PARALELISMO E DISTRIBUIÇÃO NA RECUPERAÇÃO HEURÍSTICA DO PLÁGIO EXTERNO COM LOCALITY SENSITIVE HASH

Joaquim Afonso Ferreira Viana

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Geraldo Bonorino Xexéo

Rio de Janeiro
Setembro de 2019

PARALELISMO E DISTRIBUIÇÃO NA RECUPERAÇÃO HEURÍSTICA DO PLÁGIO EXTERNO COM LOCALITY SENSITIVE HASH

Joaquim Afonso Ferreira Viana

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Geraldo Bonorino Xexéo, D.Sc.

Prof. Alexandre de Assis Bento Lima, D.Sc.

Prof. Daniel Cardoso Moraes de Oliveira, D.Sc

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2019

Viana, Joaquim Afonso Ferreira

Paralelismo e Distribuição na Recuperação Heurística do Plágio Externo com Locality Sensitive Hash/Joaquim Afonso Ferreira Viana. – Rio de Janeiro: UFRJ/COPPE, 2019.

XII, 66 p.: il.; 29, 7cm.

Orientador: Geraldo Bonorino Xexéo

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2019.

Referências Bibliográficas: p. 59 – 66.

1. Locality Sensitive Hash (LSH). 2. Recuperação Heurística. 3. Identificação de Plágio. 4. Paralelismo e Distribuição. 5. Spark. I. Xexéo, Geraldo Bonorino. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Agradecimentos

Antes de tudo, agradeço a Deus por me dar sabedoria, força, determinação, e por me permitir concluir essa etapa da minha vida.

Agradeço aos meus pais Joaquim Afonso e Lucia Antônia pelos ensinamentos, apoio e suporte, em todas as etapas da minha vida, sobretudo neste mestrado, onde demonstraram abundante compreensão e me proporcionaram uma incrível motivação para concluir esta etapa em minha vida.

Agradeço também a minha esposa Karinne Santos pelo suporte, pela sua compreensão, comprometimento, dedicação e me incentivar a concluir esse trabalho.

Junto ao meu orientador Geraldo Bonorino Xexéo, agradeço também ao Fellipe Duarte, pela grande colaboração com este trabalho, pelos valiosos ensinamentos, dedicação, paciência e boa vontade durante todo o tempo despendido na construção desse trabalho.

Agradeço também aos professores Jano Moreira de Souza e Alexandre de Assis Bento Lima pela boa vontade e colaboração com a minha pesquisa.

Ao Centro de Análise de Sistemas Navais (CASNAV), em especial ao Edgard Oliveira, Kelli, Mauricio Malburg , Laje e Valéria, pelo apoio e por proporcionarem no CASNAV um ambiente incentivador à pesquisa cientifica e a inovação.

Aos amigos Egberto Caetano, Michel Arruda, Hugo Diniz e Ricardo Luiz, companheiros de mestrado, os quais pude compartilhar conhecimento e ótimos momentos juntos.

Aos amigos Matheus Emerick, Vinícius Mororó, Rafael Risala, Gabriel Rosário, Bruno Moore, Thiago Flosino, Rafael Chaves, José Vitor, Waldeir Duarte e Matilde Santiago, Arthur, Magno, Diego, Juliana, Izadora, Clayton, Claudio, Daniel, Edson, Sebastião Alves, Priscila, Maurício e tantos outros que contribuíram na minha vida e com o meu trabalho.

"O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001."

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

PARALELISMO E DISTRIBUIÇÃO NA RECUPERAÇÃO HEURÍSTICA DO PLÁGIO EXTERNO COM LOCALITY SENSITIVE HASH

Joaquim Afonso Ferreira Viana

Setembro/2019

Orientador: Geraldo Bonorino Xexéo

Programa: Engenharia de Sistemas e Computação

A recuperação heurística na tarefa de identificação de plágio externo tem o objetivo de retornar uma lista com os documentos mais prováveis de terem sido plagiados, baseado em uma métrica de similaridade, reduzindo a carga de trabalho das etapas seguintes, caracterizadas por serem altamente custosas. O trabalho de DUARTE (2017) formalizou uma sequência de passos para realizar recuperação heurística com métodos de *Locality Sensitive Hash* (LSH) e demonstrou que, devido a capacidade de preservação da similaridade, os métodos LSH são opções viáveis para a recuperação heurística. Este trabalho propôs duas estratégias, denominadas de *paralelismo nos documentos* (PnD) e *paralelismo na permutação* (PnP), baseadas na sequência de passos de DUARTE (2017), que foram implementadas no sistema de computação distribuída Apache Spark, para apoiar a tarefa de identificação de plágio em grandes coleções de documentos. Os experimentos demonstraram que as estratégias PnD e PnP foram capazes de reduzir, em função do aumento da capacidade computacional, o tempo das atividades de representar, buscar e recuperar documentos; bem como permitem atingir um alto nível de eficácia para retornar os documentos efetivamente plagiados.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

PARALLELISM AND DISTRIBUTION IN EXTERNAL PLAGIARISM'S HEURISTIC RETRIEVAL WITH LOCALITY SENSITIVE HASH

Joaquim Afonso Ferreira Viana

September/2019

Advisor: Geraldo Bonorino Xexéo

Department: Systems Engineering and Computer Science

The heuristic retrieval on external plagiarism identification task is intended to return a list of the documents most likely to have been plagiarized, based on a similarity metric, reducing the workload of the following and highly costly steps. DUARTE (2017)'s work formalized a sequence of steps for performing heuristic retrieval with Locality Sensitive Hash (LSH) methods and demonstrated that due to their ability to preserve similarity, LSH methods are viable options for heuristic retrieval. This work proposed two strategies based on DUARTE (2017)'s sequence of steps, called *document parallelism* (PnD) and *permutation parallelism* (PnP), that were implemented in Apache Spark distributed computing system, to support the task of identifying plagiarism in large document collections. The experiments demonstrated that the PnD and PnP strategies were able to reduce, according computational capacity increases, the time of the activities of representing, searching and retrieving documents; as well as achieving a high level of effectiveness for returning effectively plagiarized documents.

Sumário

Agradecimentos **iv**

Lista de Figuras **x**

Lista de Tabelas **xii**

1 Introdução **1**
1.1 Contextualização 1
1.2 Delimitação do Problema 3
1.3 Objetivo 3
1.4 Contribuições 3
1.5 Organização 4

2 A Identificação de Plágio **5**
2.1 A Identificação Automática de Plágio 6
2.2 Cenários de Identificação de Plágio 6
2.2.1 A Identificação de Plágio Interno (IPI) 7
2.2.2 A Identificação de Plágio Externo (IPE) 7
2.3 A Recuperação Heurística na Identificação de Plágio Externo (RHPE) 8
2.3.1 Abordagens de Busca pelos Vizinhos mais Próximos 9
2.4 Busca Aproximada pelos Vizinhos mais Próximos na Recuperação Heurística 10
2.5 Identificando Plágio com Locality Sensitive Hash 12
2.6 A Recuperação Heurística com métodos LSH baseados em Permutações Aleatórias 14
2.6.1 Tokenização 14
2.6.2 Geração de *Fingerprint* 15
2.6.3 Permutação das Features 15
2.6.4 Aplicação da Função de Seleção 16
2.6.5 Avaliação da Similaridade 17
2.7 Métodos LSH baseados em Permutações Aleatórias 17
2.7.1 Minwise Hashing 17

2.7.2 Min-Maxwise Hashing 18
2.7.3 Minmax Circular Sector Arcs 19

3 Paralelismo e Escalabilidade com o Apache Spark 20
3.1 O Paralelismo 20
3.1.1 O Problema da Dependência nos Dados 21
3.1.2 Delimitação da Estratégia de Paralelismo nos Dados 21
3.2 Escalabilidade com Sistemas de Computação de Alto Desempenho . . 22
3.2.1 O Apache Spark 22
3.3 Trabalhos Relacionados 24

4 Locality Sensitive Hash com Paralelismo nos Dados 25
4.1 O Paralelismo na Indexação 26
4.1.1 Análise sobre a Dependência de Dados na Indexação 26
4.2 O Paralelismo na tarefa de Recuperação de Informação 28
4.2.1 O Paralelismo na Extração de Consultas 28
4.2.2 O Paralelismo na Recuperação de Documentos Fonte 29
4.3 A exploração do Paralelismo em Diferentes Níveis 30

5 Desafios para Escalar as Estratégias de Paralelismo com o Apache Spark 32
5.1 O Alto Custo de Permutar em Ambiente Distribuído 32
5.1.1 O Problema de Replicar os Vetores de Permutação 34
5.2 Lidando com o Consumo de Memória na Permutação das Features . . 34
5.2.1 Estratégia de Re-Indexar o Vetor (RIV) 35
5.2.2 Estratégia de Expandir o Vetor no Final (EVF) 35
5.3 Estudo sobre a Influência das Estratégias de Permutação RIV e EVF em Estimar Jaccard 36
5.4 Estudo sobre o Impacto das Partições do Spark 39

6 Avaliação Experimental 42
6.1 O Ambiente de Execução dos Experimentos 42
6.2 Coleção de Dados 43
6.3 Métricas de Avaliação 44
6.4 A Recuperação Heurística do Plágio Externo (RHPE) em ambiente Distribuído 45
6.4.1 A Eficácia na Recuperação Heurística 46
6.4.2 A Eficiência na Indexação de Documentos 49
6.4.3 A Eficiência na Recuperação de Documentos Fonte 51
6.4.4 A Eficiência em Cenários de Larga Escala 54

7 Conclusões **57**
7.1 Trabalhos Futuros 58

Referências Bibliográficas **59**

Lista de Figuras

2.1 Cenários de Identificação de Plágio. adaptado de DUARTE (2017). . 7
2.2 Etapas da identificação de plágio externo adaptado de EHSAN & SHAKERY (2016) 8
2.3 Problema da busca dos vizinhos mais próximos. Adaptado de WANG *et al.* (2016) 9
2.4 Tarefa de Recuperação Heurística. Adaptado de (EHSAN & SHAKERY, 2016) 11
2.5 Estimando o Jaccard entre os conjuntos A e B a partir de assinaturas LSH. 13
2.6 Sequência de passos para realizar busca aproximada dos vizinhos mais próximos usando métodos LSH baseados em permutações aleatórias, adaptado de DUARTE *et al.* (2017). 14

3.1 Arquitetura do Cluster Spark com o gerenciador de recursos Standalone. Adaptado de (SPARK, 2018) 23

4.1 Estratégia ideal de paralelismo da tarefa de indexação com métodos LSH. Adaptado de DUARTE *et al.* (2017) 26
4.2 Estratégia de paralelismo da tarefa de indexação com métodos LSH. Estendida da seção 4.1 27
4.3 Etapas da tarefa de Recuperação de Informação. 28
4.4 Estratégia para execução paralela da etapa de extração de consultas segundo a sequencia de passos para métodos LSH baseados em permutações. Adaptado de DUARTE *et al.* (2017) 29
4.5 Estratégia para execução paralela da etapa de recuperação de documentos fonte. 30
4.6 Permutação em diferentes níveis de paralelismo nos dados. 31

5.1 Estratégia de re-indexar o vetor de permutação. 35
5.2 Estratégia de expandir o vetor de permutação no final. 36
5.3 Impacto das Partições do Spark na Indexação 40
5.4 Impacto das Partições do Spark na Recuperação da Informação . . . 41

6.1 Recall@pos consolidado das estratégias PnD e PnP na Recuperação Heurística na coleção PAN-11 plagiarism corpus. 46
6.2 Recall@pos do método Min-Hash consolidado na estratégias PnD e PnP em comparação ao baseline, o Min-Hash@MLlib. 47
6.3 Escalabilidade na Vazão de Indexação (documentos por minuto $\times$1000) das estratégias PnD e PnP 50
6.4 Escalabilidade na Vazão de Indexação (documentos por minuto $\times$1000) das estratégias PnD e PnP em comparação ao baseline Min-Hash@MLlib. 50
6.5 Escalabilidade na Recuperação de Documentos Fonte das estratégias PnD e PnP. 52
6.6 Escalabilidade das estratégias PnD e PnP na recuperação de documentos 53
6.7 Eficiência na Indexação de documentos em larga escala. 55
6.8 Eficiência na recuperação de documentos fonte, para retornar 7983 documentos. 56

Lista de Tabelas

5.1 Consumo aproximado (em megabytes) para armazenar V em memória. 33
5.2 Tempo médio de permutação (em segundos) das abordagens de pré-computar (PCV) e de re-computar os vetores de permutação (CVET). 37
5.3 RMSE e MAE das estratégias de permutação. 38
5.4 Tempo médio de permutação (segundos) das estratégias de permutação. 38
5.5 Parâmetros de Execução do Estudo sobre o Impacto das Partições do Spark. 40

6.1 Configuração dos Componentes do Ambiente de Execução dos Experimentos. 43
6.2 Evolução do Recall do Min-Hash conforme o pré-processamento varia. 48
6.3 Escalabilidade na Vazão de Indexação (documentos por minuto $\times$1000) para todos os casos avaliados. 51
6.4 Escalabilidade na recuperação de documentos fonte (em segundos) para todos os casos avaliados. 54
6.5 Perda do Min-Hash@MLlib. 55

Capítulo 1

Introdução

Esse capítulo apresenta a contextualização da tarefa de identificação de plágio externo. Em seguida, são apresentados a delimitação do problema de pesquisa, os objetivos e as contribuições deste trabalho. Por fim, a organização do trabalho é apresentada.

1.1 Contextualização

O plágio é o ato de apresentar ideias como se fossem próprias, de modo a fazer uso da obra de terceiros sem dar os devidos créditos (OXFORD, 1999), ou apresentar como novo e original um produto derivado de uma fonte existente (MERRIAM, 2017). Determinar se um fragmento de texto foi plagiado ou se uma pessoa é culpada por cometer este delito é uma decisão que requer o julgamento humano (BARRÓN-CEDEÑO, 2010). Assim, os sistemas de detecção automática de plágio lidam com a tarefa de encontrar indícios da ocorrência de plágio, para que um especialista possa realizar uma avaliação final e conclusiva. Portanto, os desafios dos sistemas de identificação automática de plágio estão relacionados as seguintes atividades:

(i) reduzir o tempo de representar, buscar e recuperar documentos que podem ter sido copiados, visto que o especialista aguarda a lista resultante, e

(ii) aumentar a assertividade em retornar os documentos efetivamente plagiados.

A tarefa de identificação de plágio divide-se em dois cenários, de acordo com os elementos presentes para a realização da tarefa, sendo eles: **plágio interno** e **plágio externo**. No cenário de identificação de plágio interno a tarefa é encontrar indícios de que um determinado documento suspeito cometeu plágio, utilizando como base para a investigação apenas o próprio documento suspeito, enquanto que, no cenário de identificação de plágio externo, a tarefa é investigar se um documento suspeito plagiou algum documento pertencente a um corpus de referência, denominados *documentos fonte*.

Especificamente no cenário de identificação de plágio externo, para detectar se um documento suspeito plagiou algum documento do corpus de referência, o custo computacional aumenta de forma linear, conforme também aumenta o número de documentos no corpus de referência, em vista que aumenta o número de comparações par-a-par, entre o documento suspeito e os documentos fonte. Para lidar com este problema, EHSAN & SHAKERY (2016) propuseram a adoção de uma etapa de recuperação heurística na identificação de plágio externo, a qual gera uma lista reduzida de documentos candidatos, isto é, um subconjunto com os documentos mais prováveis de terem sido plagiados, extraídos do corpus de referência com base em uma métrica de similaridade, com o intuito de reduzir a carga de trabalho das técnicas de detecção de plágio, caracterizadas por serem altamente custosas.

Apesar da etapa de recuperação heurística propor-se a permitir a realização da identificação de plágio externo em cenários de larga escala, é necessário avaliar a eficácia do modelo de recuperação, tendo em vista que existe a possibilidade de os documentos que efetivamente foram plagiados não serem retornados na lista reduzida de documentos candidatos, impossibilitando, assim, que a ocorrência do delito seja identificada nas etapas seguintes. Além disso, a etapa de recuperação heurística também deve ser avaliada quanto a sua eficiência em representar, buscar e recuperar, no menor tempo possível, os documentos candidatos.

Em consonância com a motivação de identificar plágio externo em cenários de larga escala, DUARTE (2017) experimentou a viabilidade, sob os aspectos de eficiência e eficácia, da utilização de métodos de *Locality Sensitive Hash* no modelo de recuperação heurística, para lidar com o problema da maldição da alta dimensionalidade, inerente à tarefa de computar a similaridade em massivas coleções de documentos textuais. Para tanto, DUARTE (2017) formalizou uma sequência de passos para realizar recuperação heurística baseada na utilização de métodos de Locality Sensitive Hash e específica para coleções de documentos.

Os métodos de Locality Sensitive Hash possuem a característica de gerar uma representação de baixa dimensionalidade, conhecida como *hash codes* ou *assinaturas*, criadas a partir da execução de N operações sobre a representação original dos documentos, e que ainda assim mantém a preservação da localidade e por consequência, permitem a estimar a similaridade. Devido aos métodos de Locality Sensitive Hash serem abordagens aproximadas, eles lidam com a probabilidade de perda de precisão em estimar a similaridade, e por consequência, de encontrar os documentos efetivamente plagiados, isto é, lidam com a probabilidade de perda de eficácia. Todavia, essa probabilidade de perda de eficácia pode ser minimizada com o aumento do número de assinaturas utilizadas para representar os documentos, o que, em contrapartida, eleva de forma linear o custo computacional para gerar as assinaturas,

acarretando no aumento do tempo da tarefa identificação de plágio externo, o que significa uma redução na eficiência.

1.2 Delimitação do Problema

A etapa de recuperação heurística tem o objetivo de viabilizar a identificação de plágio externo em massivas coleções de documentos ao reduzir a carga de trabalho das etapas computacionalmente mais custosas. Neste contexto, DUARTE (2017) formalizou uma sequência de passos para realizar a recuperação heurística do plágio externo baseada em métodos de Locality Sensitive Hash. Contudo, o trabalho de Duarte está focado no aspecto da viabilidade em termos de eficácia, ou seja, verificar se a representação gerada pelos métodos de Locality Sensitive Hash é uma opção viável para retornar, na lista reduzida, os documentos que efetivamente foram plagiados. Assim, Duarte argumenta em sua seção de trabalhos futuros, a necessidade de estudar oportunidades de paralelismo e distribuição em sua sequência de passos, visando criar uma solução escalável, capaz de lidar com o problema da escalabilidade nos dados, ou até mesmo com o intuito de aumentar a eficiência através do aumento da capacidade computacional de processamento, reduzindo os tempos para representar, buscar e retornar os documentos.

1.3 Objetivo

Este trabalho possui como objetivo apresentar uma estrutura com paralelismo e distribuição para apoiar a tarefa de identificação de plagio externo em grandes coleções de documentos textuais, realizada pela sequência de passos proposto por DUARTE (2017) para realizar a busca aproximada dos vizinhos mais próximos com métodos LSH, possibilitando um aumento da eficiência em razão da ampliação da capacidade computacional de processamento, reduzindo o custo temporal para representar, buscar e retornar os documentos.

1.4 Contribuições

As contribuições específicas apresentadas neste trabalho incluem a formação de estratégias de paralelismo nos documentos e nas permutações, baseada na sequência de passos de DUARTE (2017) para utilizar métodos LSH, que contribua para o aumento da eficiência nas atividades de indexação e recuperação em grandes coleções de documentos textuais. As outras contribuições e estudos conduzidos neste trabalho, são:

1. Formalização das estratégias RIV e EVF para viabilizar a execução do passo de permutação das features no Spark;

2. Estudo sobre o impacto das estratégias RIV e EVF na capacidade de estimar o Jaccard;

3. Estudo sobre impacto das partições do Spark ao problema da Recuperação Heurística (indexação e recuperação de documento), e

4. Implementação no Spark dos métodos LSH Min-Max-Hash e CSA.

1.5 Organização

O presente trabalho está organizado em 7 capítulos: no capítulo 2 são descritos conceitos relacionados à tarefa de identificação de plágio externo com o uso métodos de Locality Sensitive Hash; o capítulo 3 apresenta conceitos ligados aos temas paralelismo e escalabilidade e o sistema de computação de alto desempenho Apache Spark; o capítulo 4 apresenta propostas de estratégias de paralelismo para a recuperação heurística com LSH; o capítulo 5 formaliza propostas de alteração na sequência de passos do LSH para viabilizar a execução no Spark, e estuda a melhor configuração do Spark para executar as estratégias propostas; o capítulo 6 apresenta a condução e avaliação experimental da recuperação heurística sob perspectiva da escalabilidade de recursos computacionais, bem como da escalabilidade nos dados; por fim o capítulo 7 apresenta as conclusões sobre os experimentos realizados e os trabalhos futuros.

Capítulo 2

A Identificação de Plágio

O plágio no sentido de apropriação da obra intelectual de outra pessoa, existe desde os primórdios das atividades humanas ao produzirem obras de artes e pesquisas (MAURER *et al.*, 2006a). Ao analisar os diversos conceitos e origens etimológicas e históricas do plágio, podemos identificar que sua conceituação é vaga, conforme apresentado por LIDDELL (2003).

Neste trabalho consideramos uma série de definições de plágio em relação aos principais autores. Segundo a importante empresa de detecção de plágio PLAGIARISM (2017), pensar no plágio apenas como um ato de copiar um trabalho ou tomar ideias originais de terceiros é uma visão simplória. Ainda segundo esse trabalho, o plágio é considerado como "uma atitude fraudulenta que envolve o ato de roubar o trabalho de outra pessoa e mentir sobre isso depois". Segundo a empresa BBC, um objeto pode ser considerado original sem que, necessariamente, apresente ideias novas e nunca antes expressas por um humano, basta, simplesmente, fazer o trabalho por conta própria (BBC, 2011).

Para este trabalho, o plágio é o ato de copiar, ainda que disfarçadamente, ou utilizar parcialmente um conteúdo, sem dar os devidos crédito à fonte (MAURER *et al.*, 2006b). Contudo, no meio acadêmico, a diferenciação de plágio e reutilização proporciona uma série de discussões em relação a suas fronteiras e delimitações. Segundo BARRÓN-CEDEÑO (2010), "a fronteira entre o plágio e a reutilização justa nem sempre é clara e um dos principais problemas do plágio é entender o que ele realmente é e representa".

Em um cenário informacional marcado pelo crescimento de dados textuais, principalmente em decorrência da exploração de elementos na WWW (*World Wide Web*), e em um universo caracterizado pelo elevado volume dos dados (JIFFRIYA *et al.*, 2013), a atividade de detecção de plágio, enfrenta dois grandes desafios, os quais motivam o uso de sistemas computacionais para a identificação automática de plágio. O primeiro desafio está relacionado com o elevado número de práticas atreladas ao ato de plagiar (NAIK *et al.*, 2015); o segundo desafio está relacionado

com a quantidade crescente de documentos textuais alvo de análise da presença de plágio, necessitando o uso de sistemas computacionais de detecção automática para apoiar a atividade humana do especialista em investigar a ocorrência de plágio.

2.1 A Identificação Automática de Plágio

A identificação de plágio é uma tarefa multidisciplinar, tendo em vista que o plágio pode se manifestar em diversos domínios, tais como em documentos, músicas, filmes, pinturas e outros meios manipuláveis (ALZAHRANI *et al.*, 2012a). Ainda segundo ALZAHRANI *et al.* (2012a), devido ao grande volume em que dados são produzidos na atualidade, é ineficaz e inviável a utilização apenas da capacidade humana na tarefa de identificação de plágio.

