Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

Mineração de Dados Aplicada à Construção de Bases de *Hash* em Computação Forense

MARCELO CALDEIRA RUBACK

ORIENTADORA: CÉLIA GHEDINI RALHA

CO-ORIENTADOR: BRUNO WERNECK PINTO HOELZ

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA
ÁREA DE CONCENTRAÇÃO INFORMÁTICA FORENSE E SEGURANÇA DA INFORMAÇÃO

PUBLICAÇÃO: PPGENE.DM - 84 A/11

BRASÍLIA / DF: Dezembro/2011

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

MINERAÇÃO DE DADOS APLICADA À CONSTRUÇÃO DE BASES DE *HASH* EM COMPUTAÇÃO FORENSE

MARCELO CALDEIRA RUBACK

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE PROFISSIONAL EM INFORMÁTICA FORENSE E SEGURANÇA DA INFORMAÇÃO.

APROVADA POR:

CÉLIA GHEDINI RALHA, Ph.D., UnB
(ORIENTADORA)

HELVIO PEREIRA PEIXOTO, Ph.D., DPF
(EXAMINADOR INTERNO)

ANA PAULA APPEL, Doutora, UFES
(EXAMINADOR EXTERNO)

DATA: BRASÍLIA/DF, 13 DE DEZEMBRO DE 2011.

FICHA CATALOGRÁFICA

RUBACK, MARCELO CALDEIRA
Mineração de Dados Aplicada à Construção de Bases de Hash em Computação Forense [Distrito Federal] 2011.
xxx, 169p., 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, Ano).

Dissertação de Mestrado– Universidade de Brasília, Faculdade de Tecnologia. Departamento de Engenharia Elétrica.

1. Computação Forense 2. Mineração de Dados
3. Árvores de Decisão 4.Conjuntos de *hash*

I. ENE/FT/UnB. II. Título (Série)

REFERÊNCIA BIBLIOGRÁFICA

RUBACK, M. C. (2011). Mineração de Dados Aplicada à Construção de Bases de *Hash* em Computação Forense. Dissertação de Mestrado, Publicação PPGENE.DM - 84 A/11, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 169p.

CESSÃO DE DIREITOS

NOME DO AUTOR: Marcelo Caldeira Ruback
TÍTULO DA DISSERTAÇÃO: Mineração de Dados Aplicada à Construção de Bases de *Hash* em Computação Forense.
GRAU/ANO: Mestre/2011.

Marcelo Caldeira Ruback
SQN 214, Bloco J, Apto 211 – Asa Norte
CEP 70873-100 – Brasília – DF - Brasil

À Sofia, Júlia e Maria Amélia, amores da minha vida.

AGRADECIMENTOS

Agradeço primeiramente a Deus, que esteve e está sempre presente iluminando o meu caminho.

À Profª. Célia, pela enorme dedicação, atenção e apoio; que desde o princípio incentivou o desenvolvimento deste trabalho e esteve sempre presente para ajudar a superar os desafios encontrados.

Aos colegas do SEPINF, especialmente o Bruno Werneck, co-orientador deste trabalho, por terem contribuído com ideias fundamentais para a realização deste projeto.

Aos colegas de turma do Mestrado, com quem dividi as angústias, os ensinamentos e também momentos de alegria.

Aos organizadores do projeto de Mestrado, pelo esforço e dedicação para contribuir com a melhoria da formação dos peritos criminais no Brasil.

A todos, que de alguma forma, tenham me incentivado ou ajudado durante esse árduo caminho até a conclusão do Mestrado.

Finalmente, agradeço aos meus queridos pais, pelo apoio e amor incondicionais e exemplo de vida a ser seguido e às minhas irmãs, que, mesmo distantes, transmitiram todo seu carinho e atenção.

O presente trabalho foi realizado com o apoio do Departamento de Polícia Federal, com recursos do Programa Nacional de Segurança Pública com Cidadania – PRONASCI, do Ministério da Justiça.

RESUMO

MINERAÇÃO DE DADOS APLICADA À CONSTRUÇÃO DE BASES DE *HASH* EM COMPUTAÇÃO FORENSE

Autor: Marcelo Caldeira Ruback
Orientadora: Célia Ghedini Ralha
Programa de Pós-graduação em Engenharia Elétrica
Brasília, dezembro de 2011

A grande quantidade de dados a serem processados e analisados por peritos em informática é um desafio crescente enfrentado pela comunidade de Computação Forense. O uso de conjuntos de *hashes* de arquivos conhecidos para identificar e filtrar arquivos irrelevantes é um procedimento amplamente utilizado, mas não é tão eficaz quanto poderia ser, especialmente em países cujo idioma não seja o inglês. Este trabalho propõe o uso de técnicas de mineração de dados com algoritmos de classificação baseados em árvores de decisão para encontrar novos arquivos irrelevantes para a análise forense a partir de uma amostra de computadores de uma dada região ou país. Os *hashes* dos arquivos identificados podem ser mesclados com um subconjunto selecionado de *hashes* que sejam realmente efetivos, escolhido a partir de bases de *hashes* convencionais. Os experimentos foram conduzidos para avaliar o desempenho de filtragem da solução proposta, usando amostras de evidências extraídas de casos reais. Esses experimentos demonstraram que a abordagem proposta obteve resultados de filtragem entre 15% e 30% melhores do que a base de *hash* convencional utilizada, mesmo contendo um número reduzido de valores de *hash*. Este trabalho lança luz sobre uma técnica forense comumente utilizada, mas que tem sido relegada ao uso de bases de dados em constante crescimento composto apenas de *hashes* de arquivos conhecidos cuja origem seja rigidamente rastreável. A adoção de novas soluções para lidar com bancos de dados de *hash* em Computação Forense é uma boa oportunidade para introduzir técnicas inteligentes para melhorar a forma como conjuntos de *hashes* são criados, mantidos e utilizados.

ABSTRACT

DATA MINING APPLIED TO HASHSETS CONSTRUCTION IN COMPUTER FORENSICS

Author: Marcelo Caldeira Ruback
Supervisor: Célia Ghedini Ralha
Programa de Pós-graduação em Engenharia Elétrica
Brasília, December of 2011

The large amount of data to be processed and analyzed by forensic experts is a growing challenge faced by the computer forensics community. The use of hashsets of known files to identify and filter irrelevant files is a commonly used technique, but it is not as effective as it could be, especially in non-English speaking countries. This work proposes the use of data mining techniques with decision tree learning algorithms to find new irrelevant files to the forensic analysis from a sample of computers from a given region or country. The resulting files can be merged with an optimized subset of really effectively used hashsets, which are chosen from conventional hash databases. Experiments were conducted to evaluate the filtering performance of the proposed solution, using samples from real evidence. The experimental results demonstrate that our approach obtained between 15% to 30% better filtering results than the conventional hashset database used as benchmark, even with a reduced number of hash values. This work sheds light on a commonly used forensics technique, which has relied on the use of ever-growing databases composed only of hashes from rigidly traceable known files. The adoption of new solutions to deal with hash databases in computer forensics is a good opportunity to introduce intelligent techniques to improve the way hashsets are created, maintained and used.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SÍMBOLOS, NOMENCLATURA E ABREVIAÇÕES

AC Amostra de Computadores

AD Árvore de Decisão

AES *Advanced Encryption Standard*

BD Banco de Dados

BDAE Base de Dados de Arquivos Examinados

BHAC Base de *Hashes* de Arquivos Conhecidos

BHP Base de *Hashes* Proposta

CBC *Cipher Block Chaining*

CCIPS *Computer Crime and Intellectual Property Section*

CRISP-DM *Cross Industry Standard Process for Data Mining*

CRHF *Collision Resistant Hash Function*

CTPH *Context Triggered Piecewise Hashing*

DCBD Descoberta de Conhecimento em Bases de Dados

DDL *Data Definition Language* (Linguagem de Definição de Dados)

DML *Data Manipulation Language* (Linguagem de Manipulação de Dados)

E-R Entidade-Relacionamento

ETL *Extraction, Transformation, Loading*

FBI *Federal Bureau of Investigation*

FTK *Forensic ToolKit*

GPU *Graphical Processing Unit*

IA Inteligência Artificial

KFF *Known File Filter*

KRDS *Korean Reference Data Set*

MACs *Message Authentication Codes*

MD Mineração de Dados

MD4 *Message-Digest Algorithm 4*

MD5 *Message-Digest Algorithm 5*

MDCs *Modification Detection Codes*

NDIC *National Drug Intelligence Center*

NIST *National Institute of Standards and Technology*

NSA *National Security Agency*

NSRL *National Software Reference Library*

OWHF *One-Way Hash Function*

PF Polícia Federal

RDS *Reference Data Set*

SCH Seleção de Conjuntos de *Hashes*

SGBD Sistema de Gerenciamento de Banco de Dados

SHA *Secure Hash Algorithm*

SHA-0 *Secure Hash Algorithm-0*

SHA-1 *Secure Hash Algorithm-1*

SHA-2 *Secure Hash Algorithm-2*

SHA-3 *Secure Hash Algorithm-3*

SHA-224 *Secure Hash Algorithm-224 bits*

SHA-256 *Secure Hash Algorithm-256 bits*

SHA-384 *Secure Hash Algorithm-384 bits*

SHA-512 *Secure Hash Algorithm-512 bits*

SISCRIM Sistema Criminalística da Polícia Federal

SO Sistema Operacional

SQL *Structured Query Language*

STS *Statistical Test Suite*

1. INTRODUÇÃO

O grande volume de dados armazenados e disponíveis atualmente através de sistemas computacionais, característica da atual Era da Informação, trouxe grandes avanços para nossa sociedade por meio da maior socialização do conhecimento. Porém, essa grande disponibilidade de informações trás consigo o grande desafio de como conseguir extrair informação útil e gerar conhecimento em meio a um universo tão grande de dados. De acordo com Hinshaw (2004), com o advento da Internet e de sistemas de rastreamento de dados, os volumes de dados armazenados vêm dobrando a cada nove meses, o que representa um tempo duas vezes menor que o previsto pela Lei de Moore (Moore, 1965). E a Computação Forense é uma área do conhecimento diretamente afetada por esse contexto justamente por ter como principal finalidade a análise e correlacionamento de dados armazenados em sistemas computacionais – oriundos dos mais diversos segmentos da sociedade – sob os quais recai a suspeita de conterem evidências da prática de crimes.

A proposta deste trabalho é projetar um *framework* que possibilite a criação de uma base de dados contendo informações a respeito de arquivos extraídas de casos reais de análises periciais realizadas no âmbito da Polícia Federal (PF) e implementar técnicas de Mineração de Dados (MD) sobre esses dados para encontrar arquivos potencialmente irrelevantes para exames periciais. O objetivo principal é diminuir o volume de dados a serem examinados em análises periciais futuras por meio da filtragem automática de arquivos com perfis que se mostraram irrelevantes em exames periciais já realizados. A solução proposta utiliza algoritmos de indução de Árvores de Decisão (AD) para obter uma classificação a respeito da relevância potencial dos arquivos utilizando como atributos os seus metadados e seu relacionamento com outros arquivos da base.

À base de dados de arquivos examinados (BDAE) serão acrescidas informações sobre arquivos reconhecidamente irrelevantes ou potencialmente suspeitos obtidas de bases de *hashes* de arquivos conhecidos (BHACs) tradicionais, de modo que essa base consolidada deverá funcionar como um repositório centralizado a respeito de arquivos que possam vir a ser utilizados para filtragem automática em exames periciais.

A base de *hashes* proposta (BHP) será composta pelos *hashes* dos arquivos classificados como irrelevantes pelo processo de MD e por uma seleção de *hashes* extraída de BHACs convencionais.

O que se propõe, portanto, é adotar uma nova visão a respeito de BHACs e introduzir um pouco mais de inteligência ao processo de criação e manutenção, uma vez que o

procedimento de identificação de arquivos conhecidos está aparentemente relegado pela comunidade de Computação Forense ao uso de bases de *hash* organicamente crescentes de arquivos cuja origem seja rigidamente rastreável.

A Seção 1.1, apresenta o contexto atual da Computação Forense e as perspectivas futuras que motivaram a realização deste trabalho; na Seção 1.2 são apresentados os principais objetivos a serem alcançados neste trabalho e a Seção 1.3 descreve a metodologia utilizada.

1.1. APRESENTAÇÃO DO PROBLEMA

Com a disseminação do uso de equipamentos computacionais e a substituição gradativa, crescente e irreversível de documentos em papel pelo armazenamento eletrônico em mídias, a tarefa de encontrar vestígios de crimes em meios de armazenamento digital está se tornando cada vez mais complexa. A grande quantidade de informações digitais apreendidas em operações policiais é um desafio crescente para os órgãos periciais criminais, na medida em que estes são os responsáveis pelo processamento dessas informações na busca por evidências da prática de crimes (Sommer, 2004). No ano de 2010, apenas no âmbito da PF, foram realizados em todo o país mais de quatro mil exames periciais apenas em discos rígidos e computadores, correspondentes a mais de 1 petabyte de dados[1]. Como conseqüência, a capacidade de realizar exames periciais que forneçam uma resposta em tempo adequado às necessidades da persecução penal é um dos grandes desafios enfrentados atualmente na área de Computação Forense (Beebe e Clark, 2005).

Um método amplamente utilizado em Computação Forense para reduzir a quantidade de dados que necessitam ser analisados em exames periciais consiste em excluir automaticamente da análise pericial os arquivos reconhecidamente irrelevantes (e.g., arquivos comuns de instalação de aplicativos comerciais, tutoriais, arquivos de exemplo e ajuda, dentre outros) ou destacar aqueles arquivos que possuam conteúdo potencialmente incriminador, como imagens de abuso sexual de crianças, *rootkits* ou aplicativos de esteganografia (Bunting, 2007). Para tanto, é necessária a utilização de uma base de dados que permita a identificação desses arquivos a serem filtrados do processo de exame pericial.

Esse processo de identificação de arquivos é normalmente baseado no conceito de funções criptográficas de *hash* (Schneier, 1995). As funções criptográficas de *hash* fornecem

[1] Dados obtidos através do Sistema de Criminalística da Diretoria Técnico-Científica da Polícia Federal em 12/05/2011.

um meio de identificar um arquivo sem que seja preciso armazenar todo o seu conteúdo, sendo suficiente que seja armazenada apenas a sua "impressão digital", ou seja, seu valor de *hash*. O valor de *hash* de um arquivo possui sempre um comprimento fixo e reduzido quando comparado com o tamanho médio de arquivos. As principais ferramentas periciais (como *AccessData Forensic ToolKit - FTK*[2], *Guidance Encase*[3] e *The Sleuth Kit/Autopsy*[4]) permitem a utilização dessa metodologia de filtragem por *hashes* (Steel, 2006 e Bunting, 2007), o que pode diminuir consideravelmente a quantidade de arquivos a serem analisados (Mead, 2006).

