Avaliação do impacto da remoção de características comuns em estratégias de busca de arquivos similares

João P. B. Velho[1], **Vitor H. G. Moia**[2], **Marco A. A. Henriques**[1]

[1]Faculdade de Engenharia Elétrica e de Computação (FEEC)
Universidade Estadual de Campinas (Unicamp)

[2]Samsung R&D Institute Brazil
Campinas, SP, Brasil

j218711@dac.unicamp.br vitor.m@samsung.com maah@unicamp.br

Abstract. *Digital forensic investigations face an important problem: The large amount of files that are stored in seized devices. To better assess these devices, one can use similarity search strategies, which are capable of finding identical, or even similar, files to a given set of files, using approximate matching techniques. However, this search can be impaired due to common blocks, pieces of similar or identical information (like headers, templates, etc) that are present in different files. This paper aims to assess the impact of removing common blocks on the strategies' performance. The results show a significative reduction on false positive rates, with a acceptable increase on runtime.*

Resumo. *Investigações de forense digital enfrentam um importante problema: a grande quantidade de arquivos armazenados em dispositivos apreendidos. Para analisar esses dispositivos de forma mais eficiente, utilizam-se estratégias de busca de similaridade, capazes de encontrar arquivos idênticos, ou até mesmo similares, à um dado conjunto de arquivos, usando técnicas de pareamento aproximado. No entanto, esta busca pode ser prejudicada devido a blocos comuns, como cabeçalhos, presentes em diferentes arquivos. Este trabalho objetiva avaliar o impacto da remoção de blocos comuns na performance das estratégias. Os resultados mostram uma redução significativa na taxa de falsos positivos com um aumento aceitável no tempo de execução.*

1. Introdução

Na última década, a capacidade de armazenamento de dispositivos pessoais, como computadores, celulares, *tablets*, dentre outros, cresceu de forma rápida e contínua, assim como a quantidade de arquivos presentes nestes dispositivos. Este aumento na quantidade de dados em dispositivos pessoais têm se tornado um grande desafio para os peritos de forense digital, já que estes devem realizar a análise de dispositivos que são apreendidos durante investigações em busca de arquivos suspeitos. Um método comumente empregado para tal busca é através do uso de funções hash. Contudo, estas são ineficazes para encontrar arquivos similares e uma alternativa é necessária para solucionar este problema.

Para encontrar arquivos similares, podemos utilizar as funções de pareamento aproximado. Da mesma forma que as funções hash, estas funções criam um resumo,

porém de similaridade do arquivo e, ao compará-lo com o resumo de outro arquivo, é possível determinar o quão similar estes são. Contudo, para buscar arquivos suspeitos em dispositivos com grandes quantidades de arquivos, as funções de pareamento aproximado são ineficazes (dada a grande quantidade de comparações dos resumos que serão necessárias). Para otimizar as comparações, foram propostas estratégias de busca de similaridade, as quais utilizam parte dos resumos das funções de pareamento aproximado para mapear um conjunto de arquivos em uma estrutura criada por elas e, em seguida, consultar um arquivo nessa estrutura. Com este tipo de técnica, a busca por arquivos se torna mais rápida e eficaz, evitando várias comparações desnecessárias.

Mesmo com a otimização que as estratégias de busca de similaridade oferecem, ainda há um grande empecilho que afeta de forma significativa seu desempenho: os blocos comuns. Estes blocos são trechos encontrados em arquivos (do mesmo tipo ou até em tipos diferentes) que são criados pela própria aplicação que os geraram, sendo partes de cabeçalhos, tabelas de cores, especificações de fontes etc. Em alguns casos estes blocos não são úteis para a avaliação de similaridade e apenas geram falsos positivos em uma busca; portanto, como não são gerados pelo usuário, não devem ser utilizados na avaliação de similaridade. Desta forma, ao removermos os blocos comuns, espera-se que ocorra um aumento na capacidade de encontrar dados similares pelas estratégias de busca, ao mesmo tempo em que há uma redução no número de falsos positivos.

Com tal cenário estruturado, este trabalho avalia o comportamento de algumas das estratégias presentes na literatura quanto à sua capacidade de detecção de similaridade e o tempo de execução quando removemos os blocos comuns dos arquivos durante uma busca de similaridade. Inicialmente foram escolhidas duas estratégias baseadas na ferramenta `sdhash` disponíveis na literatura (`MRSH-NET` [Breitinger et al. 2014a] e `MRSH-HBFT` [Lillis et al. 2017]) para realização de experimentos. Em seguida foram realizadas implementações alternativas destas estratégias (`NET-SD` [Velho et al. 2020] e `HBFT-SD` que foi modificado neste trabalho) e também modificações para remoção dos blocos comuns (`NET-SD-NCF` e `HBFT-SD-NCF`, ambos propostos neste trabalho). Finalmente foram realizadas comparações entre as três versões de cada uma das estratégias escolhidas a fim de analisar o impacto da remoção dos blocos comuns.

A principal contribuição deste trabalho foi avaliar o impacto da remoção dos blocos comuns em estratégias de busca de similaridade. Apesar de outros trabalhos já discutirem sobre o tema de blocos comuns e o impacto de sua remoção em ferramentas de pareamento aproximado ([Moia et al. 2020b]), nenhum trabalho na literatura analisou qual seria o impacto da remoção destes blocos em estratégias de busca de similaridade. Através desta análise, foi possível concluir que as estratégias de busca de similaridade, quando estão livres da influência dos blocos comuns, passam a ter um desempenho melhor, apresentando um aumento considerável em sua precisão. No entanto, esta maior precisão traz consigo um aumento no tempo de execução, o qual pode ser considerado aceitável dado o nível de melhoria na capacidade de encontrar similaridade. Além disso, com o aumento da precisão, o número de falsos positivos diminui, contribuindo para um menor trabalho manual dos peritos ao analisar os arquivos suspeitos.