A necessidade da criação e ampliação de técnicas e ferramentas que apoiem a tarefa de identificação automática de plágio, fomentou o surgimento da área de estudo **Pesquisa de Detecção de Plágio** (plagiarism detection research ou PD). A pesquisa de detecção de plágio estuda técnicas que sejam capazes de encontrar indícios de plágio, de maneira a permitir encaminhar a suspeita a um especialista para realizar uma avaliação final e decisiva (MEUSCHKE & GIPP, 2014; ZHANG & CHOW, 2011). Ademais, especificamente para plágio no domínio de texto, a área de pesquisa de detecção de plágio em linguagem natural (NLPD) estuda a aplicação de técnicas de processamento de linguagem natural (NLP) para encontrar evidências textuais de que um determinado documento suspeito d_{Susp} cometeu plágio.

2.2 Cenários de Identificação de Plágio

A tarefa de identificação de plágio divide-se em dois cenários, conforme visto na Figura 2.1, de acordo com a existência de um corpus de referência D, usado para verificar se um determinado documento suspeito d_{Susp} plagiou algum documento em D:

(i) cenário de identificação de **plágio interno** ou **intrínseco** (IPI), quando não existe o corpus, e

(ii) cenário de identificação de **plágio externo** ou **extrínseco** (IPE), quando existe o corpus.

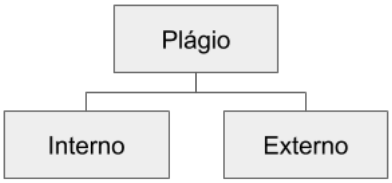


Figura 2.1 – Cenários de Identificação de Plágio. adaptado de DUARTE (2017).

2.2.1 A Identificação de Plágio Interno (IPI)

O cenário de identificação de plágio interno é caracterizado pela ausência ou não utilização de um corpus de referência. Assim, a tarefa é responder se um determinado documento suspeito d_{Susp} cometeu plágio, a partir de evidências de irregularidades e inconsistências no próprio no documento suspeito (STAMATATOS, 2009).

Portanto, a tarefa de identificação lida com a atividade de encontrar evidências de plágio, caracterizadas por possíveis incoerências em um documento suspeito, tais como alteração de vocabulário, complexidade e mal fluxo (STEIN *et al.*, 2007). Em geral, a identificação automática de plágio intrínseco está altamente relacionada com o uso de técnicas capazes de encontrar diferentes perfis de autores em um documento suspeito (STEIN *et al.*, 2007).

2.2.2 A Identificação de Plágio Externo (IPE)

O cenário de identificação de plágio externo é caracterizado pela presença e utilização de um corpus de referência. Assim, a tarefa é responder se um determinado documento suspeito d_{Susp} plagiou algum documento do corpus de referência D. Portanto, a tarefa envolve a realização de comparações entre o documento d_{Susp} com cada um dos documentos fontes $d_i \in D$, onde, a partir de uma medida de similaridade, os documentos mais próximos são considerados os mais prováveis de terem sido plagiados (ALZAHRANI *et al.*, 2012a; POTTHAST *et al.*, 2011b).

Um dos principais desafios da identificação de plágio externo, além de encontrar os documentos que efetivamente foram plagiados, é o aumento linear do custo computacional da tarefa, decorrente do maior número de comparações entre o documentos suspeito d_{Susp} e os documentos fonte $d_i \in D$, conforme aumenta o volume de documentos no corpus de referência D (LI & KÖNIG, 2011). Para tanto, com a finalidade de resolver o problema do aumento do custo computacional, e permitir a identificação de plágio em grandes coleções de referência, EHSAN & SHAKERY (2016) propuseram dividir a atividade de identificar plágio externo em três etapas: recuperação heurística; comparação detalhada; e pós processamento baseado em conhecimento.

A figura 2.2 apresenta o processo de identificação de plágio externo adaptado de EHSAN & SHAKERY (2016). A etapa de *recuperação heurística* tem o objetivo de gerar uma lista reduzida com os documentos mais prováveis de terem sido plagiados, também conhecida como lista de documentos candidatos. A lista de documentos candidatos proporciona a redução da carga de trabalho das etapas seguintes (MEUSCHKE & GIPP, 2014; THOMPSON *et al.*, 2015). Dessa forma, as etapas mais custosas da IPE, a *comparação detalhada* e o *pós processamento baseado em conhecimento*, lidam com a tarefa de identificar plágio apenas nos documentos mais prováveis de terem sido plagiados.

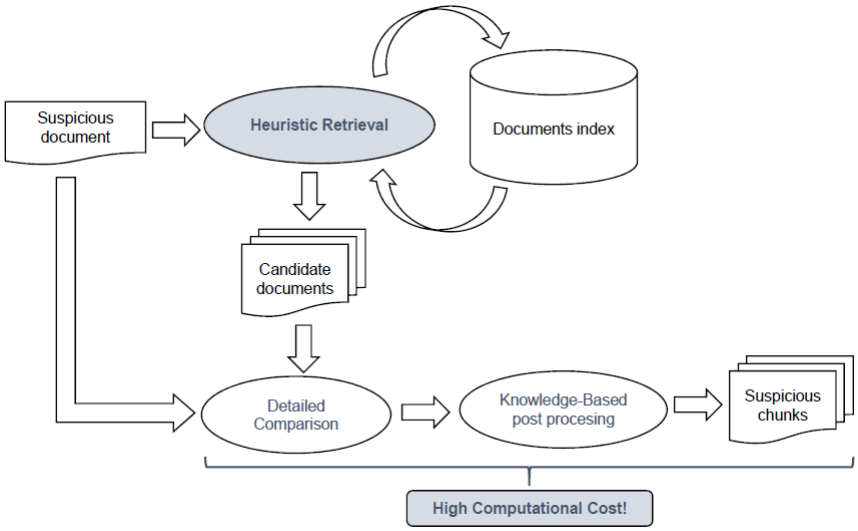


Figura 2.2 – Etapas da identificação de plágio externo adaptado de EHSAN & SHAKERY (2016)

Dentre as etapas que compõem o workflow de identificação de plágio externo proposto por EHSAN & SHAKERY (2016), este trabalho está delimitado no problema da etapa da recuperação heurística. A seção a seguir apresenta em detalhes a etapa de recuperação heurística.

2.3 A Recuperação Heurística na Identificação de Plágio Externo (RHPE)

A etapa de recuperação heurística na tarefa de identificação de plágio externo tem o objetivo de retornar uma coleção reduzida de documentos candidatos de serem alvos de plágio D_{Cand} para um determinado documento suspeito d_{Susp}. Para tanto, D_{Cand} é gerado a partir de uma função de ranqueamento $R(d_{Susp}, D)$, que avalia a probabilidade de $d_i \in D$ ser retornado para D_{Cand} (BAEZA-YATES *et al.*, 1999).

Ainda que a proposta da etapa de recuperação heurística, na tarefa de identificação de plágio externo, tenha o intuito de permitir identificar plágio em grandes coleções de documentos, sua viabilidade é avaliada pelos fatores eficácia e eficiência. A avaliação da **eficácia** verifica a capacidade da função de ranqueamento $R(d_{Susp}, D)$ em retornar para D_{Cand}, os documentos de D que foram efetivamente plagiados. Já a avaliação da eficiência está relacionada à capacidade de executar cada uma das atividades da recuperação heurística no menor tempo possível.

Uma das estratégias de recuperação heurística mais difundidas para lidar com o problema de encontrar itens similares em uma determinada coleção de dados é a **busca pelos vizinhos mais próximos**, em inglês, Nearest Neighbor Search (NNS) (DUARTE *et al.*, 2017; SHAKHNAROVICH *et al.*, 2006; WANG *et al.*, 2013). A figura 2.3 ilustra o problema da busca dos vizinhos mais próximos, onde os itens mais próximos a um determinado item de dados são considerados os similares.

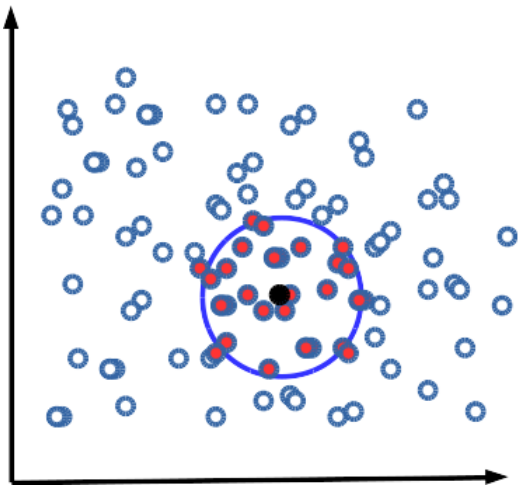


Figura 2.3 – Problema da busca dos vizinhos mais próximos. Adaptado de WANG *et al.* (2016)

Mais especificamente, a busca pelos vizinhos mais próximos pode ser realizada em diferentes abordagens: busca exaustiva dos vizinhos mais próximos; busca exata dos vizinhos mais próximos; e busca aproximada dos vizinhos mais próximos (MOHAMED & MARCHAND-MAILLET, 2015; WANG *et al.*, 2016). A seguir, as abordagens de busca pelos vizinhos mais próximos são apresentadas.

2.3.1 Abordagens de Busca pelos Vizinhos mais Próximos

A abordagem de **busca exaustiva** lida com a tarefa de encontrar os vizinhos mais próximos através da comparação entre pares de documentos (MOHAMED & MARCHAND-MAILLET, 2015). Logo, em cenários de larga escala, isto é, para grandes coleções de documentos, a solução de busca exaustiva é inviável devido a sua complexidade ser linear de acordo com a quantidade de documentos $d_i \in D$ (LI

& KÖNIG, 2011). Portanto, a busca exaustiva pelos vizinhos mais próximos é uma solução viável apenas em cenários de baixa escala, caracterizados por coleções com poucos documentos.

A abordagem de **busca exata** lida com a tarefa de encontrar os vizinhos mais próximos escapando-se do custo linear para comparar a similaridade entre todos os pares de documentos. Para tanto, utilizam técnicas como as de decomposição de matrizes e particionamento, que reduzem o número de comparações par-a-par. Contudo, devido à maldição da alta dimensionalidade, a performance das técnicas de busca exata degrada conforme a dimensionalidade aumenta (HYVÖNEN *et al.*, 2015). Portanto, a abordagem de busca exata é uma solução viável apenas em casos de baixa dimensionalidade (MOHAMED & MARCHAND-MAILLET, 2015; WANG *et al.*, 2014).

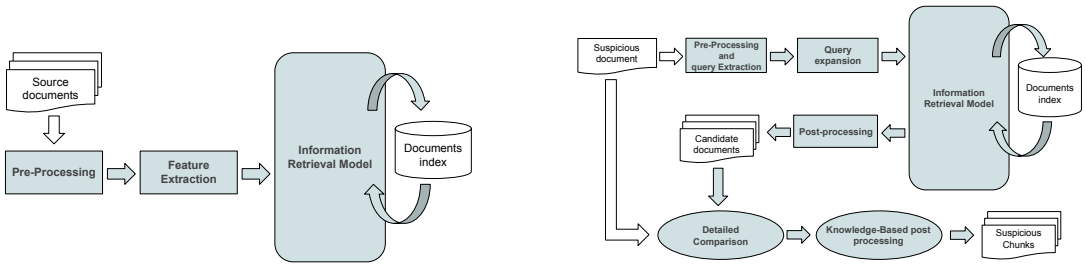
Já a abordagem de **busca aproximada** também lida com a tarefa de encontrar os vizinhos mais próximos evitando o custo linear para comparar a similaridade entre todos os pares de documentos. Para tanto, os métodos de busca aproximada extraem uma representação de baixa dimensionalidade e que ainda permitem estimar a similaridade entre documentos. Portanto, a abordagem de busca aproximada é uma solução viável para cenários de grande escala de documentos e de alta dimensionalidade (MOHAMED & MARCHAND-MAILLET, 2015; WANG *et al.*, 2014). Cabe ressaltar que, em troca de ser uma solução viável para cenários de larga escala e de alta dimensionalidade, as técnicas de busca aproximada aceitam uma pequena perda de precisão, visto que a similaridade entre documentos é computada de maneira aproximada (HYVÖNEN *et al.*, 2015).

Em consonância com a proposta de apoiar a identificação de plágio externo em grandes coleções de documentos, este trabalho está focado no uso da abordagem de busca aproximada dos vizinhos mais próximos na etapa recuperação heurística do plágio externo, a ser apresentada com mais detalhes na próxima seção.

2.4 Busca Aproximada pelos Vizinhos mais Próximos na Recuperação Heurística

A etapa de recuperação heurística, baseado na abordagem de busca aproximada pelos vizinhos mais próximos, é constituída, basicamente, por duas macro tarefas: Indexação e Recuperação de Documentos Fonte.

A tarefa de Indexação é composta por duas macro etapas, conforme apresentado na Figura 2.4a. Na etapa de pré-processamento, o texto é normalizado a fim de garantir a consistência do índice. Além disso, na etapa de Pré-processamento, as palavras dos documentos são convertidas em minúsculas, e espaços em branco, da-



(a) Sequência de passos da tarefa de Indexação.

(b) Sequência de passos da tarefa de Recuperação de Documentos Fonte.

Figura 2.4 – Tarefa de Recuperação Heurística. Adaptado de (EHSAN & SHAKERY, 2016)

tas e pontuação extras podem ser removidos. Em sequência, na etapa de extração de recursos, os documentos são convertidos em *tokens*, por exemplo, divididos em palavras, sentenças, n-gramas e etc, e os *tokens*, que podem ser usados como evidência de plágio, são selecionados como características do Modelo de Recuperação de Informações (EHSAN & SHAKERY, 2016).

A tarefa de Recuperação de Documento Fonte visa retornar a lista dos documentos mais prováveis de terem sido plagiados pelo documento suspeito d_{Susp} (POTTHAST *et al.*, 2014). A figura 2.4b apresenta as etapas a se realizar para permitir que o Modelo de Recuperação de Informações encontre evidências textuais do plágio. Nas etapas de pré-processamento e extração de consulta, de maneira semelhante a indexação (figura 2.4a), o documento suspeito é pré-processado, dividido em uma ou mais consultas e, por fim, a representação da consulta a ser usada na busca é gerada (DUARTE *et al.*, 2017). Em seguida, a etapa de expansão da consulta aprimora as consultas substituindo os recursos lexicais, sintáticos ou semânticos e, então, as consultas são submetidas ao Modelo de Recuperação de Informações, que a partir da função de ranqueamento, retorna a lista dos documentos mais similares ao documento suspeito. Por fim, a etapa de pós-processamento remove do conjunto final os documentos recuperados que apresentarem conteúdos semelhante, contudo, não foram plagiados por d_{Susp} (DUARTE *et al.*, 2017; EHSAN & SHAKERY, 2016).

Especificamente na construção do modelo de recuperação de informações, esse trabalho está delimitado no uso de métodos de Locality Sensitive Hash (LSH), que foram utilizados no trabalho de DUARTE (2017). A seção seguinte apresenta em detalhes o Locality Sensitive Hash, a sequência de passos proposta por DUARTE (2017) para realizar recuperação heurística com métodos LSH e, por fim, os métodos LSH aderentes à sequência de passos de DUARTE (2017).

2.5 Identificando Plágio com Locality Sensitive Hash

Locality Sensitive Hash (LSH) é uma família de métodos baseados em *hashs* aleatórias que possuem a propriedade de preservação da proximidade, e que devido a sua simplicidade, são amplamente utilizados na busca aproximada dos vizinhos mais próximos (WANG *et al.*, 2016). Mais especificamente, os métodos de Locality Sensitive Hash (LSH) são caracterizados pela presença de uma função *hash* $h(\cdot)$ onde a probabilidade de dois conjuntos, A e B, resultarem em ao menos um valor *hash* em comum é igual à proporção da similaridade de A e B (WANG *et al.*, 2014, 2016). Assim, como apresentado na Eq. 2.1, a probabilidade de colisão de A e B é avaliada pela similaridade par a par $sim(\cdot,\cdot)$, através de uma medida de similaridade, como por exemplo: co-seno, Hamming, Jaccard e etc. (WANG *et al.*, 2016).

$$Pr[h(A) = h(B)] = sim(A, B) \tag{2.1}$$

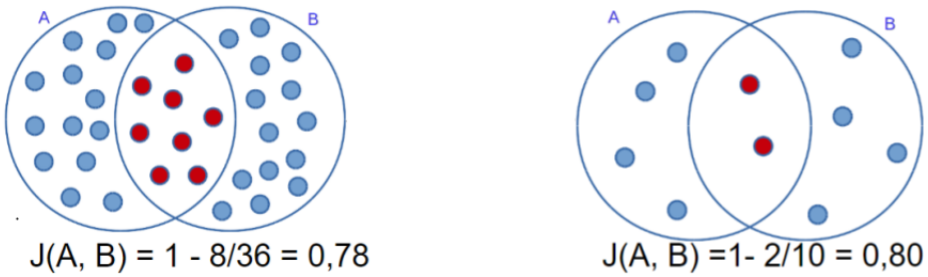
A distância de Jaccard é uma das medidas de similaridade mais populares para lidar com representações de conjuntos binários (JI *et al.*, 2013), como nos métodos LSH baseados em permutações aleatórias. Assim, conforme visto na eq 2.2, a distância de Jaccard computa a similaridade medindo o tamanho da interseção entre os conjuntos (HUANG, 2008; NIWATTANAKUL *et al.*, 2013).

$$J(A, B) = 1 - \frac{|A \bigcap B|}{|A \bigcup B|} \tag{2.2}$$

Especificamente em coleções de documentos D, cada documento $d_i \in D$ é representado por um vetor binário com um universo $\Omega = \{0, 1, 2, ..., |T|\text{ -1}\}$ de elementos, onde $|T|$ representa o tamanho do vocabulário. Dessa forma, a distância de Jaccard mede o grau de semelhança entre dois documentos d_0 e d_1, pelo tamanho da interseção do vocabulário de d_0 e d_1. Contudo, devido à maldição da alta dimensionalidade, computar a distância de Jaccard pode se tornar uma tarefa altamente custosa, além do custo de armazenar as representações dos documentos, em alta dimensionalidade, poder ser muito caro, ou até mesmo proibitivo, ao lidar com grandes coleções de documentos (LI & KÖNIG, 2011).

Por ser uma abordagem para realizar busca aproximada dos vizinhos mais próximos, os métodos LSH são capazes extrair uma representação de baixa dimensionalidade, que permite estimar o Jaccard entre dois conjuntos, sob um custo de obter resultados ligeiramente imprecisos, em comparação aos resultados obtidos em abordagens de busca exaustiva (WANG *et al.*, 2016). Todavia, os métodos LSH permitem realizar busca em grandes coleções de documentos, e a perda de acurácia pode ser minimizada com o aumento na quantidade de assinaturas em $k \in K$, resul-

tantes da aplicação de N funções *hashs* aleatórias $h(\cdot)$ e utilizadas para representar cada $d_i \in D$ (DUARTE *et al.*, 2017; WANG *et al.*, 2016).



(a) Similaridade a partir da representação original de A e B.

(b) Similaridade Estimada a partir de Assinaturas LSH.

Figura 2.5 – Estimando o Jaccard entre os conjuntos A e B a partir de assinaturas LSH.

Entretanto, em contrapartida ao aumento no número de assinaturas $k \in K$ suscitar no aumento da eficácia na identificação de plágio, devido ao aumento da precisão em estimar o Jaccard, a eficiência tende a diminuir, pois eleva o custo computacional para gerar mais assinaturas de formar linear. A título de ilustração, a figura 2.5 demonstra a capacidade de um método LSH em preservar a similaridade entre os conjuntos A e B. A figura 2.5a apresenta a distância de Jaccard de 0.78, obtida a partir de todas as features de A e B. Já a figura 2.5b ilustra a representação de menor dimensionalidade, extraída por um método LSH, e a distância de Jaccard estimada de 0.80, com base nas K assinaturas selecionadas. É possível notar nessa ilustração que houve uma imprecisão no Jaccard estimado, contudo, tal imprecisão tende a ser minimizada conforme o número de K assinaturas aumenta. Contudo, ao aumentar o número de assinaturas, o custo computacional da geração das assinaturas também aumenta.

Os trabalhos de WANG *et al.* (2016) e WANG *et al.* (2014) organizam os métodos LSH em três diferentes paradigmas, de acordo a abordagem utilizada para gerar as assinaturas, sendo eles:

(i) métodos baseados em projeções;

(ii) métodos baseados em permutações aleatórias, e

(iii) métodos baseados no aprendizado.

Para tanto, WANG *et al.* (2016) e WANG *et al.* (2014) formalizam um workflow genérico para realizar busca aproximada dos vizinhos mais próximos, abrangendo os três paradigmas. Entretanto, o trabalho de DUARTE (2017) formalizou uma sequência de passos mais detalhada, específica para métodos baseados em permutações aleatórias, para realizar busca aproximada dos vizinhos mais próximos em

coleções de documentos. A seção seguinte apresenta a tarefa de recuperação heurística com base no *pipeline* para métodos LSH baseados em Permutações Aleatórias de DUARTE (2017).

2.6 A Recuperação Heurística com métodos LSH baseados em Permutações Aleatórias

A recuperação heurística com métodos da família *Locality Sensitive Hash* lida com a tarefa de encontrar documentos cuja à probabilidade de serem considerados similares é dada pela maior quantidade de colisões entre as assinaturas. Para tanto, a figura 2.6 apresenta a sequência de passos proposta por DUARTE *et al.* (2017), composta por cinco passos, para realizar busca aproximada dos vizinhos mais próximos com métodos LSH baseados em permutações aleatórias, a ser descrito a seguir.

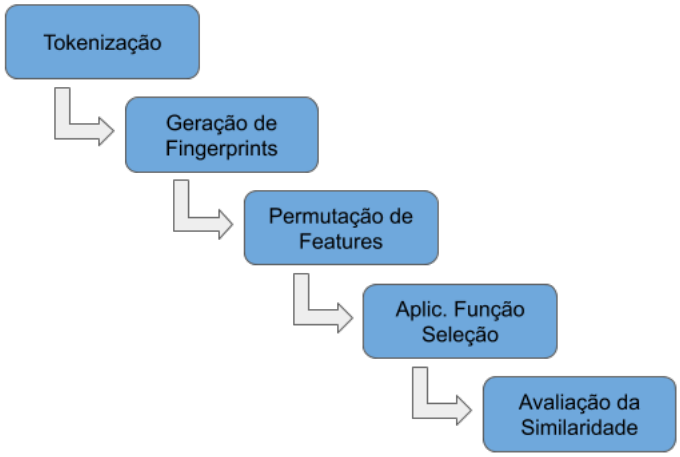


Figura 2.6 – Sequência de passos para realizar busca aproximada dos vizinhos mais próximos usando métodos LSH baseados em permutações aleatórias, adaptado de DUARTE *et al.* (2017).

2.6.1 Tokenização

No passo de tokenização, o objetivo é transformar os dados em uma representação de conjunto de *tokens*. Mais especificamente, de cada documento presente em uma coleção de documentos D, o conjunto de termos é extraído e representado em uma matriz termo-documento binária $M \in \{0,1\}^{j*i}$, onde cada coluna d_i de M representa um documento genérico correspondente ao seu conjunto S_i e cada tupla de t_j representa um termo do vocabulário T da coleção de documentos.