A solução atualmente adotada na comunidade de Computação Forense para a criação e manutenção de uma BHAC consiste em se fazer o acompanhamento, a importação dos dados dos arquivos e a atualização da BHAC a cada novo software lançado ou a cada nova atualização de um software já existente (Mead, 2006 e Kim et al., 2009). Considerando a grande diversidade de empresas de desenvolvimento de softwares existentes no mercado e a grande quantidade de softwares disponíveis, é cada vez mais difícil conseguir realizar essa tarefa a contento. Além disso, cada país precisa realizar individualmente esse mesmo esforço para acompanhar a evolução do parque de softwares específicos de seu próprio mercado.

As principais BHACs disponíveis atualmente para serem utilizadas com o propósito de filtragem de arquivos são criadas e organizadas por entidades estrangeiras e, por consequência, estão baseadas em aplicativos utilizados em seus países de origem. Entre as principais BHACs, pode-se citar a *National Software Reference Library (NSRL)* do *National Institute of Standards and Technology* (NIST)[5] dos Estados Unidos, a agora extinta *HashKeeper Library* do *National Drug Intelligence Center* (NDIC)[6] dos Estados Unidos e a *Known File Filter (KFF) Library*[7] da empresa americana de *softwares* forenses *AccessData*. Aplicativos em versões para a língua portuguesa (contendo, por exemplo, arquivos de tutoriais e ajuda específicos para o nosso idioma) e aplicativos específicos do nosso país (como os aplicativos disponibilizados pela Receita Federal do Brasil), em geral não estão presentes nessas BHACs estrangeiras ou estão presentes de forma limitada, conforme pode ser visto na Tabela 1.1, que mostra a distribuição por idioma dos *hashes* nos conjuntos de dados de referência (*Reference Data Sets – RDS*) versão 2.32 da NSRL.

[2] http://accessdata.com/products/computer-forensics/ftk
[3] http://www.guidancesoftware.com/forensic.htm
[4] http://www.sleuthkit.org/sleuthkit/
[5] http://www.nsrl.nist.gov/
[6] http://www.justice.gov/ndic/domex/hashkeeper.htm
[7] http://accessdata.com/support/adownloads

Tabela 1.1 – Distribuição por idioma dos *softwares* referentes aos conjuntos de *hashes* da NSRL versão RDS 2.32.

Idioma	***Hashes únicos***	***Percentual***
Português apenas	38.877	0,20%
Português e outros	268.016	1,39%
Multi-linguagem	190.423	0,99%
Outros	18.757.880	97,42%
Total	**19.255.196**	**100,00%**

Portanto, para diminuir essa dependência atual dos órgãos periciais brasileiros em relação às BHACs predominantemente estrangeiras e também para melhorar a capacidade de realizar a filtragem automática de arquivos conhecidos em exames periciais, é necessária a implementação de uma solução viável para criar e manter BHACs que atenda a realidade nacional.

Os inúmeros casos de análise de mídias de armazenamento computacional que são objeto de perícia criminal são uma fonte preciosa de dados a respeito de arquivos de cunho potencialmente criminoso e de arquivos irrelevantes que são comumente encontrados em exames periciais. Esses dados podem ser utilizados para extrair informações e gerar uma base de dados que facilite análises periciais posteriores ao permitir a identificação automática de arquivos potencialmente suspeitos ou irrelevantes.

1.2. OBJETIVOS

O objetivo principal deste trabalho é melhorar a eficácia e diminuir o custo computacional do processo de filtragem automática de arquivos irrelevantes em exames periciais. Propõe-se alcançar esse objetivo através da implementação de um processo de MD baseado em AD que utiliza como atributos os metadados de arquivos extraídos de casos reais de análises periciais para classificá-los de acordo com seu potencial de relevância.

Esse objetivo principal pode ser dividido em três objetivos intermediários e complementares:

- projetar um *framework* para alimentar uma base de dados unificada formada por metadados de arquivos extraídos de casos reais de análises periciais e por BHACs tradicionais obtidas de fontes externas;

- realizar a classificação dos arquivos da BDAE de acordo com seu potencial de relevância para exames periciais e incluir os *hashes* dos arquivos classificados como potencialmente irrelevantes na BHP a ser utilizada no processo de filtragem automática de arquivos. Espera-se, assim, melhorar a eficácia do processo de filtragem de arquivos conhecidos;
- realizar uma seleção dos conjuntos de *hashes* (SCH), obtidos das BHACs tradicionais, que sejam efetivos para a filtragem de arquivos, excluindo aqueles referentes a softwares obsoletos ou pouco utilizados na região ou país. Pretende-se, dessa forma, diminuir o custo computacional do processo de filtragem de arquivos conhecidos.

1.3. METODOLOGIA

Para realizar este trabalho foi feito inicialmente um levantamento das principais BHACs atualmente disponíveis, além de uma revisão da literatura a respeito de Computação Forense, *hashes* criptográficos e aspectos relacionados à sua segurança, assim como técnicas de MD e suas aplicações.

Através da revisão da literatura foi possível identificar um trabalho na Coréia do Sul para criar uma BHAC mais adequada à realidade do conjunto de *softwares* utilizados naquele país, o que veio a demonstrar que a preocupação em melhorar a eficácia de BHACs no cenário nacional apresentava correspondência em outros países.

Para avaliar o potencial de ganho na filtragem por *hashes* a partir da utilização de metadados de arquivos obtidos de casos de perícias reais foi realizado um teste preliminar de viabilidade. Nesse teste foram identificados arquivos que apareciam repetidamente em exames não correlatos (indicando tratarem-se de arquivos comuns e, portanto, potencialmente irrelevantes), mas que não foram filtrados utilizando as BHAC convencionais.

Considerando os resultados positivos do teste preliminar, foi realizada a coleta de dados de arquivos de várias mídias computacionais e de BHACs tradicionais. Em seguida, foi planejada a arquitetura geral do sistema e implementado o *framework* para realizar as operações de ETL (*Extraction, Transformation, Loading*) sobre as diversas fontes de dados, importando os dados para a base de dados modelada.

Como parte do desenvolvimento do processo de MD, foram realizadas entrevistas com especialistas em exames periciais com o propósito de fazer um levantamento dos possíveis

atributos a serem utilizados na classificação da relevância de arquivos no escopo das perícias em computadores. As informações obtidas foram modeladas e agregadas à BDAE.

Através do processo de MD foram identificados novos *hashes* de arquivos irrelevantes e estes foram integrados à SCH obtida a partir das BHAC tradicionais, formando a BHP. A BHP definida foi testada por meio de estudos de caso utilizando mídias computacionais objeto de exames periciais da PF. Alguns dos experimentos realizados, assim como a solução proposta, foram descritos no artigo *Improving Hashsets in Computer Forensics*, a ser apresentado na *8th Annual IFIP WG 11.9 International Conference on Digital Forensics*, na África do Sul (Ruback, Hoelz e Ralha, 2012).

Os capítulos seguintes deste trabalho estão assim distribuídos: os Capítulos 2, 3 e 4 contêm os fundamentos teóricos que serviram de base para o trabalho; no Capítulo 5 é apresentada a solução proposta; o Capítulo 6 contém o detalhamento dos experimentos realizados e os resultados obtidos; finalmente, o Capítulo 7 apresenta as conclusões e as sugestões para trabalhos futuros.

2. COMPUTAÇÃO FORENSE

Neste capítulo são apresentados os principais conceitos relacionados à área de Computação Forense, bem como as técnicas e procedimentos ligados às perícias digitais e as perspectivas futuras de pesquisa vinculadas ao trabalho proposto.

2.1. CONCEITOS

A Computação Forense é uma área do conhecimento que nasceu da necessidade de apresentar em tribunais de Justiça, respeitando todos os requisitos legais e processuais, as evidências obtidas através da análise de equipamentos computacionais (US-CERT, 2008). É, portanto, uma área originária da interseção de outras duas grandes áreas do conhecimento humano: a Ciência da Computação e o Direito.

De forma semelhante à estabelecida por Hoelz (2009), será utilizado neste trabalho o termo Perícia em Computadores como análogo ao termo em inglês *Computer Forensics*, enquanto o termo Computação Forense, derivado do termo em inglês *Digital Forensic Science*, será utilizado para referenciar a área científica de pesquisa correlata.

De acordo com a definição utilizada pela *High Technology Investigative Unit* do Departamento de Justiça dos Estados Unidos, a perícia em computadores é "*o processo de identificar evidências potenciais em meios eletrônicos para serem utilizadas em tribunais de Justiça. Ele envolve a preservação, extração, documentação e interpretação de dados de computadores*". De acordo com Carrier (2005) uma evidência digital é um "*objeto digital que contém informação confiável que apóia ou refuta uma hipótese*". As perícias em computadores são muitas vezes classificadas de forma mais restritiva, como uma subárea das perícias digitais, definidas pelo termo em inglês *Digital Forensics*. Essa classificação subdivide as perícias digitais em diversos ramos, como perícia em computadores, perícia em redes, perícia em banco de dados e perícia em dispositivos móveis. O foco deste trabalho será a perícia em computadores no seu sentido mais restrito.

A Computação Forense, por sua vez, faz parte das Ciências Forenses, que englobam todas as áreas do conhecimento que podem oferecer subsídios para a elucidação de crimes, tendo como objetivo determinar a dinâmica, a materialidade e a autoria das práticas criminosas (Damasceno, Zacca e Nogueira, 2006). Uma definição de Computação Forense é assim apresentada por Palmer (2001): "*o uso de métodos cientificamente procedentes e comprovados para a preservação, coleta, validação, identificação, análise, interpretação,*

documentação e apresentação de evidências digitais provenientes de fontes digitais com a finalidade de facilitar ou promover a reconstrução de eventos criminosos, ou ajudar a antecipar ações não autorizadas que mostrarem ser prejudiciais para operações planejadas".

Neste trabalho, a Computação Forense será tratada com foco em sua utilização por forças da lei, ou seja, para investigação de práticas criminosas na sociedade civil. Entretanto, os estudos aqui apresentados também podem ser aplicados na área forense militar ou comercial.

2.2. TÉCNICAS E PROCEDIMENTOS

As técnicas e procedimentos utilizados nas perícias em computadores têm início antes mesmo da chegada do material a ser analisado em um laboratório forense. Já durante a obtenção do material é necessário que sejam realizados os procedimentos adequados para a preservação da cadeia de custódia. Conforme Schweitzer (2003), "*a cadeia de custódia rastreia a evidência desde sua fonte original até o que é oferecido como evidência em um tribunal, demonstrando que a evidência coletada é autêntica*". No caso de evidências digitais, a preservação da cadeia de custódia pode ser realizada com a cópia forense dos dados e cálculo do *hash* do conteúdo copiado ainda durante a fase de obtenção do material (Eleutério e Machado, 2011). O valor do *hash* poderá, então, ser posteriormente recalculado para verificar a autenticidade da evidência.

Uma metodologia para padronizar o processo de perícias digitais foi desenvolvida pelo Laboratório de Crimes Cibernéticos da *Computer Crime and Intellectual Property Section* (CCIPS) do Departamento de Justiça dos Estados Unidos após consultas a diversos peritos em Computação Forense de diversas agências federais americanas (Carroll, Brannon e Song, 2008). Essa metodologia é descrita através de uma série de fluxogramas. O fluxograma que apresenta a visão geral do processo pericial é apresentado na Figura 2.1.

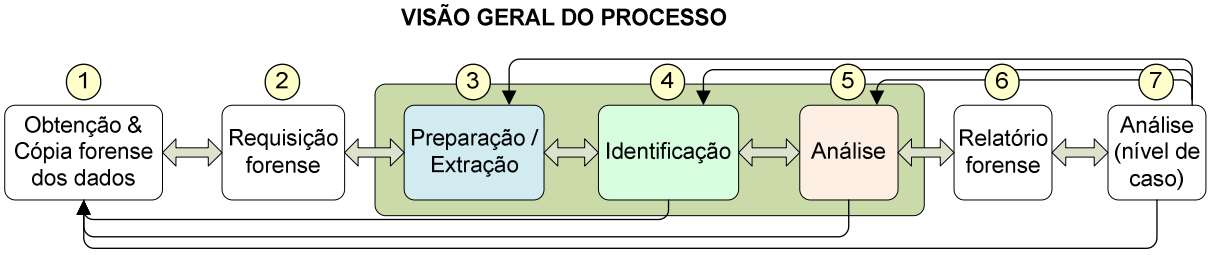

Figura 2.1 - Fluxograma com a descrição da visão geral do processo de perícias digitais (adaptada de Carroll, Brannon e Song, 2008).

A metodologia desenvolvida pela CCIPS é composta de sete etapas e o processo é iterativo e pode ser repetido quantas vezes for necessário. Cada etapa pode ser executada por grupos de pessoas diferentes, havendo a troca de informações entre esses grupos. A seguir, será apresentado um melhor detalhamento de cada uma dessas etapas.

1. Obtenção e cópia forense dos dados

A etapa inicial ocorre com a obtenção dos equipamentos a serem apreendidos e a realização de uma cópia forense (*bit-a-bit*) dos dados. Via de regra, os materiais apresentados como evidências nos tribunais devem ser os originais, razão pela qual estes são geralmente apreendidos. Mas também é possível a admissão legal apenas das cópias, desde que os dados destas sejam apropriadamente duplicados e desde que apenas os dados contidos nos originais sejam objeto de questionamento e não o material em si (Mandia, Prosise e Pepe, 2003). As ações dessa etapa geralmente seguem os termos especificados por um mandado judicial de busca e apreensão e são executadas por equipe policial na qual é comum a presença de um perito em Computação Forense. Solomon, Barrett e Broom (2005) destacam a importância de haver um planejamento prévio para identificar antecipadamente qual é o ambiente computacional que espera-se encontrar no local onde irá ocorrer a apreensão de materiais de informática, de modo a serem tomadas as providências necessárias.

2. Requisição forense

Na sequência, é necessário que seja feita a requisição para a realização da análise forense propriamente dita. É comum haver comunicação entre o requisitante e a equipe de peritos que irá atender o pedido, de modo a avaliar como as técnicas periciais podem melhor atender as necessidades da investigação em curso.

3-5. Preparação, extração, identificação e análise

Estas três etapas do processo consistem dos procedimentos principais de um exame pericial, que irão resultar na elaboração do laudo pericial na etapa de *Relatório forense*. Essas etapas são geralmente executadas em conjunto e em sequência por um único grupo de peritos e formam o principal objeto de estudo na área de Computação Forense.

Neste trabalho será apresentada uma divisão dessas etapas em quatro fases, ilustradas na Figura 2.2, de acordo com os procedimentos adotados nos exames periciais de informática da PF e descritos em Hoelz et al. (2007).

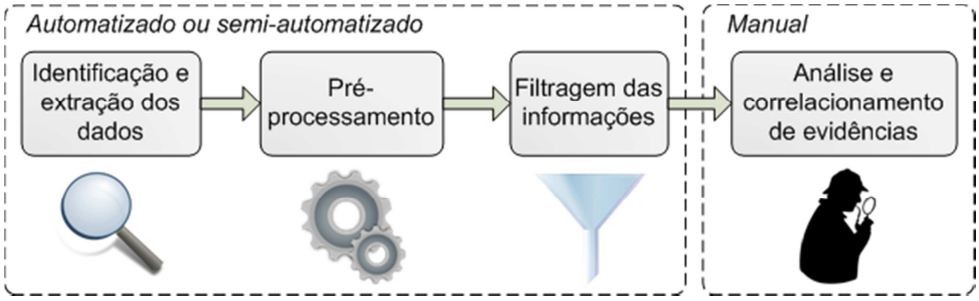

Figura 2.2 - Fases da análise pericial digital.