Este artigo está estruturado da seguinte forma: inicialmente é feito um embasamento teórico na Sec. 2, apresentando alguma das ferramentas mais utilizadas da área; em seguida, é detalhada a contribuição deste trabalho na Sec. 3; por fim, são apresentados

os dados e métricas na Sec. 4 para na sequência, discutir os experimentos realizados na Sec. 5, juntamente com os resultados. Finalizando com as conclusões na Sec. 6.

2. Trabalhos relacionados

Ao longo de uma investigação de forense computacional, um dos maiores desafios de um perito é a busca por informações (suspeitas) em um dispositivo apreendido. Esta tarefa causa dificuldades devido à grande quantidade de dados presentes nos dispositivos atualmente. Uma primeira tática é o uso de funções hash para a busca de arquivos idênticos, calculando o hash de todos os arquivos e comparando-os um a um. No entanto, as funções hash apresentam uma grande limitação neste cenário, a incapacidade de identificar arquivos similares, pois uma simples alteração na entrada da função, modifica completamente o resultado.

Uma forma de mitigar a limitação das funções hash e assim realizar uma busca que permita encontrar tanto arquivos idênticos como similares é por meio das funções de pareamento aproximado. Essas funções geram um resumo do arquivo que, ao ser comparado com outro resumo, mostra o quão similar são os arquivos aos quais estes resumos pertencem. Desta forma, mesmo que alterações sejam realizadas no arquivo, os resumos são afetados de maneira proporcional e uma correlação ainda é possível. Exemplos de funções que implementam os conceitos de pareamento aproximado são: `sdhash` [Roussev 2010], `ssdeep` [Kornblum 2006], `MRSH-V2` [Breitinger and Baier 2013], `TLSH` [Oliver et al. 2013], `LZJD` [Raff and Nicholas 2018], dentre outras. Mais informações sobre as funções de pareamento aproximado podem ser encontradas em um report do NIST [Breitinger et al. 2014b].

No entanto, quando é necessário comparar um grande volume de arquivos, o número de comparações de resumos de similaridade é elevado e, portanto, o tempo necessário aumenta e torna-se até impraticável, dependendo do tamanho dos conjuntos a serem comparados e, por isso, torna-se necessária outra abordagem para este problema. Buscando reduzir o tempo de comparação de grandes quantidades de arquivos em busca de similaridade, foram desenvolvidas, com base nas funções de pareamento aproximado, as estratégias de busca de similaridade. Estas estratégias utilizam alguns conceitos e processos das funções de pareamento para realizar a comparação dos arquivos de forma mais eficiente. Alguns exemplos de estratégias encontradas na literatura são `MRSH-NET` [Breitinger et al. 2014a], `MRSH-HBFT` [Lillis et al. 2017] e `F2S2` [Winter et al. 2013].

O método para tornar a busca de similaridade mais eficiente varia para cada estratégia, podendo ser a utilização de tabelas hash ou de árvores de filtros de Bloom [Moia and Henriques 2017]. Apesar das vantagens trazidas por estas estratégias, elas apresentam problemas em relação à alta quantidade de falsos positivos, isto é, os pares de arquivos que as estratégias afirmam serem similares não se provam similares quando se examina o conteúdo do dado, gerado pelo usuário.

Um dos agentes causadores da alta taxa de falsos positivos encontrados nas estratégias são os blocos comuns [Moia et al. 2020a]. Tais blocos são trechos encontrados em vários arquivos (do mesmo ou de diferentes tipos) gerados pelas próprias aplicações, como partes de cabeçalhos, tabelas de cores, especificações de fontes etc., que não são relacionadas ao conteúdo gerado pelos usuários e, portanto, não são de interesse em alguns casos de busca de similaridade. Em algumas situações, estes blocos podem ser de grande

ajuda, por exemplo, quando desejamos encontrar arquivos de um determinado tipo (*pdf*, *doc*, *xls* etc), mas em outros casos, quando o foco é encontrar similaridade em arquivos relacionada ao conteúdo gerado por usuários, eles acabam aumentando a quantidade de falsos positivos durante uma busca, o que dificulta o trabalho do perito. Apesar da questão de blocos comuns já ter sido discutida e analisada na literatura de pareamento aproximado [Moia et al. 2020a, Moia et al. 2020b], ainda há uma carência de avaliações de comportamento das estratégias mediante a remoção dos blocos comuns dos arquivos e, portanto, este artigo se dedica a este estudo.

3. Contribuição

Para compreendermos a contribuição deste trabalho, primeiro é necessário selecionar as estratégias que serão avaliadas. Neste estudo foram selecionadas as estratégias `MRSH-NET` e `MRSH-HBFT`. Além disso, foram incluídas propostas de melhorias nessas estratégias, assim como versões de ambas nas quais são removidos blocos comuns. Um resumo dessas estratégias pode ser encontrado na Tabela 1.

Tabela 1. Estratégias de busca de similaridade avaliadas neste trabalho.