Como ilustração, suponha uma coleção de documentos ($D = \{d_0, d_1\} = \{$ *"this car is good"*, *"his car is red"*$\}$ e que cada palavra presente em D será considerada um termo. Logo, o vocabulário T é composto por $\{$ *"this"*, *"his"*, *"car"*, *"is"*, *"good"*,

"red"}, visto que $d_0 = \{$*"this", "car", "is", "good"*$\}$ e que $d_1 = \{$*"his", "car", "is", "red"*$\}$ e, portanto, D apresentará uma matriz M igual a :

$$M = \begin{array}{r} \\ this \\ his \\ car \\ is \\ good \\ red \end{array} \begin{pmatrix} d_0 & d_1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$

2.6.2 Geração de *Fingerprint*

No passo de geração de *fingerprint*, o objetivo é gerar um identificador g_j inteiro e não negativo para cada termo t_j do vocabulário da coleção T. Dessa forma, a função de transformação $\psi(t_j)$ irá produzir o mesmo valor de *fingerprint* g_j em todos os documentos onde o termo t_j aparece. Então, cada documento $d_i \in D$ passa a ser representado por um conjunto de *fingerprints* L, conforme apresentado na eq 2.3, gerado pela aplicação de uma função de transformação ψ no vocabulário da coleção T.

$$L = [\psi(t_j) \text{ para cada } t_j \in T], \text{ onde } \psi(t_j) : T \to \mathbb{N} \tag{2.3}$$

Seguindo com a ilustração do passo anterior, ao aplicar a função $\psi(t_j)$, que representa t_j em T através do índice j, produzirá $L = [0, 1, 2, 3, 4, 5]$, que corresponde a T=$\{$*"this", "his", "car", "is", "good", "red"*$\}$. Logo, L é utilizado para representar os termos de D na matriz M, resultando na matriz M'.

$$M' = \begin{array}{r} \\ \psi(this) = \\ \psi(his) = \\ \psi(car) = \\ \psi(is) = \\ \psi(good) = \\ \psi(red) = \end{array} \begin{array}{c} L \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{pmatrix} d_0 & d_1 \\ 0 & \varnothing \\ \varnothing & 1 \\ 2 & 2 \\ 3 & 3 \\ 4 & \varnothing \\ \varnothing & 5 \end{pmatrix}$$

2.6.3 Permutação das Features

No passo de permutação das features o objetivo é gerar uma nova matriz $L'_{i,N}$ para cada L_i, que corresponde a sequência de *fingerprints* de $d_i \in D$. Para tanto, inicialmente, $v_n \in V$vetores de permutação são gerados com a aplicação de π_n funções de

aleatórias de permutação, sobre o conjunto de *fingerprints* L, que corresponde aos termos t_j do vocabulário da coleção T. Para gerar v_n, a função $h(.)$ da permutação π_n pode ser uma simples função de embaralhamento, então, cada *fingerprint* g_j do vocabulário assume um novo valor, de acordo com a permutação π_n. Em seguida, após gerar os vetores $v_n \in V$ de permutação, para permutar $d_i \in D$, as *fingerprints* g_j em L_i assumem um novo valor, considerando que a ordem da *fingerprints* de d_i corresponde a ordem dos elementos de v_n. Assim, conforme apresentado na eq 2.4 , $L'_{i,N}$ é gerado após realizar N permutações.

$$L'_{i,N} = \{v_1(L_i), v_2(L_i). \ldots, v_n(L_i)\} \tag{2.4}$$

Seguindo a ilustração do passo anterior, suponha que $d_i \in D$ será permutado com $|N| = 2$ permutações. Para tanto, v_0=[5, 3, 0 ,2, 4, 1] e v_1 = [2, 5, 4, 1, 0, 3] foram gerados através de duas operações aleatórias de embaralhamento de L. Logo, $L'_{i,0}$ e $L'_{i,1}$ são gerados após permutar cada $d_i \in D$.

$$\begin{array}{cc} L & v_0 \\ 0 & 5 \\ 1 & 3 \\ 2 & 0 \\ 3 & 2 \\ 4 & 4 \\ 5 & 1 \end{array} L_{i,0} = \begin{pmatrix} \pi_0(d_0) & \pi_0(d_1) \\ 5 & \varnothing \\ \varnothing & 3 \\ 0 & 0 \\ 2 & 2 \\ 4 & \varnothing \\ \varnothing & 1 \end{pmatrix} \quad \begin{array}{c} v_1 \\ 2 \\ 5 \\ 4 \\ 1 \\ 0 \\ 3 \end{array} L_{i,1} = \begin{pmatrix} \pi_1(d_0) & \pi_1(d_1) \\ 2 & \varnothing \\ \varnothing & 5 \\ 4 & 4 \\ 1 & 1 \\ 0 & \varnothing \\ \varnothing & 3 \end{pmatrix}$$

2.6.4 Aplicação da Função de Seleção

No passo de aplicação da função de permutação o objetivo é gerar k assinaturas, para cada documento $d_i \in D$, a partir da sumarização de $L'_{i,N}$, resultante da aplicação de N permutações sobre o conjunto de *fingerprints* L_i. Assim, para reduzir o alto custo para computar a distância entre pares de documentos, $L'_{i,N}$ é sumarizado em k assinaturas, geradas através da função de seleção $sel(.)$ de um método LSH, que seleciona uma ou mais *fingerprints* em $L'_{i,n}$, correspondentes aos tokens $S_{i,j} \in S_j$, em $d_i \in D$.

Seguindo a ilustração do passo anterior, suponha que para sumarizar $L'_{i,N}$, a função de seleção $minimum(.)$ do método Min-Hash seja utilizada. Logo K_i, ou seja , K assinaturas para cada $d_i \in D$, são geradas após sumarizar cada vetor permutado $L'_{i,n}|n \in N$ de *fingerprints*, através da função de seleção do Min-Hash, resultando na seguinte representação:

$$K_i = \begin{array}{cc} d_i \in D & [\pi_0, \pi_1] \\ d_0 & [0,0] \\ d_1 & [0,1] \end{array}$$

Após sumarizar $L'_{i,N}$ com aplicação da função de seleção, especificamente na tarefa de indexação de documentos, a tarefa produz o seguinte índice invertido orientado a assinaturas, que armazena os documentos da coleção D, com o intuito de facilitar a tarefa de busca:

$$\text{Índice Invertido} = \begin{pmatrix} 0_{\pi_0} & [d_0, d_1] \\ 0_{\pi_1} & [d_0] \\ 1_{\pi_1} & [d_1] \end{pmatrix}$$

2.6.5 Avaliação da Similaridade

No passo de avaliação da similaridade, cada assinatura extraída da consulta é submetida a um mecanismo de pesquisa baseado no índice invertido, criado na etapa anterior. Assim, a similaridade é avaliada entre a consulta e documentos indexados, e a recuperação é realizada pelo mecanismo de busca, baseado no ranqueamento do número de colisões nas assinaturas.

Na próxima seção são apresentados os métodos LSH baseados em permutações aleatórias, aderente a sequência de passos 2.6 proposto por DUARTE *et al.* (2017).

2.7 Métodos LSH baseados em Permutações Aleatórias

Os diferentes métodos LSH baseados em permutações aleatórias visam explorar possíveis otimizações nas etapas de *Permutação das features* e *Aplicação da Função de Seleção*, também conhecidas como de etapa de permutação, do *workflow* de recuperação heurística com métodos LSH (2.6). Em geral, as otimizações visam aumentar a eficiência ao realizar a tarefa de permutação no menor tempo possível, tendo em vista que o custo computacional para permutar e selecionar muitas assinaturas é linear, sendo estes os passos mais custosos do *pipeline*. A seguir serão apresentados os métodos LSH aderentes ao *pipeline* de DUARTE (2017).

2.7.1 Minwise Hashing

O Minwise Hashing, também conhecido como Min-Hash, foi proposto por BRODER (1997), com o objetivo de armazenar apenas um esboço do documento, de forma a permitir estimar a similaridade de Jaccard. Para tanto, N permutações aleatórias

$\pi : \Omega \rightarrow \Omega$ são geradas, e então, o menor valor *hash* , de cada permutação, é selecionado como assinatura pela função de seleção: $min(\pi_k(A))$. A probabilidade de haver colisão em ao menos uma assinatura $Pr[X_\pi = 1]$ (eq. 2.6), para dois conjuntos não vazios A e B é dada pela variável aleatória X_π (eq. 2.5).

$$X_\pi = \begin{cases} 1, & min(\pi_k(A)) = min(\pi_k(B)) \\ 0, & otherwise \end{cases} \tag{2.5}$$

$$Pr[X_\pi = 1] = \frac{|A \bigcap B|}{|A \bigcup B|} = J(A, B) \tag{2.6}$$

Logo, após gerar um um conjunto de N permutações aleatórias $\pi_1, \pi_2, ..., \pi_k$ correspondentes a $X_1, X_2, ..., X_n$ variáveis, é possível estimar a similaridade de Jaccard conforme apresentado na eq. 2.7, através da contagem do número de colisões nas assinaturas de A e B ($X_i = 1 \mid min(\pi_i(A)) = min(\pi_i(B))$), com variância igual a $Var[X] = \frac{1}{N} J(A, B)(1 - J(A, B))$ e esperança $\mathbb{E}[X_\pi] = Pr[X_\pi = 1] = J(A, B)$.

$$X = \frac{1}{k} \sum_{i=1}^{n} X_i \tag{2.7}$$

Com o intuito de gerar mais assinaturas a um custo computacional menor, a literatura apresentou evoluções a partir do Min-Hash, como o Min-Max-Hash e o CSA (DUARTE *et al.*, 2017; JI *et al.*, 2013), a serem apresentadas a seguir.

2.7.2 Min-Maxwise Hashing

O Min-Maxwise Hashing, também conhecido por Min-Max-Hash, foi proposto por JI *et al.* (2013) e, é capaz de gerar o dobro de assinaturas que o Min-Hash, com o mesmo número de permutações. Para tanto, a função de seleção do Min-Max-Hash seleciona tanto o menor valor $min(\pi_k(A))$ de um permutação π_k, como o maior valor, $max(\pi_k(A))$, para $k/2$ permutações. Portanto, o Min-Max-Hash é capaz de selecionar o mesmo número de assinaturas que o Min-Hash, reduzindo o tempo de permutação pela metade, e, ainda assim, mantém a capacidade de estimar a distância entre pares de documentos, sem inserir viés, com a mesma probabilidade do algoritmo Min-Hash (JI *et al.* (2013)), conforme apresentado na eq. 2.9, dado pela variável aleatória Z_π (eq. 2.8).

$$Z_\pi = \begin{cases} 1, & max(\pi_k(A)) = max(\pi_k(B)) \\ 0, & otherwise \end{cases} \tag{2.8}$$

$$Pr[Z_\pi = 1] = \frac{|A \bigcap B|}{|A \bigcup B|} = J(A, B) = Pr[X_\pi = 1] \tag{2.9}$$

2.7.3 Minmax Circular Sector Arcs

O Minmax Circular Sector Arcs, também conhecido como CSA, foi proposto por DUARTE *et al.* (2017) e, é capaz de gerar, ao custo do mesmo número de permutações, duas vezes mais assinaturas que o Min-Max-Hash, e consequentemente, quatro vezes mais assinaturas que o Min-Hash. Para tanto, na função de seleção do CSA, além de selecionar o menor e o maior valor, também são selecionados o valor supremo $K^{sup}_{S_i,n}$ e o valor ínfimo $K^{inf}_{S_i,n}$ de $K_{S_i,n}$, onde $K_{S_i,n}$ é a propriedade triangular que codifica a permutação de dois conjuntos, conforme visto na equação 2.10.

$$K_{S_{i,n}} = \sqrt{max^2_{S_{i,n}} - min^2_{S_{i,n}}} \tag{2.10}$$

Ainda que o CSA possa gerar resultados ligeiramente um pouco mais imprecisos, devido a probabilidade de colisão estar relacionada com a combinação da probabilidade das variáveis aleatórias ($Pr[X_\pi = 1 \mid Z_\pi = 1]$), DUARTE *et al.* (2017) demonstrou que o método CSA é capaz de selecionar um número muito maior de assinaturas, reduzindo o tempo de permutação, e a perda de precisão pode ser minimizada utilizando um pouco mais de assinaturas para representar os documentos.

Capítulo 3

Paralelismo e Escalabilidade com o Apache Spark

Esse capítulo apresenta conceitos relacionados ao paralelismo de uma tarefa, a ser utilizados por este trabalho para cumprir a proposta de aumentar a eficiência na recuperação heurística da tarefa de identificação de plágio externo. Em seguida, com o intuito de propor uma solução capaz lidar com grandes coleções de documentos, o Apache Spark, um sistema de computação de alto desempenho, é apresentado em detalhes. Por fim, são apresentados os trabalhos relacionados e a definição do nosso *baseline*.

3.1 O Paralelismo

Com o intuito de acelerar a execução de uma atividade através do paralelismo, as estratégias denominadas **paralelismo na tarefa** e **paralelismo nos dados** são amplamente difundidas (HARDWICK, 1997; KUMAR, 2002; SITARAM & MANJUNATH, 2011).

No paralelismo na tarefa, a tarefa é dívida em sub-tarefas independentes entre si, ou seja, o produto de uma sub-tarefa não é utilizado como entrada de uma outra sub-tarefa (KUMAR, 2002). No paralelismo nos dados, a tarefa é mapeada em diferentes unidades de processamentos, onde cada unidade de processamento executa operações semelhantes, em uma parte dos dados (KUMAR, 2002). No paralelismo nos dados, devido às diferentes unidades de processamento executarem os mesmos cálculos, é possível obter um balanceamento de carga satisfatório, apenas com um bom método de particionamento de dados, enquanto que no paralelismo na tarefa, obter um balanceamento de carga satisfatório pode ser não trivial, tendo em vista que as diferentes sub-tarefas independentes podem ser ter diferentes custos computacionais (KUMAR, 2002).

Apesar de a ideia de dividir um problema em partes independentes, sejam sub-tarefas ou subconjuntos de dados, parecer simples, para fazer uso de múltiplas unidades de processamento e acelerar a execução de um algoritmo, muitas tarefas computacionais não são plausíveis de serem executadas em partes independentes e simultâneas, devido ao problema da dependência nos dados (KUMAR, 2002), que será apresentado em mais detalhes a seguir.

3.1.1 O Problema da Dependência nos Dados

O problema da dependência nos dados é caracterizado por um gargalo na execução de uma parte da tarefa, devido às partes da tarefa dependerem do produto resultante de uma outra parte, o que dificulta sua à execução em unidades de processamento distintas e de forma simultânea (KUMAR, 2002).

Na estratégia de paralelismo na tarefa, a dependência nos dados é caracterizada pela existência de sub-tarefas que necessitam de produtos de outra sub-tarefa. Dessa forma, existe um gargalo, e, para que uma sub-tarefa seja iniciada, outra sub-tarefa precisa finalizar sua execução. Na estratégia de paralelismo nos dados, a dependência nos dados é caracterizada pelo cenário onde não é possível executar por completo uma tarefa em um sub-conjunto dos dados, devido à existência de um ponto onde as unidades de processamento necessitam de dados mapeados em outras unidades de processamento. Nessa situação, é necessário aguardar todas as múltiplas unidades de processamento atingirem o mesmo ponto, para que então os dados possam ser trafegados entre as distintas unidades de processamentos, e então finalmente a execução da atividade em paralelo é retomada.

3.1.2 Delimitação da Estratégia de Paralelismo nos Dados

Conforme visto na figura 2.6, os passos da sequência de DUARTE (2017) são caracterizados pela dependência de resultados produzidos pelo passo anterior, o que dificulta a adoção da estratégia de paralelismo na tarefa. Portanto, este trabalho está delimitado na estratégia de paralelismo nos dados, que além ser capaz de executar simultaneamente a sequência de passos de DUARTE (2017), em diferentes partes da coleção de documentos, também pode ser facilmente escalável, quando implantado em um sistema de computação distribuída, com base na adoção de um modelo gerenciador-trabalhador, onde o nó gerenciador é o responsável por realizar o particionamento dos dados e delegar as partes aos nó(s) trabalhadores.

3.2 Escalabilidade com Sistemas de Computação de Alto Desempenho

Com a proposta de apoiar a identificação de plágio externo em grandes coleções de documentos, onde a utilização de uma única maquina é inviável e proibitiva, tanto do ponto de vista de armazenamento dos dados, bem como do alto custo computacional inerente à tarefa, esta dissertação se propõe a fazer uso de um sistema de computação de alto desempenho, para viabilizar, em função do aumento da capacidade computacional, a identificação de plágio externo em cenários de grandes e crescentes corpus de referência.

Uma das ferramentas mais populares para utilização de supercomputadores e clusters com o objetivo de escalar e paralelizar uma tarefa, permitindo sua execução em um tempo satisfatório, é o Apache Hadoop (DEAN & GHEMAWAT, 2004). O Hadoop é um sistema de computação distribuída que provê uma implementação de código aberto do sistema *MapReduce* do Google. Um dos principais componentes do Apache Hadoop é seu sistema de arquivos distribuídos (HDFS), que o permite ser um sistema capaz de atingir uma alta disponibilidade e tolerância a falhas, sem a necessidade de nenhum hardware especial para construir o cluster. Recentemente, o Hadoop foi superado pelo Apache Spark, um sistema de computação distribuída que aprimora o processamento em grandes volumes de dados, principalmente através do melhor uso de memória principal, resultando, em alguns casos, na execução de um algoritmo até 100 vezes mais rápido que o Hadoop (SPARK, 2018).

3.2.1 O Apache Spark

O Apache Spark é um sistema de computação distribuída de alto desempenho, de uso geral, que suporta a criação de aplicativos em diversas linguagens de programação, como Python, R, Java e Scala (SPARK, 2018). Ademais, o Spark também suporta um vasto conjunto de ferramentas de alto nível, tais como: MLlib, usado para aprendizado de máquina; Spark SQL e Dataframes, para processamento de dados estruturados; e Spark Streaming, para processamento de streams (SPARK, 2018). O Spark pode ser implantado em um cluster Hadoop com o gerenciador de recursos YARN, no Apache Mesos ou em seu modo independente, conhecido como Standalone.

Neste trabalho o cluster Spark foi implementado em seu modo nativo, o Standalone, e com a configuração de não compartilhamento de recursos (*shared-nothing).* A figura 3.1 ilustra sua arquitetura, onde o nó *Driver Program* é responsável por submeter uma tarefa a ser executada; o nó *Cluster Manager* é responsável por receber a tarefa e gerenciar a distribuição do trabalho; e os nó(s) *Workers* são os responsáveis

por executarem o trabalho, de acordo com a gerência do Cluster Manager (SPARK, 2018).

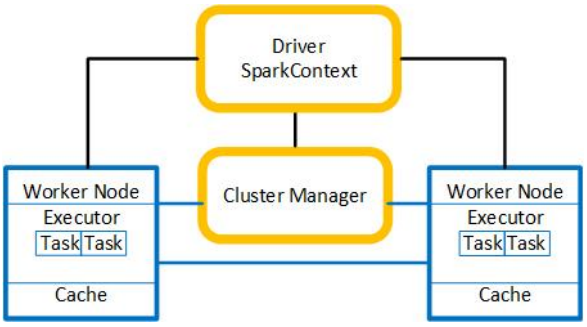


Figura 3.1 – Arquitetura do Cluster Spark com o gerenciador de recursos Standalone. Adaptado de (SPARK, 2018)

O Spark trabalha com uma abstração dos dados, denominada Resillient Distributed Dataset (RDD), que otimiza a utilização de memória principal, evitando acessos desnecessários ao disco. RDD é uma coleção de objetos imutáveis, fisicamente particionada entre os nó(s) do cluster, de modo a permitir que cada nó do cluster realize uma parte do trabalho em um subconjunto dos dados, provendo um eficiente balanceamento da carga de trabalho (SPARK, 2018).

O Spark é um sistema de computação distribuída que implementa nativamente a estratégia de paralelismo nos dados. Para tanto, a coleção de dados é divida em partições, isto é, subconjuntos dos dados espalhados pelos nó(s) trabalhadores de uma tarefa, que permitem que cada nó realize apenas uma parte do trabalho. Para que todos os núcleos das CPUs dos nó(s) trabalhadores possam realizar uma parte do trabalho, o Spark necessita de ao menos uma partição por núcleo, assim, para evitar a sub-utilização de recursos, é necessário haver ao menos uma partição por núcleo (SPARK, 2018). A título de ilustração, suponha que uma tarefa seja submetida ao Spark, implantado em um cluster com dois nó(s) trabalhadores, sendo cada um com 8 cores, portanto, 16 cores no total; para que não ocorra sub-utilização de recursos, a coleção de dados precisa ser dividida em ao menos 16 partições.

Entretanto, é importante ressaltar que lidar com um número muito grande de partições pode resultar no aumento de complexidade, elevando a carga de trabalho do nó Cluster Manager. Logo, a melhor configuração do número de partições pode variar de acordo com a coleção de dados e o trabalho a ser executado, o que torna necessário uma avaliação em cada caso, para encontrar o número de partições que maximize o desempenho da execução da tarefa.

3.3 Trabalhos Relacionados

Este trabalho tem o intuito de levar a proposta de DUARTE (2017) um passo adiante, através da exploração de oportunidades de paralelismo na sequência de passos de métodos LSH, com o intuito de reduzir os tempos das etapas de indexação e recuperação de informação, proporcionando o aumento da eficiência na recuperação heurística da tarefa de identificação de plágio externo.

A motivação deste trabalho em atingir um maior nível de eficiência ao paralelizar e distribuir uma tarefa também é objeto de motivação de muitos outros, tais como: o trabalho de JÚNIOR (2011), que a explorou do uso do paralelismo nos dados para aumentar a eficiência de algoritmos de processamento de imagem; o trabalho de GUPTA & KULARIYA (2016), que utilizou o Spark para criar uma estrutura de detecção rápida e eficiente de intrusões em redes de segurança cibernética; e o trabalho de PANIGRAHI *et al.* (2016), que utilizou o Spark para criar um eficiente mecanismo de recomendação baseado em técnicas de filtragem colaborativa.

Entretanto, o principal trabalho relacionado com nossa proposta é o trabalho de MENG *et al.* (2016), que realizou uma implementação do método Min-Hash no Spark. A implementação do método Min-Hash realizada por MENG *et al.* (2016), denominada por este trabalho de Min-Hash@MLlib, tem o objetivo de apoiar a atividade de busca aproximada pelos vizinhos mais próximos, o que permite sua aplicabilidade na tarefa de recuperação heurística da identificação de plágio externo, e, portanto, é utilizado neste trabalho como base de comparação (*baseline*).

Cabe ressaltar que o Min-Hash@MLlib é uma implementação fechada de um método, enquanto que este trabalho tem o intuito de propor uma estratégia genérica, capaz de executar qualquer método LSH baseado em permutações aleatórias. Além disso, o Min-Hash@MLlib não se baseia na sequência de passos de DUARTE (2017), e não realiza todos os passos propostos por ela. Especificamente na indexação, o Min-Hash@MLlib não realiza o passo de criação do índice invertido, proposto com o intuito de otimizar a etapa de recuperação de informação.

Capítulo 4

Locality Sensitive Hash com Paralelismo nos Dados

Esse capítulo apresenta uma proposta de estratégia para apoiar a tarefa de recuperação heurística com métodos de Locality Sensitive Hash em cenários de larga escala. Além disso, a estratégia proposta tem o intuito de aumentar a eficiência da recuperação heurística da identificação de plágio externo, isto é, reduzir o tempo nas tarefas de representar, buscar e retornar a lista de documentos candidatos, em função do aumento da capacidade computacional de processamento.

