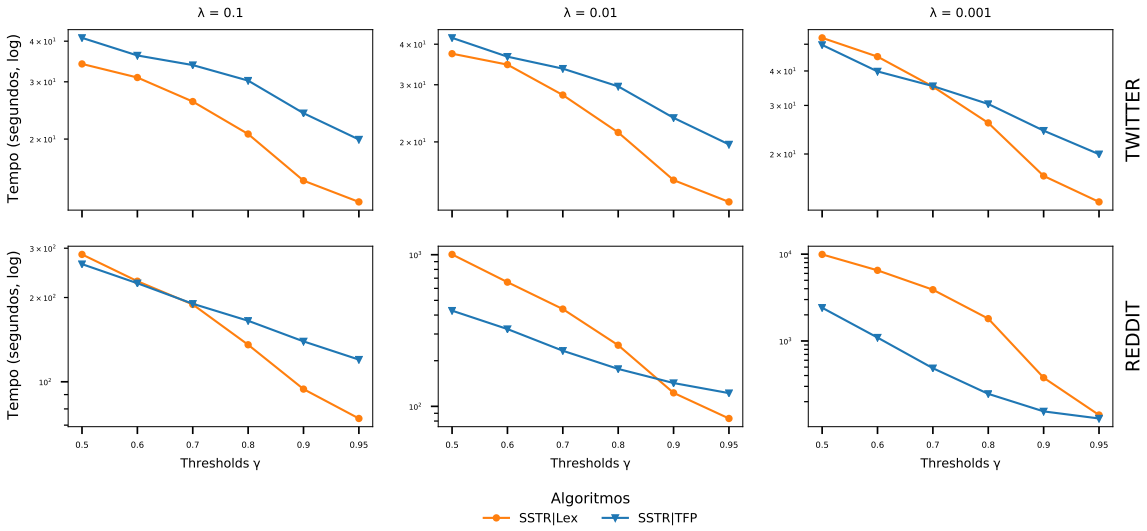


**Figura 5.6:** *Tempo de execução para datasets reais.*

Por fim, para o *dataset* REDDIT podemos perceber novamente a presença do padrão apresentado nos resultados para o WIKI, ou seja, uma curva deslizante sobre a medida que os valores de $\lambda$ ficam menores. Diferentemente do WIKI, para valores de $\lambda = 0.001$ o SSTR|TFP apresenta desempenho superior ou igual para todos os casos de testes. A explicação para tal desempenho pode se basear na densidade dos elementos na janela temporal, pois como visto anteriormente, a ordenação por TFP demonstra melhores desempenhos para menores valores de lambda e pode ser entendido como explicação plausível para os resultados obtidos.

### 5.2.4 SSTR|TFP vs SSTR-LI

O algoritmo SSTR-LI terá seu desempenho analisado em relação ao *dataset* do TWITTER, pois para todos os outros casos de testes realizados, ordenações possíveis e *datasets*, o tempo de execução obtido é equivalente aos resultados anteriormente mostrados, ou seja, manteve o desempenho apresentado nas configurações anteriores dos algoritmos. Por esse motivo, somente o *dataset* do TWITTER é mostrado na Figura 5.7.

Analisando os resultados entre SSTR e STTR-LI é possível de perceber que para todos os casos testados, o algoritmo SSTR-LI demonstra um desempenho superior em relação ao seu concorrente. A explicação para o desempenho se explica na distribuição dos objetos dentro do fluxo de dados, pois o *dataset* do TWITTER foi coletado a partir de

um fluxo real de objetos e demonstra um maior espaçamento temporal entre os objetos e permite que a indexação tardia realiza o trabalho de filtragem entre as janelas temporais e iniba a comparação desnecessária entre conjuntos.

