XXIX Congresso de Iniciação Científica UNICAMP 2021

Aperfeiçoamento do processo de busca de similaridade usando funções de pareamento aproximado para investigações forenses

João P. B. Velho, Vitor H. G. Moia, Marco A. A. Henriques

Resumo

A grande quantidade de arquivos presentes em dispositivos é um dos principais problemas com os quais as investigações forenses lidam. Neste trabalho são propostas melhorias nas estratégias de busca de similaridade para analisar dispositivos em busca de arquivos similares ou idênticos a um conjunto de arquivos. No entanto, um fator que prejudica a busca de similaridade são os blocos comuns, os quais podem ser caracterizados como partes do arquivo, como cabeçalhos, presentes em vários arquivos e que não contribuem para a investigação. O trabalho avalia ainda o impacto da remoção destes blocos na performance das estratégias. Os resultados mostram que há uma redução considerável na taxa de falsos positivos, ao mesmo tempo que um aumento no tempo de execução causados pela remoção de tais blocos comuns. Entretanto, tais aumentos de tempo são mais que compensados pela diminuição significativa do número de arquivos suspeitos que os investigadores terão que focar suas análises mais profundas.

Keywords: Forense Digital, Busca de Similaridade, Funções de Pareamento Aproximado, Estratégias de Busca de Similaridade, Blocos Comuns

1 Contextualização

A grande quantidade de arquivos, presentes em dispositivos eletrônicos, é um grande problema para os peritos de forense digital, já que estes, ao longo de uma investigação, devem realizar a análise destas mídias em busca de arquivos suspeitos. 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.

Para tornar possível a busca de arquivos similares, ao invés de utilizar-se funções hash, passa-se a utilizar funções de pareamento aproximado. As funções deste tipo geram um resumo do arquivo que, ao ser comparado com outro resumo, determina 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] e `ssdeep` [Kornblum 2006], dentre outras. Mais informações sobre as funções de pareamento aproximado podem ser encontradas em um report do NIST [Breitinger et al. 2014b].

Uma desvantagem que se torna evidente quando as funções de pareamento aproximado lidam com grandes volumes de dados é o tempo gasto para comparar todos os resumos de similaridade, podendo tornar-se impraticável. Tendo este problema em vista, um método de comparação mais rápido e com a mesma qualidade na determinação da similaridade de arquivos passa a ser necessário.

Para que seja possível lidar com muitos arquivos, as funcionalidades das funções de pareamento aproximado foram expandidas para as estratégias de busca de similaridade. Estas estratégias utilizam alguns conceitos e processos que as funções de pareamento utilizam para realizar a comparação dos arquivos de forma mais eficiente. Alguns exemplos de estratégias encontradas na literatura são `MRSH-NET` [Breitinger et al. 2014a] e `MRSH-HBFT` [Lillis et al. 2017].

O procedimento para manipular os resumos de similaridade e realizar a comparação destes de forma mais eficiente é característico de cada estratégia, podendo, por exemplo, ser utilizadas tabelas hash [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.

Para tornar a busca de similaridade mais eficiente, podemos nos voltar a um outro problema das estratégias: a alta taxa de falsos positivos, causada pela presença dos blocos comuns nos arquivos analisados [Moia et al. 2020b]. 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.

2 Contribuição

Para compreendermos a contribuição deste trabalho, primeiro é necessário selecionar as estratégias que serão avaliadas. Neste estudo foram selecionadas: `MRSH-NET` [Breitinger and Baier 2013] e `MRSH-HBFT` [Lillis et al. 2017]. A primeira estratégia realiza a extração de *features* de cada arquivo (usando a função de pareamento aproximado `MRSH-V2`) e insere todas as *features* extraídas em um único filtro de Bloom. Neste contexto, uma *feature* é basicamente o hash de uma parte do arquivo (sequência de bytes) de tamanho variável que foi selecionado pela estratégia . Após o mapeamento de todas as features dos arquivos desejados (base de referência) na estrutura criada pelo `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, 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 similar na base de referência.

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 contem 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) consecutivas for encontrado, 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`, cuja implementação foi batizada de `HBFT-SD`. Este nome de deve à troca do processo de extração de *features* originalmente implementado usando o `MRSH-V2` pelo processo do `sdhash`, uma vez que este apresenta melhor desempenho [Moia and Henriques 2017]. Esta troca também foi realizada no `MRSH-NET`, que será referido neste trabalho como `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*). Ambas implementações se encontram disponíveis no *GitHub*[1]. Por fim, este trabalho avalia o desempenho das duas estratégias (`NET-SD` e `HBFT-SD`) quando removemos os blocos comuns dos resumos criados dos arquivos. A remoção dos blocos (ou *features*) gera versões com o sufixo *NCF - No Common features*, e 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. 2020b].

Para cada arquivo analisado, são extraídos *features* e *metadados* que são armazenados em tabelas do banco de dados. A tabela *features* contém as informações de cada *feature* extraída dos arquivos, com 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 ar-

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

quivos 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.