Família	Estratégia	Extrator de *features*	Principal característica
NET	`MRSH-NET`	`MRSH-V2`	Filtro de Bloom único
	`NET-SD`	`sdhash`	Adoção do `sdhash` em `MRSH-NET`
	`NET-SD-NCF`	`sdhash`	Remoção de *features* comuns
HBFT	`MRSH-HBFT`	`MRSH-V2`	Árvore de filtros de Bloom
	`HBFT-SD`	`sdhash`	Adoção do `sdhash` em `MRSH-HBFT`
	`HBFT-SD-NCF`	`sdhash`	Remoção de *features* comuns

A estratégia `MRSH-NET` realiza a extração de *features* de cada arquivo (similar à função de pareamento aproximado `MRSH-V2` em que ela é baseada) e insere todas as *features* extraídas em um único filtro de Bloom. Uma *feature* pode ser definida basicamente como um hash de um fragmento extraído do arquivo (sequência de bytes) de tamanho fixo (`sdhash`) ou variável (`MRSH-V2`), a depender da função de pareamento utilizada. Após o mapeamento de todas as features dos arquivos desejados (base de referência) na estrutura criada pela `MRSH-NET`, é possível consultar outros arquivos (conjunto de arquivos suspeitos) na estrutura por meio do mesmo processo, mas dessa vez consultando as *features* no filtro ao invés de inseri-las. Desta forma, ao encontrar um número (por padrão, o valor é seis) de *features* consecutivas contidas no filtro da estratégia, podemos definir que existe um arquivo inserido nela que é similar ao que estamos buscando. Contudo, a resposta da consulta é binária, sendo positiva (existe algum arquivo similar) ou negativa (não existe arquivo similar), não retornando qual é o arquivo da base de referência que possui similaridade.

Já a estratégia `MRSH-HBFT` utiliza o mesmo processo de extração de *features* do `MRSH-NET`, mas se diferencia na estrutura que é criada para armazenamento das *features*. Neste caso, é utilizada uma árvore binária cujos nós são compostos por filtros de Bloom: o nó raiz contém um filtro de Bloom que representa todo o conjunto de arquivos; os nós-folha contêm filtros de Bloom com um ou mais arquivos; a cada nível abaixo da raiz, o conjunto de arquivos é dividido pela metade e cada parte é inserida em um filtro de Bloom

filho. Isso ocorre até que reste apenas um único filtro com um só arquivo (nó folha), no qual metadados sobre o arquivo são adicionados para posterior identificação. Para realizar uma busca, o arquivo alvo tem suas *features* extraídas e consultadas na árvore; caso uma *feature* seja encontrada no filtro raiz, a busca procede nos demais níveis da árvore até que uma folha seja alcançada; caso contrário, a *feature* é descartada. Se um certo número de *features* (por padrão, seis) forem encontradas de forma consecutiva, definimos que o item consultado tem uma versão similar inserida na estrutura e podemos localizar qual é o item através dos metadados presentes no nó folha que obteve o número mínimo de *features* encontradas durante a busca.

Este trabalho propõe melhorias na estratégia `MRSH-HBFT`, rebatizada como `HBFT-SD`. Foi proposta a troca do processo de extração de *features* originalmente implementado usando o `MRSH-V2` pelo processo do `sdhash`, uma vez que este apresenta melhores resultados [Velho et al. 2020]. Esta troca também foi realizada na `MRSH-NET`, chamada doravante de `NET-SD`. Além disto, foram realizadas outras pequenas modificações no código para melhorar o desempenho da ferramenta, como por exemplo, o mapeamento de apenas um arquivo por folha, evitando uma comparação adicional quando uma busca chega em um filtro folha, bastando apenas que seja encontrado um certo número de *features* consecutivas para considerarmos uma comparação como similar (*match*). As implementações se encontram disponíveis no *GitHub*[1].

Por fim, este trabalho avalia o desempenho das estratégias `NET-SD` e `HBFT-SD` quando removemos os blocos comuns. Neste ponto é necessário esclarecer que, como as estratégias de busca de similaridade utilizam o conceito de *features*, em razão da precisão do termo, de agora em diante iremos nos referir aos blocos comuns como *features* comuns. A remoção das *features* comuns (implementada em cada uma das estratégias que são identificadas através do sufixo *NCF - No Common Features*) é realizada utilizando um banco de dados relacional (`SQLite3`) contendo todas as *features* de um conjunto de referência mapeado, similar à implementação descrita em [Moia et al. 2020a].

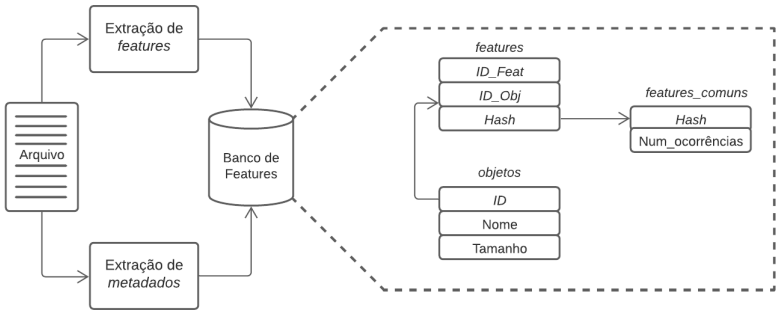

Figura 1. Estrutura do banco de dados de *features* utilizado para realizar a identificação de *features* comuns.

Na Fig. 1, apresentamos um esquema simplificado da estrutura do banco de dados utilizado para armazenar as informações dos arquivos utilizados no experimento relacionados às *features* comuns. Para cada arquivo analisado, são extraídos *features* e *metadados* que são armazenados nas tabelas do banco de dados *features* e *objetos*, respectiva-

[1] `https://github.com/regras/hbft-sd` e `https://github.com/regras/net-sd`

mente. A tabela *features* contém as informações de cada *feature* extraída dos arquivos, contendo um identificador (ID), valor (hash) e uma identificação do arquivo do qual foi extraída; a tabela *objetos* contem os metadados dos arquivos, como identificação (ID), nome e tamanho. Por fim, temos uma tabela chamada de *features_comuns*, a qual reúne todas as *features* da tabela *features* e o número de ocorrências de cada uma em todo o conjunto de arquivos que foi armazenado na base de dados. Durante a execução das estratégias, tanto no processo de criação da estrutura como na busca, uma *feature* extraída é consultada na base de dados (é verificado se ela existe na tabela *features_comuns*) e, caso ela seja encontrada em N diferentes arquivos (este valor é configurado previamente), esta *feature* é considerada comum e ignorada pela estratégia. O processo de inserção/consulta de uma feature no banco de dados é mostrado na Fig. 2.