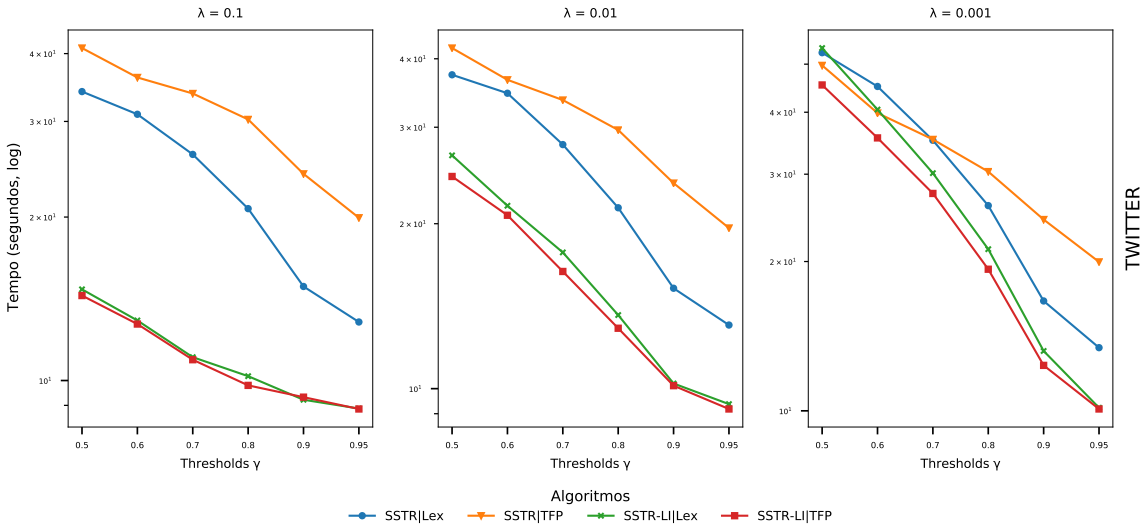


**Figura 5.7:** *Tempo de execução para o dataset Twitter.*

Comparando as duas formas de ordenação para o algoritmo SSTR-LI é perceptível que a ordenação baseada na TFP demonstrou um melhor desempenho em relação a ordenação léxica. Tal resultado, entra em contrapartida com resultados anteriores, pois para o *dataset* em questão, a ordenação léxica sempre havia se mostrado superior. Novamente, após extensivas análises, não foram encontradas justificativas que respondessem satisfatoriamente a inversão de desempenho apresentada dentro do algoritmo. Observando o cenário como um todo, é notável que o desempenho do SSTR-LI é superior para todos os casos presentes na Figura 5.7.

CAPÍTULO 6

# Conclusões

## 6.1 Conclusões

No presente trabalho, propomos três algoritmos para junção por similaridade em fluxos de dados, ao qual, os elementos são representados como conjuntos e processados pelos algoritmos SSPJ,SSTR e SSTR-LI. Adotamos também o conceito da similaridade temporal e exploramos suas propriedades para reduzir os custos de processamento e o uso de memória. Para lidar com a ordenação dos tokens no dentro do contexto de fluxo de dados, ao qual, os conjuntos possuem período de vida limitado dentro do processamento dos algoritmos, é proposto a amostragem TFT para contribuir com a ordenação dos tokens.

Em relação aos algoritmos, como demonstrado no Capítulo 5, concluímos que o SSTR-LI demonstrou melhores resultados em relação com as outras propostas. Essa conclusão se baseia no tempo de processamento obtido para o *dataset* do Twitter, demonstrado na Figura 5.7, e demonstra capacidade de execução para casos reais de processamento. Ao fato que, fluxos de dados reais não possuem um comportamento contínuo e lacunas são esperadas dentro do fluxo. O SSTR-LI demonstrou ser capaz de processar dados reais com mais eficiência e nos piores casos manteve o mesmo desempenho das outras propostas.

A tabela de frequência parcial (TFP) demonstrou ser bastante complexa em suas utilizações e não promoveu o desempenho esperado para todos os casos. O contexto de processamento para fluxos de conjuntos é recente na atual literatura e não se é possível encontrar trabalhos relacionados que justifiquem o seu mal comportamento para alguns casos. Acreditamos que o seu baixo desempenho se dê em característica da baixa representatividade dos tokens iniciais durante o processo de amostragem, mas o tema não foi abordado e será melhor explorado na Seção 6.2.