Fase I – Identificação e extração dos dados

Inicialmente, após a estação pericial ser preparada e configurada com os *softwares* forenses adequados e após a execução do procedimento de cópia forense dos dados da mídia a ser analisada, é realizada a fase de busca por dados e informações que serão utilizados posteriormente durante a fase de análise pericial. Portanto, nesta fase ocorre a identificação e expansão dos dados que deverão ser utilizados pelo perito na busca por evidências que irão confirmar ou refutar as hipóteses da prática criminosa. Alguns dos procedimentos realizados nesta fase de identificação e extração dos dados e informações são:

- localização de partições;
- identificação de sistemas de arquivos;
- recuperação de arquivos apagados;
- identificação e extração dos dados de *slack space*;
- identificação de arquivos *containers* (arquivos compactados, arquivos que armazenam mensagens de e-mail, entre outros) e extração dos arquivos e informações neles contidos.

Fase II – Pré-processamento

A fase seguinte consiste na realização de procedimentos automatizados para processar todos os dados encontrados na fase anterior, de modo a facilitar o trabalho de análise pericial. Dentre os procedimentos que podem ser realizados nessa fase, podemos citar:

- cálculo dos *hashes* de todos os arquivos encontrados;
- identificação das assinaturas dos arquivos;
- categorização dos arquivos de acordo com a sua assinatura (documentos, planilhas, bases de dados, imagens, vídeos, entre outros);
- identificação de inconsistências entre a assinatura e a extensão dos arquivos;

- geração de índice de palavras a serem utilizadas em busca automatizada;
- identificação de arquivos criptografados.

Fase III – Filtragem das informações

Uma vez que todos os dados e informações foram encontrados e processados, são realizados procedimentos automatizados ou semi-automatizados para filtrar aqueles considerados irrelevantes e destacar aqueles que apresentam potencial interesse para a análise pericial. Em oposição à fase de identificação e extração dos dados, essa fase visa reduzir a quantidade de informações que necessitarão ser analisadas diretamente pelo perito. Podemos citar os seguintes procedimentos a serem realizados nessa fase:

- identificação e, opcionalmente, filtragem de arquivos conhecidos (com o uso de BHACs);
- identificação e, opcionalmente, filtragem de arquivos duplicados (por meio de seus valores de *hash*);
- busca por arquivos contendo palavras-chave especificadas.

Fase IV – Análise e correlacionamento de evidências

Finalmente, a última fase de análise requer a intervenção direta do perito, em que são realizados procedimentos de visualização, seleção e correlacionamento das evidências encontradas na mídia em análise. Atualmente, esses procedimentos são realizados de maneira predominantemente manual, mas diversos estudos apresentam propostas para utilizar técnicas de Inteligência Artificial (IA) e MD para auxiliar o trabalho pericial durante essa fase, como Hoelz, Ralha e Geeverghese (2009) e Beebe e Clark (2005). Alguns dos procedimentos que podem ser realizados nessa fase são:

- análise visual do conteúdo dos arquivos;
- identificação de aplicativos presentes na mídia analisada e que possam vir a ter interesse para o caso em análise, como programas para transmissão de dados, esteganografia, limpeza de dados, contábeis, entre outros;
- análise de informações contidas no sistema operacional ou em aplicativos de interesse específico para o caso investigado (*registry* do sistema operacional Windows, arquivos de log, cache e histórico de navegação na Internet, entre outros);
- organização cronológica das informações encontradas;
- agrupamento de arquivos/informações correlatos;

- restauração de informações obtidas a partir de dados recuperados;
- quebra de senhas de arquivos criptografados.

6. Relatório forense

Uma vez terminadas as fases da análise pericial, os resultados obtidos são apresentados em um laudo pericial que, geralmente, é acompanhado de uma mídia digital na qual são copiadas as evidências digitais encontradas. Para garantir a integridade e autenticidade da mídia anexa, é calculado o *hash* de seu conteúdo e o resultado é colocado no corpo do laudo.

7. Análise (nível de caso)

Os resultados de cada exame pericial devem ser analisados sob a perspectiva da investigação como um todo, de modo a se ter a convicção a respeito da ocorrência, ou não, de fato criminoso. Novas ações (como novas buscas ou pedidos de mandado judicial) ou iterações do processo podem vir a ser executadas de acordo com os achados e seus impactos sobre a investigação.

O foco deste trabalho está na etapa de análise pericial e, principalmente, na fase de filtragem das informações que deverão ser processadas e analisadas pelo perito, mais especificamente no procedimento de identificação e filtragem automática de arquivos com o uso de BHACs, o qual será melhor detalhado a seguir.

2.2.1. Identificação e filtragem de arquivos por *hash*

Funções criptográficas de *hash* são amplamente utilizadas em Computação Forense como forma de identificação única de arquivos e, como consequência, para filtrar arquivos conhecidos com o uso de bases de *hashes* (Mead, 2006 e Bunting, 2007).

As principais ferramentas forenses atualmente utilizadas (*AccessData FTK*, *Guidance Encase* e *Autopsy*) implementam o procedimento de filtragem de arquivos por meio do uso de BHACs. Essas ferramentas permitem classificar os arquivos conhecidos em duas categorias:

1. Conhecido (*known*) ou ignorável (*ignorable*): arquivos que fazem parte de aplicativos comerciais comuns e sem interesse específico para análises forenses (ex: arquivos comuns de instalação de sistemas operacionais; processadores de texto; planilhas gráficas; jogos, dentre outros);
2. Alerta (*alert*) ou notável (*notable*): arquivos que podem ter especial interesse em exames periciais, por possuírem conteúdo já destacado anteriormente em análises periciais (ex: imagens relacionadas a racismo, apologia às drogas,

contendo abuso sexual a criança ou adolescente; assim como códigos de malwares, dentre outros) ou estarem associados a aplicativos comumente utilizados para esconder informações ou dificultar o trabalho pericial (ex: softwares para realizar a criptografia de dados, para esconder informações com o uso de esteganografia, para apagar dados fisicamente do disco e até mesmo ferramentas forenses).

A Figura 2.3 mostra uma tela da ferramenta forense FTK (versão 1.81) em que são apresentadas algumas opções de processamentos a serem realizados na criação de um caso de exame pericial em mídia de armazenamento computacional. Dentre as opções está destacada aquela relacionada à filtragem de arquivos conhecidos (opção "*KFF Lookup*").

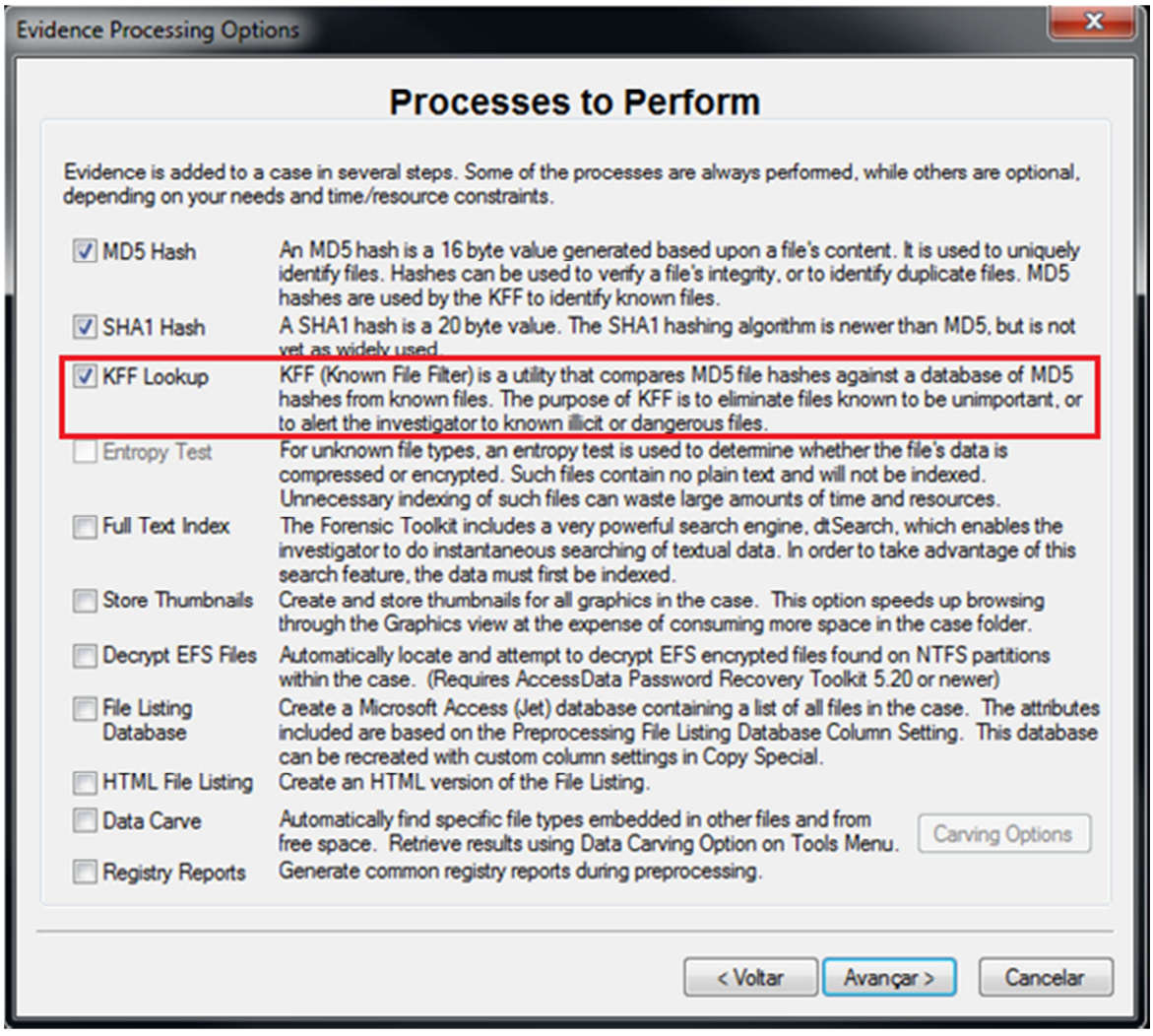

Figura 2.3: Opções de processamentos a serem realizados na criação de um caso pericial pela ferramenta forense *AccessData Forensic ToolKit* versão 1.81. Destaque para a opção de filtragem com o uso de BHACs.

As ferramentas forenses permitem que sejam adicionadas novas entradas à BHAC utilizada ou que seja alterada a classificação (*ignorável* ou *alerta*) dos conjuntos de *hash*

existentes. Entretanto, a manipulação dos dados da BHAC é geralmente difícil e custosa para o usuário da ferramenta.

Dentre as BHACs atualmente disponíveis para serem utilizadas em exames periciais, a mais conhecida e utilizada é a NSRL (National Institute of Standards and Technology, 2011). A NSRL consiste de conjuntos de *hashes* (MD5, SHA-1 e, de forma limitada, SHA-256) extraídos de uma coleção de *softwares* obtidos de diversas fontes e é disponibilizada sob a forma de RDS que são publicados trimestralmente. A NSRL tem como objetivo "*promover o uso eficiente e efetivo de tecnologia computacional em investigações de crimes envolvendo computadores*" e é apoiada pelo Instituto Nacional de Justiça do Departamento de Justiça dos Estados Unidos, por forças da lei americanas federais (como o *Federal Bureau of Investigation* - FBI), estaduais e locais e pelo NIST, que é uma agência do Departamento de Comércio dos Estados Unidos.

Uma análise empírica nos mais de dez milhões de arquivos únicos da base de *hashes* da NSRL, realizada no ano de 2006, mostrou não haver nenhuma colisão em sua base e que colisões aleatórias de *hashes* MD5 ou SHA-1 são probabilisticamente improváveis, tendo como base os estudos estatísticos realizados (Mead, 2006).

O RDS da NSRL disponibilizado em julho de 2011 - versão 2.33 - continha 20.129.213 *hashes* únicos de arquivos conhecidos.

Além da NSRL, outra BHAC conhecida e que também foi organizada por uma entidade americana é a HashKeeper. A HashKeeper foi criada em 1998 pelo NDIC, uma agência vinculada ao Departamento de Justiça dos Estados Unidos, e consistia em um software com uma base de *hashes* para fazer especificamente a filtragem de arquivos conhecidos utilizando o algoritmo MD5. Sua base de dados era composta por *hashes* enviados por forças policiais de diversos países e classificava os arquivos em "conhecidos por serem bons" (*known to be good*) e "conhecidos por serem ruins" (*known to be bad*). Entretanto, com o avanço das ferramentas forenses que passaram a realizar o processo de filtragem de arquivos, o software HashKeeper não é mais mantido ou distribuído pelo NDIC, assim como sua base de *hashes*.

A terceira fonte de *hashes* de arquivos conhecidos é a *Known File Filter* (KFF) da empresa americana de softwares forenses *AccessData*. A KFF, entretanto, consiste de uma compilação feita pela empresa *AccessData* das bases de *hashes* da NSRL e HashKeeper, acrescida de *hashes* de arquivos fornecidos pelo Departamento de Segurança Interna (*Department of Homeland Security*) dos Estados Unidos.

Além do uso de BHACs para excluir arquivos conhecidos, as ferramentas forenses também utilizam os valores de *hashes* para identificar arquivos que estão duplicados em um mesmo caso de exame pericial, evitando que o perito necessite analisar mais de uma vez arquivos com o mesmo conteúdo.

2.3. PERSPECTIVAS FUTURAS

Grande parte da motivação para estudos na área de Computação Forense está relacionada à rápida evolução no volume de dados que necessitam ser analisados em exames periciais, conforme destacado por Roussev, Richard III e Marziale (2008):

> "*Perícias digitais em larga escala apresentam ao menos dois desafios fundamentais. O primeiro é atender as necessidades computacionais de uma grande quantidade de dados a serem processados. O segundo é extrair informação útil dos dados brutos de uma maneira automatizada. Ambos os problemas podem resultar em longos tempos de processamento que podem prejudicar seriamente uma investigação.*"

Garfinkel (2010) faz uma análise dos desafios a serem enfrentados e as perspectivas das pesquisas na área de Computação Forense para os próximos anos. Como exemplos de desafios, são citados:

- o uso de armazenamento e processamento em nuvem significa que os dados muitas vezes podem nem mesmo ser encontrados;
- o aumento de tamanho dos dispositivos de armazenamento dificulta a cópia forense e o processamento dos dados em um tempo satisfatório;
- a proliferação de sistemas de arquivos (notadamente os voltados para *smartphones*, como Android, iOS, Blackberry e Windows Mobile) e formatos de arquivos levam a um aumento da complexidade para a exploração dos dados;
- a difusão do uso de criptografia pode inviabilizar o acesso aos dados.

Todos esses desafios trazem consigo oportunidades de pesquisa e a necessidade de modificação nos métodos atualmente utilizados para realizar exames periciais.