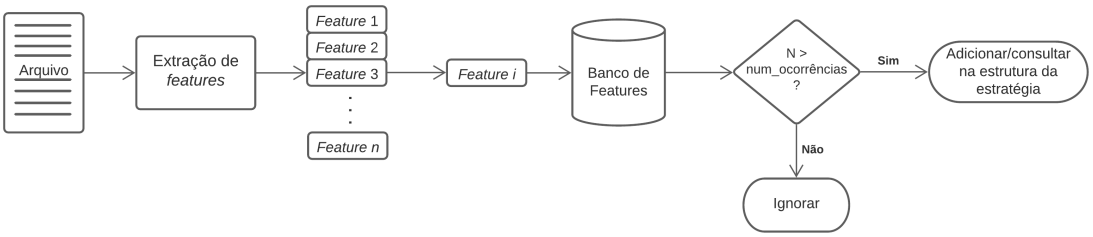

Figura 2. Inserção/Consulta de *feature* no banco de dados para se determinar se ela é comum (e deve ser descartada) ou não (e deve ser inserida/consultada).

4. Dados e métricas

Para avaliação do desempenho das estratégias, foram utilizados dois subconjuntos da base de arquivos *t5* [Roussev 2011]. A relação dos tipos de arquivos e suas quantidades se encontra disponível na Tabela 2. Para cada estratégia, utilizou-se o conjunto *known* como base de referência e inserção na estrutura criada e, em seguida, foram usados os arquivos do conjunto *target* para a realização das consultas.

Tabela 2. Relação dos tipos de arquivos e suas quantidades para os conjuntos de dados utilizados nos experimentos.

Conjunto	html	pdf	doc	xls	ppt	text	jpg	gif	Total
known	1066	882	464	193	240	654	351	61	3911
target	20	20	20	10	10	10	5	5	100

Para determinar o desempenho das estratégias, foi utilizada uma lista de pares de arquivos similares de acordo com os tipos de similaridades existentes entre os conjuntos *known* e *target*, apresentados na Tabela 3. Essa lista e a definição dos tipos de similaridade utilizados, foram extraídos do estudo [Moia et al. 2020a]. O tipo de similaridade *UGC (User Generated Content)* é relacionado ao conteúdo criado por um usuário, como um texto, tabela, figura de um documento. *TC (Template Content)* é uma similaridade também inserida pelo usuário mas que se repete em outros arquivos, como por exemplo, o template de um documento Word de uma empresa. Já o *AGC (Application Generated Content)* são os dados criados por aplicações e inseridos em um arquivo, como por exemplo, informações de cabeçalho. Neste trabalho são propostos dois cenários diferentes,

sendo que no primeiro, o perito forense está interessado apenas em similaridades do tipo *UGC* e *TC*, mas não do tipo *AGC* (que será considerado um falso positivo). No segundo cenário, o perito tem interesse em obter similaridade apenas do tipo *UGC*, e similaridades do tipo *TC* e *AGC* serão ambas falsos positivos.

Tabela 3. Número de comparações similares por tipo de similaridade entre os conjuntos de dados *known* e *target*.

Tipo de similaridade	Quantidade
UGC	39
AGC	79
TC	76

As estratégias `MRSH-NET` e `MRSH-HBFT`, assim como suas versões modificadas, apresentam o resultado de uma busca de duas formas diferentes. A `MRSH-NET` e derivados são capazes de indicar apenas se o arquivo consultado possui similaridade ou não com algum dos arquivos do conjunto mapeado em seu filtro de Bloom. Portanto elas fornecem uma resposta binária de pertencimento ou não ao conjunto ou não apenas. Já a `MRSH-HBFT` é capaz de detectar a similaridade e ainda indicar quais são os arquivos similares ao consultado. Portanto, é necessário interpretar os resultados de cada estratégia de forma diferente. Neste trabalho, nos referiremos às estratégias `MRSH-NET` e `NET-SD` como sendo do tipo `NET` e as estratégias `MRSH-HBFT` e `HBFT-SD` do tipo `HBFT`.

Ao avaliar as estratégias do tipo `NET`, como sua resposta é binária, um *match* (comparação entre dois arquivos considerada como similar) é considerado um verdadeiro positivo (*tp*), se na lista de referência o arquivo possui alguma similaridade do tipo *UGC* e/ou *TC* (dependendo do cenário considerado). Caso não haja ou seja de um tipo indesejável, será considerada um falso positivo (*fp*). Quanto às comparações dos arquivos que a estratégia não indicar similaridade ao conjunto mapeado, consideramos um falso negativo (*fn*) caso este arquivo tenha na lista de referência alguma similaridade dos tipos desejados, e verdadeiro negativo (*tn*) caso não conste na lista de referência ou conste, mas com similaridades indesejáveis.

Para a avaliação da estratégia `MRSH-HBFT`, a classificação dos resultados é feita da seguinte forma: comparações de arquivos que foram indicadas similares pela estratégia e que constem na lista de referência sendo similaridades dos tipos *UGC* e/ou *TC* (dependendo do cenário considerado) são *tp*, já as comparações indicadas que constarem como similaridade *AGC* (ou dos tipos indesejáveis) serão considerados *fp*. Quanto aos arquivos que não apresentarem nenhuma similaridade de acordo com a estratégia, se na lista este arquivo apresentar similaridade dos tipos desejáveis para um dado cenário, será considerado um *fn*, ou um *tn* se houver apenas comparações com este aquivo de similaridade dos tipos indesejáveis ou não constar nada na lista (para cada par de similaridade que conste na lista, e a ferramenta não encontre, será acrescido um *fn* ou *tn* na contagem).