Por fim, ressalto que extensivos experimentos foram realizados em todos os cenários passíveis de experimentação e com a utilização de *datasets* artificiais e reais para comprovar a eficácia de todos os algoritmos propostos e parte do atual trabalho pode ser também analisada em [27].

## 6.2 Trabalhos Futuros

Para os trabalhos futuros visamos à criação de novas formas para se ordenar os objetos dentro do conjunto. Primeiramente, novas fórmulas estatísticas podem estudadas para serem utilizadas na obtenção da frequência parcial dos elementos. Pois, atualmente a TFP apresenta uma troca justa por informação em relação ao ganho de tempo, mas essa troca não se faz verdadeira para todos os casos de testes e precisa ser analisada mais severamente em oportunidades futuras.

Uma próxima abordagem a ser considerada e integrada ao trabalho é a aplicação de pesos nos tokens, ou seja, a utilização do IDF (*Inverse Document Frequency*) durante o pré-processamento realizado aos conjuntos e acrescentando um maior grau de dificuldade ao trabalho, pois novas variáveis serão adicionadas para o processamento da similaridade.

Consideramos no presente trabalho apenas a aplicação de algoritmos de similaridade para fluxos de dados representados em conjunto, ao qual, todo processamento realizado é feito de forma sequencial, ou seja, sobre um único core do CPU. Acrescentamos também uma amostragem baseada no fluxo para que fosse possível obter uma frequência parcial dos tokens, mas devido ao tempo de duração do projeto, não foram realizados intensivos estudos sobre a tabela de frequência parcial (TFP). Ambos problemas apresentados são desafiadores e importantes para serem estudados futuramente.

# Referências Bibliográficas


[1] ABADI, D. J.; AHMAD, Y.; BALAZINSKA, M.; ÇETINTEMEL, U.; CHERNIACK, M.; HWANG, J.; LINDNER, W.; MASKEY, A.; RASIN, A.; RYVKINA, E.; TATBUL, N.; XING, Y.; ZDONIK, S. B. **The Design of the Borealis Stream Processing Engine**. In: *Proceedings of the Conference on Innovative Data Systems Research*, p. 277–289, 2005.

[2] AMAGATA, D.; HARA, T.; XIAO, C. **Dynamic Set kNN Self-Join**. In: *Proceedings of the ICDE Conference*, p. 818–829, 2019.

[3] ANASTASIU, D. C.; KARYPIS, G. **L2AP: fast cosine similarity search with prefix L-2 norm bounds**. In: *Proceedings of the ICDE Conference*, p. 784–795, 2014.

[4] ARASU, A.; GANTI, V.; KAUSHIK, R. **Efficient exact set-similarity joins**. In: *Proceedings of the 32nd international conference on Very large data bases*, p. 918–929. VLDB Endowment, 2006.

[5] BAAYEN, R. H. **Word Frequency Distributions**, volume 18 de **Text, Speech and Language Technology**. Kluwer Academic Publishers, 2001.

[6] BABCOCK, B.; BABU, S.; DATAR, M.; MOTWANI, R.; WIDOM, J. **Models and Issues in Data Stream Systems**. In: *Proceedings of the ACM Symposium on Principles of Database Systems*, p. 1–16, 2002.

[7] BAUMGARTNER, J. **Reddit May 2019 Submissions**. Harvard Dataverse, 2019.

[8] BAYARDO, R. J.; MA, Y.; SRIKANT, R. **Scaling up All Pairs Similarity Search**. In: *Proceedings of the WWW Conference*, p. 131–140. ACM, 2007.

[9] BRODER, A. Z.; CHARIKAR, M.; FRIEZE, A. M.; MITZENMACHER, M. **Min-Wise Independent Permutations (Extended Abstract)**. In: *Proceedings of the STOC Symposium*, p. 327–336. ACM, 1998.