Garfinkel (2010) destaca, também, que as ferramentas forenses estão atualmente voltadas para a busca por evidências, onde a simples posse das evidências caracteriza o crime. As ferramentas são feitas para examinar crimes onde as evidências residem no computador e não para analisar crimes cometidos com o uso de computadores ou contra outros

computadores. Elas auxiliam muito pouco o perito a fazer correlacionamento de evidências, encontrar informações que estão fora do padrão, fora do lugar ou modificadas de forma sutil.

Prevendo uma futura crise nas perícias digitais, decorrente de todas as dificuldades advindas das tendências apontadas, Garfinkel (2010) acredita que a cooperação entre a comunidade de perícias digitais tem um papel importante no futuro da área: "*Com cuidadosa atenção à cooperação, padronização e desenvolvimento compartilhado, a comunidade de pesquisa em perícias digitais pode simultaneamente diminuir os custos de desenvolvimento e melhorar a qualidade de nossos esforços de pesquisa. Essa é provavelmente uma das poucas técnicas a nossa disposição para sobreviver à crise que se aproxima nas perícias digitais*".

Beebe (2009) apresenta sua opinião sobre a situação atual e futura das pesquisas em Computação Forense, assim como a visão de vinte e um pesquisadores e praticantes da área. A autora mostra, como pontos positivos atuais:

- o aumento da valorização das evidências digitais em investigações, o que trouxe a incorporação do planejamento e uso de procedimentos de Computação Forense já nas fases iniciais das investigações;
- a percepção de que os fundamentos científicos das perícias digitais têm se fortalecido, com vários esforços sendo empregados para a adoção de padrões, melhores práticas, testes e validações de ferramentas e processos;
- a grande evolução ocorrida nas pesquisas relacionadas à "arqueologia" de artefatos digitais, ou seja, a identificação, extração e exame de artefatos digitais.

Mas a autora destaca, também, a mudança de foco nas pesquisas atuais, que caminharam das etapas iniciais do processo de perícias digitais (Preparação → Resposta a incidente → Coleta → Análise → Apresentação → Encerramento do incidente), passando a focar mais a etapa de análise.

Como problemas e dificuldades atuais, Beebe (2009) destaca a hiperformalização de procedimentos, que muitas vezes engessam a maneira de agir e dificultam a adoção de técnicas fora do padrão por parte do perito, além do foco que é dado aos ambientes computacionais mais comuns (sistemas operacionais Windows e Linux, sistemas de arquivos NTFS, FAT e EXT, por exemplo), o que causa grandes dificuldades para a execução de perícias em ambientes não convencionais. A autora também destaca que, por ser uma área relativamente nova, os rigores científicos na área de Computação Forense são ainda incipientes, apesar de apresentarem sensível melhora.

Beebe (2009) também indica quatro temas principais que têm importância estratégica nas pesquisas futuras em Computação Forense:

- volume e escalabilidade;
- abordagens analíticas inteligentes;
- perícias em ambientes computacionais não convencionais;
- desenvolvimento de ferramentas forenses.

Relacionados a esses temas, são apresentadas diversas oportunidades e desafios de pesquisa: aquisição digital forense seletiva; processamento analítico distribuído; processos de busca baseados em MD; classificação de arquivos para auxílio à análise; agrupamento temático de resultados de pesquisa por palavras-chave; *threading* massivo com utilização de *graphical processing units* (GPUs); uso de *hashing* de similaridade; além de outros tópicos como análise *live*; perícias em bancos de dados; detecção, extração e análise de dados escondidos por esteganografia, dentre outros.

Conforme destacado nesta Seção, são diversas as possibilidades de pesquisa na área de Computação Forense e grande parte dos estudos estão priorizando formas de automatizar e auxiliar o trabalho de análise da grande quantidade de informações que necessitam ser periciadas. A intenção deste trabalho é auxiliar na redução dessa quantidade de informação a ser processada – por meio da melhoria no procedimento de identificação e filtragem de arquivos conhecidos – com o objetivo final de diminuir o custo computacional e humano envolvido em análises periciais.

2.4. TRABALHOS CORRELATOS

Nesta Seção são apresentados trabalhos relacionados à criação de BHACs com finalidades periciais e ao uso de técnicas de MD na área de Computação Forense. O primeiro trabalho apresenta um estudo sobre a NSRL, a principal BHAC atualmente disponível para uso pericial, enquanto o segundo trabalho mostra a experiência de criação de uma base de *hashes* na Coréia do Sul com foco em *softwares* voltados para as especificidades daquele país.

Ambos os trabalhos se baseiam em metadados e *hashes* de arquivos conhecidos extraídos diretamente dos pacotes de instalação de *softwares*. Não foram encontrados trabalhos que, assim como o trabalho aqui proposto, buscam identificar arquivos conhecidos diretamente das mídias de armazenamento analisadas em exames periciais. O terceiro trabalho apresenta uma visão sobre as perspectivas do uso de técnicas de MD para auxiliar a tarefa de análise pericial.

2.4.1. Identificação Única de Arquivos na NSRL

Conforme já citado na Seção 2.2.1, a mais conhecida e utilizada BHACs para fins forenses é a NSRL, organizada pelo NIST, uma agência do Departamento de Comércio dos Estados Unidos. O projeto NSRL começou no ano de 1999 atendendo a pedidos de órgãos governamentais americanos como o FBI, o Centro de Crimes Cibernéticos do Departamento de Defesa e o Instituto Nacional de Justiça (Mead, 2006).

No ano de 2006 foi realizado um estudo (Mead, 2006) sobre a base de *hashes* da NSRL com a finalidade de verificar a possível ocorrência de colisões nas assinaturas dos arquivos e, assim, avaliar a qualidade e viabilidade da base de *hashes* para uso em filtragem de arquivos conhecidos em exames periciais. Foram realizados testes empíricos e estatísticos, assim como uma análise dos possíveis impactos decorrentes dos ataques de criptoanálise para produzir colisões de *hashes* disponíveis à época.

Os estudos empíricos mostraram que a ocorrência de colisões aleatórias dos *hashes* MD5 e SHA-1 na base da NSRL são estatisticamente improváveis. A Tabela 2.1 mostra que a distância média dos valores de *hash* na base NSRL, versão 2.9, com um total de 10.533.771 assinaturas totais, foi próxima do valor teórico esperado. Ou seja, conforme desejado, as funções de *hash* MD5 e SHA-1 realizam uma distribuição uniforme dos valores de *hash* para mensagens de entrada aleatórias. As distâncias máximas e mínimas dos valores de *hash* também não indicaram grandes desvios na distribuição dos valores e consequente tendência das funções para gerarem colisões além da expectativa teórica para esses casos.

Tabela 2.1 - Distância entre valores de *hashes* de arquivos na NSRL 2.9 (adaptada de Mead, 2006).

	SHA-1	**MD5**	**CRC-32**
Média (esperada)	$1{,}387443 \times 10^{41}$	$3{,}230395 \times 10^{31}$	$4{,}077331 \times 10^{2}$
Mínimo (observado)	$7{,}088510 \times 10^{33}$	$4{,}218731 \times 10^{23}$	0
Máximo (observado)	$2{,}313508 \times 10^{42}$	$5{,}398685 \times 10^{32}$	$6{,}47000 \times 10^{2}$
Média (observada)	$1{,}387444 \times 10^{41}$	$3{,}230393 \times 10^{31}$	$4{,}07000 \times 10^{2}$

Também foram realizados testes estatísticos para avaliar se as funções de *hash* possuem as características de pseudo-aleatoriedade esperadas. Os testes realizados foram baseados no *Statistical Test Suite* (STS), conforme descrito por Rukhin et al. (2001). A Tabela 2.2 apresenta um sumário dos testes realizados e os resultados obtidos para os algoritmos SHA-1, MD5 e CRC-32 e, conforme pode ser verificado, tanto o SHA-1 quanto o MD5 apresentaram resultados favoráveis.

Tabela 2.2 - Sumário dos testes estatísticos realizados (adaptada de Mead, 2006).

Teste estatístico	**SHA-1**	**MD5**	**CRC-32**
Teste de frequência (mono-bit)	Passou	Passou	Passou
Teste de frequência com um bloco	Passou	Passou	Falhou
Testes de somas-cumulativas {1, 2}	Passou	Passou	Passou
Runs test	Passou	Passou	Passou
Teste de maior sequência de 1's em um bloco	Passou	Passou	Falhou
Teste de rank de matriz binária aleatória	Passou	Passou	Passou
Teste de transformada discreta de Fourier (spectral)	Passou	Passou	Passou
Teste de correspondência de sobreposição (periódica) de modelo	Passou	Passou	Passou
Teste estatístico universal de Maurer	Passou	Passou	Passou
Teste de entropia aproximada (Apen)	Passou	Passou	Falhou
Teste serial {1, 2}	Passou	Passou	Falhou
Teste de complexidade linear	Passou	Passou	Passou
Testes de excursões aleatórias {1-8}	Passou	Passou	Passou

Além disso, também foi feita uma representação gráfica em três dimensões de cada conjunto de assinaturas de arquivos com o objetivo de tentar identificar algum tipo de padrão ou agrupamento dos dados que pudessem indicar que o algoritmo de *hash* utilizado não realiza uma distribuição uniforme conforme seria desejável. A análise visual dos gráficos obtidos para os algoritmos MD5 e SHA-1, entretanto, não identificou nenhum padrão claro ou sistemático de agrupamento nos dados, o que corroborou com os resultados dos testes anteriores.

Assim, com base nos resultados dos testes realizados, o autor conclui que "*os algoritmos MD5 e SHA-1 utilizados para gerar assinaturas de hash claramente suportam seu uso dentro da comunidade forense para a identificação de conteúdo conhecido*".

2.4.2. Construção de um RDS de *softwares* na Coréia do Sul

A falta de uma fonte de *hashes* de arquivos conhecidos que seja voltada para a realidade de países com idiomas diferentes do Inglês é um dos problemas destacados neste trabalho e também foi fator motivador para o trabalho de Kim et al. (2009), no qual é apresentado um modelo para a construção de um conjunto de dados de referência de *softwares* na Coréia do Sul.

A proposta de Kim et al. (2009) foi composta por dois objetivos e executada em duas fases correspondentes:

- fase I: realizar um estudo para avaliar a efetividade e a consistência dos dados da BHAC da NSRL;

- fase II: criar um RDS de softwares consolidado composto por metadados e *hashes* importados da NSRL (após a exclusão, realizada na fase I, de dados inconsistentes ou que se mostraram pouco efetivos) acrescida de metadados e *hashes* de novos arquivos mais voltados para a realidade sul-coreana, ou seja, softwares domésticos, softwares mais largamente utilizados naquele país e softwares com suporte ao alfabeto Hangul. A esse RDS de softwares consolidado foi dado o nome de *Korean Reference Data Set* (KRDS).

A Figura 2.4 apresenta, em detalhes, o modelo de construção da solução proposta por Kim et al. (2009).

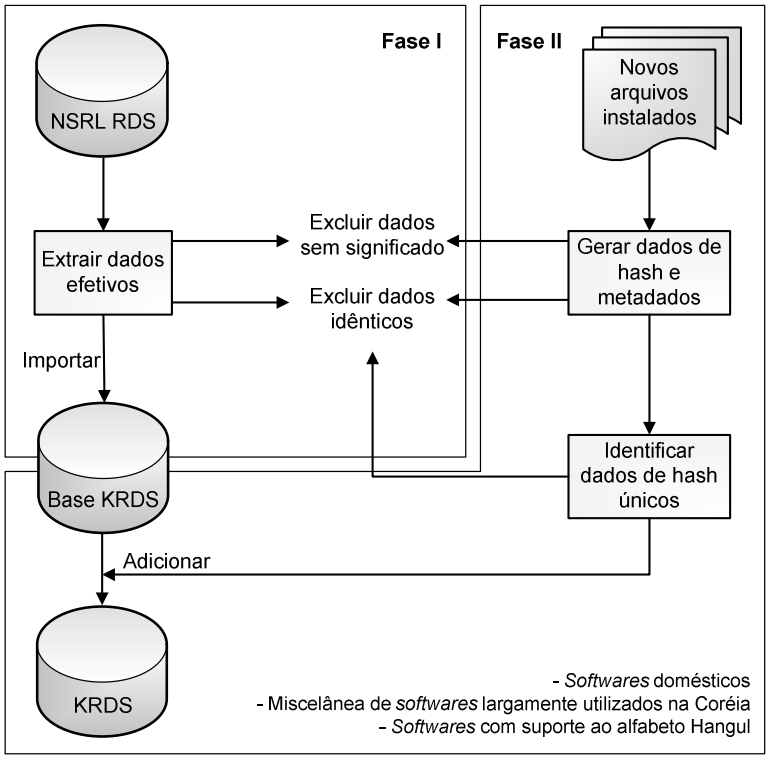

Figura 2.4 - Modelo de construção do KRDS (adaptada de Kim et al., 2009).

Durante a fase I foram filtrados os registros da NSRL para os quais os valores dos códigos do sistema operacional ou do produto eram marcados como “desconhecidos”. Também foram filtradas entradas duplicadas antes da inserção no KRDS. Essa fase teve por objetivo deixar a base KRDS a mais limpa e consistente possível.

Na fase II, a inclusão das informações a respeito dos arquivos de *softwares* sul-coreanos foi feita diretamente através dos pacotes de instalação dos *softwares*, assim como é feito na NSRL. Os pacotes foram instalados em uma máquina base e, por meio do uso de um algoritmo para identificar arquivos significativos e para excluir registros idênticos de um

mesmo *software*, os *hashes* e os metadados dos arquivos de interesse foram extraídos e incluídos no KRDS.

Os *softwares* sul-coreanos incluídos na fase II consistiam de:

- 15 versões do sistema operacional Microsoft Windows com suporte ao idioma coreano;
- 19 *softwares* comerciais;
- 49 *softwares freeware* ou *shareware* listados entre os mais populares em sites de *download* da Coréia do Sul.

Ao término do processo, conforme pode ser observado na Tabela 2.3, o KRDS final foi composto por 9.053 registros de metadados e 34.744.693 *hashes*, dentre os quais 8.707 registros de metadados e 34.483.804 *hashes* foram importados da NSRL e 346 registros de metadados e 260.889 *hashes* foram adicionados a partir de informações de *softwares* específicos da Coréia do Sul.

Tabela 2.3 - Quantidade de registros no KRDS (adaptada de Kim et al., 2009).

Origem		Total	NSRL RDS	Específico da Coréia do Sul
Metadados	Produto	7.386	7.090	296
	Sistema operacional	1.312	1.277	35
	Fabricante	355	340	15
	Soma parcial	**9.053**	**8.707**	**346**
Hashes	Sistemas operacionais		34.483.804	180.079
	Softwares comerciais			70.426
	Freeware/*shareware*			10.384
	Soma parcial	**34.744.693**	**34.483.804**	**260.889**

2.4.3. O uso de MD na área de Computação Forense

As dificuldades encontradas na realização de perícias em computadores decorrentes do aumento do volume de dados a serem examinados são a motivação apresentada por Beebe e Clark (2005) para a urgente adoção de técnicas de MD na área de Computação Forense e investigação criminal: "*Estamos pedindo maiores investigações sobre o uso de técnicas de mineração de dados para ajudar investigações digitais em duas áreas principais: detecção de crime e investigação criminal. Técnicas de mineração de dados descritivas, preditivas e de recuperação de conteúdo podem ser livremente mapeadas para cada uma dessas áreas. Nós*

acreditamos que tal mapeamento e aplicação resultará em: (i) redução do tempo de processamento por parte de sistemas e seres humanos associado à análise de dados; (ii) maior eficácia analítica e qualidade da informação, e (iii) redução dos custos monetários associados a investigações digitais". Como benefício adicional ao uso de MD, é citada também a possibilidade de identificar padrões ou tendências nos dados que geralmente passariam despercebidos para um analista humano.