Para avaliar o desempenho das estratégias, foram utilizadas três diferentes métricas recorrentes do cenário de busca de similaridade: *precision*, *recall* e *F-Score*.

$$precision = \frac{t_p}{t_p+f_p} \qquad recall = \frac{t_p}{t_p+f_n} \qquad F\text{-}Score = 2 \cdot \frac{precision \cdot recall}{precision+recall}$$

O *precision* pode ser descrito como a razão entre o número de pares realmente similares encontrados e o total de pares com alguma similaridade encontrados. Já o *recall* é a razão

entre a quantidade de pares realmente similares encontrados e o total de pares realmente similares que deveriam ser encontrados. Por fim, *F-Score* é a média harmônica entre as duas métricas, sendo útil para determinar o quão balanceada é a ferramenta.

Um último ponto que deve ser ressaltado são os parâmetros utilizados das estratégias utilizados durante os experimentos. Para as estratégias do tipo `NET` (`MRSH-NET`, `NET-SD` e `NET-SD-NCF`), devemos definir o tamanho do filtro de Bloom utilizado, o qual é calculado por meio da Eq. 1, proposta originalmente em [Breitinger et al. 2014a],

$$m = -\frac{k \cdot s \cdot 2^{14}}{ln(1 - \sqrt[k \cdot r_{min}]{p_f})} \tag{1}$$

onde, k representa a ordem do filtro, isto é, o número de funções hash que são calculados a partir da *feature*. No caso, utilizou-se $k = 5$, valor que representa um bom compromisso entre desempenho e tamanho do filtro. O parâmetro s é referente ao tamanho total dos arquivos da base a serem mapeados em MiB (a constante 2^{14} faz a conversão do tamanho total dos arquivos em MiB para o número esperado de *features* dos mesmos, considerando uma média de 160 *features* por $10KiB$ de arquivo) e r_{min} é o número mínimo de *features* consecutivas para considerarmos um *match* dentro do filtro (foi utilizado o valor padrão da estratégia de $r_{min} = 6$). Por fim, p_f representa a taxa de falsos positivos esperada no filtro, a qual foi definida como $p_f = 10^{-6}$. Portanto, como a base de arquivos que foi utilizada tem tamanho total de aproximadamente 580 MiB, o filtro necessário para representar todas as features deste conjunto de arquivos é de aproximadamente 8 MiB (devido a detalhes de implementação foi necessário arredondar para a potência de 2 superior mais próxima ao resultado obtido).

Quanto às estratégias `HBFT`, os dois principais parâmetros são o número de folhas da árvore e o tamanho do filtro raiz. Na implementação utilizada, é necessário que um nó folha contenha apenas um arquivo, ou seja, como temos 3911 arquivos e sabendo que a árvore é binária, o número de folhas necessário para armazenar todos os arquivos é 4096 (potência de 2 superior mais próxima do número de arquivos). Quanto ao tamanho do nó raiz, foi calculado primeiramente o tamanho dos nós folha (que terão todos o mesmo tamanho) com base no maior aquivo da base, e em seguida, multiplicou-se o resultado obtido pelo número de folhas da árvore. Este cálculo pode ser realizado já que a árvore criada tem a mesma quantidade de memória em todos os níveis distribuída entre os filtros de Bloom do nível. Logo, se o maior arquivo a ser mapeado possui cerca de $0,99MiB$, o filtro folha terá 10 *KiB* aproximadamente, e o filtro raiz totalizará 64 *MiB* (como são 10 *KiB* por folha, com 4096 folhas temos, arredondando novamente para a potência de 2 superior mais próxima, 64 *MiB* para o filtro raiz).

Por fim, o último parâmetro a ser definido é em relação à remoção de *features* comuns. Como mencionado na Sec. 3, as estratégias *NCF* consultam uma base de dados para verificar quantas vezes uma feature aparece ao longo do conjunto de arquivos e, caso este número seja maior do que N, esta *feature* é considerada comum e deve ser ignorada; caso contrário, a estratégia a utiliza para avaliação de similaridade. O valor de N deve ser configurado antes da execução da estratégia e nos experimentos realizados foram utilizados três valores diferentes, 3, 10 e 20, já que foram os valores que permitiram uma representação mais clara do impacto de se remover diferentes tipos de features comuns.

5. Experimentos e resultados

Nesta seção, apresentamos os experimentos feitos com as estratégias de busca de similaridade e os resultados obtidos. Inicialmente, são discutidos os experimentos das comparações dos conjuntos *known* e *target* sob as métricas de *precision*, *recall* e *F-Score* para as estratégias com e sem a remoção de *features* comuns; em seguida, vem o número mínimo de comparações necessárias durante o processo de busca para as estratégias do tipo `HBFT` (uma vez que estratégias do tipo `NET` requerem apenas uma comparação por item pesquisado, para definir se o arquivo está ou não inserido no conjunto); por fim, são apresentadas as medições de tempo das estratégias.

Destacamos que para a realização dos experimentos apresentados aqui, foi utilizado um computador i5-8300H 2.30 GHz com 8 GB de RAM rodando Ubuntu 20.04 LTS em um SSD *Crucial* BX500 de 480 GB. Para a medição do tempo, foi utilizado o comando *time*, somando o tempo de *system* com *user* para se obter o tempo efetivamente gasto na execução. Além disso, cada experimento foi executado dez vezes e foi obtida a média, tomando o cuidado de limpar o cache de todo o sistema entre cada execução.