De forma específica, com o intuito de reduzir o custo computacional do passo mais custoso do LSH, o passo de permutação das features, e portanto, melhorar a eficiência ao realizar a tarefa num menor tempo, os trabalhos de JI *et al.* (2013) e de DUARTE *et al.* (2017) apresentam métodos LSH que aumentam a capacidade de selecionar assinaturas por permutação em, respectivamente, duas e quatro vezes mais, em comparação com o método de BRODER (1997). Além disso, DUARTE (2017) atestou a viabilidade do uso de métodos LSH aplicados ao problema da recuperação heurística, e, para tanto, formalizou uma sequência de passos (2.6) para realizar busca aproximada dos vizinhos mais próximos usando métodos LSH.

Portanto, o intuito desse capitulo é levar o trabalho de DUARTE (2017) um passo adiante, visto que o mesmo aponta como uma oportunidade para trabalhos futuros, a exploração de pontos do pipeline LSH que possam ser paralelizados e escalados em ambientes distribuídos, para enfim consolidar a viabilidade da recuperação heurística com LSH em cenários de larga escala. Para tanto, os passos do pipeline LSH de Duarte são incorporados na estratégia para prover o paralelismo e a distribuição das etapas de indexação, extração de consulta e recuperação de documentos fonte, da recuperação heurística do plágio externo.

4.1 O Paralelismo na Indexação

A tarefa de indexação com métodos LSH é responsável por transformar a coleção de documentos fonte em uma representação de índice invertido orientado a assinaturas. No contexto da estratégia de paralelismo nos dados, o objetivo é permitir a execução simultânea dos passos do pipeline LSH (Fig. 2.4a), para diferentes subconjuntos da coleção de documentos.

A estratégia ideal para a execução paralela dos passos LSH na tarefa de indexação é representada na figura4.1. Inicialmente cada unidade de processamento (UP) recebe um sub-conjunto da coleção de documentos. Em seguida, cada unidade de processamento realiza a sequência os passos de tokenização, geração de fingerprints, permutação das features e aplicação da função de seleção, com base apenas em seu subconjunto de documentos, e ao final, os resultados são combinados para compor o índice invertido.

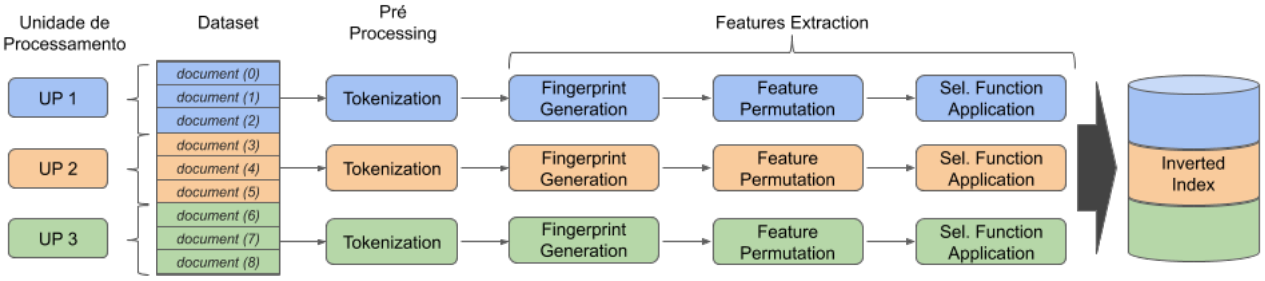


Figura 4.1 – Estratégia ideal de paralelismo da tarefa de indexação com métodos LSH. Adaptado de DUARTE *et al.* (2017)

A título de ilustração, suponha a execução da indexação em um sistema de computação distribuída, implantado com o conceito de *shared-nothing*. Segundo a estratégia ideal (4.1), cada nó trabalhador deve ser capaz de executar toda a sequência de passos, com base apenas em seu sub-conjunto de dados. Contudo, devido o problema da dependência nos dados, é necessária uma análise sobre sua eventual incidência, em cada um dos passos do pipeline LSH. A análise da incidência do problema da dependência nos dados tem o intuito de identificar possíveis gargalos, isto é, passos do pipeline LSH em que uma UP necessite dos resultados processados por outra UP, acarretando no alto custo de trafegar dados entre diferentes nó(s).

4.1.1 Análise sobre a Dependência de Dados na Indexação

A figura 4.1 representa a estratégia ideal para a indexação, segundo a estratégia de paralelismo nos dados. Contudo, devido o problema da dependência nos dados, visto na seção 3.1, faz-se necessário uma análise para verificar um a possível incidência do problema em algum dos passos do pipeline de Duarte.

(*i*) **Dataset** - Na etapa de leitura da coleção de documentos são há dependência nos dados, cada unidade de processamento é responsável por ler um subconjunto de documentos. A título de ilustração, suponha que um unidade de processamento represente um nó em um cluster, cada nó do cluster pode realizar a leitura dos documentos através de um sistema gerenciador de arquivos distribuídos, como por exemplo o Hadoop Distributed File System (HDFS), ou até mesmo por meio de um simples acesso a disco local.

(*ii*) **Pré-Processamento** A etapa de pré-processamento do workflow (figura 2.4a) é realizada no pipeline de LSH no passo de tokenização (seção 2.6.1) e é responsável por realizar sub-tarefas como: limpeza dos dados, tokenização do texto, remoção de stop-words e construção do vocabulário da coleção. Vale destacar que na atividade de construção do vocabulário da coleção há o problema da dependência nos dados, o que representa um gargalo, devido a necessidade de troca de dados entre os nó(s).

(*ii*) **Extração das *features*** Na etapa de extração das *features* (Figura 2.4a), que corresponde ao passo de geração de *fingerprints* (seção 2.6.2), não há o problema da dependência nos dados para gerar identificadores numéricos para os *tokens*.

(*iii*) **Modelo de Recuperação de Informação** Nas etapa de permutação das features e aplicação da função de seleção cada unidade de processamento é capaz de gerar assinaturas sem realizar troca de dados. Contudo, após gerar as assinaturas, especificamente na atividade de indexação, a etapa de aplicação da função de seleção é responsável por gerar o índice invertido de assinaturas, o qual exige uma grande troca de dados entre as unidades de processamento.

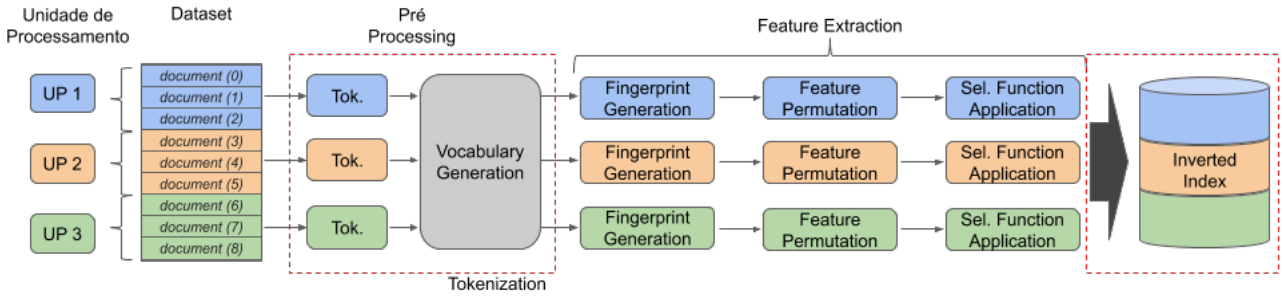


Figura 4.2 – Estratégia de paralelismo da tarefa de indexação com métodos LSH. Estendida da seção 4.1

Com o intuito de representar o gargalo ocasionado pelo problema da dependência dos dados, manifestado nos itens (*ii*) e (*iii*), esse trabalho propõe uma extensão de estratégia, apresentada na figura 4.2, adaptada do modelo ideal de paralelismo nos dados. Assim, a figura 4.2 divide o passo de tokenização nas atividades que não possuem dependência nos dados e, portanto, podem ser paralelizadas, e em seguida pela sub-tarefa de geração do vocabulário, onde todas as unidades de processamento

precisam trocar dados, sendo o gargalo da execução da tarefa de indexação em paralelo.

4.2 O Paralelismo na tarefa de Recuperação de Informação

A tarefa de recuperação de Informação, do workflow de identificação de plágio externo, conforme visto na seção 2.4, é responsável por, dado um documento suspeito, retornar os documentos fonte mais prováveis de terem sido plagiados. A tarefa de recuperação de informação é dividida em duas etapas (figura 4.3): extração de consulta e recuperação de documentos fonte.

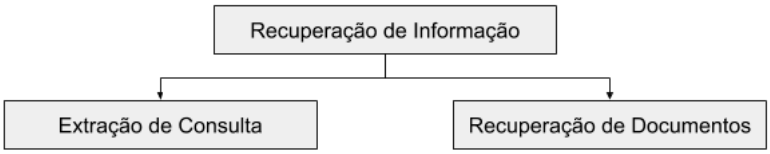


Figura 4.3 – Etapas da tarefa de Recuperação de Informação.

As sub-seções a seguir formalizam propostas de paralelismo para as etapas de extração de consulta e recuperação de documentos fonte da tarefa de recuperação de informação.

4.2.1 O Paralelismo na Extração de Consultas

A extração de consulta da tarefa de recuperação de informação, assim como na tarefa de indexação, também contempla os passos de tokenização, geração de fingerprints, permutação das features e aplicação da função de seleção. Entretanto, apesar de pontos em comum, a etapa de extração de consulta utiliza o vocabulário gerado pela tarefa de indexação, e portanto, não há o problema da dependência dos dados na etapa de extração de consultas. Além disso, diferente da tarefa de indexação, que ao final produz um índice invertido orientado as assinaturas, a extração de consultas apenas gera e armazena as assinaturas em um vetor, a ser usado como entrada de dados da etapa da tarefa de recuperação de informação.

Dessa forma, no contexto da estratégia de paralelismo nos dados, esse trabalho formaliza, na figura 4.4, uma proposta de estratégia para a execução paralela da etapa de extração de consulta. Devido a ausência do problema da dependência dos dados na etapa de extração consulta, essa proposta é totalmente aderente a estratégia de paralelismo nos dados, não havendo nenhum ponto de gargalo, ou seja, uma unidade de processamento é capaz por executar por completo todas as

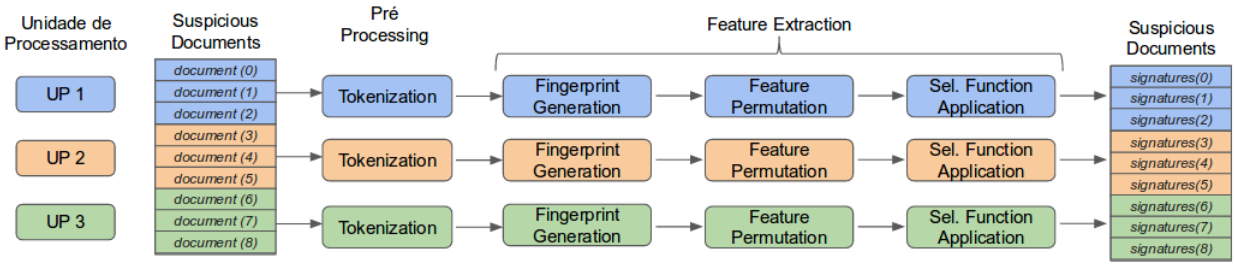


Figura 4.4 – Estratégia para execução paralela da etapa de extração de consultas segundo a sequencia de passos para métodos LSH baseados em permutações. Adaptado de DUARTE *et al.* (2017)

instruções em uma parte dos dados, sem se preocupar com a execução das outras partes em outras unidades.

4.2.2 O Paralelismo na Recuperação de Documentos Fonte

A etapa de recuperação de documentos fonte da tarefa de recuperação de informação é responsável receber um documento suspeito, representado por um vetor de assinaturas, e realizar a operação de pesquisa no índice invertido para retornar os documentos mais prováveis de terem sido plagiados.

Do ponto de vista do problema da dependência dos dados, a etapa de recuperação de documentos fonte depende dos resultados da tarefa de indexação e da etapa de extração de consulta, portanto necessita que a tarefa de indexação e a etapa de extração de consulta seja executada. Contudo, independentemente de a recuperação de documentos fonte depender do produto resultantes das tarefas anteriores, no contexto da estratégia de paralelismo nos dados, há oportunidades de paralelismo da execução da etapa de recuperação de documentos fonte. Duas abordagens para a implementação da estratégia de paralelismo nos dados foram identificadas:

1. **Paralelismo na consulta**: é a capacidade de receber múltipla consultas simultaneamente e delegar cada consulta para uma unidade de processamento diferente, onde essa unidade de processamento precisa ter acesso a todo a estrutura do índice invertido.

2. **Paralelismo da pesquisa ao índice**: é a capacidade de varrer o índice invertido simultaneamente, em pequenas partes do índice, através de múltiplas unidades de processamento.

Entretanto, devido o objetivo desse trabalho estar relacionado a capacidade de reduzir o tempo da tarefa de identificação de plágio, em razão da escalabilidade de recursos computacionais, apenas a abordagem de paralelismo da pesquisa ao índice torna-se viável, visto que na abordagem de paralelismo na consulta, o aumento dos

recursos computacionais não reduz o tempo de uma consulta. Além disso, a tática de paralelismo da pesquisa ao índice leva vantagem, do ponto de vista da resiliência, pois em caso de falha de uma unidade de processamento, uma outra unidade de processamento pode assumir a parte da tarefa que falhou, executando apenas em um subconjunto dos dados.

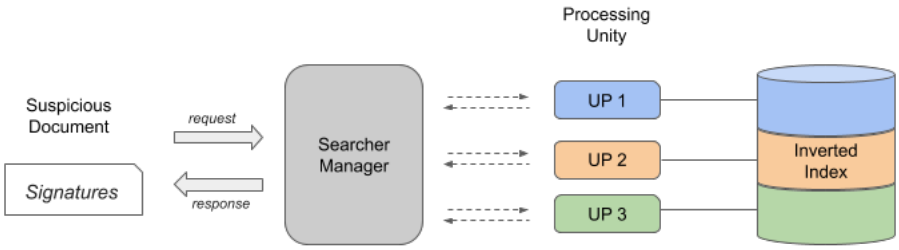


Figura 4.5 – Estratégia para execução paralela da etapa de recuperação de documentos fonte.

A figura 4.5 formaliza a estratégia proposta para a execução paralela da etapa de recuperação de documentos fonte com a abordagem de paralelismo da pesquisa ao índice. As assinaturas dos documentos suspeitos são submetidas ao gerenciador de busca, responsável por delegar para cada unidade de processamento a tarefa de pesquisa em uma parte do índice. Cada unidade de processamento realiza uma varredura em sua parte do índice, de forma que quanto maior for o número de unidades de processamento, menor é o tamanho das partes. Logo, ao aumentar o número de unidades de processamento o custo da varredura por UP é reduzido e, após todas as UPs finalizarem a pesquisa nas partes do índice, o resultado é combinado para retornar os documentos fontes mais prováveis de serem plagiados.

4.3 A exploração do Paralelismo em Diferentes Níveis

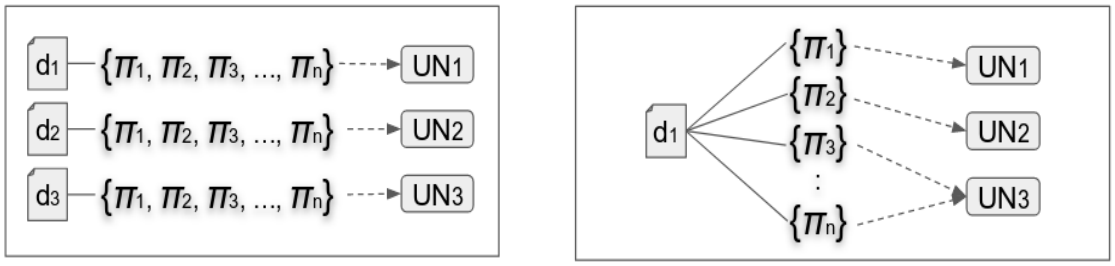
A estratégia proposta acima, para realizar em paralelo as tarefas de indexação e recuperação de documentos fonte, foram baseadas na estratégia de paralelismo nos dados. De forma geral, cada unidade de processamento lidou com um subconjunto da coleção de documentos, para permitir a execução simultânea dos passos do pipeline LSH, em diferentes documentos. Logo, do ponto de vista do nível do paralelismo nos dados, a estratégia, acima proposta, explora o paralelismo no nível de documento (PnD).

Entretanto, especificamente na atividade de extração das features do workflow de identificação de plágio externo (seção 2.4), presente na tarefa de indexação e na

etapa de extração de consulta e composta pelos passos de permutação das features e aplicação da função de seleção do pipeline LSH (seção 2.6), há a oportunidade de realizar um paralelismo mais granular, a nível de permutação.

Para cada documento, os passos de permutação das features e aplicação da função de seleção, também chamados de **estágio de permutação**, realizam N operações sequenciais de permutação, realizadas através de um loop. Assim, no paralelismo nos dados a nível de documento, múltiplas unidades de processamento processam sequencialmente o estágio de permutação, de múltiplos documentos.

Nesse contexto, esse trabalho explora a oportunidade de, especificamente no estágio de permutação, fragmentar a tarefa a nível de permutação (PnD). A figura 4.6 ilustra a diferença entre o paralelismo nos documentos (PnD) e o paralelismo na permutação (PnP), onde π representa a tarefa de permutar. É possível notar no item no 4.6a que a tarefa de permutar um documento é realizada sequencialmente, enquanto que no item 4.6b a tarefa de permutar é fragmentada em sub-tarefas menores, em nível de permutação (PnP).



(a) Paralelismo nos documentos (PnD). (b) Paralelismo nas permutações (PnP).

Figura 4.6 – Permutação em diferentes níveis de paralelismo nos dados.

Uma das vantagens de trabalhar com tarefas menores é o aumento da resiliência, tendo em vista que no caso de falha uma unidade de processamento, a unidade de processamento que assumir a responsabilidade da executar a tarefa vai realizar um trabalho menor. Em contrapartida, o ato de fragmentar uma tarefa, além do custo de dividir, pode acarretar no aumento do custo para trocar dados e combinar resultados entre as unidades de processamento.

Capítulo 5

Desafios para Escalar as Estratégias de Paralelismo com o Apache Spark

Esse capítulo tem o objetivo estudar os desafios para escalar as estratégias de paralelismo nos documentos (PnD) e paralelismo nas permutações (PnP) no sistema de computação distribuída Apache Spark. A escalabilidade das estratégias de paralelismo visa tornar a tarefa de recuperação heurística com métodos LSH viável para cenários de massivas coleções de documentos.

O presente capítulo está organizado 4 partes: inicialmente, as dificuldades para executar o pipeline LSH em ambiente distribuído são caracterizadas. Em seguida, adaptações nos passos do pipeline LSH são propostas com o intuito otimizar sua execução no Spark. Já na terceira parte, é apresentado a condução de um estudo para verificar se as adaptações propostas impactam na capacidade dos métodos LSH em estimar a similaridade. Por fim, um estudo para observar as melhores configurações do Spark no problema da recuperação heurística com as estratégias PnD e PnP.

5.1 O Alto Custo de Permutar em Ambiente Distribuído

As estratégias de paralelismo nos documentos (PnD) e na permutação (PnP) foram elaboradas sob a perspectiva da estratégia do paralelismo nos dados. Com o intuito de permitir a recuperação heurística em larga escala, esse trabalho se propõe a implementar as estratégias PnD e PnP em um sistema de computação de alto desempenho, nesse caso o Apache Spark.

Conforme visto em 2.6, a etapa de permutação das features do workflow LSH tem o objetivo de gerar, para cada permutação π_n em N permutações, uma nova sequência $L_n = \pi_n(L_i)$ resultante da aplicação de uma simples função $h(.)$ sobre vetor de fingerprints L_i, correspondentes a identificadores numéricos dos termos de um documento $d_i \in D$, onde D representa a coleção de documentos.

A função $h(.)$ da permutação π_n é responsável por gerar o mesmo valor, em todos os documentos $d_i \in D$, para uma mesma fingerprint g_j , que compõe o vetor L_i e corresponde a um determinado termo t_j do vocabulário da coleção T. Cada função $h(.)$ da permutação π_n de N permutações é composta por um vetor v_n do tamanho do vocabulário da coleção T, gerado a partir do embaralhamento das fingerprints g_j em L.

A título de ilustração, suponha que uma determinada coleção de documentos tenha o seguinte vocabulário $T = [p_1, p_2, p_3, p_4, p_5]$, onde p represente uma palavra qualquer; então o vetor de fingerprints $L = [1, 2, 3, 4, 5]$ é gerado a partir do vocabulário T; assim a permutação π_n gera o vetor $v_n = [4, 1, 5, 3, 2]$ decorrente do embaralhamento aleatório de L.

Com a finalidade de otimizar a execução da etapa de permutação das features, do ponto de vista de implementação, por serem imutáveis, os vetores $v_n \in V$ são pré-computados e utilizados para permutar todos os documentos $d_i \in D$. A estratégia de pré-computar V reduz para $\Delta_t = v_n | n \in N$ o custo linear $\Delta_t = (v_n | n \in N)^D$ de computar V , em tempo de execução, para cada $d_i \in D$.

Entretanto, em troca do custo linear para re-computar V, em tempo de execução, para cada $d_i \in D$, a estratégia de pré-computar V eleva o consumo de memória, tendo em vista que V precisa ser armazenado e acessado várias vezes, durante a tarefa de permutação. A tabela 5.1 apresenta o consumo aproximado para armazenar V em memória, variando o número de N permutações. Para tanto, foi considerado um vocabulário de 420.000 palavras e o tamanho aproximado de 3,36MB para armazenar um vetor de inteiros em python (numpy).

Tabela 5.1 – Consumo aproximado (em megabytes) para armazenar V em memória.

Permutação	**Consumo Memória**
100	336 mb
200	672 mb
400	1.344 mb
800	2.688 mb

Apesar de a estratégia de pré-computar V evitar o custo linear em $d_i \in D$. Devido a característica do Spark de não compartilhar recursos (*shared-nothing*), onde cada nó trabalhador realiza uma tarefa apenas com base nos dados de entrada. Para

realizar a tarefa de permutar um documento d_i com a estratégia de pré-computar os vetores $v_n \in V$, juntamente com as fingerprints g_i de $d_i \in D$, os vetores de permutação V precisam ser replicados em cada tupla de d_i e, então, ser mapeado na entrada de dados da função de permutação. Para tanto, a seguir será discutido o problema, em relação ao Spark, para realizar a tarefa de replicar V em cada tupla de documentos d_i.

5.1.1 O Problema de Replicar os Vetores de Permutação

Para utilizar os vetores V na tarefa de permutação, pré-computados com o intuito de reduzir o custo de re-computar para cada $d_i \in D$, é necessário que os vetores sejam mapeados para a função de permutação, juntamente com cada documento d_i.

Todavia, replicar V em cada $d_i \in D$ torna-se proibitivo devido ao aumento linear do consumo de memória, igual a $|V|^D$, de acordo com o tamanho de D e tendo em vista que V possui um alto custo de armazenamento. Para tanto, em detrimento ao respectivo problema, esse trabalho propõe duas novas estratégias que visam reduzir o custo para replicar V e viabilizar a tarefa de permutação das features em ambiente distribuído, através da redução de V, a serem apresentadas na próxima seção.