Os autores propõem a adoção de maneira progressiva das classes de técnicas de MD (descritiva → preditiva → recuperação de conteúdo) durante a fase de análise de dados em um processo de exame pericial. A fase de análise de dados é apresentada pelos autores como composta de três sub-fases: levantamento dos dados, extração de dados e exame dos dados.

Dessa forma, técnicas de modelagem descritiva poderiam ser aplicadas durante a sub-fase de levantamento dos dados para caracterizar o perfil de uso e de atividade em um dispositivo computacional; técnicas de MD de classificação poderiam ser aplicadas para ajudar a reduzir o volume de dados a serem analisados e técnicas de recuperação de conteúdo poderiam ser utilizadas para avaliar a importância ou relevância de documentos em um determinado contexto.

Como este trabalho tem por objetivo auxiliar o processo de redução do volume de dados que necessitam ser analisados, é interessante destacar a observação de Beebe e Clark (2005) sobre esse assunto específico: "*É importante enfatizar que técnicas de redução de dados devem ser baseadas em classificação, conforme descrito, em oposição a técnicas de caracterização descritiva (sumarização), devido à perda de dados associada a esta última*".

Mas há limitações potenciais que devem ser levadas em consideração quanto ao uso de MD em investigações digitais, como a escassez de testes e validação do uso de técnicas de MD na área de Computação Forense e a falta de conhecimento dessas técnicas pela comunidade de Computação Forense. Porém, como a MD tem sido utilizada com sucesso em diversas áreas, os autores acreditam que há razões para confiar na efetividade de seu uso também na disciplina de Computação Forense.

Os autores concluem o trabalho citando a necessidade do avanço nas pesquisas em MD voltadas para as perícias em computadores: "*Conjuntos de dados com terabytes de capacidade já estão desafiando analistas e investigadores. Portanto, um fluxo ativo de pesquisas estendendo os estudos em mineração de dados para as perícias em computadores e investigações digitais é desesperadamente necessário*".

2.4.4. Análise comparativa com a proposta deste trabalho

A forma usual de construir e manter BHACs, conforme apresentado nas Seções 2.4.1 e 2.4.2, consiste em extrair os *hashes* dos arquivos presentes nos pacotes de *software* nos quais são distribuídos. Espera-se que esses arquivos sejam encontrados posteriormente em computadores que serão objeto de análises periciais futuras e, assim, os arquivos conhecidos poderão ser identificados e filtrados, conforme descrito e traduzido de Mead (2006): "*A NSRL é usada para identificar arquivos conhecidos. Isso pode reduzir o tempo gasto examinando um computador. Correspondências com sistemas operacionais ou aplicativos usuais não precisam ser pesquisadas, manualmente ou eletronicamente, por evidência*".

Entretanto, como as principais BHACs atualmente disponíveis são estrangeiras e a realidade do parque de *softwares*, considerando aspectos de idioma, pode variar entre países distintos (problema destacado por Kim et al. (2009), conforme apresentado na Seção 2.4.2), este trabalho busca implementar um método para melhorar a eficácia das BHACs por meio da identificação de arquivos potencialmente ignoráveis que representem mais fielmente esse parque de *softwares* em uso em um dado mercado.

Para atingir esse objetivo, ao invés de armazenar na BHAC os *hashes* de arquivos presentes em *softwares* que esperamos encontrar em vários computadores de um dado mercado, a solução apresentada neste trabalho propõe a utilização da lógica inversa: usar uma amostra dos computadores de um dado mercado para identificar, dentre aqueles arquivos que são comumente encontrados, quais apresentam características para serem adicionados à uma BHAC. Conforme descrito no trabalho apresentado na Seção 2.4.3, o uso de técnicas de MD podem auxiliar a implementação dessa solução, que será detalhada no Capítulo 5.

A capacidade das funções criptográficas de *hash* de prover, com alto grau de segurança, o resumo de um conteúdo digital que seja probabilisticamente único é um conceito fundamental para a criação de BHACs. Seria computacionalmente inviável armazenar o conteúdo completo de dezenas de milhões de arquivos e compará-los com os conteúdos completos dos arquivos presentes em computadores a cada exame pericial realizado. Portanto, por ser um elemento chave para o presente trabalho, será apresentado no próximo capítulo um estudo sobre as funções criptográficas de *hash*.

3. FUNÇÕES CRIPTOGRÁFICAS DE *HASH*

As funções criptográficas de *hash* são utilizadas na área de segurança da informação para auxiliar nas mais diversas tarefas, como: verificação de corrupção de dados, autenticação de mensagens, assinatura digital, geração de números pseudo-aleatórios e para identificar univocamente mensagens ou arquivos. Esta última aplicação de funções criptográficas de *hash* é a que possui especial interesse para este trabalho.

Será apresentada neste capítulo uma revisão das principais características das funções criptográficas de *hash*, com foco no uso na área de Computação Forense. Na Seção 3.4 será apresentado um resumo a respeito de algoritmos de *hashing* por similaridade, cuja utilização em Computação Forense vem apresentando avanços nos últimos anos.

3.1. VISÃO GERAL

As funções criptográficas de *hash* são também referenciadas na literatura como funções unidirecionais de *hash* fortes (*strong one-way hash functions*) (Preneel, 1993). De acordo com Schneier (1995), uma função unidirecional de *hash*, *H(M)*, opera sobre uma mensagem *M* – também chamada de pré-imagem – de tamanho arbitrário e deve retornar um valor de *hash*, *h*, de tamanho fixo, conforme a Equação 1.

$$h = H(M), \text{ onde } h \text{ possui tamanho fixo } m \quad (1)$$

Além disso, para ser considerada unidirecional, uma função de *hash* deve possuir as seguintes características:

1. dada uma pré-imagem *M*, deve ser fácil calcular seu valor de *hash h*;
2. dado um valor de *hash h*, deve ser inviável calcular *M* tal que *H(M) =h*; e
3. dada uma mensagem *M*, deve ser inviável encontrar outra mensagem, *M*', tal que *H(M) = H(M')*.

Adicionalmente, para ser considerada uma função criptográfica de *hash* (*strong one-way hash function*), também é necessário que a função seja resistente a colisões, ou seja, deve ser inviável encontrar duas mensagens arbitrárias, *M* e *M'*, tal que *H(M) = H(M')* (Preneel, 1993).

Em resumo, portanto, a essência de uma função criptográfica de *hash* é prover uma "representação condensada" ou uma "impressão digital" que seja probabilisticamente única e

de tamanho fixo para um conteúdo de entrada de tamanho arbitrário. Para atingir esse objetivo é necessário que apresente as características citadas.

A Figura 3.1 apresenta um exemplo de uso da função criptográfica de *hash Message-Digest Algorithm 5* (MD5) que recebe algumas mensagens de tamanho variável como entrada e retorna os valores de *hash* correspondentes de tamanho fixo de 128 bits. É possível perceber que mesmo uma pequena variação na mensagem de entrada, como o acréscimo de um ponto ao final da terceira mensagem de exemplo, resulta em valores de *hash* completamente distintos.

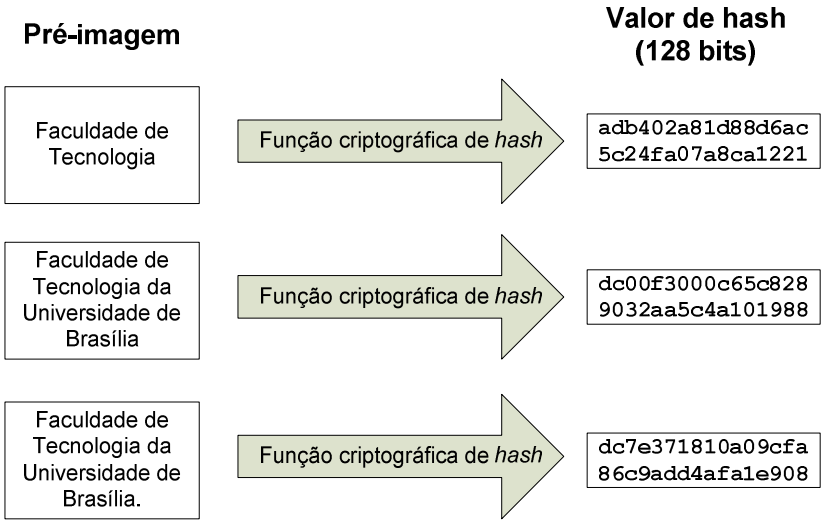

Figura 3.1 - Exemplos de entradas e saídas da função criptográfica de *hash* MD5.

As funções criptográficas de *hash*, assim como toda função de *hash*, também devem ter a propriedade de gerar valores de *hash* pseudo-aleatórios e uniformemente distribuídos quando aplicadas sobre grandes conjuntos de mensagens de entrada (Stallings, 2010). Se pequenas alterações nos dados de entrada levarem a grandes mudanças no valor de saída, diz-se que a função possui um bom efeito avalanche (*avalanche effect*). Se cada bit da pré-imagem pode afetar o resultado de todos os bits do resultado de saída, então a função é considerada completa (*complete*).

A Tabela 3.1 apresenta um resumo dos requisitos necessários de uma função criptográfica de *hash*, assim como uma breve descrição de cada um deles.

Tabela 3.1: Requisitos de uma função criptográfica de *hash* H (adaptada de Stallings, 2010).

Requisito	**Descrição**
Tamanho variável de entrada	H pode ser aplicada a um bloco de dados de qualquer tamanho.
Tamanho fixo de saída (compressão)	H produz uma saída de comprimento fixo.
Eficiência	H(x) é relativamente fácil de computar para qualquer x dado, fazendo ambas as implementações em *hardware* e *software* práticas de implementar.
Resistência a pré-imagem (propriedade unidirecional)	Para qualquer valor de *hash* h dado, é computacionalmente inviável encontrar y, tal que H(y) = h.
Resistência a segunda pré-imagem (resistência a colisão fraca)	Para qualquer bloco x dado, é computacionalmente inviável encontrar $y \neq x$, tal que H(y) = H(x).
Resistência a colisão (resistência a colisão forte)	É computacionalmente inviável encontrar qualquer par (x, y), tal que H(x) = H(y).
Pseudo-aleatoriedade	Saída de H atende testes padrão para pseudo-aleatoriedade.

Menezes, Oorschot e Vanstone (1996) apresentam uma taxonomia das funções de *hash* na Figura 3.2. Em um nível mais alto, elas podem ser separadas em duas classes:

- funções de *hash* sem chave (*unkeyed hash functions*): possuem como parâmetro de entrada apenas a mensagem;
- funções de *hash* com chave (*keyed hash functions*): possuem como parâmetro de entrada a mensagem e também uma chave secreta.

Com relação à classificação funcional, as funções de *hash* são separadas em:

- códigos de detecção de modificação (*modification detection codes* – MDCs): são uma subclasse das funções de *hash* sem chave e têm como principal objetivo assegurar a integridade de dados. Podem ser subdividas em:
  - funções unidirecionais de *hash* (*one-way hash functions* – OWHF): deve ser inviável encontrar uma mensagem que resulte em um valor pré-definido de *hash*;
  - funções de *hash* resistentes a colisão (*collision resistant hash functions* – CRHF): deve ser inviável encontrar duas mensagens quaisquer que resultem em um mesmo valor de *hash*;
- códigos de autenticação de mensagens (*message authentication codes* – MACs): são uma subclasse das funções de *hash* com chave e têm como objetivo facilitar a verificação da origem e da integridade de uma mensagem.

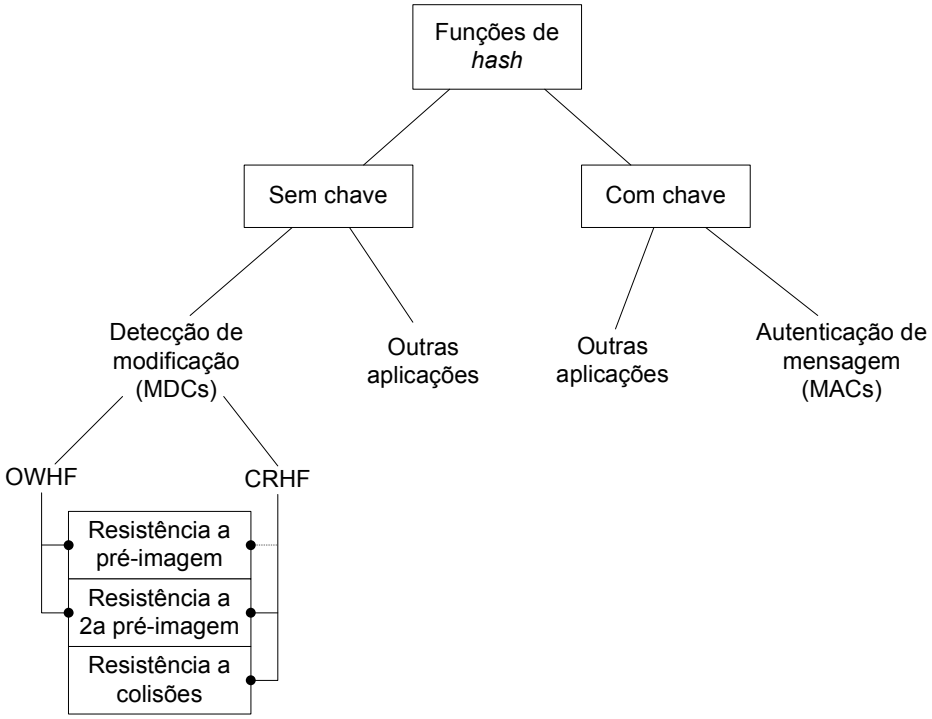

Figura 3.2: Classificação simplificada de funções criptográficas de *hash* e aplicações (adaptada de Menezes, Oorschot e Vanstone, 1996)

3.1.1. Estrutura geral de uma função de *hash* iterada

As principais funções criptográficas de *hash* utilizadas atualmente utilizam internamente um algoritmo que envolve o uso repetido de uma função de compressão que recebe dois valores de tamanho fixo como entrada e retorna um valor também de tamanho fixo como saída. Para uma dada função *f* de compressão, um dos valores de entrada, chamado de variável de encadeamento (*chaining variable*), possui geralmente tamanho *n* de bits igual ao tamanho da saída da função e o outro valor de entrada, referenciado como *tamanho de bloco*, possui tamanho *b* de bits em geral maior do que *n* – razão para uso do termo *compressão* (Stallings, 2010).