5.1. Comparações entre os conjuntos de dados para avaliação das estratégias de busca de similaridade

De forma a se avaliar a capacidade de detecção das estratégias e analisar o impacto da remoção dos *features* comuns, foi realizada a comparação dos conjuntos *known*, uma lista de referência usada por um perito durante a investigação, e *target*, simulando o dispositivo apreendido pelo perito que requer análise. Para cada uma das estratégias sob avaliação neste trabalho foi criada uma estrutura em memória (filtro de Bloom para estratégias do tipo `NET` e árvore de filtros de Bloom para as do tipo `HBFT`) e mapeado o conjunto *known* nela. Então, foi utilizado o conjunto *target* para realização de consultas a fim de se verificar se os itens similares (já conhecidos e presentes na lista de referência - ver Sec. 4) são encontrados pelas estratégias. Para classificar os resultados, consideraremos dois cenários distintos, onde no cenário um são considerados apenas as similaridades dos tipos *UGC* e *TC* como verdadeiras, e no cenário dois apenas as similaridades do tipo *UGC* são as de interesse.

Os resultados das comparações entre os conjuntos para ambos os cenários são apresentados na Tabela 4. Aqui, mostramos o desempenho das estratégias sob as três métricas de interesse (*precision*, *recall* e *F-Score*). Os resultados enfatizam a melhoria apresentada ao se mudar o modelo de extração de *features* das estratégias, do *MRSH-V2* para o `sdhash`, em ambos os cenários considerados. É importante destacar que as *features* comuns ainda não foram removidos neste experimento. Como podemos analisar pelos resultados, o valor de *precision* está bem baixo, principalmente para as estratégias do tipo `HBFT`. Também vale mencionar que a grande diferença no valor de *precision* entre as estratégias do tipo `NET` e `HBFT` é causada principalmente pela maneira com que cada estratégia apresenta seus resultados, sendo a primeira uma resposta binária e a segunda, uma lista de potenciais candidatos similares ao item pesquisado, o que aumenta as quantidades de falsos positivos e negativos.

Em relação à mudança observada nos resultados do cenário um para o cenário dois, destacamos que se deve ao fato das similaridades de *TC* passarem de verdadeiros positivos (cenário um) para falsos positivos (cenário dois), impactando negativamente o

valor de *precision* das estratégias. Pouco (ou nenhum) efeito foi observado no valor de *recall*, uma vez que a capacidade de detecção das estratégias não foi afetada.

Tabela 4. Resultados das comparações entre os conjuntos *known* e *target* pelas estratégias de busca de similaridade sem a remoção de *features* comuns.

Estratégia	Cenário 1 - UGC + TC			Cenário 2 - UGC		
	precision	*recall*	*F-Score*	*precision*	*recall*	*F-Score*
MRSH-NET	0,480	1,000	0,649	0,210	1,000	0,347
NET-SD	0,564	1,000	0,721	0,247	1,000	0,396
MRSH-HBFT	0,006	0,965	0,012	0,002	0,949	0,004
HBFT-SD	0,012	0,991	0,024	0,004	1,000	0,008

Já na Tabela 5, apresentamos os resultados das comparações entre os conjuntos para ambos cenários somente para as versões das estratégias melhoradas, com e sem a remoção das *features* comuns para analisar o impacto desta remoção no desempenho das estratégias. Para as versões NCF (com remoção de *features* comuns), mostramos os resultados quando variamos o valor de N (parâmetro que define se uma *feature* é considerada comum), adotando os melhores valores encontrados e sugeridos no estudo de [Moia et al. 2020a]. É importante enfatizar que para valores baixos de N (3), a estratégia pode ser considerada rigorosa, pois qualquer *feature* com três ou mais ocorrências na base de dados de *features* (ou seja, que foi encontrada em três ou mais arquivos diferentes do conjunto *known*) é descartada. Já para valores mais elevados (10 ou 20), é exigido um número maior de repetições dentre os arquivos na base para que a *feature* seja considerada comum e então descartada. Com isto, quando aumentamos o valor desse parâmetro, esperamos que os valores de *precision* e de *recall* se tornem mais próximos daqueles apresentados pela estratégia que não considera *features* comuns. Deve ser ressaltado que é importante encontrar um equilíbrio para a escolha de N considerando ambas as métricas (o que pode ser observado com a ajuda da métrica *F-score*).

Como pode ser observado nos resultados, a remoção das *features* comuns traz algumas vantagens no processo, sendo a principal delas a redução no número de falsos positivos retornados em ambos cenários abordados. Para ambos tipos de estratégias, a melhora no *precision* é significativa. Contudo, podemos observar também uma queda no *recall*, principalmente para valores baixos de N (3), onde similaridades do tipo *TC* têm grande parte de suas *features* removidas (este tipo de similaridade é encontrada nos conjuntos de dados do experimento em alguns poucos arquivos), não sendo encontradas pelas estratégias devido a redução da similaridade entre os arquivos comparados. Este fato é corroborado pelo fato de que quando aumentamos o valor de N (10 ou 20), o valor do *recall* aumenta de forma significativa, pois as comparações do tipo *TC* não têm suas *features* removidas; entretanto, para este mesmo caso mas no cenário dois, o *precision* é reduzido como consequência, pois agora os *matches* de template permanecem entre as comparações (aqui, eles são comparações indesejáveis).

Também podemos comparar o impacto da remoção das *features* comuns entre os tipos de estratégias. Primeiro, é evidente que existe uma melhoria (em geral) ao se realizar este processo, demonstrado pelo *F-score* em ambos cenários e para os dois tipos de estratégias. Segundo, a melhoria é significativa para as estratégias do tipo HBFT devido ao seu formato de saída dos resultados, onde para cada consulta, uma lista com um ou mais

itens candidatos a similaridade é retornada; isso sugere que, para um grande número de comparações, a remoção das *features* comuns se torna ainda mais evidente, e esperamos resultados mais expressivos. Por último, destacamos o papel do valor de N, que deve ser escolhido de acordo com o cenário abordado e também levando em consideração a métrica (*precision* ou *recall*) de maior interesse em uma investigação, que poderia variar de acordo com o caso abordado.

Tabela 5. Resultados das comparações entre os conjuntos *known* e *target* pelas estratégias de busca de similaridade com a remoção de *features* comuns.