5.2 Lidando com o Consumo de Memória na Permutação das Features

Como visto anteriormente, re-computar em tempo de execução, para cada documento $d_i \in D$, os vetores $v_n \in V$ é uma opção ineficaz, tendo em vista o inerente custo linear da operação. Por outro lado, devido ao alto custo para trafegar e armazenar V em memória, torna-se proibitivo replicar Vpara cada documento $d_i \in D$.

Para reduzir o alto custo de replicar na RDD de $d_i \in D$ cada vetor de permutação $v_n \in V$, esse trabalho propõe duas novas estratégias de permutação, que visam reduzir o número de N vetores $v_n \in V$, para apenas um vetor v_0, reduzindo assim o custo, até então, linear em N permutações, para o custo fixo de armazenar e replicar apenas um único vetor v_0. A redução de N vetores para um único vetor, lida com o custo adicional gerar para N permutações, N distribuições distintas computadas em tempo de execução da tarefa atividade de permutação, a partir de v_0.

A seguir são apresentadas duas estratégias hibridas, isto é, que combinam as abordagem de pré-computar com a de re-computar em tempo de execução: **Re-Indexar o Vetor (RIV)** e **Expandir o Vetor no Final (EVF)**, que permitem realizar N permutações a partir de um único vetor v_0.

5.2.1 Estratégia de Re-Indexar o Vetor (RIV)

A estratégia de re-indexar o vetor (RIV) visa obter N diferentes distribuições do vetor aleatório de permutação v_0, através da realização de uma operação re-indexação do vetor, durante a execução da tarefa de permutação. Dessa forma, essa estratégia reduz o consumo de memória ao replicar V, em troca do custo, em tempo de execução, de re-indexar v_0 para N permutações.

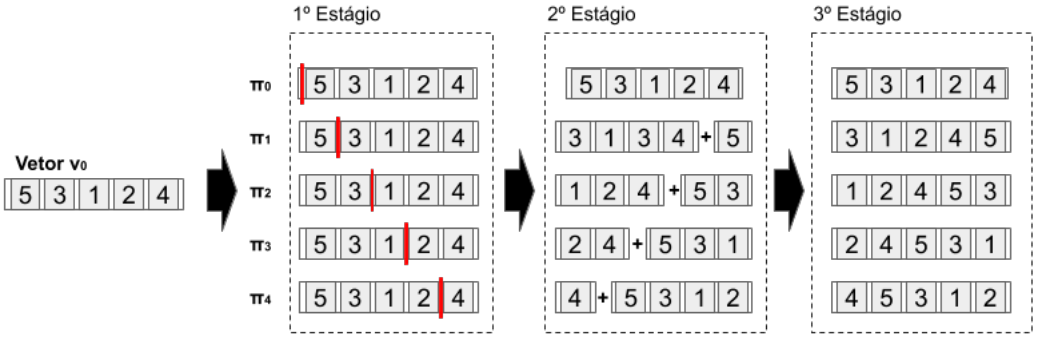


Figura 5.1 – Estratégia de re-indexar o vetor de permutação.

A figura 5.1 ilustra o processo de geração do vetor de permutação com a estratégia RIV, para uma coleção com um vocabulário de 5 palavras. Dado o vetor aleatório v_0, o ponto de divisão do vetor é definido de acordo com a o identificado da permutação. No segundo estágio, o vetor v_0 é dividido em duas partes, que em seguida, são invertidas; e no estágio final as partes invertidas são concatenadas formando a versão do vetor a ser utilizado na permutação. A obtenção de N distribuições do vetor v_0 com a estratégia RIV é demonstrada no algorítimo 1.

Algorithm 1: Obtenção de N distribuições do vetor v_0 com a estratégia RIV

```
1 Input:N = Número de operações de permutações, v_0 = Vetor único de
    permutação gerado aleatoriamente e g_i = Fingerprints do documento d_i a
    ser permutado ;
2 for n in range(N) do
3     vetor = v_0[n:] + v_0[:n];
4     permutar(g_i, vetor);
```

5.2.2 Estratégia de Expandir o Vetor no Final (EVF)

A estratégia de expandir o vetor no final (EVF), assim como a estratégia RIV, visa obter N diferentes distribuições do vetor aleatórios v_0, para N permutações. Para tanto, diferente da abordagem tradicional, onde cada vetor V_n possui o tamanho do vocabulário $|T|$ da coleção, a proposta é gerar o vetor aleatório de V_0 com o

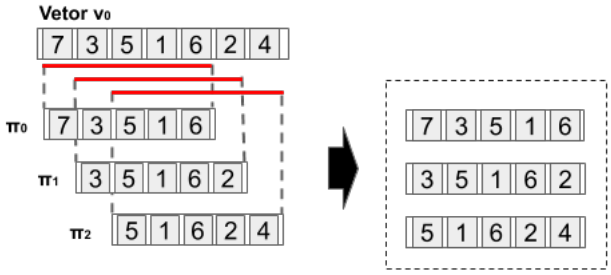


Figura 5.2 – Estratégia de expandir o vetor de permutação no final.

tamanho de $(|T| + |N| - 1)$, ou seja, o tamanho de V_0 é igual ao número de palavras do vocabulário T adicionado do número de N permutações.

A figura 5.2 ilustra o processo de geração de três distribuições de v_0 com a estratégia RIV, para uma coleção com um vocabulário de 5 palavras e 3 permutações. Dado o vetor aleatório v_0, de acordo com a permutação a ser executada, o vetor de permutação a ser utilizado é resultante de um sub-produto de v_0, extraído a partir do identificador da permutação. A obtenção de N distribuições do vetor v_0 com a estratégia EVF é demonstrada no algorítimo 2.

Algorithm 2: Obtenção de N distribuições do vetor v_0 com a estratégia

```
1 N = Número de operações de permutações;
2 v_0 = Vetor único de permutação gerado aleatoriamente ;
3 T_size = Número de Termos no Vocabulário;
4 g_i = Fingerprints do documento d_i a ser permutado ;
5 for n in range(N) do
6 |   vetor = v_0[n : T_size] ;
7 |   permutar(g_i, vetor);
```

Na seção seguinte apresenta um estudo para verificar se as novas estratégias de permutação impactam na capacidade de preservação da similaridade dos métodos LSH.

5.3 Estudo sobre a Influência das Estratégias de Permutação RIV e EVF em Estimar Jaccard

A presente seção apresenta a condução de um estudo para verificar o impacto das estratégias RIV e EVF na atividade de estimar a similaridade. Para tanto, as estratégias RIV e EVF serão avaliadas com o intuito de: (i) observar se há alguma perda da capacidade dos métodos LSH em estimar entre pares de documentos; e (ii) verificar a redução do tempo de permutação.

Para verificar se a capacidade de preservação de um método LSH sofre alteração com as estratégias RIV e EVF, foi realizado um experimento conhecido como: Estimar a Similaridade de Jaccard par-a-par (ESJP). No ESJP, inicialmente, as similaridades entre os pares de documentos são computadas, utilizando todos os termos da matriz termo-documento. Em seguida, o processo de extração das features é executado e produz a representação de assinaturas geradas, assim, a similaridade entres os pares de documentos são computadas com base nas assinaturas. Por fim, a capacidade de preservação de similaridade pode ser medida através da métricas (apresentadas em 6.3): RMSE, que mede a magnitude média do erro e é influenciada pela variância dos erros, atribuindo pesos maiores aos maiores erros; e MAE, que é mais voltada para medir o comportamento médio do erro.

Esse estudo foi realizado utilizando o método LSH Min-Hash para estimar a similaridade dos pares de documentos do corpus PSA (apresentado na seção 6.2). Foram avaliadas as combinações de 100, 200, 400 e 800 permutações. Para obter uma coesão estatística, esse estudo foi repedido 1000 vezes.

Tabela 5.2 – Tempo médio de permutação (em segundos) das abordagens de pré-computar (PCV) e de re-computar os vetores de permutação (CVET).

Permutações	**CVET**	**PCV**
100	2.575	0.080
200	5.230	0.238
400	10.694	0.557
800	23.829	1.216

A tabela 5.2 apresenta o tempo gasto para permutar um documento com a estratégia tradicional de permutação de DUARTE (2017). Para tanto, a tabela compara a abordagem de pré-computar V (PCV) e com a de re-computar V em tempo de execução (CVET), para cada documento $d_i \in D$. É possível notar que apesar da abordagem PCV, que pré-computa V, segundo a estratégia de Duarte, ser proibitiva em ambiente distribuído, devido ao alto custo para replicar V, o resultado demonstra uma grande redução do tempo de permutação em pré-computar V, em detrimento da abordagem CVET.

A ineficiência da estratégia de re-gerar os vetores de permutação para cada documento (CVET), juntamente com a inviabilidade de utilizar a abordagem PCV, devido ao custo proibitivo em ambientes distribuídos, contextualizam a relevância da proposta das estratégias RIV e EVF. Portanto, a seguir serão avaliadas as estratégias RIV e EVF, as quais também lidam com a abordagem de pré-computar o vetor de permutação, e ainda assim, são viáveis de serem executas em ambiente distribuído. Ademais, abordagem de CVET será utilizada como baseline, em vista é possível ser implementada no Spark, ainda que de maneira ineficiente.

Tabela 5.3 – RMSE e MAE das estratégias de permutação.

Permutações	**RMSE**			**MAE**		
	CVET	**RIV**	**EVF**	**CVET**	**RIV**	**EVF**
100	0.0213	0.0212	0.0214	0.1216	0.1219	0.1213
200	0.0209	0.0206	0.0208	0.1225	0.1217	0.1214
400	0.0206	0.0204	0.0205	0.1225	0.1219	0.1218
800	0.0204	0.0202	0.0202	0.1221	0.1216	0.1213

A tabela 5.3 apresenta o RMSE, mais sensível a variância no comportamento, e o MAE, mais sensível ao comportamento médio, obtido nas estratégias CVET, RIV e EVF. Em geral, tanto em termos de RMSE e MAE, todas as três estratégias apresentam um comportamento similar, na mesma ordem de magnitude. É possível notar a tendência de redução da variação do erro (RMSE) ao aumentar o número de permutações.

Tabela 5.4 – Tempo médio de permutação (segundos) das estratégias de permutação.

Permutações	**CVET**	**RIV**	**EVF**
100	2.575	0.976	0.091
200	5.230	2.016	0.184
400	10.694	3.956	0.378
800	23.829	7.923	0.791

A tabela 5.4 apresenta o tempo médio de permutação obtido nas estratégias CVET, RIV e EVF. Considerando que a estratégia CVET é o baseline, a estratégia proposta RIV obteve uma redução do tempo de permutação entre 2.6 e 3 vezes, em comparação ao baseline. Por sua vez, a estratégia EVF obteve um desempenho ainda melhor que a estratégia RIV, variando em uma redução entre 10.01 e 10.95 vezes no tempo de permutação, em comparação ao RIV e entre 26.02 e 32.85 vezes, em comparação ao baseline.

Portanto, conforme visto na tabela 5.3, as estratégias RIV e EVF não geram impacto na capacidade dos métodos LSH em estimar a similaridade. Ainda assim, conforme visto na tabela 5.4, as estratégias RIV e EVF foram capazes de reduzir o tempo gasto na tarefa de permutação, sendo que, em especial, a estratégia EVF obteve o maior nível de redução do tempo de permutação. Logo, esse trabalho implementa a estratégia de permutação EVF para viabilizar a execução da tarefa em ambiente distribuído.

5.4 Estudo sobre o Impacto das Partições do Spark

O Spark realiza o gerenciamento de dados através de partições, que são *chunks* lógicos de dados, onde cada *chunk* armazena uma parte do conjunto de dados. As partições ajudam no paralelismo e na distribuição da execução de uma tarefa com o objetivo de minimizar o troca de dados entre os nó(s) trabalhadores.

Assim, cada nó trabalhador recebe do nó gerenciador uma ou mais partições, onde, inicialmente, os nó(s) trabalhadores realizam tarefas locais em paralelo e, então, os resultados são coletados e combinados pelo nó gerenciador. Dessa forma, o número de partições possui impacto direto no desempenho de um programa a ser executado no Spark. A melhor configuração do número de partições pode variar de acordo com o tipo de problema, o tamanho da coleção, o número de iterações e outros fatores. Nesse contexto, torna-se necessário a condução de uma avaliação experimental visando encontrar a melhor configuração do número de partições para o problema a ser abordado neste trabalho.

Para definir o número de partições a serem avaliadas, é importante considerar que cada CPU dos nó(s) trabalhadores necessita de ao menos uma partição, para não haver sub-utilização de recursos. Além disso, devido ao fato de a quantidade de CPU ser variável, em função do número de nó(s) trabalhadores do cluster, o melhor número de partições é dado pela variável $fator$, que representa o número de partições por CPU (Eq.5.1). Dessa forma, o $fator = 1$ representa o menor número de partições, viável para que não haja sub-utilização de recursos. Portanto, a condução da avaliação da melhor configuração de partições visa encontrar o melhor $fator$, isto é, a melhor quantidade de partições por CPU.

$$fator = (\text{workers} \times \text{cores}) \tag{5.1}$$

Esse estudo foi realizado utilizando o método LSH Min-Hash para estimar a similaridade dos pares de documentos do corpus PSA (apresentado na seção 6.2). Foram avaliadas as combinações de 100, 200, 400 e 800 permutações. Para obter uma coesão estatística, esse estudo foi repedido 1000 vezes.

Esse estudo foi realizado utilizando a estratégia de paralelismo nos documentos (PnD) considerando as etapas de indexação, recuperação de informação, da tarefa de recuperação heurística do plágio externo. Ademais, a tabela 5.5 apresenta todas a configurações usadas nesse estudo, sendo elas: 3 nó(s) trabalhadores no cluster; 8 CPU por nó trabalhador; variações em 500, 1000, 2000, 4000, 8000 e 15966 documentos fonte, extraídos da coleção de documentos PAN11 (seção 6.2); variações em 48, 96, 192, 384, 768 e 1536 assinaturas geradas usando o método Min-Max-Hash; e

Parâmetro	**Configurações**
Nó(s) trabalhadores	3
Quantidade de documentos fonte	{500, 1000, 2000, 4000, 8000, 15966}
Quantidade de assinaturas	{48, 96, 192, 384, 768, 1536}
Método LSH	Min-Max-Hashing
Estratégia	Paralelismo nos Documentos (PnD)
fator	{1, 2, 4}

Tabela 5.5 – Parâmetros de Execução do Estudo sobre o Impacto das Partições do Spark.

variação da variável *fator* em 1, 2 e 4, representando a quantidade de partições por CPU.

O impacto das partições do Spark na etapa de indexação é abordado na figura 5.3, que apresenta o tempo gasto na tarefa de indexar diferentes cargas de documentos, sob o custo da geração de várias assinaturas. A figura 5.3 aponta um comportamento próximo de serem sobrepostos, para o fator 1, 2 e 4; nas diversas combinações, de modo que não apresentou alterações significativa no tempo gasto na indexação.

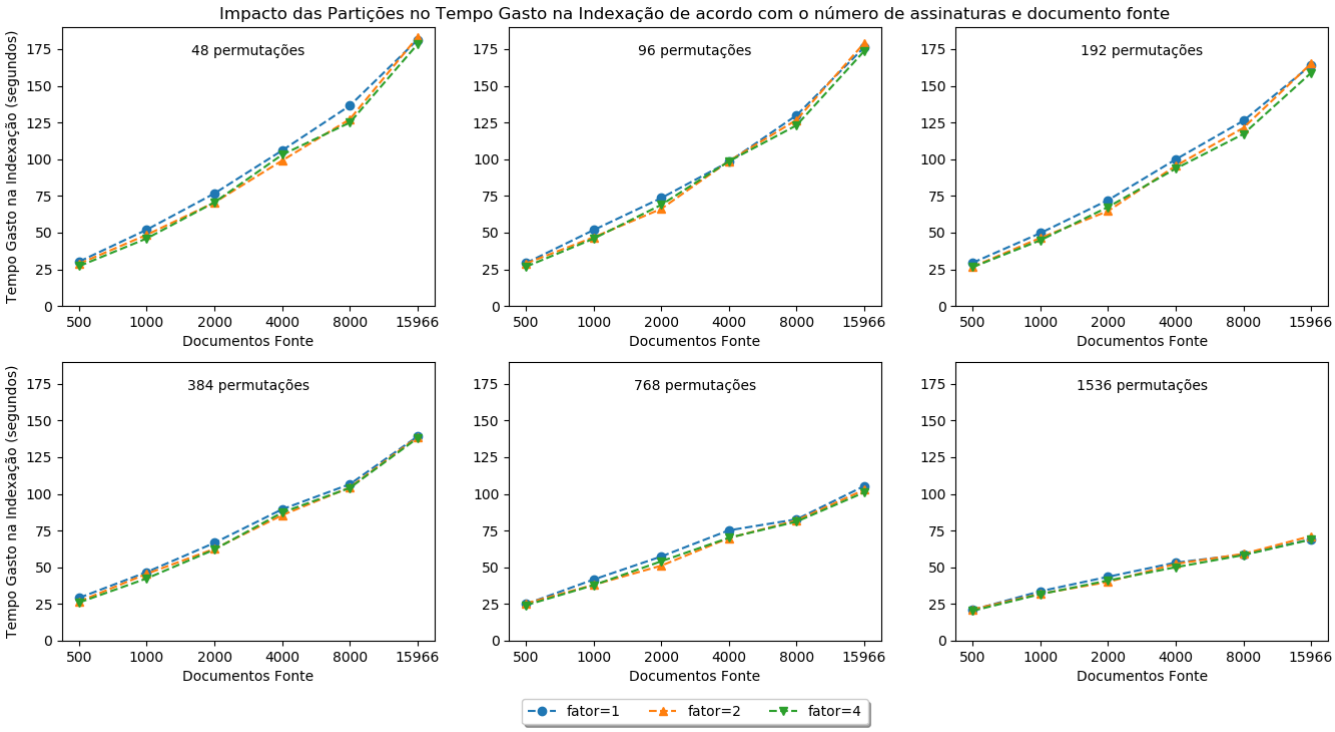


Figura 5.3 – Impacto das Partições do Spark na Indexação

Ainda que o utilizar mais partições não tenha gerado um impacto significativo no desempenho da indexação, lidar com um número maior de partições tende a aumentar a tolerância a falha, visto que a quantidade de documentos a serem processados

diminui, assim, caso ocorra a falha de um nó trabalhador, um outro nó assumiria uma tarefa menor, ou seja, com menos documentos.

O impacto das partições do Spark na etapa de Recuperação da Informação é abordado na figura 5.4, que apresenta a variação do tempo médio para pesquisar um documento suspeito d_{Susp} no índice. Em contraste com os resultados obtidos na indexação, é possível notar que na recuperação de documentos fonte (figura 5.4), o número de partições por núcleo de CPU, registrou um impacto significativo, sendo que em todos os caso, o tempo de recuperação da informação, registrou um aumento de no mínimo 50% no tempo de recuperação.

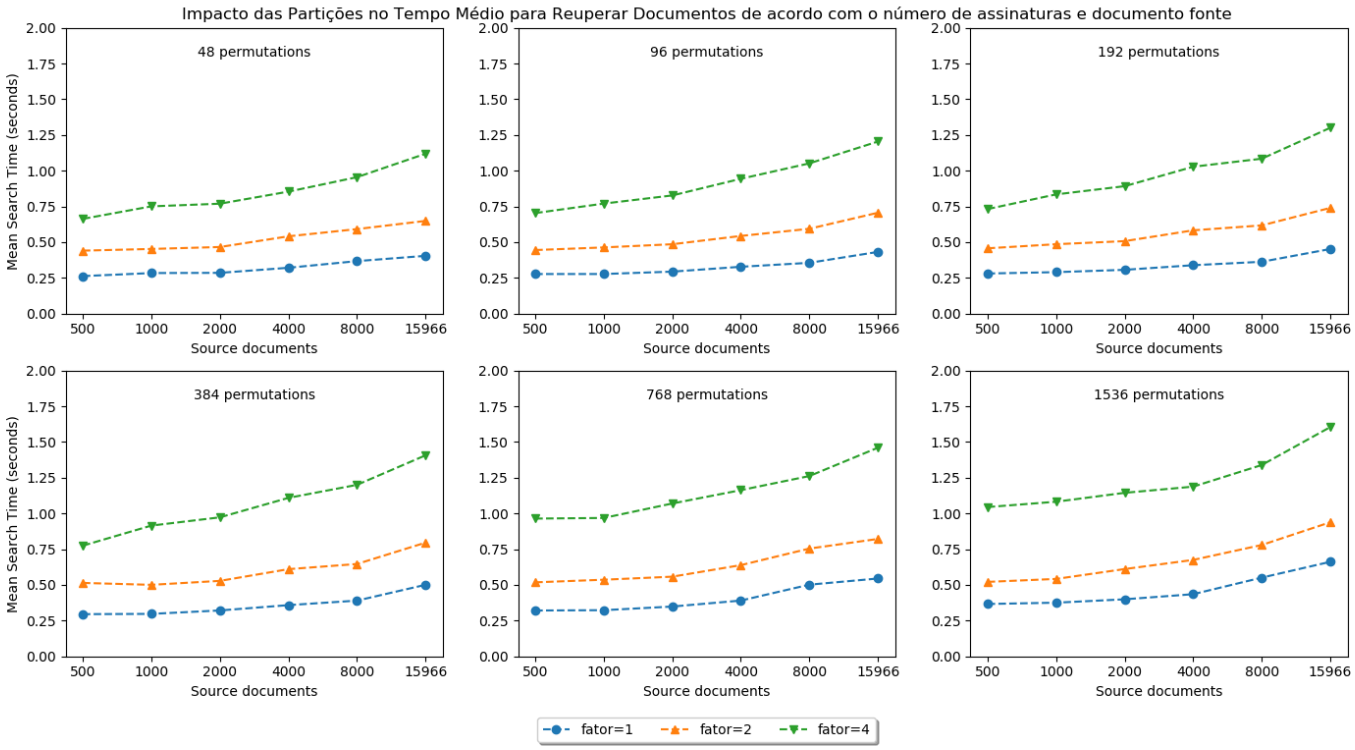


Figura 5.4 – Impacto das Partições do Spark na Recuperação da Informação

Portanto, apesar dos benefícios de utilizar um número maior de partições, com a perda de desempenho na tarefa de recuperação da informação, os experimentos do capítulo seguinte 5.4 serão configurados com o $fator = 1$, isto é o Spark será configurado com uma partição por núcleo de CPU disponível. Ademais, ainda que exista a possibilidade de utilizar diferentes configurações de partições, para as diferentes etapas, através de uma técnica de re-particionamento do índice gerado na indexação, por não ser o objeto deste trabalho, essa opção abordada na seção de trabalhos futuros.

Capítulo 6

Avaliação Experimental

A avaliação experimental realizada nesse trabalho tem o objetivo de verificar o aumento da eficiência das estratégias de Paralelismo nos Documentos(PnD) e Paralelismo nas Permutações(PnP) em acelerar a tarefa de recuperação heurística, e a manutenção da eficácia em conservar a capacidade de retornar os documentos que efetivamente foram plagiados.