Como as funções de *hash* aceitam entradas de tamanhos arbitrários e comumente maiores do que o *tamanho de bloco*, é necessário implementar algum modelo para permitir o uso sucessivo da função de compressão.

A maioria das funções criptográficas de *hash* utiliza uma estrutura geral conhecida como função de *hash* iterada (Stallings, 2010). Essa estrutura foi proposta por Merkle (1979) em sua tese na Universidade de Stanford e é também conhecida como construção Merkle-Damgard (*Merkle-Damgard construction*). Nessa construção, a função de *hash* realiza um

pré-processamento da mensagem de entrada, acrescentando bits extras (*padding*) à mensagem até tornar o seu tamanho múltiplo do tamanho do bloco da função de compressão.

Por razões de segurança, também é acrescentado, ao final da mensagem, um valor indicando o tamanho da mensagem original (o processo de *padding* deve levar em consideração esse valor acrescido à mensagem). A inclusão do tamanho da mensagem torna o trabalho de um oponente mais difícil, pois ou ele terá que encontrar duas pré-imagens de mesmo tamanho e que resultem em um mesmo valor de *hash*, ou então duas mensagens de tamanhos diferentes que, acrescidas do valor de seus próprios tamanhos, gerem valores de *hash* iguais (Stallings, 2010).

Depois do pré-processamento de uma mensagem *M*, esta é dividida em blocos M_i de tamanho fixo de *b* bits. Conforme descrito por Menezes, Oorschot e Vanstone (1996), cada bloco M_i serve sequencialmente de entrada para a função interna de compressão *f*, que a cada passo calcula um novo resultado intermediário de comprimento fixo de *n* bits, como função:

- do resultado anterior de *n* bits (variável de encadeamento) e
- do bloco seguinte M_i de tamanho *b* bits.

Para o cálculo inicial do *hash* do primeiro bloco, é utilizado como variável de encadeamento um vetor de inicialização *VI* que é especificado como parte do algoritmo de *hash*.

Assim, conforme apresentado por Menezes, Oorschot e Vanstone (1996), considerando H_i como o resultado parcial da aplicação da função de compressão no estágio *i*, o processo geral para uma função de *hash* iterada com entrada M = $M_1M_2...M_t$ pode ser descrito pelo sistema de Equações 2:

$$\begin{aligned} &H_0 = VI; \\ &H_i = f(H_{i-1}, M_i), 1 \leq i \leq t; \\ &h(M) = g(H_t) \end{aligned} \tag{2}$$

H_{i-1} funciona como a variável de encadeamento entre o estágio *i*–1 e o estágio *i*. Uma função de transformação opcional *g* pode ser utilizada como passo final para mapear a variável de encadeamento de *n* bits para um resultado final de *m* bits (geralmente a função *g* não é utilizada e $g(H_t) = H_t$).

A motivação para o uso desta estrutura iterativa parte das pesquisas realizadas por Merkle (1989) e Damgard (1989) mostrando que se a função de compressão for resistente a colisões e for utilizado um processo de *padding* adequado, então a função de *hash* iterada resultante também será resistente a colisões. Dessa forma, o problema de projetar uma função

de *hash* segura que opera sobre pré-imagens de tamanhos arbitrários se reduz ao problema de projetar uma função de compressão resistente a colisões que opera sobre entradas de tamanhos fixos (Stallings, 2010).

As diferenças entre as diversas funções de *hash* existentes são geralmente determinadas pelo modo como seu algoritmo trata o pré-processamento da mensagem de entrada, pela função de compressão utilizada e pela função de transformação da saída (Menezes, Oorschot e Vanstone, 1996).

A Figura 3.3 mostra uma representação da estrutura de uma função de *hash* iterada para o algoritmo SHA-256 (*Secure Hash Algorithm*-256), que possui tamanho de *hash* de 256 bits e blocos de 512 bits. Dada uma mensagem de n bits ($512 < n < 1024$), esta deve ser pré-processada para acrescentar o preenchimento (*padding*) necessário para que seu comprimento seja múltiplo do bloco da função de compressão utilizada e também adicionar, ao final, o valor referente ao tamanho n da mensagem. A mensagem é então dividida em blocos de 512 bits (*M0* e *M1*), que são submetidos em sequência à função de compressão, juntamente com os resultados da função de compressão do passo imediatamente anterior (sendo utilizado um vetor de inicialização *H0* no primeiro passo). O resultado da função de compressão do último passo será o valor de *hash* final para a mensagem de entrada.

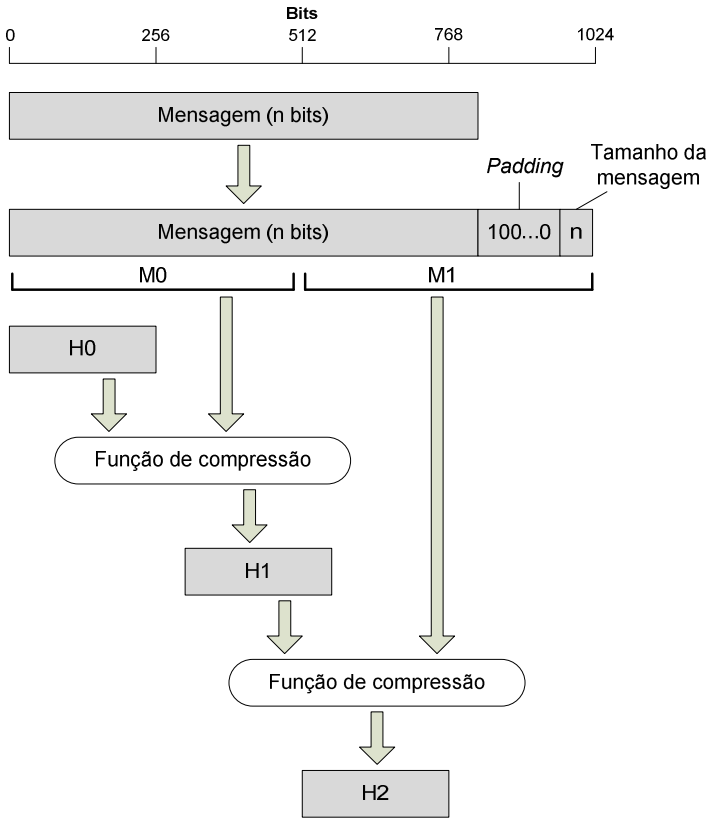

Figura 3.3: Estrutura de função de *hash* iterada (SHA-256) (adaptada de Kaminsky, 2004a).

Concatenação de funções de *hash*

Conforme descrito por Menezes, Oorschot e Vanstone (1996), uma estratégia que pode ser utilizada para aumentar a resistência a colisões consiste em criar uma função de *hash* que opera através do uso de duas funções de *hash* distintas concatenadas. Se ambas as funções utilizadas forem resistentes a colisões, então a função de *hash* resultante também será resistente a colisões e terá tamanho fixo de saída igual à soma dos tamanhos de saída das funções de *hash* intermediárias.

Ou seja, dadas duas funções de *hash h1* e *h2*, com tamanhos de saída de *n* e *m* bits respectivamente, podemos criar uma terceira função de *hash h* de tal forma que *h(x) = h1(x) || h2(x)*. O tamanho da saída de *h* será igual a *n + m* bits. Além disso, se *h1* e *h2* forem funções mutuamente independentes, será necessário encontrar colisões simultaneamente para *h1* e *h2* para uma mesma pré-imagem *x* para que também ocorra uma colisão de *h* em *x*.

3.1.2. Classificação de acordo com o funcionamento interno de funções de *hash*

O componente central de uma função de *hash* é a sua função de compressão e, sob o ponto de vista do tipo de operações que esta realiza, podemos destacar três categorias para as funções de *hash*:

- baseadas em cifras de bloco;
- baseadas em aritmética modular;
- baseadas em funções específicas de *hash*.

Funções baseadas em cifras de bloco

De acordo com Stallings (2010), diversas propostas foram feitas para criar funções de *hash* baseadas em cifras de bloco operando em modo CBC (*cipher block chaining*). A ideia básica é dividir a mensagem de entrada *M* em blocos de tamanho fixo *M1*, *M2*, ... *Mn* e utilizar uma cifra simétrica, sem chave secreta, como função de compressão para computar o valor de *hash G*, como apresentado no sistema de Equações 3:

$$\begin{aligned} &H_0 = VI; \\ &H_i = E(M_i, H_{i-1}), 1 \leq i \leq n; \\ &G = H_n \end{aligned} \tag{3}$$

Entretanto, cifras de bloco não possuem as propriedades de funções aleatórias e, na prática, exibem certas regularidades e fraquezas adicionais. Para exemplificar, cifras de bloco

são invertíveis, uma característica não muito condizente com os requisitos de construção de uma função unidirecional. Assim, não são exatamente claros quais requisitos de uma cifra de bloco são suficientes para construir uma função de *hash* segura e é possível que aqueles adequados para criar uma boa cifra de bloco não garantam também a geração de uma boa função de *hash* (Menezes, Oorschot e Vanstone, 1996).

Funções baseadas em aritmética modular

Já as funções de *hash* baseadas em aritmética modular, além de possuírem desempenho inferior às baseadas em funções específicas de *hash*, também não se tornaram muito populares devido ao histórico não muito favorável de projetos com falhas de segurança. De acordo com Menezes, Oorschot e Vanstone (1996), as poucas funções de *hash* baseadas em aritmética modular que suportaram ataques tendem a ser relativamente ineficientes.

Funções baseadas em funções específicas de *hash*

As funções criptográficas de *hash* mais utilizadas historicamente são as baseadas em funções específicas de *hash*. Como exemplo podemos citar os algoritmos *Message Digest* 4 e 5 (MD4 e MD5) e a família *Secure Hash Algorithm* (SHA), que se baseiam todos no algoritmo MD4. Por serem projetadas especificamente com o propósito de serem utilizadas por funções de *hash*, tendem a ser mais eficientes e não carregam consigo uma série de propriedades desnecessárias ou indesejáveis.

Essa classe de algoritmos utiliza operações lógicas do tipo XOR, AND, OR e NOT, além de adição modular e rotações de bits. Diversas dessas operações são executadas a cada estágio ou rodada (*round*). As operações são realizadas sobre palavras (words) de tamanho definido e os blocos da pré-imagem são por sua vez quebrados em palavras para que sejam realizadas as operações. Ao final de um conjunto de rodadas os resultados das operações são armazenados em valores intermediários de *hash* (*chaining values*), cada um deles de tamanho igual à palavra utilizada pela função. O valor final de *hash* é composto pela concatenação dos *chaining values* finais.

A Figura 3.4 ilustra como o algoritmo MD5 realiza o processamento de um bloco de 512 bits, o qual é dividido em quatro palavras de 128 bits que são submetidas, cada uma, a uma rodada composta de dezesseis passos (operações elementares) para produzir o resultado da função de compactação.

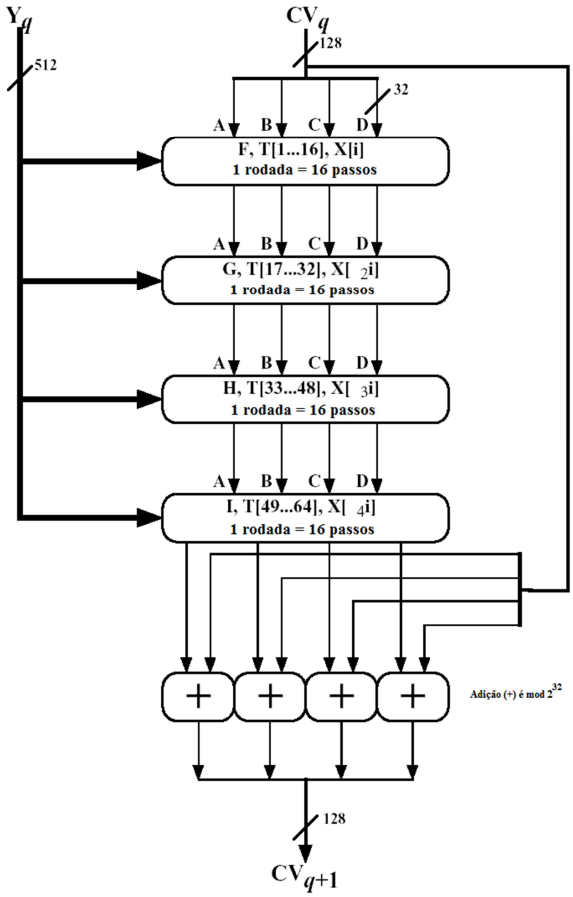

Figura 3.4: Processamento de um bloco de 512 bits no MD5 (adaptada de Stallings, 2005).

A Figura 3.5 apresenta a estrutura interna de cada passo, ou operação elementar, do algoritmo MD5.

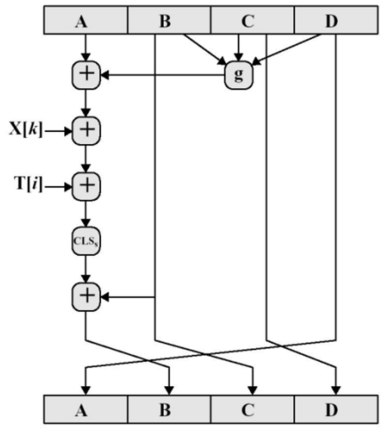

Figura 3.5: Uma operação elementar (um passo) do MD5 (adaptada de Stallings, 2005).

A Tabela 3.2 lista informações das principais funções de *hash* atualmente conhecidas.

Tabela 3.2: Detalhes de algumas das principais funções criptográficas de *hash*.

Algoritmo	Tamanho de *hash*	Rodadas x Passos por rodada	Tamanho da palavra	Tamanho de bloco	Ano	Desenvolvedor
MD4	128 bits	3x16	32 bits	512 bits	1990	Ronald Rivest
MD5	128 bits	4x16	32 bits	512 bits	1991	Ronald Rivest
SHA-0	160 bits	4x20	32 bits	512 bits	1993	NSA
SHA-1	160 bits	4x20	32 bits	512 bits	1995	NSA
Tiger	192 bits	3x8	64 bits	512 bits	1996	Ross Anderson e Eli Biham
RIPEMD-128	128 bits	4x16	32 bits	512 bits	1996	Hans Dobbertin, Antoon Bosselaers e Bart Preneel
RIPEMD-160	160 bits	5x16	32 bits	512 bits	1996	Hans Dobbertin, Antoon Bosselaers e Bart Preneel
Whirlpool	512 bits	10	8 bits	512 bits	2000	Paulo Barreto e Vincent Rijmen
SHA-2 (256/224)	256/224 bits	64	32 bits	512 bits	2002 / 2004	NSA
SHA-2 (512/384)	512/384 bits	80	64 bits	1024 bits	2002	NSA

Conforme já citado, os principais algoritmos de *hash* em uso atualmente são baseados na lógica e estrutura do algoritmo MD4, que foi criado por Ronald Rivest em 1990 (Rivest, 1992a).

O algoritmo MD5 (Rivest, 1992b), ainda é bastante utilizado, apesar de terem sido documentados ataques que atingiram seriamente sua credibilidade. Ele foi projetado originalmente como uma versão fortalecida do MD4 e incluiu mudanças nas operações internas e a adição de uma quarta rodada de dezesseis operações.