Estratégia	N	Cenário 1 - UGC + TC			Cenário 2 - UGC		
		precision	*recall*	*F-Score*	*precision*	*recall*	*F-Score*
NET-SD	–	0,564	1,000	0,721	0,247	1,000	0,396
NET-SD-NCF	3	0,661	0,937	0,775	0,294	0,952	0,449
	10	0,635	0,979	0,770	0,284	1,000	0,442
	20	0,626	0,979	0,764	0,280	1,000	0,437
HBFT-SD	–	0,012	0,991	0,024	0,004	1,000	0,008
HBFT-SD-NCF	3	0,369	0,443	0,403	0,181	0,641	0,282
	10	0,264	0,886	0,407	0,101	1,000	0,183
	20	0,188	0,991	0,316	0,064	1,000	0,121

Apesar dos dados apresentados na Tabela 5 apontarem um valor de *F-score* relativamente baixo para todas as estratégias, é evidente que a remoção das *features* comuns contribuiu para a melhoria da precisão, uma vez que as estratégias, mesmo em suas versões originais, são muito sensíveis na detecção de similaridade e acabam gerando vários falsos positivos. Como impacto prático, a remoção das *features* comuns acaba diminuindo significativamente o trabalho manual de peritos, que evitarão analisar arquivos (falso suspeitos) desnecessariamente; portanto, qualquer melhoria na precisão das estratégias de busca de similaridade apresenta um grande impacto dos profissionais desta área. Contudo, uma calibração nestas estratégias é um dos passos futuros deste trabalho para tentar ajustar os parâmetros internos das estratégias (por exemplo r_{min}) que, combinados com os parâmetros de remoção de *features* comuns (N), consigam melhorar ainda mais a precisão dos resultados.

5.2. Número de comparações dos conjuntos *known* e *target* de cada estratégia de busca de similaridade

Durante a busca de similaridade, as estratégias retornam uma resposta binária indicando a presença ou não de um item (pertencente ao conjunto *known*) similar ao consultado (pertencente ao conjunto *target*), nas estratégias do tipo NET, ou uma lista de arquivos cuja similaridade com o item buscado é maior que um valor pré-definido (um número mínimo r_{min} de *features* consecutivas encontradas, no caso das estratégias do tipo HBFT). Logo, como temos apenas 100 arquivos no conjunto *target*, as estratégias NET executam exatamente 100 buscas de similaridade, enquanto as estratégias do tipo HBFT realizam 391100 ($100 * 3911$) comparações.

Para as estratégias do tipo HBFT, podemos avaliar quantas comparações são retornadas de forma a se ter uma noção melhor da métrica *precision* e observar a redução do número de comparações realizadas quando removemos as *features* comuns. Os números

de pares realmente similares são apresentados na Tabela 3. Vale destacar que o número de similaridades apresentado independe do cenário considerado (um ou dois), pois o que muda é a forma de interpretação dos resultados. Neste caso, para o cenário um, temos 115 (39+76) similaridades, enquanto no cenário dois, apenas 39. Como pode ser observado pelos resultados, a remoção de *features* comuns ajuda a reduzir o número de similaridades retornadas de forma significativa, principalmente para valores de N menores.

Tabela 6. Número de similaridades retornadas pelas estratégias do tipo `HBFT`.

Estratégias	`MRSH-HBFT`	`HBFT-SD`	`HBFT-SD-NCF`		
			$N = 3$	$N = 10$	$N = 20$
Num. de comparações	16.201	9.377	138	385	605

5.3. Tempos de execução das estratégias

A terceira análise realizada neste trabalho é em relação ao tempo de execução de cada uma das estratégias durante a busca por similaridade. Na Tabela 7, apresentamos os tempos médios (em segundos) obtidos na execução das estratégias (e suas versões melhoradas) em duas fases da busca: preparação e operação. Além destas duas medidas, também apresentamos a diferença percentual entre as estratégias (coluna $\Delta(\%)$), a qual representa a diferença de tempo entre duas linhas adjacentes. Apesar de não constar na tabela, também foi calculado o desvio padrão das várias execuções o qual variou de $0,02$ a $2,89$. Para os experimentos desta seção, consideramos o valor de N=20 por apresentar melhores resultados para *recall*; contudo, os resultados são similares (um pouco menores) para valores N = 10 e N = 3.

A primeira fase consiste na criação da estrutura pela estratégia e inserção de todos os arquivos do conjunto *known*, incluindo o tempo de extração de *features* e mapeamento na estrutura. A criação da base de dados de *features* não está contabilizada neste processo, pois é uma etapa computacionalmente custosa, que pode ser realizada antes de qualquer investigação e reaproveitada para quaisquer outras buscas que venham a ser realizadas. Já a fase de operação consiste no tempo total necessário para consultar todos os 100 arquivos do conjunto *target* na estrutura, considerando apenas o tempo de extração de *features* e pesquisa. Enfatizamos que para as estratégias `NCF` foram considerados em ambas as fases os tempos de consultas das *features* extraídas dos arquivos na base de dados, para verificação se determinada *feature* é comum e deve ser desconsiderada.