[10] CARBONE, P.; KATSIFODIMOS, A.; EWEN, S.; MARKL, V.; HARIDI, S.; TZOUMAS, K. **Apache Flink™: Stream and Batch Processing in a Single Engine**. *IEEE Data Engineering Bulletin*, 38(4):28–38, 2015.

[11] CHANDEL, A.; HASSANZADEH, O.; KOUDAS, N.; SADOGHI, M.; SRIVASTAVA, D. **Benchmarking declarative approximate selection predicates**. In: *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, p. 353–364, 2007.

[12] CHAUDHURI, S.; GANTI, V.; KAUSHIK, R. **A Primitive Operator for Similarity Joins in Data Cleaning**. In: *Proceedings of the ICDE Conference*, p. 5. IEEE Computer Society, 2006.

[13] CHRISTIANI, T.; PAGH, R.; SIVERTSEN, J. **Scalable and Robust Set Similarity Join**. In: *Proceedings of the ICDE Conference*, p. 1240–1243. IEEE Computer Society, 2018.

[14] CORMODE, G.; HADJIELEFTHERIOU, M. **Finding Frequent Items in Data Streams**. *Proceedings of the VLDB Endowment*, 1(2):1530–1541, 2008.

[15] DENG, F.; RAFIEI, D. **Approximately Detecting Duplicates for Streaming Data using Stable Bloom Filters**. In: *Proceedings of the SIGMOD Conference*, p. 25–36, 2006.

[16] DO CARMO OLIVEIRA, D. J.; BORGES, F. F.; RIBEIRO, L. A.; CUZZOCREA, A. **Set similarity joins with complex expressions on distributed platforms**. In: *Proceedings of the Symposium on Advances in Databases and Information Systems*, p. 216–230, 2018.

[17] DUTTA, S.; NARANG, A.; BERA, S. K. **Streaming quotient filter: A near optimal approximate duplicate detection approach for data streams**. *Proceedings of the VLDB Endowment*, 6(8):589–600, 2013.

[18] GRAVANO, L.; IPEIROTIS, P. G.; JAGADISH, H. V.; KOUDAS, N.; MUTHUKRISHNAN, S.; SRIVASTAVA, D.; OTHERS. **Approximate string joins in a database (almost) for free**. In: *VLDB*, volume 1, p. 491–500, 2001.

[19] KRAUS, N.; CARMEL, D.; KEIDAR, I. **Fishing in the Stream: Similarity Search over Endless Data**. In: *bigdata*, p. 964–969, 2017.

[20] LEVENSHTEIN, V. I. **Binary codes capable of correcting deletions, insertions and reversals**. *Soviet Physics Doklady*, 10:707–710, 1966.

[21] LIAN, X.; CHEN, L. **Efficient Similarity Join over Multiple Stream Time Series**. *IEEE Transactions on Knowledge and Data Engineering*, 21(11):1544–1558, 2009.

[22] LIAN, X.; CHEN, L. **Set Similarity Join on Probabilistic Data**. *Proceedings of the VLDB Endowment*, 3(1):650–659, 2010.

[23] LIAN, X.; CHEN, L. **Similarity Join Processing on Uncertain Data Streams**. *IEEE Transactions on Knowledge and Data Engineering*, 23(11):1718–1734, 2011.

[24] MANN, W.; AUGSTEN, N.; BOUROS, P. **An Empirical Evaluation of Set Similarity Join Techniques**. *PVLDB*, 9(9):636–647, 2016.

[25] METWALLY, A.; AGRAWAL, D.; EL ABBADI, A. **Duplicate Detection in Click Streams**. In: *Proceedings of the WWW Conference*, p. 12–21, 2005.

[26] MORALES, G. D. F.; GIONIS, A. **Streaming Similarity Self-Join**. *Proceedings of the VLDB Endowment*, 9(10):792–803, 2016.

[27] PACÍFICO, L.; RIBEIRO, L. A. **Sstr: Set similarity join over stream data.** In: *ICEIS (1)*, p. 52–60, 2020.