Esse capitulo está organizado em quatro seções. A seção 6.1 apresenta o ambiente de execução e a seção 6.2, a descrição das coleções de dados utilizadas nos experimentos. Na seção 6.3, as métricas utilizadas nos experimentos e estudos são apresentadas. Já a seção 6.4, apresenta os experimentos de Recuperação Heurística do Plágio Externo (RHPE), onde, inicialmente, é avaliado a capacidade de preservação da eficácia dos métodos LSH nas estratégias PnD e PnP; em seguida, a eficiência das estratégias PnD e PnP é avaliada, tanto no âmbito da indexação, bem como na recuperação de documentos fonte, sob os vários aspectos, tais como: o aumento da capacidade computacional (escalabilidade dos recursos) e o aumento do número de assinaturas, em diferentes métodos LSH. Por fim, a coleção de documentos é aumentada para verificar a eficiência alcançada, em cenários de grande escala, em comparação ao Min-Hash@MLlib (baseline).

6.1 O Ambiente de Execução dos Experimentos

Os experimentos foram realizados em um cluster Spark, implantado no Departamento de Ciência da Computação (DCC) da Universidade Federal Rural do Rio de Janeiro (UFRRJ), com o gerenciador Standalone (seção3.2).

O Driver Program e o Cluster Manager foram implantados em um único nó. Além disso, foram instanciados quatro nó(s) trabalhadores. Cada nó do cluster possui 16GB de memória principal e CPU de 12 cores. Entretanto, com o objetivo de minimizar o impacto de possíveis execuções de daemons dos sistemas operacionais, cada nó trabalhador foi limitado a utilizar apenas 12GB de memória principal

e 8 cores de CPU. É importante ressaltar que na condução dos experimentos de escalabilidade, apenas os nó(s) trabalhadores são considerados.

Tabela 6.1 – Configuração dos Componentes do Ambiente de Execução dos Experimentos.

Componente	**Versão**
Apache Spark	Spark Release 2.4.0
Sistema Operacional	Ubuntu Server 17.10 (Artful Aardvark)
Python	3.6
Anaconda	3.5.1
Java jdk	1.8

Conforme apresentado na tabela 6.1, o cluster foi implementado no sistema operacional Ubuntu Server. As estratégias PnD e PnP foram implementadas na linguagem de programação Python. A versão do Spark utilizada foi a 2.4.0 realease, que possui como pré-requisito a instalação do Java 1.8. A linguagem de programação utilizada foi o Python versão 3.6, distribuído pelo gerenciador de pacotes, Anaconda versão 3.5.1, que também fornece todas as dependências necessárias à implementação desse trabalho.

6.2 Coleção de Dados

Esse trabalho fez uso das coleções de documentos *PAN plagiarism 2011 (PAN-11)* e *Plagiarised Short Answers (PSA)*, criadas para permitir a avaliação automática da eficácia dos algoritmos de detecção de plágio.

A coleção PSA (CLOUGH & STEVENSON, 2011b) simula plágio através de respostas baseadas em revisões ou cópias aproximadas de documentos fonte, indagadas a 19 participantes. Esta é composta por 100 documentos, sendo 5 documentos fonte, retirados da Wikipédia, e 95 documentos com respostas escritas pelos participantes do experimento. A coleção PSA foi utilizada no estudo sobre a capacidade de preservar a similaridade das estratégias de permutação RIV e EVF (5.3).

A coleção PAN-11 (POTTHAST *et al.*, 2011a) é composta por 26.939 documentos, 61.064 fragmentos de texto classificados como plágio, e 686.668.842 palavras. Na coleção PAN-11, 50% dos documentos são classificados como documentos originais, os quais podem ser plagiados por outros documentos; 25% dos documentos são falsos positivos, ou seja, documentos que são suspeitos e contudo não contém casos de plágio; e o outros 25% são os verdadeiros positivos, documentos suspeitos que contem plágio, os quais os algoritmos devem identificar. Entretanto, apenas documentos em inglês foram considerados nos experimentos; especificamente para composição da base de documentos suspeitos, apenas documentos com ao menos

um fragmento de plágio foram considerados, assim, foram selecionados 15.966 documentos fonte e 4.992 documentos suspeitos com um vocabulário de 457.092 palavras distintas.

É importante ressaltar que corpus PAN-11 também foi utilizado no trabalho de Duarte (DUARTE, 2017), que experimentou a aplicação de técnicas de LSH aplicadas ao problema de detecção de plágio, permitindo assim uma comparação dos resultados obtidos neste trabalho.

6.3 Métricas de Avaliação

A presente seção apresenta as métricas *Root Mean Squared Error (RMSE)* e *Mean Absolute Error (MAE)*, que foram utilizadas no estudo sobre a capacidade de preservar a similaridade das estratégias de permutação RIV e EVF (5.3), e as métricas *Recall* e a *Vazão*, utilizadas no experimento de Recuperação Heurística do Plágio Externo (RHPE) (6.4) .

Suponha que y_{true} é o valor da similaridade de Jaccard entre dois documentos d_0 e d_1, y_{est} é o valor da similaridade de Jaccard calculado a partir do conjunto de assinaturas de d_0 e d_1. Logo, a equação 6.1 apresenta o *MAE*, que é a média da diferença absoluta entre y_{true} e y_{est}, onde todas as diferenças de valores possuem o mesmo peso na média final. Já a seção 6.2 apresenta o *RMSE*, que dá pesos maiores a erros maiores e, portanto, é mais influenciado pela variância dos valores dos erros que o *MAE*. Logo, o RMSE foi utilizado para avaliar o comportamento geral dos métodos, incluindo também os resultados fora da média, enquanto que o comportamento médio foi avaliado a partir do MAE. .

$$MAE = \frac{1}{N}\sum_{j=1}^{n}|y_{true} - y_{est}| \tag{6.1}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{j=1}^{n}(y_{true} - y_{est})^2} \tag{6.2}$$

Nos experimentos de Recuperação Heurística do Plágio Externo (RHPE), a métrica *recall* é utilizada para verificar a capacidade do motor de busca retornar os documentos que efetivamente foram plagiados. Assim, suponha que $d_q \in D_{Susp}$ é um documento suspeito a ser submetido ao motor de busca e que $F_q \sqsubset D$ é o sub-conjunto de documentos que foram plagiados por d_q. Para tanto, um motor de busca retornará um conjunto de documentos $R_q \sqsubset D$ e, portanto, o *recall*, conforme a equação 6.3, medirá a quantidade de documentos relevantes (F_q) que foram retornados em R_q. Por fim, a métrica Vazão, demonstrada na equação 6.4, mede em

um determinado intervalo de tempo Δ_t, a capacidade de indexar grandes coleções de documentos.

$$\text{Recall} = \frac{1}{|D_{Susp}|} \sum_{d_q \in D_{Susp}} \frac{|F_q \cap R_q|}{|F_q|} \tag{6.3}$$

$$\text{Vazão} = \frac{|D_{Fonte}|}{\sum\limits_{d_i \in D_{Fonte}} \Delta_t \text{ para indexar } d_i} \tag{6.4}$$

6.4 A Recuperação Heurística do Plágio Externo (RHPE) em ambiente Distribuído

A recuperação heurística é responsável por gerar uma coleção reduzida de documentos candidatos de terem sido alvos de plágio, com a finalidade de reduzir a carga de trabalho das etapas mais custosas do workflow de identificação de plágio externo.

Para tanto, a presente seção avalia o desempenho das estratégias de Paralelismo nos Documentos(PnD) e Paralelismo nas Permutações(PnP), através da análise experimental sob, inicialmente, a perspectiva de **eficácia**, ao verificar se as estratégias apresentam perda de capacidade de retornar os documentos que efetivamente foram plagiados, em comparação com os resultados obtidos no trabalho de DUARTE (2017); e, em seguida, sob a perspectiva de **eficiência**, ao verificar se o objetivo de acelerar a tarefa de Recuperação Heurística, tanto na etapa de indexação, assim como, na etapa de recuperação de documentos fontes, é alcançado. Ademais, os experimentos foram repetidos 25 vezes para se obter uma coesão estatística.

Os resultados serão comparados com baseline Min-Hash@MLlib, uma implementação do método Min-Hash em ambiente distribuído, disponível no pacote MLlib do Spark (MENG *et al.*, 2016). Entretanto, é importante ressaltar que o Min-Hash@MLlib, não segue, necessariamente, todos os passos do pipeline 2.6 para realizar a recuperação heurística utilizando métodos LSH, por exemplo: a tarefa de indexação não realiza a etapa de geração do índice invertido, dessa forma, a busca aproximada da tarefa de recuperação da informação se dá através da comparação par-a-par das assinaturas dos documentos, enquanto que as estratégias PnD e PnP geram um índice invertido. Assim, a estratégias PnD e PnP possuem um custo a mais na tarefa de indexação, contudo, evitam comparações par-a-par na recuperação de documentos fonte. Além disso, um outro fator que pode gerar impacto nos resultados são as diferentes sobrecargas de linguagem programação, pois o Min-Hash@MLlib é implemento em Java enquanto que as estratégias PnD e PnP foram implementadas em Python.

Ainda assim, a comparação com o Min-Hash@MLlib é importante no intuito de verificar o nível de maturidade das estratégias PnD e PnP, ressaltando que, ao

contrário da implementação fechada do método Min-Hash no MLlib, as estratégias PnD e PnP permitem executar qualquer método LSH baseado em permutação aleatória, tendo como exemplo os métodos Min-Max-Hash e CSA.

6.4.1 A Eficácia na Recuperação Heurística

A presente seção visa verificar se estratégias de Paralelismo nos Documentos(PnD) e Paralelismo nas Permutações(PnP) geram, para os métodos LSH, perda na capacidade de encontrar os documentos efetivamente plagiados.

É importante ressaltar que, conforme esperado, todos os métodos avaliados registraram os mesmos níveis de recall, nas estratégias PnD e PnP, haja vista que as duas estratégias utilizaram o mesmo vocabulário e as mesmas funções de permutação. Dessa forma, com o intuito e facilitar e simplificar a análise dos resultados de eficácia das estratégias, esta seção apresentará o recall de maneira consolidada para cada método LSH avaliado. Ademais, esse trabalho repete método de avaliação realizado por DUARTE (2017), onde para cada documento suspeito $d_q \in D_{susp}$, são avaliados os "recalls at pos"(recall@k), para $pos \in [1597, 3991, 7983, 11975]$, que correspondem a um percentual de retorno de, respectivamente, 10%, 25%, 50% e 75% da coleção de documentos D.

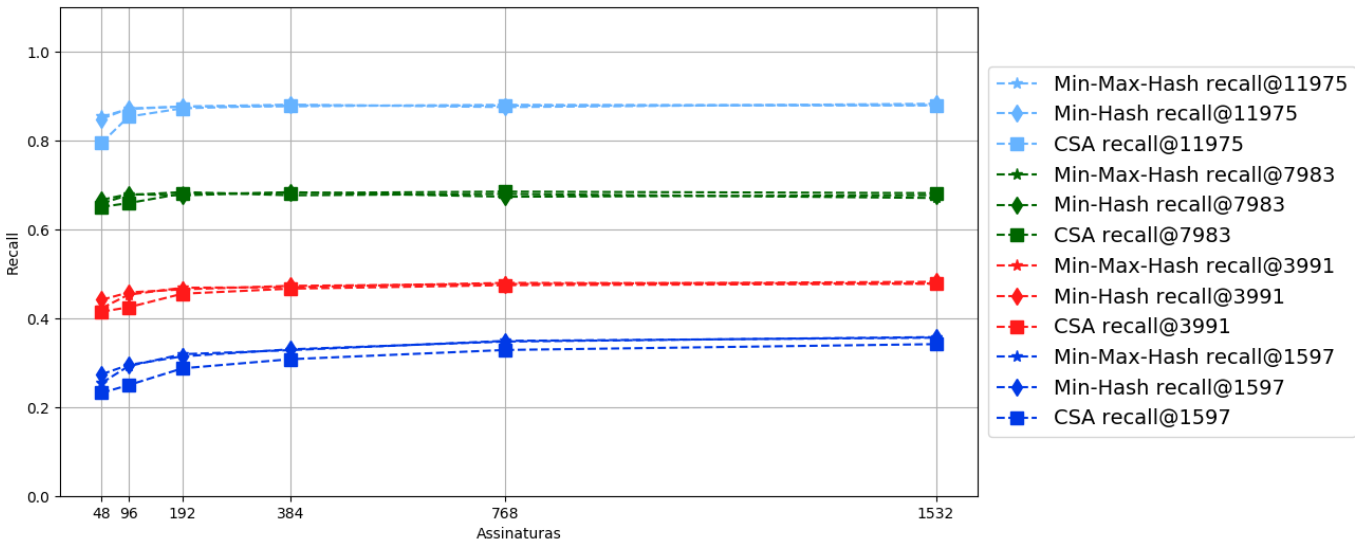


Figura 6.1 – Recall@pos consolidado das estratégias PnD e PnP na Recuperação Heurística na coleção PAN-11 plagiarism corpus.

A figura 6.1 apresenta o recall@k dos métodos Min-Hash, Min-Max-Hash e CSA, sendo que cada valor de pos possui uma cor associada, e, portanto, todos os resultados em azul claro são resultados do recall@11975, enquanto que os resultados vermelhos, azuis e verdes estão associados ao recall@7983, recall@1597 e recall@3991,

respectivamente. De forma geral, apesar de utilizar um algoritmo de tokenização diferente, o que pode resultar em um vocabulário diferente, o comportamento dos níveis de recall@k obtidos nas estratégias PnD e PnP é o mesmo do encontrado no trabalho de (DUARTE, 2017). É possível notar que os níveis de recall melhoram conforme aumenta o número de assinaturas e que a partir de 384 assinaturas há uma redução da variância e os níveis de recall convergem para um valor próximo para todos os métodos avaliados.

A figura 6.2 apresenta o recall@k dos métodos Min-Hash, consolidado nas estratégias PnD e PnP, em comparação ao Min-Hash@MLlib, também implementado no Spark. É possível notar que o Min-Hash@MLlib apresentou, no *pos* igual a 1597, uma tendência diferente do Min-Hash deste trabalho, onde o melhor resultado foi obtido com a menor quantidade de assinaturas e que conforme o número de assinaturas aumentou, houve uma perda de recall, que se estabilizou a partir de 384 assinaturas. Entretanto, tal comportamento do Min-Hash@MLlib não se repetiu para um *pos* igual a 3391, 7983 e 11975, que de maneira oposta, registrou, para todas as assinaturas, um nível de recall no máximo sobreposto e em vários casos, um nível menor de recall. Todavia, ao considerar que as diferentes implementações do método Min-Hash utilizam diferentes funções de permutações, a alternância nos níveis de recall era esperada.

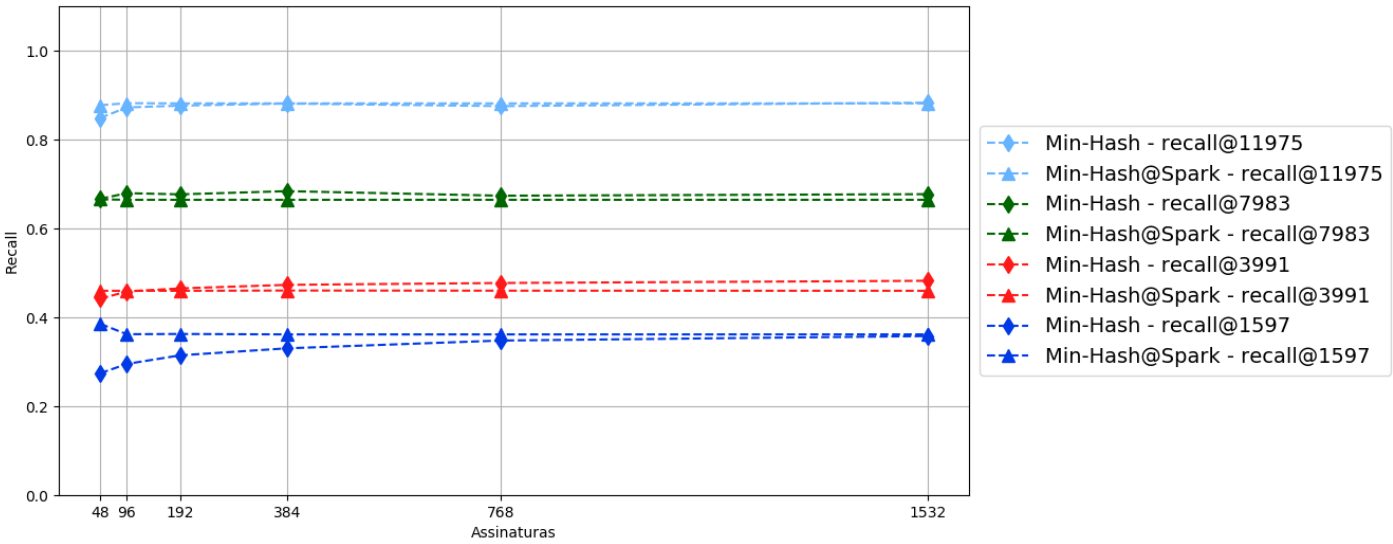


Figura 6.2 – Recall@pos do método Min-Hash consolidado na estratégias PnD e PnP em comparação ao baseline, o Min-Hash@MLlib.

Com o intuito de corroborar com a viabilidade de utilizar algoritmos LSH no problema da RHPE, formalizada no trabalho de DUARTE (2017), foi avaliada uma pequena alteração no pré-processamento. Para aumentar os níveis de recall, as

palavras que mais aparecem nos documentos da coleção, portanto, as palavras menos representativas, foram removidas do vocabulário da coleção. Cabe ressaltar que existe uma gama de possibilidades para realizar um pre-processamento que favoreça a tarefa de identificação de plágio, contudo, por não ser o objeto central deste trabalho, essas possibilidades são abordadas na seção de trabalhos futuros.

Tabela 6.2 – Evolução do Recall do Min-Hash conforme o pré-processamento varia.

pos	**Max Documents**	**Recall for Signatures**					
		48	**96**	**192**	**384**	**768**	**1536**
1597	10%	**0.498**	**0.547**	**0.619**	**0.662**	**0.707**	**0.720**
1597	15%	0.417	0.494	0.583	0.593	0.634	0.676
1597	30%	0.371	0.442	0.461	0.511	0.55	0.514
1597	45%	0.344	0.305	0.406	0.346	0.447	0.485
1597	original	0.274	0.295	0.315	0.33	0.348	0.358
3991	10%	**0.631**	**0.703**	**0.752**	**0.783**	**0.806**	**0.809**
3991	15%	0.606	0.668	0.714	0.738	0.781	0.773
3991	30%	0.532	0.600	0.640	0.669	0.681	0.66
3991	45%	0.528	0.490	0.554	0.534	0.609	0.616
3991	original	0.442	0.458	0.465	0.473	0.477	0.482
7983	10%	0.714	**0.818**	**0.876**	**0.907**	**0.916**	**0.922**
7983	15%	**0.731**	0.813	0.870	0.892	0.901	0.914
7983	30%	0.727	0.790	0.831	0.844	0.849	0.836
7983	45%	0.719	0.741	0.789	0.771	0.824	0.798
7983	original	0.667	0.679	0.677	0.684	0.674	0.677
11975	10%	0.721	0.847	0.939	**0.974**	**0.980**	**0.984**
11975	15%	0.761	0.882	**0.945**	0.972	0.976	0.981
11975	30%	0.838	**0.907**	0.932	0.957	0.951	0.953
11975	45%	0.841	0.893	0.916	0.925	0.937	0.932
11975	original	**0.849**	0.872	0.876	0.881	0.875	0.883

A tabela 6.2 apresenta os níveis de recall do Min-Hash, com o vocabulário original, ou seja, sem alterar o pré-processamento, em comparação a limitação do vocabulário em utilizar apenas palavras que aparecem em no máximo, 10%, 15%, 30% e 45% dos documentos da coleção. Os melhores níveis de recall, para cada valor *pos*, são destacados em negrito.

O vocabulário original possui o maior número de features, o que potencializa a possibilidade de colisão, contudo, reduz a probabilidade de a colisão ser representativa a identificação do plágio. Conforme o vocabulário é reduzido, a probabilidade de colisão também é reduzida, entretanto, é maior a probabilidade de a colisão ser mais representativa. Tal fator pode ser observado na tabela 6.2, onde os menores valores de *pos*, que exigem uma maior precisão na recuperação, alcançaram melhor desempenho utilizando o vocabulário compostos de palavras que aparecem em até 15% dos documentos. Ao aumentar o valor de *pos*, é desejado se atingir o recall

maior, contudo, com um vocabulário muito reduzido e com poucas assinaturas, a probabilidade de colisão fica muito reduzida. Tal fator pode ser observado no *pos* igual a 11975 e com 48 assinaturas, nesta combinação, o vocabulário original obteve o melhor desempenho. Contudo, conforme o número de assinaturas aumenta, a probabilidade dos métodos LSH encontrar os documentos que foram plagiados também aumenta, assim, para o mesmo *pos* igual a 11975, a partir de 96 assinaturas, ao limitar apenas ao uso das palavras mais representativas se obteve um melhor nível de recall.

É importante ressaltar que apesar de as figuras 6.1 e 6.2 não demonstrarem uma vantagem em utilizar mais que 384 assinaturas, a tabela 6.2 demonstra que, de acordo com o pré-processamento, ao utilizar mais assinaturas, é possível atingir melhores níveis de recall. Essa constatação corrobora ainda mais a relevância deste trabalho, que busca viabilizar a identificação de plágio em coleções massivas de documentos.

6.4.2 A Eficiência na Indexação de Documentos

A presente seção avalia a eficiência das estratégias de Paralelismo nos Documentos(PnD) e Paralelismo nas Permutações(PnP) na etapa de indexação da tarefa de Recuperação Heurística. Para tanto, o nível de escalabilidade das estratégias é verificado através a medição do aumento da vazão de indexação, conforme os recursos computacionais aumentam, i.e., nó(s) trabalhadores. As figuras 6.3 e 6.4 apresentam os resultados da vazão de indexação, de forma simplificada, para apenas 96, 384 e 1536 assinaturas, devido ao comportamento uniforme entre todas as assinaturas avaliadas. Ainda assim, a tabela 6.3 apresenta a vazão de indexação completa, para todas as assinaturas avaliadas.

A figura 6.3 apresenta a escalabilidade na vazão de indexação das estratégias PnD e PnP. A visualização vertical dos gráficos demonstra o comportamento da escalabilidade das estratégias em diferentes métodos LSH, sendo eles, respectivamente, o Min-Hash, Min-Max-Hash e CSA. Enquanto que a visualização horizontal dos gráficos demonstra o comportamento da escalabilidade de cada método LSH segundo o número de k assinatura, sendo elas: 96, 384, 1532. As linhas continuas representam os verdadeiros valores obtidos, enquanto que as linhas pontilhadas representam o aumento linear da vazão de indexação ao aumentar o número de nó(s) trabalhadores. É possível notar que tanto a estratégia PnD, como a PnP obtiveram uma aceleração acima da vazão linear para todas as assinaturas e métodos LSH. Entretanto, apesar de ambas as estratégias apresentarem desempenhos muitos próximos de vazão de indexação, a estratégia PnD no mínimo registrou o mesmo desempenho e em alguns casos houve uma pequena melhora na vazão de indexação.