O algoritmo SHA-1 (National Institute of Standards and Technology, 1995), que se apresentou na prática como o principal substituto do MD5, foi desenvolvido pela NSA (*National Security Agency*) e publicado pelo NIST e também incluiu mudanças nas operações internas, além do acréscimo de uma quinta rodada de operações e mudança no tamanho de imagem gerado, que passou a ser de 160 bits. O SHA-1 difere do padrão original SHA-0 (publicado em 1993 como FIPS 180) apenas pela inclusão de uma rotação de bit na expansão

de blocos de 16 para 80 palavras, corrigindo assim erros detectados na proposta original (Menezes, Oorschot e Vanstone, 1996).

No ano de 2002, o NIST divulgou uma nova versão de funções de *hash* padrão, publicada como FIPS 180-2, conhecidas como família SHA-2 (National Institute of Standards and Technology, 2002). Foram especificados três novos algoritmos com tamanhos de imagem iguais a 256, 384 e 512 bits, conhecidos como SHA-256, SHA-384 e SHA-512, respectivamente. Esses algoritmos apresentam a mesma estrutura básica e utilizam os mesmos tipos de operações de lógica binária e aritmética modular que o SHA-1. Em 2008 foi publicado pelo NIST o FIPS 180-3 (National Institute of Standards and Technology, 2008), adicionando à família SHA-2 o algoritmo de 224 bits SHA-224. A família de algoritmos SHA-2 também foi publicada como RFC 4634 (Stallings, 2010).

Cabe citar também que a função de *hash* Whirlpool (Rijmen e Barreto, 2001) foi criada pelo brasileiro Paulo Barreto (juntamente com Vincent Rijmen) e utiliza uma cifra de bloco interna similar ao algoritmo de chave simétrica Rijndael (que venceu o concurso de seleção lançado pelo NIST para definição do algoritmo de chave simétrica que se tornaria padrão no governo norte-americano, recebendo o nome de AES – *Advanced Encryption Standard*).

3.2. SEGURANÇA DAS FUNÇÕES CRIPTOGRÁFICAS DE *HASH*

Assim como ocorre com os algoritmos de criptografia, existem duas formas principais de ataque a funções de *hash*: ataques por força-bruta e criptoanálise. Ataques por força-bruta a funções de *hash* consistem em se fazer tentativas exaustivas para encontrar mensagens distintas que resultem em um mesmo valor de *hash*. Esse tipo de ataque não depende do algoritmo específico utilizado, mas apenas do comprimento do valor de *hash*. A criptoanálise, ao contrário, é um ataque que consiste em encontrar e explorar fraquezas em um algoritmo criptográfico específico, de modo a tornar o esforço computacional para quebrá-lo menor do que aquele requerido para um ataque usando apenas força-bruta (Stallings, 2010).

Os principais ataques a uma função criptográfica de *hash* podem ser classificados em:

- Ataques de colisão
- Ataques de pré-imagem (*pre image*)
- Ataques de segunda pré-imagem (*second pre image*)

A Tabela 3.3 apresenta qual seria a força ideal esperada de uma função de *hash* para resistir aos ataques listados. A força ideal de uma boa função de *hash* é aquela correspondente

ao custo de um ataque por força-bruta. Funções de *hash* que possuam resistência aos ataques igual à força ideal requerida para cada um deles podem ser consideradas idealmente seguras.

Tabela 3.3: Força ideal esperada de uma função de *hash*, com tamanho de imagem igual a *n* bits, para diferentes ataques (adaptada de Menezes, Oorschot e Vanstone, 1996).

Objetivo de projeto	Força ideal	Objetivo do ataque
Resistência a ataques de pré-imagem	2^n	Descobrir uma pré-imagem para uma dada imagem
Resistência a ataques de segunda pré-imagem	2^n	Encontrar uma segunda pré-imagem que produza a mesma imagem
Resistência a ataques de colisão	$2^{n/2}$	Produzir qualquer colisão

O foco da criptoanálise de funções de *hash*, por sua vez, conforme descrito por Stallings (2010), está na estrutura interna da função de compressão *f* e procura encontrar técnicas de produzir colisões para uma única execução de *f*. O ataque deve levar em consideração o valor do vetor de inicialização, assim como a análise dos padrões de mudança nos bits a cada rodada de execução do algoritmo.

3.2.1. Ataques de colisão

Como uma função de *hash* mapeia um conjunto de domínio *D* de tamanho arbitrariamente finito para um conjunto de imagem *R* de comprimento fixo de *n* bits ($h : D \rightarrow R$ e $|D| > |R|$), a função é, então, do tipo muitos-para-um e a existência de colisões é inevitável. Se a função de *hash* for "aleatória" de modo que todas as saídas existentes sejam igualmente possíveis, então a probabilidade de que duas pré-imagens escolhidas aleatoriamente resultem em um mesmo valor de *hash* é igual a 2^{-n} e depende, portanto, apenas do tamanho de saída da função de *hash* (Menezes, Oorschot e Vanstone, 1996). Assim, a propriedade de resistência a colisões de uma função de *hash* é diretamente dependente do tamanho dos valores de *hash* que ela gera. Se a saída de uma função de *hash* não possuir comprimento suficientemente grande, ela fatalmente não será resistente a colisões, mesmo que atenda satisfatoriamente os requisitos de pseudo-aleatoriedade e não inversão.

Um ataque de colisão por força-bruta requer um esforço computacional que pode ser explicado por meio do paradoxo do aniversário (*birthday paradox*). O paradoxo do aniversário consiste em determinar qual é o número mínimo *p* de pessoas necessárias para ter uma chance maior que 50% de que ao menos duas delas façam aniversário na mesma data. Esse número *p* é consideravelmente baixo e corresponde a aproximadamente: $p = 1{,}18 *$

$\sqrt{n} \approx \sqrt{n}$, sendo *n* o número de dias do ano (para cálculo detalhado ver Stallings, 2010). Assim, de modo análogo, para ter uma chance maior de 50% de encontrar duas pré-imagens que resultem em um mesmo valor de *hash* devemos testar aproximadamente $\sqrt{2^n} = 2^{n/2}$ pré-imagens distintas, sendo *n* o número de bits do código de *hash* da função utilizada. Como exemplo, para realizar um ataque de colisão por força-bruta ao algoritmo SHA-1 (160 bits), seria necessário um esforço computacional da ordem de $2^{160/2} = 2^{80}$.

3.2.2. Ataques de pré-imagem e segunda pré-imagem

Nos ataques de segunda pré-imagem é conhecida pelo menos uma mensagem que produz o valor de *hash* a ser buscado, enquanto nos ataques de pré-imagem não há nenhuma imagem conhecida que produza o valor de *hash* procurado. Esse fato, porém, geralmente apresenta pouca vantagem para um atacante que realiza criptoanálise. Dessa forma, para realizar um ataque de pré-imagem e segunda pré-imagem, a abordagem de um atacante é basicamente a mesma e consiste em descobrir, para um valor de *hash h* específico, uma mensagem *x* que resulte no valor *h* quando aplicada à função de *hash*.

Um ataque de força bruta para quebrar a resistência a pré-imagem consiste em escolher mensagens aleatoriamente, calcular seu valor de *hash* e testar até encontrar uma colisão com o valor *h*. O custo computacional para esse ataque é da ordem de 2^n, sendo *n* o tamanho em bits do valor de *hash*. Como, em termos probabilísticos, é necessário testar, em média, ao menos metade do número total de *hashes* possíveis até encontrar uma colisão específica, um adversário terá que testar, em média, 2^{n-1} mensagens até encontrar uma que resulte no valor de *hash h*.

De acordo com Menezes, Oorschot e Vanstone (1996), se um oponente pré-calcular e armazenar um grande número de pares de saída/entrada de uma função de *hash*, será possível trocar esse esforço de processamento e armazenamento por menor tempo de ataque posteriormente. O conjunto dos pares de entrada e saída pré-calculados e tabulados utilizando essa técnica de ataque é conhecido como *Rainbow Tables*. No caso de uma função de *hash* de 128 bits, se forem pré-calculados os valores de *hash* de 2^{40} mensagens escolhidas aleatoriamente, o custo esperado para encontrar uma pré-imagem cai de 2^{128} para 2^{88}.

Já em situações em que, ao invés de apenas uma pré-imagem, existam *t* imagens alvo para as quais se deseja encontrar pré-imagens correspondentes e seja preciso encontrar apenas uma pré-imagem que gere alguma das imagens alvo, então a probabilidade de sucesso aumenta *t* vezes em relação à probabilidade original (Menezes, Oorschot e Vanstone,1996).

Esse é o caso de ataques para tentar enganar um processo de filtragem de arquivos por uso de uma BHAC. O atacante precisa apenas produzir um documento relevante que gere valor de *hash* igual a qualquer um dos arquivos da BHAC, sendo que esta pode ser da ordem de 2^{25} *hashes*.

3.2.3. Ataques conhecidos a funções criptográficas de *hash*

O algoritmo MD5 foi, por muito tempo, o algoritmo mais utilizado em Computação Forense para a montagem de BHACs. Contudo, devido aos diversos ataques a que foi submetido e que demonstraram suas vulnerabilidades, foi sendo substituído gradativamente pelo algoritmo SHA-1. Entretanto, este também vem apresentando vulnerabilidades expostas por ataques de criptoanálise.

Já em 1993 foi apresentada uma pseudo-colisão do algoritmo MD5 (Boer e Bosselaers, 1993). Pseudo-colisões são colisões encontradas variando não apenas a pré-imagem, mas também o vetor de inicialização. Esse tipo de ataque mais restrito, segundo Menezes, Oorschot e Vanstone (1996), apesar de não representar uma fraqueza direta da função de *hash*, pode ser estendido a ataques completos através do uso de algumas técnicas padronizadas.

Em 1996, Hans Dobbertin divulgou ter encontrado uma colisão na função de compressão do MD5 (Dobbertin, 1996). Conforme já explicado, a segurança de uma função de *hash* está intimamente ligada à segurança de sua função de compressão.

Finalmente, em 2004, Wang et al. (2004) apresentaram colisões do algoritmo MD5, entre outros algoritmos, na conferência Crypto04. Wang e Yu (2005) detalharam posteriormente o ataque realizado, um tipo de ataque diferencial chamado diferencial modular, para encontrar colisões nos algoritmos MD4, MD5, HAVAL-128, RIPEMD e SHA-0. Com um custo computacional da ordem de 2^{39}, foi possível encontrar colisões para o MD5 em aproximadamente 1 hora utilizando um computador IBM P690.

Em 2007, um ataque prático a certificados digitais X.509 usando MD5 foi realizado com o uso de computação distribuída com cerca de 1200 máquinas, através do projeto conhecido como “HashClash” (Stevens, Lenstra e Weger, 2009). Esse ataque prático implementou as técnicas descritas pelos autores em Stevens, Lenstra e Weger (2007). Por meio dessas técnicas, para quaisquer dois prefixos de mensagem P e P' escolhidos, podem ser criados dois sufixos S e S' de tal forma que os valores concatenados $P||S$ e $P'||S'$ apresentam valores de *hash* MD5 que colidem, com um custo computacional estimado da ordem de 2^{50} chamadas à função de compressão do MD5. Utilizando essas características, foram

construídos dois certificados X.509 baseados em MD5 com *hashes* iguais, mas com valores diferentes nos campos chave pública e *Distinguished Name*. Conforme os próprios autores descreveram: "*Apesar do potencial prático do ataque desta construção de colisões por prefixo escolhido ser limitado, é de preocupação maior que as colisões aleatórias para MD5.*"

Com relação a ataques de pré-imagem, Sazaki e Aoki publicaram em 2009 um ataque teórico com complexidade de $2^{116,9}$ para gerar pseudo pré-imagens para o MD5 e com complexidade de $2^{123,4}$ para gerar pré-imagens (Sazaki e Aoki, 2009). Apesar de computacionalmente inviáveis nos dias atuais, esses ataques teóricos elevam as vulnerabilidades do algoritmo MD5 para além dos ataques de colisão.

Para o algoritmo SHA-1 ainda não são conhecidos estudos de criptoanálise efetivos para ataques de pré-imagem, mas existem ataques de colisão teóricos publicados que exigem esforço computacional menor do que as cerca de 2^{80} operações esperadas para obter sucesso em um ataque de força-bruta.

Em 2005 foi publicado um artigo por Rijmen e Oswald acerca de um ataque de colisão a uma versão reduzida do SHA-1, com 53 rodadas ao invés de 80, que requeria esforço computacional menor do que 2^{80} operações (Rijmen e Oswald, 2005).

Wang, Yin e Yu, apresentaram, também em 2005, o primeiro ataque de colisão à versão completa do SHA-1 (Wang, Yin e Yu, 2005). Esse ataque mostrou uma combinação de técnicas que diminuíam a complexidade necessária para encontrar colisões para menos de 2^{69} operações de *hash*. Ainda em 2005, Wang fez uma apresentação na Crypto05, em conjunto com Andrew e Frances Yao, mostrando melhorias no ataque de colisão que diminuíam o esforço computacional para 2^{63} operações (Wang, Yao e Yao, 2005).

Novos estudos a respeito de vulnerabilidades do SHA-1 continuam a ser feitos, podendo ser citado o trabalho de Manuel (2008), que apresenta uma metodologia que pode ser utilizada para ataques de colisão ao SHA-1 e o projeto "HashClash" (Stevens, Lenstra e Weger, 2009) que obteve sucesso com ataques de colisão ao MD5 e também busca obter colisões para o algoritmo SHA-1 .

3.3. PERSPECTIVAS

Como os algoritmos MD5 e SHA-1 ainda continuam a ser utilizados em Computação Forense, especialmente como forma de detecção de alterações em conteúdos digitais e identificação única de arquivos com a montagem de BHACs, é preciso fazer considerações a

respeito do impacto para essas aplicações decorrente dos ataques conhecidos às funções de *hash*.

Para o caso específico de BHACs, o problema seria a existência de ataques de pré-imagem viáveis, que possibilitariam a criação de arquivos que possuam o mesmo valor de *hash* de algum arquivo que esteja na BHAC e, assim, o arquivo criado seria erroneamente filtrado em um exame pericial. Nesse caso, só faria sentido utilizar como pré-imagem um arquivo modificado que tivesse significado para um atacante (Steel, 2006). A inclusão de arquivos desprovidos de significado não traria grandes prejuízos ao processo de filtragem de arquivos pelo seu valor de *hash*, pois, se não possuem significado, então deveriam mesmo ser filtrados da análise pericial.

Já para o processo de detecção de alterações de conteúdos digitais, os ataques de pré-imagem são potencialmente mais perigosos, pois um atacante pode modificar um arquivo que tenha conteúdo incriminador de modo que este continue resultando no mesmo valor de *hash* anterior, mas o arquivo, dessa vez, não precisa ter significado, pois o objetivo é simplesmente apagar o seu conteúdo original.