[28] QUIRINO, R. D.; RIBEIRO-JÚNIOR, S.; RIBEIRO, L. A.; MARTINS, W. S. **fgssjoin: A GPU-based Algorithm for Set Similarity Joins**. In: *International Conference on Enterprise Information Systems*, p. 152–161. SCITEPRESS, 2017.

[29] RIBEIRO, L. A.; CUZZOCREA, A.; BEZERRA, K. A. A.; DO NASCIMENTO, B. H. B. **Sjclust: Towards a framework for integrating similarity join algorithms and clustering**. In: *International Conference on Enterprise Information Systems*, p. 75–80. SCITEPRESS, 2016.

[30] RIBEIRO, L. A.; CUZZOCREA, A.; BEZERRA, K. A. A.; DO NASCIMENTO, B. H. B. **SjClust: A Framework for Incorporating Clustering into Set Similarity Join Algorithms**. *LNCS Transactions on Large-Scale Data- and Knowledge-Centered Systems*, 38:89–118, 2018.

[31] RIBEIRO, L. A.; HÄRDER, T. **Generalizing Prefix Filtering to Improve Set Similarity Joins**. *Information Systems*, 36(1):62–78, 2011.

[32] RIBEIRO, L. A.; SCHNEIDER, N. C.; DE SOUZA INÁCIO, A.; WAGNER, H. M.; VON WANGENHEIM, A. **Bridging Database Applications and Declarative Similarity Matching**. *Journal of Information and Data Management*, 7(3):217–232, 2016.

[33] RIBEIRO-JÚNIOR, S.; QUIRINO, R. D.; RIBEIRO, L. A.; MARTINS, W. S. **Fast Parallel Set Similarity Joins on Many-core Architectures**. *Journal of Information and Data Management*, 8(3):255–270, 2017.

[34] SARAWAGI, S.; KIRPAL, A. **Efficient Set Joins on Similarity Predicates**. In: *Proceedings of the SIGMOD Conference*, p. 743–754. ACM, 2004.

[35] SHEN, Z.; CHEEMA, M. A.; LIN, X.; ZHANG, W.; WANG, H. **A Generic Framework for Top-k Pairs and Top-k Objects Queries over Sliding Windows**. *IEEE Transactions on Knowledge and Data Engineering*, 26(6):1349–1366, 2014.

[36] SIDNEY, C. F.; MENDES, D. S.; RIBEIRO, L. A.; HÄRDER, T. **Performance Prediction for Set Similarity Joins**. In: *Proceedings of the SAC Conference*, p. 967–972, 2015.

[37] STONEBRAKER, M.; ÇETINTEMEL, U.; ZDONIK, S. B. **The 8 Requirements of Real-time Stream Processing**. *SIGMOD Record*, 34(4):42–47, 2005.

[38] UKKONEN, E. **Approximate string-matching with q-grams and maximal matches**. *Theoretical Computer Science*, 92(1):191 – 211, 1992.

[39] VERNICA, R.; CAREY, M. J.; LI, C. **Efficient Parallel Set-similarity Joins using MapReduce**. In: *Proceedings of the SIGMOD Conference*, p. 495–506. ACM, 2010.

[40] VITTER, J. S. **Random Sampling with a Reservoir**. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.

[41] WANG, X.; QIN, L.; LIN, X.; ZHANG, Y.; CHANG, L. **Leveraging Set Relations in Exact Set Similarity Join**. *Proceedings of the VLDB Endowment*, 10(9):925–936, 2017.

[42] XIAO, C.; WANG, W.; LIN, X.; YU, J. X.; WANG, G. **Efficient Similarity Joins for Near-duplicate Detection**. *ACM Transactions on Database Systems*, 36(3):15:1–15:41, 2011.

[43] ZOBEL, J.; MOFFAT, A. **Exploring the similarity space**. In: *Acm Sigir Forum*, volume 32, p. 18–34. ACM New York, NY, USA, 1998.