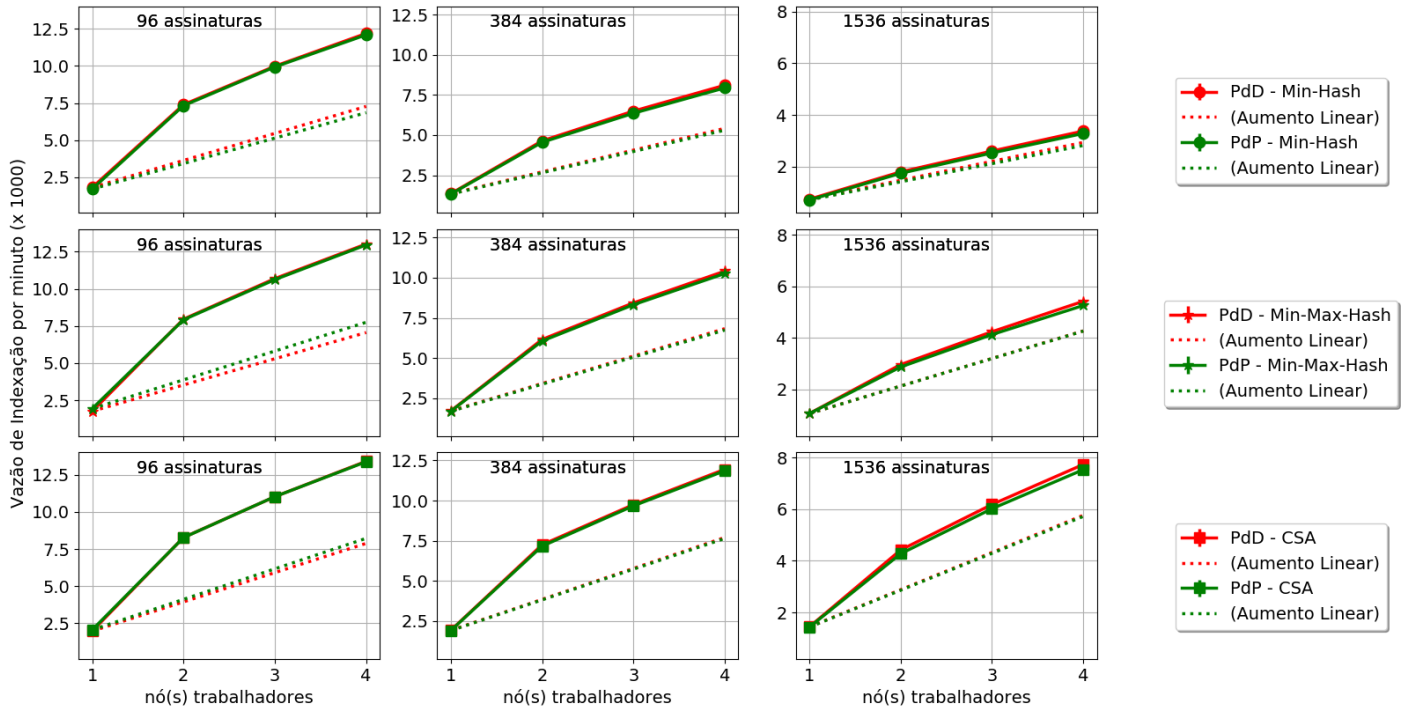


Figura 6.3 – Escalabilidade na Vazão de Indexação (documentos por minuto ×1000) das estratégias PnD e PnP .

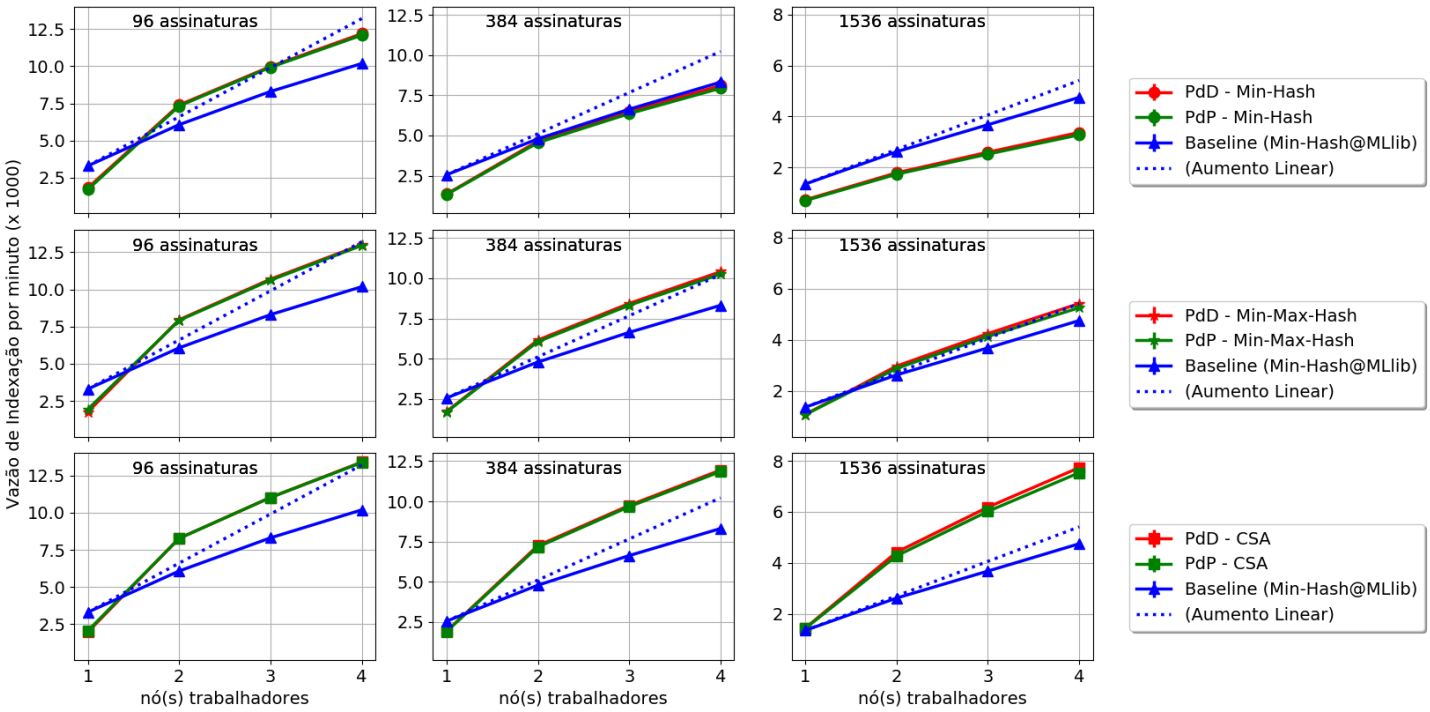


Figura 6.4 – Escalabilidade na Vazão de Indexação (documentos por minuto ×1000) das estratégias PnD e PnP em comparação ao baseline Min-Hash@MLlib.

A figura 6.4 compara a escalabilidade na vazão de indexação dos métodos Min-Hash, Min-Max-Hash e CSA nas estratégias PnD e PnP, com o Min-Hash@MLlib, considerado o baseline nesse trabalho. É possível notar que apesar de Min-

Hash@MLlib não apresentar uma aceleração acima da vazão linear esperada, para gerar a partir de 384 assinaturas, o Min-Hash@MLlib atinge um melhor desempenho na vazão de indexação, em comparação ao método Min-Hash nas estratégias PnD e PnP. Tal fator pode ser explicado pelo custo adicional dos métodos LSH nas estratégias PnD e PnP para gerar um índice invertido, o que não ocorre no Min-Hash@MLlib. Entretanto, é importante considerar que o Min-Hash@MLlib é uma implementação fechada de um método LSH, e que as estratégias PnD e PnP permitem a implementação de qualquer método LSH baseado em permutações aleatória, de maneira que, o melhor desempenho dos métodos Min-Max-Hash e CSA, em termos de escalabilidade, para todas as permutações, são considerados uma forte contribuição deste trabalho.

Tabela 6.3 – Escalabilidade na Vazão de Indexação (documentos por minuto ×1000) para todos os casos avaliados.

Permutações	**Nodes**	**Baseline**	**PnD**			**PnP**		
		Min MLlib	**Min**	**Min-Max**	**CSA**	**Min**	**Min-Max**	**CSA**
96	1	3.302	1.820	1.766	1.969	1.714	1.938	2.054
	2	6.074	7.35	7.908	8.23	7.293	7.893	8.217
	3	8.306	9.697	10.394	10.73	9.649	10.352	10.725
	4	10.113	11.665	12.396	12.803	11.61	12.405	12.756
192	1	2.996	1.729	1.952	2.006	1.711	1.937	2.059
	2	5.573	6.187	7.311	7.877	6.125	7.208	7.675
	3	7.683	8.327	9.684	10.317	8.203	9.605	10.273
	4	9.425	10.108	11.597	12.336	9.987	11.524	12.299
384	1	2.555	1.354	1.707	1.927	1.326	1.685	1.911
	2	4.789	4.614	6.126	7.208	4.527	6.028	7.136
	3	6.634	6.335	8.267	9.532	6.229	8.123	9.434
	4	8.257	7.775	10.033	11.464	7.668	9.886	11.389
768	1	1.970	1.126	1.316	1.651	1.096	1.282	1.624
	2	3.737	3.022	4.557	5.986	2.924	4.43	5.853
	3	5.221	4.246	6.258	8.066	4.143	6.112	7.909
	4	6.581	5.333	7.699	9.824	5.221	7.534	9.643
1536	1	1.353	0.734	1.068	1.441	0.704	1.066	1.430
	2	2.621	1.783	2.951	4.397	1.729	2.863	4.256
	3	3.678	2.551	4.148	6.043	2.474	4.05	5.894
	4	4.717	3.263	5.225	7.456	3.178	5.054	7.244

6.4.3 A Eficiência na Recuperação de Documentos Fonte

A presente seção avalia o desempenho das estratégias PnD e PnP na etapa de recuperação de documentos fonte, da tarefa de Recuperação Heurística. Para tanto, a escalabilidade das estratégias é verificada através da medição do tempo médio na

recuperação de documentos fonte, conforme os recursos computacionais aumentam, i.e., nó(s) trabalhadores. Além disso, a avaliação considerou um valor de $pos = 50$, isto é, as estratégias foram avaliadas na tarefa de recuperar 50% dos documentos da coleção PAN11 (7.983 documentos fonte). As figuras 6.5 e 6.6 apresentam os tempos médios de recuperação de documento fonte, de forma simplificada, para apenas 96, 384 e 1536 assinaturas, devido ao comportamento uniforme entre todas as assinaturas avaliadas. Ainda assim, a tabela 6.3 apresenta todos os tempos médios de recuperação de documento fonte, para todas as assinaturas avaliadas.

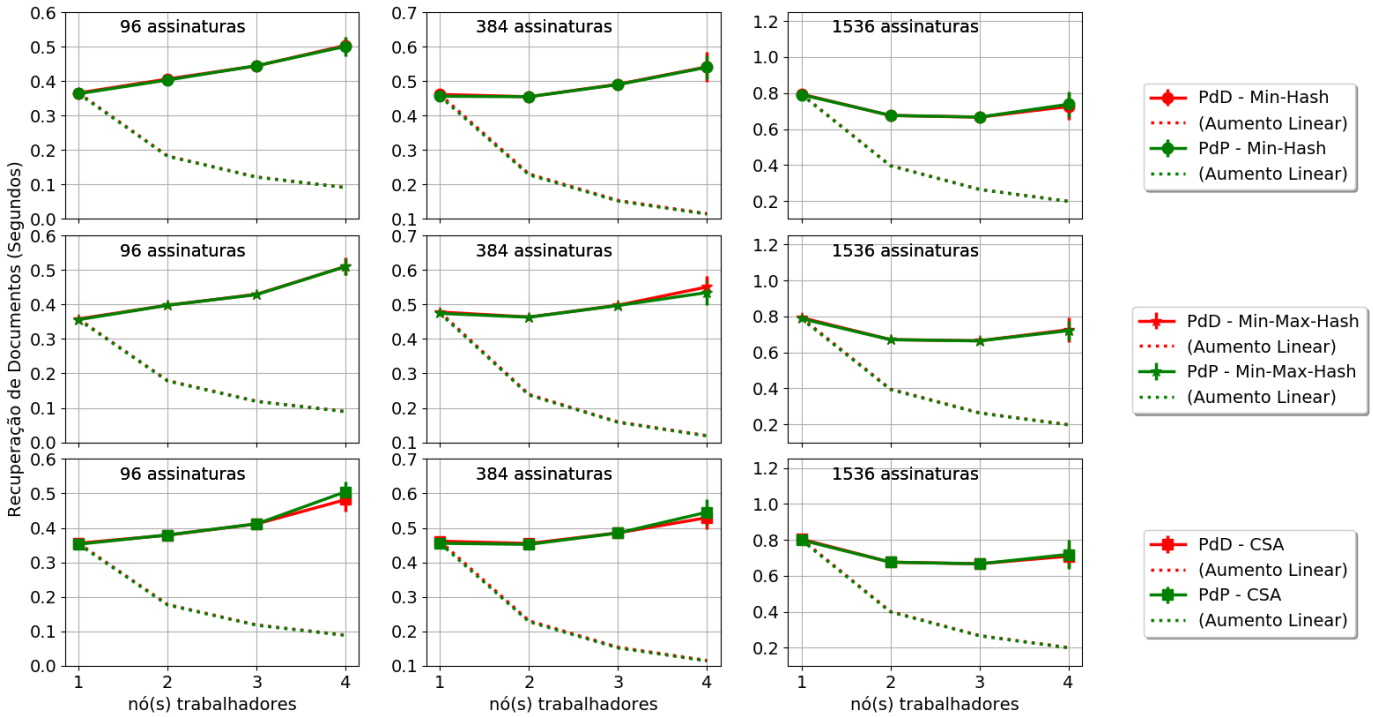


Figura 6.5 – Escalabilidade na Recuperação de Documentos Fonte das estratégias PnD e PnP.

A figura 6.5 apresenta a escalabilidade na recuperação de documentos fonte das estratégias PnD e PnP. É possível notar que em todos os métodos avaliados, para todas as permutações, o tempo de recuperação observado ficou acima do esperado. Entretanto, conforme o número de assinaturas aumenta, mais o tempo observado se aproxima do esperado. Tal comportamento pode ser verificado nos gráficos de 96 e 1536 assinaturas, ao aumentar de um para dois nó(s). Contudo, especificamente para gerar 1536 assinaturas, ao utilizar mais que dois no(s), novamente, os tempos observados se distanciam do valor linear esperado.

Dessa forma, é possível analisar que, para todos os métodos avaliados nas estratégias PnD e PnP, só é vantajoso utilizar muitos nó(s) em problemas altamente custosos. Como o custo para gerar 1536 assinaturas não foi suficiente para valer a pena utilizar quatro nó(s), uma das hipóteses é que quando a tarefa a ser executada é muito pequena, é mais custoso dividir sua execução em muitos nó(s) do que

executar a tarefa em poucos nó(s). Para tanto, na seção 6.4.4 o comportamento das estratégias sobre uma alta carga de documentos é avaliado, aumentando assim o custo computacional da tarefa, com o intuito de verificar a respectiva hipótese.

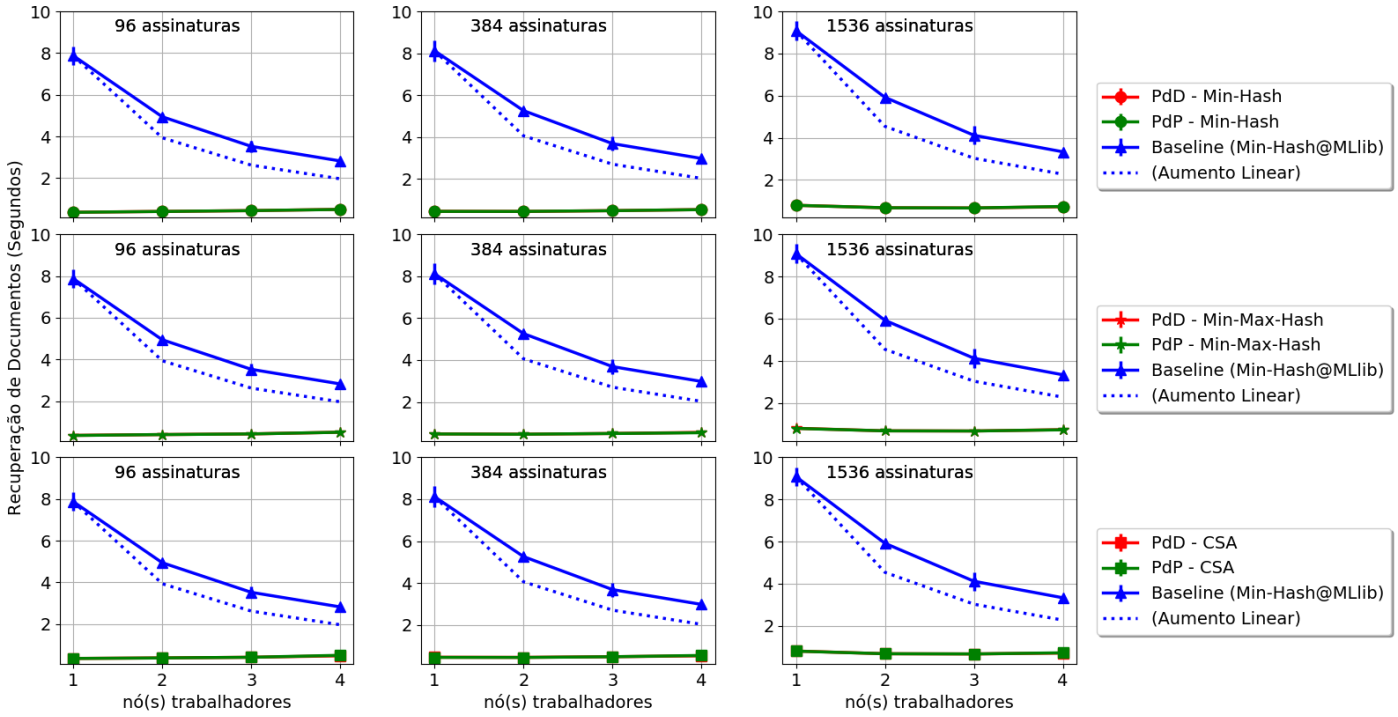


Figura 6.6 – Escalabilidade das estratégias PnD e PnP na recuperação de documentos .

A figura 6.6 compara a escalabilidade na recuperação de documentos fontes dos métodos Min-Hash, Min-Max-Hash e CSA nas estratégias PnD e PnP, com o Min-Hash@MLlib, considerado o baseline nesse trabalho. É possível notar que o Min-Hash@MLlib apresenta, em todos os casos, uma redução no tempo de recuperação de documentos até 4 nó(s), diferente das estratégias PnD e PnP, que a partir do uso de 3 nó(s), registrou, para todos os casos avaliados, um aumento no tempo de recuperação de documentos fonte. Contudo, apesar do Min-Hash@MLlib, em termos de escalabilidade, apresentar um desempenho observado mais próximo de linear, o Min-Hash@MLlib está em uma ordem magnitude consideravelmente maior que os métodos nas estratégias PnD e PnP. Ademais, a tabela 6.4 apresenta a a escalabilidade completa na recuperação de documentos fontes, para todas as assinaturas avaliadas.

Tabela 6.4 – Escalabilidade na recuperação de documentos fonte (em segundos) para todos os casos avaliados.

Permutações	Nodes	Baseline	PnD			PnP		
		Min MLlib	Min	Min-Max	CSA	Min	Min-Max	CSA
96	1	3,302	0,628	0,93	1,193	0,614	0,915	1,164
	2	6,074	7,35	7,908	8,23	7,293	7,893	8,217
	3	8,306	9,697	10,394	10,73	9,649	10,352	10,725
	4	10,113	11,665	12,396	12,803	11,61	12,405	12,756
192	1	2,996	1,245	1,222	1,45	1,126	1,415	1,569
	2	5,573	6,187	7,311	7,877	6,125	7,208	7,675
	3	7,683	8,327	9,684	10,317	8,203	9,605	10,273
	4	9,425	10,108	11,597	12,336	9,987	11,524	12,299
384	1	2,555	0,897	0,870	1,159	0,779	1,088	1,338
	2	4,789	4,614	6,126	7,208	4,527	6,028	7,136
	3	6,634	6,335	8,267	9,532	6,229	8,123	9,434
	4	8,257	7,775	10,033	11,464	7,668	9,886	11,389
768	1	1,970	0,776	0,555	0,844	0,486	0,752	1,036
	2	3,737	3,022	4,557	5,986	2,924	4,43	5,853
	3	5,221	4,246	6,258	8,066	4,143	6,112	7,909
	4	6,581	5,333	7,699	9,824	5,221	7,534	9,643
1536	1	1,353	0,48	0,749	1,061	0,467	0,727	1,024
	2	2,621	1,783	2,951	4,397	1,729	2,863	4,256
	3	3,678	2,551	4,148	6,043	2,474	4,05	5,894
	4	4,717	3,263	5,225	7,456	3,178	5,054	7,244

6.4.4 A Eficiência em Cenários de Larga Escala

A presente seção tem o objetivo avaliar a eficiência nas etapas de indexação e recuperação de documentos fonte, em cenários de larga escala. Para tanto, foram utilizados 4 nó(s) trabalhadores para gerar 1536 assinaturas, e os documentos fonte D, da coleção PAN11, foram replicados em [10.000, 20.000, 30.000, 40.000], para permitir a avaliação da eficiência na escalabilidade nos dados sob alto custo computacional.

Na avaliação da eficiência na etapa de recuperação de documentos fonte (seção6.4.3), o melhor desempenho do baseline Min-Hash@MLlib foi observado no uso de quatro nó(s) trabalhadores. Em oposto, as estratégias PnD e PnP registraram o pior desempenho no uso de quatro nó(s), apesar de as estratégias PnD e PnP apresentarem uma tendência de melhora conforme o custo computacional aumentou, ao gerar mais assinaturas. Portanto, essa seção tem o intuito de observar o comporta-

mento das estratégias em comparação com o Min-Hash@MLlib, em cenário de alto custo computacional, aumentando o volume de dados nas operações. Com o intuito de simplificar a análise, o desempenho do Min-Hash@MLlib foi comparado apenas com os métodos na estratégias PnD, tendo em vista que a avaliação da eficiência na indexação e recuperação de documentos fonte apontaram um desempenho similar em todos os casos avaliados.

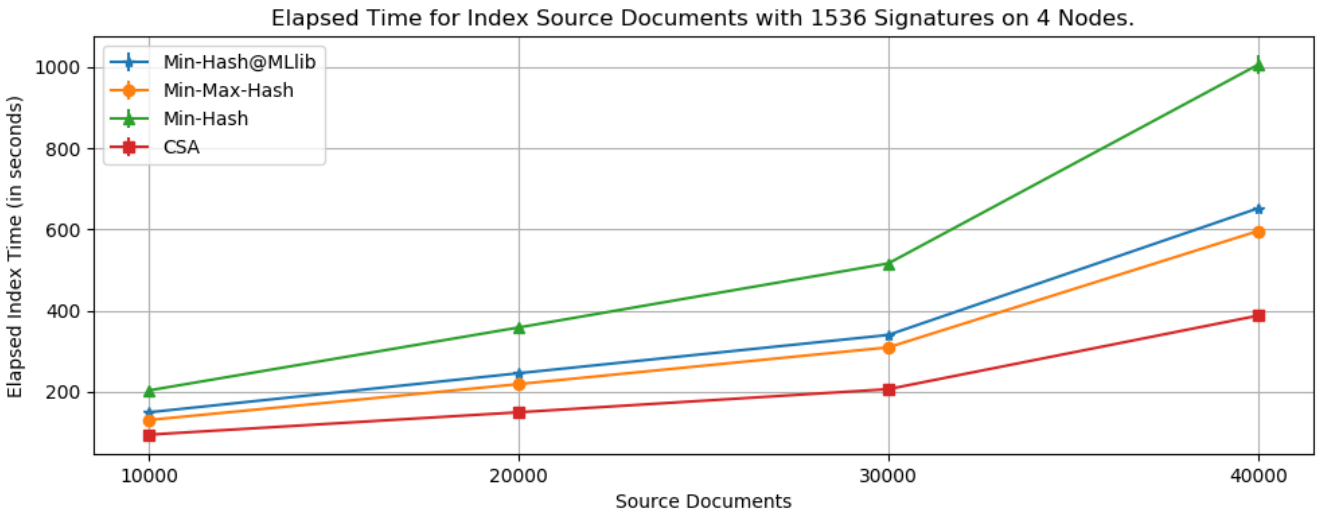


Figura 6.7 – Eficiência na Indexação de documentos em larga escala.