3 Dados e métricas

Para avaliação do desempenho das estratégias, foram utilizados dois subconjuntos da base de arquivos *t5* [Roussev 2011]. 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.

Para determinar o desempenho das estratégias, foi utilizada uma lista contendo os pares de arquivos similares de acordo com o tipo da similaridade existentes entre os conjuntos *known* e *target* . Esta lista, assim como também a definição dos tipos de similaridade utilizados aqui, 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.

As estratégias `NET-SD` e `HBFT-SD`, apresentam o resultado de uma busca de duas formas diferentes. A `NET-SD` é capaz de indicar apenas se o arquivo consultado possui similaridade ou não com algum dos arquivos do conjunto mapeado em seu filtro de Bloom; portanto, ela apenas fornece uma resposta binária de pertencimento ao conjunto ou não. Já a `HBFT-SD` 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, como detalhado a seguir.

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 `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, 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*.

$$precision = \frac{t_p}{t_p+f_p} \qquad recall = \frac{t_p}{t_p+f_n}$$

$$F\text{-}Score = 2 \cdot \frac{precision \cdot recall}{precision+recall}$$

A *precision* pode ser descrita como a razão entre o número de pares realmente similares de acordo com um determinado critério encontrados e o total de pares com alguma similaridade apontada pela estratégia. Já o *recall* é a razão entre a quantidade de pares realmente similares encontrados e o total de pares realmente similares existentes na base e 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. As informações sobre parâmetros e configurações utilizadas se encontram nos repositórios referenciados.

4 Experimentos e resultados

Nesta seção, apresentamos os experimentos realizados com as estratégias de busca de similaridade e os resultados obtidos. De forma a se avaliar a capacidade de detecção das estratégias e analisar

Tabela 1: 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

o impacto da remoção dos blocos 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 e 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) são encontrados pelas estratégias. Para classificar os resultados, consideraremos dois cenários distintos, onde no cenário um são consideradas apenas as similaridades dos tipos *UGC* e *TC* como verdadeiras, e no cenário dois apenas as similaridades do tipo *UGC* são as de interesse.

Na Tabela 1, 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 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.

Em contrapartida, o *recall* tende a ser maior para valores de N maiores, já que pares de arquivos que apresentam um pequeno trecho similar (que para valores de N menores não eram encontrados pelas estratégias) passam a ser encontrados pelas estratégias. 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 à redução da similaridade entre os arquivos comparados. Isto 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 produz 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.

Foram realizados testes para avaliar o tempo de execução também, utilizando o comando `time` e coletando a soma dos tempos *system* e *user*. A tabela contendo os dados se encontra no repositório referenciado (no arquivo `times.md`). Conhecendo os tempos de execução de cada estratégia foi possível concluir que a melhora nas métricas quando passamos a remover blocos comuns é acompanhada de um aumento considerável no tempo de execução, já que é necessária uma consulta no banco de dados a cada *feature* consultada. No entanto, este aumento é totalmente aceitável, já que, ao reduzir a taxa de falsos positivos, o trabalho do perito de verificar manualmente cada par de arquivo potencialmente similar é reduzido significativamente.

5 Conclusões e trabalhos futuros

Ao longo do projeto, foram avaliadas duas estratégias aprimoradas de busca de similaridade, `NET-SD` e `HBFT-SD`. Focamos no impacto da remoção dos blocos comuns encontrados em diferentes arquivos, sendo eles do mesmo tipo (*pdf*, *txt*, *html* etc) ou não. Com esta análise, foi observado um aumento significativo na taxa de *precision* com uma pequena redução no *recall*, a depender do critério adotado para considerar um bloco como comum e do cenário considerado. A combinação dessas duas métricas em uma terceira, F-Score, mostrou uma melhoria significativa no desempenho das estratégias analisadas, o que revela um melhor equilíbrio entre as duas após a remoção de blocos comuns. Esta melhoria foi acompanhada de um custo adicional de tempo de execução das estratégias. Contudo, este tempo maior de processamento é aceitável pela economia de tempo que as estratégias propostas irão proporcionar aos investigadores, já que elas reduzem de forma significativa o número de similaridades entre arquivos que são falsos positivos e exigiriam uma análise manual. Futuramente, pretendemos investigar outros parâmetros das estratégias, como o número mínimo de *features* consecutivas mencionado, e avaliar como refiná-los para se obter melhores resultados quando removemos os blocos comuns, além de adicionar outras estratégias à comparação, como a F2S2 [Winter et al. 2013].

Referências

[Breitinger and Baier 2013] Breitinger, F. and Baier, H. (2013). Similarity preserving hashing: Eligible properties and a new algorithm mrsh-v2. In *Digital Forensics and Cyber Crime: 4th International Conference, ICDF2C 2012, Lafayette, IN, USA, October 25-26, 2012, Revised Selected Papers*, pages 167–182, Berlin, Heidelberg. Springer Berlin Heidelberg.

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

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

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

[Lillis et al. 2017] 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 et al. 2020a] 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 et al. 2020b] 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 and Henriques 2017] 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.

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

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

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