Tabela 7. Tempos para as fases de preparação e operacional e o tempo médio por consulta com os conjuntos *target* e *known* para N = 20

Estratégia	**Tempo de execução (segundos)**			
	Fase de preparação	$\Delta(\%)$	**Fase operacional**	$\Delta(\%)$
`MRSH-NET`	$22,5$	-	$2,7$	-
`NET-SD`	$19,5$	$-13,3$	$2,4$	$-11,1$
`MRSH-HBFT`	$30,6$	-	$17,1$	-
`HBFT-SD`	$25,9$	$-15,4$	$10,6$	$-37,4$
`NET-SD-NCF`	$119,7$	-	$22,9$	-
`HBFT-SD-NCF`	$143,6$	$+20,0$	$20,2$	$-11,8$

Conforme mostrado na Tabela 7, as versões com `sdhash` (sufixo `SD`) apresentaram um desempenho melhor do que as versões originais, tanto na fase de preparação quanto na fase de operação. Esta melhora demonstra que o novo módulo de extração de *features* (do `sdhash`) é mais rápido do que o anterior, permitindo que o tempo necessário para mapear um arquivo na estrutura, ou buscá-lo, seja menor do que a versão original. Ao analisar o tempo das estratégias ao remover as *features* comuns (`NCF`), notamos um aumento expressivo no tempo que varia quase duas até mais de nove vezes o tempo em relação as versões modificadas para os dois tipos. Este aumento se deve ao fato de que para cada *feature* extraída de um arquivo, é realizada uma consulta no banco de *features* para verificar se esta *feature* é comum (e deve ser descartada) ou não; este processo é realizado em ambas as fases. Por fim, nota-se o maior tempo gasto na fase de preparação pelas estratégias da família `HBFT` em relação à `NET`. Isto se deve ao fato de que as estratégias `HBFT` necessitam inserir todas as *features* na árvore de filtros de Bloom, enquanto as `NET` o fazem em apenas um filtro; consequentemente o tempo é menor para esta última.

Embora os tempos com as estratégias `NCF` sejam superiores, acreditamos que o custo computacional extra exigido é válido devido ao benefício que estas estratégias trazem para um perito, pois diminui o trabalho manual que ele deverá realizar para analisar os vários casos de falsos positivos retornados. Como mostrado na Tabela 6, o número de comparações quando se utiliza a remoção de *features* comuns cai $98,5\%$ (N = 3) ou $93,5\%$ (N = 20), valores de grande impacto em uma investigação com muitos arquivos.

6. Conclusões e trabalhos futuros

Neste trabalho, foram avaliadas as estratégias de busca de similaridade `MRSH-NET` e `MRSH-HBFT` e versões aprimoradas destas estratégias. O principal foco das avaliações foi o impacto da remoção dos blocos (ou mais precisamente, das *features*) comuns encontrados em diferentes arquivos do mesmo tipo ou não. Com isso, observou-se um aumento significativo na taxa de *precision* com uma pequena (em alguns casos nula) redução no *recall*, a depender do critério adotado para considerar um bloco como comum e do cenário considerado. Estes benefícios foram atingidos com um custo adicional de tempo de execução das estratégias. Este tempo maior de processamento é plenamente aceitável pela economia de tempo que as estratégias propostas irão proporcionar aos investigadores, já que elas reduzam de forma significativa o número de similaridades entre arquivos que são falsos positivos e exigiriam uma análise manual mais detalhada. Como trabalhos futuros, pretendemos investigar outros parâmetros de funcionamento das estratégias e verificar como refiná-los para se obter melhores resultados quando removemos os blocos comuns, além de adicionar outras estratégias à comparação. A investigação de outras formas de melhorar a eficiência do processo de remoção de blocos comuns também é algo que planejamos como futuro tópico de estudo.

Agradecimentos

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

Referências

Breitinger, F. and Baier, H. (2013). Similarity preserving hashing: Eligible properties and a new algorithm mrsh-v2. In *Digital Forensics and Cyber Crime: 4th Internati-*

onal Conference, ICDF2C 2012, Lafayette, IN, USA, pages 167–182. Springer Berlin Heidelberg.

Breitinger, F., Baier, H., and White, D. (2014a). On the database lookup problem of approximate matching. *Digital Investigation*, 11:S1–S9.

Breitinger, F., Guttman, B., McCarrin, M., Roussev, V., and White, D. (2014b). Approximate matching: definition and terminology. *NIST Special Publication*, 800:168.

Kornblum, J. (2006). Identifying almost identical files using context triggered piecewise hashing. *Digital investigation*, 3:91–97.

Lillis, D., Breitinger, F., and Scanlon, M. (2017). Expediting mrsh-v2 approximate matching with hierarchical bloom filter trees. In *International Conference on Digital Forensics and Cyber Crime*, pages 144–157. Springer.

Moia, V. H. G., Breitinger, F., and Henriques, M. (2020a). Understanding the effects of removing common blocks on approximate matching scores under different scenarios for digital forensic investigations. *XIX Brazilian Symposium on information and computational systems security, Brazilian Computer Society (SB)*.

Moia, V. H. G., Breitinger, F., and Henriques, M. A. A. (2020b). The impact of excluding common blocks for approximate matching. *Computers & Security*, 89:101676.

Moia, V. H. G. and Henriques, M. A. A. (2017). Similarity digest search: A survey and comparative analysis of strategies to perform known file filtering using approximate matching. *Security and Communication Networks*, pages 1–17.

Oliver, J., Cheng, C., and Chen, Y. (2013). TLSH–a locality sensitive hash. In *Cybercrime and Trustworthy Computing Workshop (CTC), 2013 Fourth*, pages 7–13. IEEE.

Raff, E. and Nicholas, C. (2018). Lempel-ziv jaccard distance, an effective alternative to ssdeep and sdhash. *Digital Investigation*, 24:34–49.

Roussev, V. (2010). Data fingerprinting with similarity digests. In *IFIP International Conf. on Digital Forensics*, pages 207–226. Springer.

Roussev, V. (2011). An evaluation of forensic similarity hashes. *Digital investigation*, 8:34–41.

Velho, J. P. B., Moia, V. H. G., and Henriques, M. A. A. (2020). Entendendo e melhorando a capacidade de detecção de estratégias de busca de similaridade em investigações forenses. *XX Brazilian Symposium on information and computational systems security, Brazilian Computer Society (SB)*.

Winter, C., Schneider, M., and Yannikos, Y. (2013). F2s2: Fast forensic similarity search through indexing piecewise hash signatures. *Digital Investigation*, 10(4):361–371.