A figura 6.7 apresenta o tempo total gasto para os indexar os documentos fonte. O Min-Hash@MLlib é representado pela linha de cor azul, o Min-Hash na cor verde, o Min-Max-Hash na cor laranja e o CSA na cor vermelha. Para todas as variações de $|D|$, o comportamento obtido na avaliação com 15966 documentos foi mantido, onde Min-Hash@MLlib se consolidou com uma melhor eficiência na indexação, em detrimento do Min-Hash. Contudo, foi superado pelos métodos Min-Max-Hash e CSA.

Com o intuito avaliar a eficiência na recuperação de documentos fonte apenas em razão da quantidade de documentos indexados, a quantidade de documentos a ser retornados foi fixada em 7983, ao invés de ser variável para o $pos = 50\%$, o que aumentaria o número de documentos retornados, conforme a quantidade de documentos indexados aumentassem.

Tabela 6.5 – Perda do Min-Hash@MLlib.

Documentos	Min-Hash	Min-Hash@MLlib	Aumento
10000	0,559	2,554	4,570
20000	0,713	3,755	5,263
30000	0,783	4,890	6,243
40000	0,948	8,087	8,527

]

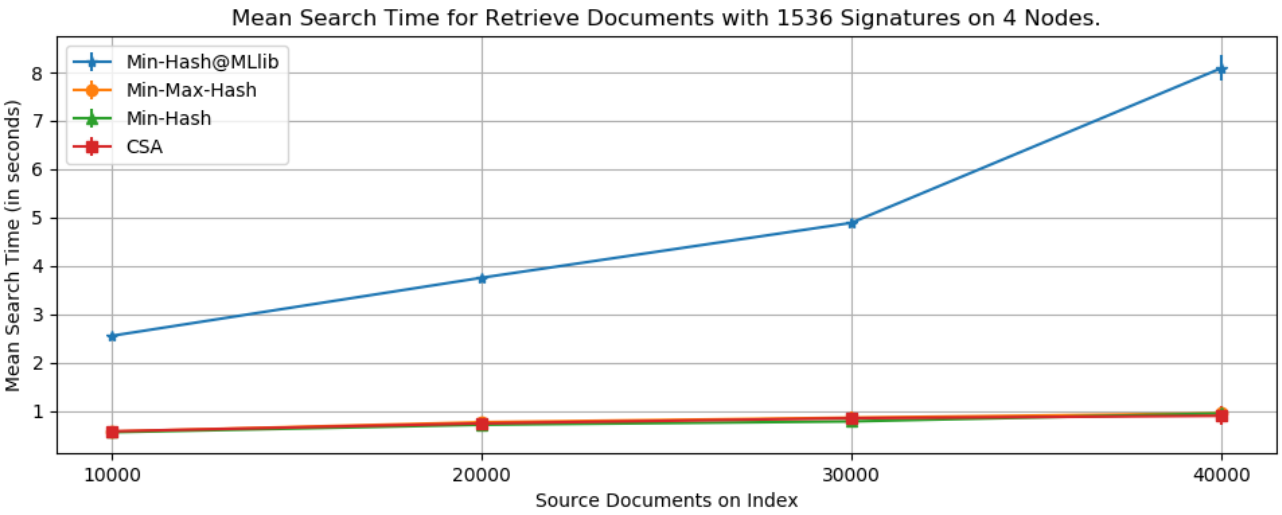


Figura 6.8 – Eficiência na recuperação de documentos fonte, para retornar 7983 documentos.

A figura 6.8 apresenta o tempo médio de recuperação de documentos. É possível notar que todos os métodos apresentaram um aumento no tempo de recuperação, contudo, devido a não utilização de um índice invertido, o método Min-Hash@MLlib se mostrou menos eficiente na recuperação de documentos em larga escala. A tabela 6.5 consolida o aumento da diferença entre o Min-Hash@MLlib e o Min-Hash, conforme $|D|$ aumenta.

Capítulo 7

Conclusões

Este trabalho contextualizou a necessidade de aumentar a eficiência da etapa de recuperação heurística na redução dos tempos para representar, buscar e retornar os documentos mais prováveis de terem sido plagiados, para viabilizar a identificação de plágio externo em grandes coleções de documentos.

Com a proposta de aumentar a eficiência sem a perda de eficácia na recuperação heurística com a utilização de métodos LSH, esse trabalho propôs a formação de duas estratégias, denominadas de **paralelismo nos documentos (PnD)** e **paralelismo nas permutações (PnP)**, a partir do pipeline de DUARTE *et al.* (2017).

Para permitir a implementação das estratégias PnD e PnP no Spark, viabilizando sua aplicabilidade em grandes coleções de documentos, esse trabalho propôs duas estratégias de permutação, denominadas de **Re-indexar o Vetor (RIV)** e **Aumentar o Vetor ao Final (EVF)**, as quais reduzem o custo proibitivo para armazenar e replicar o vetor de permutação, e ainda assim mantêm a propriedade dos métodos LSH em preservar a similaridade.

Os experimentos realizados com as estratégias PnD e PnP demonstram um aumento da eficiência na etapa de indexação, ao proporcionar uma ampliação na vazão de documentos, considerando a ampliação da capacidade computacional, sem comprometer a eficiência na tarefa de recuperação heurística com métodos LSH. Na etapa de recuperação de informação, os experimentos demonstraram que as estratégias PnD e PnP são mais eficientes que o baseline apresentado, que obteve uma alta degradação conforme aumenta o número de documentos.

No tocante ao desempenho das estratégias PnD e PnP, nas etapas de indexação e recuperação de documentos fonte, não houve uma alteração substancial nos resultados, contudo, na maioria dos casos avaliados, a estratégia PnD registrou um desempenho ligeiramente melhor que estratégia PnP.

7.1 Trabalhos Futuros

Este trabalho considerou as seguintes atividades como oportunidades a serem abordada no futuro:

(i) Avaliar as estratégias PnD e PnP sob uma maior capacidade computacional, com o intuito de observar o ponto de degradação na etapa de indexação, isto é, descobrir até que ponto o aumento da capacidade computacional é capaz de aumentar a vazão de indexação.

(ii) Avaliar de forma especifica, para cada um dos passos executados, a melhor configuração de partições do Spark.

(iii) Explorar diferentes oportunidades de pré-processamento, com o intuito de aumentar a probabilidade do modelo de recuperação de informação retornar os documentos efetivamente plagiados.

(iv) Utilizar e verificar a viabilidade das estratégias PnD e PnP no problema de recuperação heurística de áreas co-relacionadas à identificação de plágio.

Referências Bibliográficas

ALZAHRANI, S. M., SALIM, N., ABRAHAM, A., 2011, "Understanding plagiarism linguistic patterns, textual features, and detection methods", *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, v. 42, n. 2, pp. 133–149.

ALZAHRANI, S. M., SALIM, N., ABRAHAM, A., 2012a, "Understanding plagiarism linguistic patterns, textual features, and detection methods", *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, v. 42, n. 2, pp. 133–149.

ALZAHRANI, S. M., SALIM, N., ABRAHAM, A., 2012b, "Understanding plagiarism linguistic patterns, textual features, and detection methods", *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, v. 42, n. 2, pp. 133–149.

BAEZA-YATES, R., RIBEIRO-NETO, B., OTHERS, 1999, *Modern information retrieval*, v. 463. ACM press New York.

BARRÓN-CEDEÑO, A., VILA, M., MARTÍ, M., et al., 2013, "Plagiarism Meets Paraphrasing: Insights for the Next Generation in Automatic Plagiarism Detection", *Comput. Linguist.*, v. 39, n. 4 (dec.), pp. 917–947. ISSN: 0891-2017. doi: 10.1162/COLI_a_00153. `http://dx.doi.org/10.1162/COLI_a_00153` .

BARRÓN-CEDEÑO, A., 2010, "On the mono-and cross-language detection of text reuse and plagiarism". In: *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pp. 914–914. ACM, July.

BBC, 2011. "The Ctrl+C, Ctrl+V boom". `https://www.bbc.com/news/magazine-12613617` .

BRADLEY, J., NI, Y., CHU, K., 2017. "Detecting Abuse at Scale: Locality Sensitive Hashing at Uber Engineering".

databricks.com/blog/2017/05/09/detecting-abuse-scale-locality-sensitive-hashing-uber-engineering.html. [Online; accessed 19-July-2018].

BRODER, A. Z., 1997, "On the resemblance and containment of documents". In: *Compression and Complexity of Sequences 1997. Proceedings*, pp. 21–29. IEEE, June.

BRODER, A. Z., GLASSMAN, S. C., MANASSE, M. S., et al., 1997, "Syntactic clustering of the web", *Computer Networks and ISDN Systems*, v. 29, n. 8, pp. 1157–1166.

BRODER, A. Z., CHARIKAR, M., FRIEZE, A. M., et al., 2000, "Min-wise independent permutations", *Journal of Computer and System Sciences*, v. 60, n. 3, pp. 630–659.

CLOUGH, P., 2000, *Plagiarism in natural and programming languages: an overview of current tools and technologies*. Relatório técnico, University of Sheffield.

CLOUGH, P., STEVENSON, M., 2011a, "Developing a corpus of plagiarised short answers", *Language Resources and Evaluation*, v. 45, n. 1, pp. 5–24.

CLOUGH, P., STEVENSON, M., 2011b, "Developing a corpus of plagiarised short answers", *Language Resources and Evaluation*, v. 45, n. 1, pp. 5–24. ISSN: 1574-0218. doi: 10.1007/s10579-009-9112-1. `https://doi.org/10.1007/s10579-009-9112-1` .

DAHLGAARD, S., KNUDSEN, M. B. T., ROTENBERG, E., et al., 2015, "Hashing for statistics over k-partitions". In: *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pp. 1292–1310. IEEE, October.

DEAN, J., GHEMAWAT, S., 2004, "MapReduce: Simplified Data Processing on Large Clusters". In: *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, pp. 137–150, San Francisco, CA, November.

DITTMAR, C., HILDEBRAND, K. F., GAERTNER, D., et al., 2012, "Audio forensics meets music information retrieval—a toolbox for inspection of music plagiarism". In: *Signal Processing Conference (EUSIPCO), 2012 Proceedings of the 20th European*, pp. 1249–1253. IEEE, August.

DUARTE, F., CALED, D., XEXÉO, G., 2017, "Minmax Circular Sector Arc for External Plagiarism's Heuristic Retrieval stage", *Knowledge-Based Systems*, v. 137, n. Supplement C, pp. 1 – 18. ISSN: 0950-7051. doi: https:

//doi.org/10.1016/j.knosys.2017.08.013. http://www.sciencedirect.com/science/article/pii/S0950705117303696 .

DUARTE, F. R., 2017, *IDENTIFICANDO PLAGIO EXTERNO COM LOCALITY-SENSITIVE HASHING*. Ph.D. Thesis, Universidade Federal do Rio de Janeiro.

EHSAN, N., SHAKERY, A., 2016, "Candidate document retrieval for cross-lingual plagiarism detection using two-level proximity information", *Information Processing & Management*, v. 52, n. 6, pp. 1004 – 1017. ISSN: 0306-4573. doi: http://dx.doi.org/10.1016/j.ipm.2016.04.006. http://www.sciencedirect.com/science/article/pii/S0306457316300784 .

EINSTEIN, A., 1905, "Zur Elektrodynamik bewegter Körper. (German) [On the electrodynamics of moving bodies]", *Annalen der Physik*, v. 322, n. 10, pp. 891–921. doi: http://dx.doi.org/10.1002/andp.19053221004.

FAWZY, F., 2016. "From speeches to Ph.D.'s: Politicians called out for copying". http://edition.cnn.com/2016/07/19/politics/politicians-plagiarism/index.html. [Online; accessed 01-march-2018].

FETTERLY, D., MANASSE, M., NAJORK, M., et al., 2003, "A large-scale study of the evolution of web pages". In: *Proceedings of the 12th international conference on World Wide Web*, pp. 669–678. ACM, May.

GAIZAUSKAS, R., FOSTER, J., WILKS, Y., et al., 2001, "The METER corpus: a corpus for analysing journalistic text reuse". In: *The Corpus Linguistics 2001 Conference*, pp. 214–223, March.

GORDON, M. I., THIES, W., AMARASINGHE, S., 2006, "Exploiting coarse-grained task, data, and pipeline parallelism in stream programs", *ACM SIGARCH Computer Architecture News*, v. 34, n. 5, pp. 151–162.

GUARDIAN, 2011. "German defence minister resigns in PhD plagiarism row". www.theguardian.com/world/2011/mar/01/german-defence-minister-resigns-plagiarism. [Online; accessed 01-march-2018].

GUPTA, G. P., KULARIYA, M., 2016, "A framework for fast and efficient cyber security network intrusion detection using apache spark", *Procedia Computer Science*, v. 93, pp. 824–831.

HARDWICK, J. C., 1997, *Practical parallel divide-and-conquer algorithms*. Relatório técnico, CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE.

HUANG, A., 2008, "Similarity measures for text document clustering". In: *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, pp. 49–56, April.

HYVÖNEN, V., PITKÄNEN, T., TASOULIS, S., et al., 2015. "Fast k-NN search". `http://arxiv.org/abs/1509.06957` .

JI, J., LI, J., YAN, S., et al., 2013, "Min-max hash for jaccard similarity". In: *2013 IEEE 13th International Conference on Data Mining*, pp. 301–309. IEEE, December.

JIFFRIYA, M., JAHAN, M. A., RAGEL, R. G., et al., 2013, "AntiPlag: Plagiarism detection on electronic submissions of text based assignments". In: *2013 IEEE 8th International Conference on Industrial and Information Systems*, pp. 376–380. IEEE, December.

JÚNIOR, P. R. M., 2011, *USO DE PARALELISMO DE DADOS PARA MAIOR EFICIÊNCIA DE ALGORITMOS DE PROCESSAMENTO DE IMAGENS*. Relatório técnico, Departamento de Computação/Universidade Federal de Ouro Preto.

KUMAR, V., 2002, *Introduction to parallel computing*. Addison-Wesley Longman Publishing Co., Inc.

LEE, D. C., KE, Q., ISARD, M., 2010, "Partition min-hash for partial duplicate image discovery". In: *11 European Conference on Computer Vision*, pp. 648–662. Springer, September.

LESKOVEC, J., RAJARAMAN, A., ULLMAN, J. D., 2014, *Mining of massive datasets*. Cambridge University Press.

LI, J., ZHENG, R., CHEN, H., 2006, "From Fingerprint to Writeprint", *Commun. ACM*, v. 49, n. 4 (apr.), pp. 76–82. ISSN: 0001-0782. doi: 10.1145/1121949.1121951. `http://doi.acm.org/10.1145/1121949.1121951` .

LI, P., KÖNIG, A. C., 2011, "Theory and applications of b-bit minwise hashing", *Communications of the ACM*, v. 54, n. 8 (August), pp. 101–109.

LI, P., KÖNIG, C., 2010, "b-Bit minwise hashing". In: *Proceedings of the 19th international conference on World wide web*, pp. 671–680. ACM, April.

LI, P., OWEN, A., ZHANG, C.-H., 2012, "One permutation hashing". In: *Advances in Neural Information Processing Systems*, pp. 3113–3121, December.

LIDDELL, J., 2003, "A comprehensive definition of plagiarism", *Community & Junior College Libraries*, v. 11, n. 3, pp. 43–52.

LOWE, D. G., 2004, "Distinctive Image Features from Scale-Invariant Keypoints", *Int. J. Comput. Vision*, v. 60, n. 2 (nov.), pp. 91–110. ISSN: 0920-5691. doi: 10.1023/B:VISI.0000029664.99615.94. `https://doi.org/10.1023/B:VISI.0000029664.99615.94` .

MARTIN, B., 1994, "Plagiarism: a misplaced emphasis", *Journal of Information Ethics*, v. 3, n. 2 (Autumn), pp. 36–47.

MAURER, H., KAPPE, F., ZAKA, B., 2006a, "Plagiarism - A Survey", *Journal of Universal Computer Science*, v. 12, n. 8 (aug), pp. 1050–1084. `http://www.jucs.org/jucs_12_8/plagiarism_a_survey` .

MAURER, H. A., KAPPE, F., ZAKA, B., 2006b, "Plagiarism-A survey." *J. UCS*, v. 12, n. 8, pp. 1050–1084.

MENG, X., BRADLEY, J., YAVUZ, B., et al., 2016, "Mllib: Machine learning in apache spark", *The Journal of Machine Learning Research*, v. 17, n. 1, pp. 1235–1241.

MERRIAM, 2017. "Definition of PLAGIARIZING". `https://www.merriam-webster.com/dictionary/plagiarizing` .

MEUSCHKE, N., GIPP, B., 2014, "Reducing computational effort for plagiarism detection by using citation characteristics to limit retrieval space". In: *IEEE/ACM Joint Conference on Digital Libraries*, pp. 197–200, Sept. doi: 10.1109/JCDL.2014.6970168.

MOHAMED, H., MARCHAND-MAILLET, S., 2015, "Quantized Ranking for Permutation-based Indexing", *Inf. Syst.*, v. 52, n. C (aug.), pp. 163–175. ISSN: 0306-4379. doi: 10.1016/j.is.2015.01.009. `http://dx.doi.org/10.1016/j.is.2015.01.009` .

NAIK, R. R., LANDGE, M. B., MAHENDER, C. N., 2015, "A review on plagiarism detection tools", *International Journal of Computer Applications*, v. 125, n. 11.

NIWATTANAKUL, S., SINGTHONGCHAI, J., NAENUDORN, E., et al., 2013, "Using of Jaccard coefficient for keywords similarity". In: *Proceedings of*

the International MultiConference of Engineers and Computer Scientists, v. 1, pp. 13–15, March.

OGASAWARA, E., DE OLIVEIRA, D., CHIRIGATI, F., et al., 2009, "Exploring many task computing in scientific workflows". In: *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers - MTAGS '09*, pp. 1–10. ACM Press, November. ISBN: 978-1-60558-714-1. doi: 10.1145/1646468.1646470. http://portal.acm.org/citation.cfm?doid=1646468.1646470 .

OXFORD, 1999. "plagiarism | Definition of plagiarism in US English by Oxford Dictionaries". https://en.oxforddictionaries.com/definition/us/plagiarism .

PAN, J., MANOCHA, D., 2012, "Bi-level locality sensitive hashing for k-nearest neighbor computation". In: *2012 IEEE 28th International Conference on Data Engineering*, pp. 378–389. IEEE, April.

PANIGRAHI, S., LENKA, R. K., STITIPRAGYAN, A., 2016, "A hybrid distributed collaborative filtering recommender engine using apache spark", *Procedia Computer Science*, v. 83, pp. 1000–1006.

PLAGIARISM, 2017. "What is Plagiarism? - Plagiarism.org". http://www.plagiarism.org/article/what-is-plagiarism .

POSNER, R. A., 2007, *The little book of plagiarism*. Pantheon.

POTTHAST, M., EISELT, A., BARRÓN-CEDEÑO, A., et al., 2011a, "Overview of the 3rd International Competition on Plagiarism Detection". In: Petras, V., Forner, P., Clough, P. (Eds.), *Working Notes Papers of the CLEF 2011 Evaluation Labs*, sep.a. ISBN: 978-88-904810-1-7. http://www.clef-initiative.eu/publication/working-notes .

POTTHAST, M., EISELT, A., BARRÓN CEDEÑO, L. A., et al., 2011b, "Overview of the 3rd international competition on plagiarism detection". In: *CEUR workshop proceedings*, v. 1177. CEUR Workshop Proceedings, b.

POTTHAST, M., HAGEN, M., BEYER, A., et al., 2014, "Overview of the 6th International Competition on Plagiarism Detection". In: Cappellato, L., Ferro, N., Halvey, M., et al. (Eds.), *Working Notes Papers of the CLEF 2014 Evaluation Labs*, CEUR Workshop Proceedings. CLEF and CEUR-WS.org, sep. http://www.clef-initiative.eu/publication/working-notes .

SAMUELSON, P., 1994, "Self-plagiarism or Fair Use", *Commun. ACM*, v. 37, n. 8 (aug.), pp. 21–25. ISSN: 0001-0782. doi: 10.1145/179606.179731. http://doi.acm.org/10.1145/179606.179731 .

SHAKHNAROVICH, G., DARRELL, T., INDYK, P., 2006, *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice (Neural Information Processing)*. The MIT Press. ISBN: 026219547X.

SITARAM, D., MANJUNATH, G., 2011, *Moving to the cloud: Developing apps in the new world of cloud computing*. Elsevier.

SPARK, 2018. "Overview - Spark 2.4.0 Documentation". https://spark.apache.org/docs/2.4.0/ .

STAMATATOS, E., 2009, "Intrinsic plagiarism detection using character n-gram profiles", *threshold*, v. 2, n. 1,500.

STEIN, B., ZU EISSEN, S. M., POTTHAST, M., 2007, "Strategies for Retrieving Plagiarized Documents". In: *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pp. 825–826, New York, NY, USA, July. ACM. ISBN: 978-1-59593-597-7. doi: 10.1145/1277741.1277928. http://doi.acm.org/10.1145/1277741.1277928 .

THOMPSON, V. U., PANCHEV, C., OAKES, M., 2015, "Performance evaluation of similarity measures on similar and dissimilar text retrieval". In: *2015 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K)*, v. 01, pp. 577–584, Nov.

WANG, J., SHEN, H. T., SONG, J., et al., 2014, "Hashing for similarity search: A survey", *arXiv preprint arXiv:1408.2927*.

WANG, J., LIU, W., KUMAR, S., et al., 2016, "Learning to hash for indexing big data—A survey", *Proceedings of the IEEE*, v. 104, n. 1, pp. 34–57.

WANG, X., DING, X., TUNG, A. K. H., et al., 2013, "Efficient and Effective KNN Sequence Search with Approximate N-grams", *Proc. VLDB Endow.*, v. 7, n. 1 (sep.), pp. 1–12. ISSN: 2150-8097. doi: 10.14778/2732219.2732220. http://dx.doi.org/10.14778/2732219.2732220 .

WOOD, G., 2004, "Academic original sin: Plagiarism, the internet, and librarians", *The Journal of Academic Librarianship*, v. 3, n. 30, pp. 237–242.

ZHANG, H., CHOW, T. W., 2011, "A coarse-to-fine framework to efficiently thwart plagiarism", *Pattern Recognition*, v. 44, n. 2, pp. 471 – 487. ISSN: 0031-3203. doi: http://dx.doi.org/10.1016/j.patcog.2010.08.023. http://www.sciencedirect.com/science/article/pii/S0031320310004097 .

ZU EISSEN, S. M., STEIN, B., KULIG, M., 2007, "Plagiarism detection without reference collections". In: *Advances in data analysis*, Springer, pp. 359–366.