Thompson (2005) apresenta uma análise a respeito do impacto para a Computação Forense resultante das colisões no algoritmo MD5 encontradas por Wang et al. (2004). Ele destaca que os ataques de colisão realizados requerem a existência de um conjunto de características nas variáveis internas do MD5 e é irrealístico acreditar que tais características ocorram naturalmente ou possam ser reproduzidas para criar conteúdo inteligível. Contudo, o ataque apresentado por Stevens, Lenstra e Weger (2007) demonstra que o algoritmo MD5, mesmo fora do contexto estrito da criptografia e suas elevadas exigências de segurança, não oferece perspectivas favoráveis. Ou, conforme afirma Kaminsky (2004b): "*na comunidade de segurança, nós tendemos a reclamar da natureza de 'mudança de fase' dos nossos sistemas que repentinamente colapsam de seguros para inseguros com a descoberta de um exploit 'zero day'. A mudança de fase para o MD5 ainda não chegou, mas ela virá, algum dia. Ninguém irá se surpreender quando o dia chegar*".

Com as vulnerabilidades expostas do MD5, o algoritmo SHA-1 foi gradativamente assumindo o papel de principal substituto. Os ataques de colisão ao SHA-1 publicados não apresentam preocupações imediatas para as aplicações em Computação Forense. Contudo, conforme destacado por Stallings (2010), o NIST anunciou em 2005 a intenção de substituir o SHA-1 pela família de algoritmos SHA-2 até 2010, apesar de ainda continuar disponibilizando os RDS completos da NSRL apenas com os valores de *hash* MD5 e SHA-1.

Dessa forma, o SHA-1 não deve subsistir por muito tempo como a principal função de *hash* utilizada.

SHA-3

Entretanto, em resposta aos avanços que vêm sendo obtidos na criptoanálise de funções de *hash* e como até mesmo a família de *hashes* SHA-2 compartilha a mesma estrutura e operações matemáticas de seus predecessores, o NIST anunciou uma competição pública em novembro de 2007 para o desenvolvimento de um novo algoritmo criptográfico de *hash*, antecipando uma possível aposentadoria da família SHA-2. O NIST recebeu inicialmente sessenta e quatro propostas e selecionou cinquenta e uma para avançar para a primeira rodada da competição.

A segunda rodada classificou quatorze candidatos, que foram submetidos à apreciação pública por um ano. Em 09 de novembro de 2010 o NIST selecionou cinco algoritmos finalistas - *BLAKE, Grøstl, JH, Keccak e Skein* – baseado nas avaliações públicas e internas dos algoritmos classificados na segunda rodada. Os finalistas ainda puderam fazer pequenas modificações em seus algoritmos e submetê-los até o dia 16 de janeiro de 2011. O NIST pretende realizar uma conferência com os candidatos na primavera de 2012 (nos Estados Unidos) para discutir as avaliações públicas a respeito de seus algoritmos e selecionar o vencedor posteriormente ainda no ano de 2012. O algoritmo vencedor receberá o nome de SHA-3.

3.4. *HASHING* DE SIMILARIDADE

Além do uso tradicional para identificar unicamente conteúdos digitais, uma outra forma de utilizar funções de *hash* na área de Computação Forense vem ganhando espaço ultimamente: trata-se de *hashing* de similaridade (*similarity hashing* ou *fuzzy hashing*). A idéia por trás do conceito de *hashing* de similaridade consiste em avaliar qual é o grau de similaridade entre dois ou mais arquivos, ao invés de simplesmente apresentar uma resposta binária, sim ou não, à pergunta a respeito da igualdade entre dois conteúdos digitais.

Para calcular o grau de similaridade, o método comumente utilizado consiste em calcular *hashes* parciais de pedaços ou blocos do conteúdo digital, de forma que dois arquivos que contenham alguns blocos de conteúdo com o mesmo *hash*, mesmo não possuindo todo o conteúdo igual, possuem algumas partes iguais e, portanto, possuem algum grau de similaridade.

A abordagem mais simples seria simplesmente dividir os conteúdos digitais em tamanhos fixos e calcular o *hash* de cada uma dessas partes (Figura 3.6). Os *hashes* de cada bloco indicariam quais arquivos compartilham aquele mesmo conteúdo. Uma implementação de *hashes* parciais sobre blocos de tamanho fixo é encontrada na ferramenta de cópia de dados "dcfldd" (Nicholas, 2006). Entretanto, essa abordagem não funciona na prática para a análise de similaridade de conteúdos, pois uma simples inserção ou deleção no começo de um conteúdo digital irá alterar o conteúdo de todos os blocos subsequentes e, mesmo que um arquivo tenha somente 1 bit a mais de diferença que outro de mesmo conteúdo, será classificado como não similar.

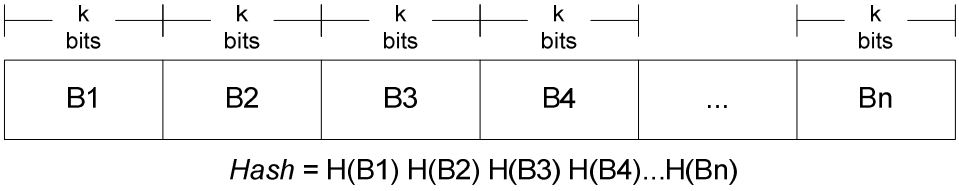

Figura 3.6: *Hashes* parciais de um conteúdo digital dividido em blocos de igual tamanho.

Para evitar esse problema, a solução mais utilizada é definir a posição de ínicio e fim de cada bloco a partir dos valores presentes no próprio conteúdo a ser analisado. Para esse fim, é utilizado um *hash* chamado de *hash* de contexto (*context hash*) ou *hash* móvel (*rolling hash*). O *hash* de contexto é calculado sobre uma janela deslizante (*sliding window*) que percorre o conteúdo a ser analisado e, quando o *hash* de contexto assume um valor pré-determinado, o gatilho (*trigger*) para determinar a borda de um bloco é disparado e o *hash* de conteúdo daquele bloco é calculado (ver Figura 3.7). Essa solução suporta bem inserções e retiradas de dados que venham a ocorrer entre dois conteúdos digitais semelhantes, pois as partes que continuarem iguais vão disparar o *hash* de contexto nas mesmas posições e, assim, os blocos para os quais serão calculados os *hashes* de conteúdo serão exatamente os mesmos.

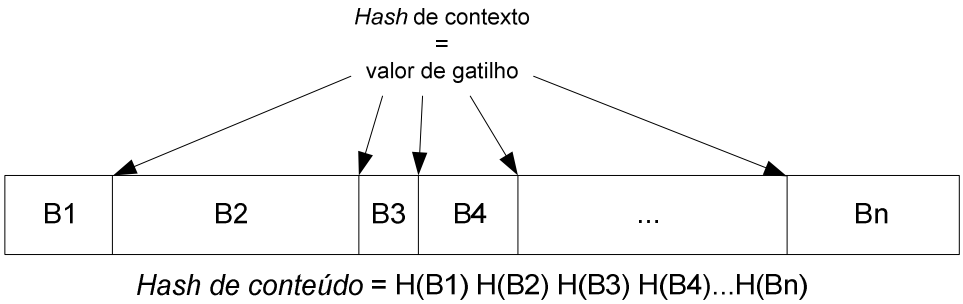

Figura 3.7: *Hash* de similaridade de um conteúdo digital com blocos com tamanhos definidos por *hash* de contexto.

Context Triggered Piecewise Hashing

Um dos algoritmos precursores do uso de *hashes* de similaridade no âmbito da Computação Forense foi proposto por Kornblum (2006) baseado em uma adaptação do algoritmo para detecção de SPAMs em e-mails conhecido como "spamsum" (Tridgell, 2002). O algoritmo de Kornblum é o *Context Triggered Piecewise Hashing* – CTPH (*hashing* seccionado disparado por contexto) e foi implementado na ferramenta "ssdeep" (Kornblum, 2011).

O ssdeep utiliza uma janela deslizante de 7 bytes para o cálculo do *hash* de contexto, que é obtido com o uso da função de *checksum* Adler32. Quando os *k* bits menos significativos do *hash* de contexto forem iguais a um, o gatilho é disparado para determinar a borda de um bloco (*k* é chamado de tamanho de bloco – *block size*). O valor de *k* é calculado de acordo com o tamanho do conteúdo de entrada, sendo maior para conteúdos maiores, de modo que conteúdos de tamanhos variados produzam um valor final de *hash* de tamanho semelhante.

O ssdeep usa simultaneamente dois gatilhos de *hash* de contexto, usando tamanho de bloco igual a *k* e a *2k*, obtendo assim *hashes* parciais de conteúdo com dois níveis de granularidade. O *hash* de conteúdo dos blocos é calculado usando uma codificação base64 dos 6 bits menos significativos (*least significant 6 bits* - LS6B) da função de *hash* não criptográfica conhecida como *Fowler-Noll-Vo* (FNV) na sua versão de 32 bits.

A assinatura final do conteúdo digital gerada pelo ssdeep é composta pelo tamanho de bloco *k*, os dois conjuntos de *hash* LS6B (para tamanho de bloco igual a *k* e a *2k*) e o nome do arquivo entre aspas. Um exemplo de assinatura CTPH gerada pelo ssdeep pode ser visto na Figura 3.8.

```
1536:T0tUHZbAzIaFG91Y6pYaK3YKqbaCo/6Pqy45kwUnmJrrevqw+oWluBY5b32TpC0:
T0tU5s7ai6ptg7ZNcqMwUArKvqfZlMC0,"/music/Dragostea Din Tei.mp4"
```

Figura 3.8: Exemplo de assinatura CTPH gerada pelo ssdeep (adaptada de Kornblum, 2006).

As diferenças entre dois conteúdos *s1* e *s2* (distância de edição) são definidas como "*o menor número de pontos de mutação requeridos para transformar s1 em s2*". O ssdeep usa uma versão de cálculo de distância de edição com peso, em que inserções e deleções têm peso 1, alterações têm peso 3 e inversões (caracteres em ordem inversa) têm peso 5. A Figura 3.9 mostra um exemplo de uso do ssdeep para comparação de similaridade entre arquivos.

```
$ ls -l exemplo.jpg
-rw-rw---- 1 usernm usernm 313478 Apr 11 11:05 exemplo.jpg

$ dd if=exemplo.jpg of=primeiro bs=1 count=120000
100000+0 records in
100000+0 records out

$ dd if=exemplo.jpg of=ultimo bs=1 skip=200000
113478+0 records in
113478+0 records out

$ ssdeep -b exemplo.jpg > sigs

$ ssdeep -bm sigs primeiro ultimo
primeiro matches exemplo.jpg (54)
ultimo matches exemplo.jpg (65)
```

Figura 3.9: Exemplo de comparação de similaridade entre arquivos utilizando a ferramenta ssdeep. São utilizados como comparação um arquivo original e dois outros arquivos com conteúdos parciais do arquivo original (adaptada de Kornblum, 2006).

Multi-Resolution Similarity Hashing

Outra proposta de *hashing* de similaridade é apresentada por Roussev, Richard III e Marziale (2007) e é conhecida como *Multi-Resolution Similarity Hashing* (*hashing* de similaridade de multi-resolução). O objetivo proposto pelos autores é utilizar essa solução para complementar o uso de cálculo de *hash* tradicional durante a passagem inicial pelos dados brutos da mídia a ser analisada. O algoritmo proposto utiliza também um *hash* de contexto (dado pela função de *hash* "djb2") sobre uma janela de 7 bytes e, para o *hash* de conteúdo, calcula *hashes* MD5 de cada bloco, sendo que 44 bits selecionados do *hash* MD5 de cada bloco são inseridos em um ou mais filtros de Bloom (Bloom, 1970) que são concatenados para gerar o resultado final do *hashing* de similaridade. Ao se definir diferentes tamanhos de bits para a seleção do gatilho

do *hash* de contexto (geralmente 8, 12, 16, 20, 24, 28 e 32 bits) são obtidos diferentes níveis de granularidade para os *hashes* resultantes, ampliando a flexibilidade na detecção de similaridades.

Utilização de *hashing* de similaridade

A ferramenta forense *AccessData FTK*, na sua versão 3, implementa a funcionalidade de *hashing* de similaridade utilizando o algoritmo CTPH. Conforme Hurlbut (2009), o processamento de *hashing* de similaridade pode ser utilizado para comparar arquivos ativos no sistema de arquivo com conteúdos parciais recuperados do espaço não alocado (indicando que o arquivo original provavelmente já esteve presente na mídia digital analisada), para detectar alterações maliciosas com o objetivo de evitar a detecção de arquivos específicos e para examinar conteúdos parciais de arquivos recuperados em comparação com arquivos incriminadores (como imagens de abuso sexual de criança ou adolescente).

A NSRL também oferece uma listagem de assinaturas CTPH para 8.334.077 arquivos de sua base (conforme consulta[8] realizada em setembro de 2011). Essa listagem contém os *hashes* SHA-1 e as assinaturas ssdeep 2.0 correspondentes.

Também é possível criar e utilizar uma base de assinaturas de *hashing* de similaridade de arquivos *alerta* para ser utilizada para auxiliar a localizar arquivos incriminadores que foram apenas parcialmente recuperados pelo perito ou que foram modificados propositalmente com o intuito de camuflar sua presença na mídia analisada.

Foram detalhadas neste capítulo as principais características das funções criptográficas de *hash*, os aspectos relacionados à sua segurança – com enfoque para aqueles que impactam mais diretamente a construção de BHACs – e suas perspectivas futuras de utilização. Também foram apresentados algoritmos de *hashing* de similaridade, que começam a ser utilizados na área de Computação Forense para identificar arquivos que possuem algum nível de semelhança de conteúdo e que também podem vir a integrar BHACs.

No capítulo seguinte serão apresentados os conceitos relacionados ao processo de MD para classificar um conteúdo digital – identificado pelo seu valor de *hash* – de acordo com sua potencial relevância para exames periciais.

[8] http://www.nsrl.nist.gov/ssdeep.htm

4. MINERAÇÃO DE DADOS

Uma das possíveis análises que podem ser realizadas em bases de dados com o uso de MD é a predição da classificação. Neste trabalho, o interesse principal é conseguir realizar a classificação de arquivos quanto à sua relevância para exames periciais. Nesse contexto, o uso de algoritmos de aprendizagem baseados em AD são uma possibilidade interessante e promissora tendo em vista os objetivos propostos.

Este capítulo apresenta os conceitos relacionados à criação de um banco de dados que possa ser utilizado para implementar procedimentos de descoberta de conhecimento através de técnicas de classificação com o uso de AD.

4.1. BANCOS DE DADOS

Fontes de dados podem ser disponibilizadas em diversos formatos, como arquivos de texto ou planilhas eletrônicas, mas o tipo mais utilizado é através do uso de banco de dados (BD). Segundo Frawley et al. (1992), um BD é uma coleção de dados integrada logicamente, armazenada em um ou mais arquivos e organizada para facilitar o armazenamento, modificação e obtenção de informações relacionadas de maneira eficiente. Além da coleção de dados, um sistema gerenciador de BD (SGBD) é composto por um conjunto de programas que permitem aos usuários acessar e modificar os dados. A proposta principal de um SGBD é fornecer aos usuários uma visão abstrata dos dados e um ambiente seguro e eficiente de armazenamento e de acesso às informações (Silberschatz et al., 2001).

Segundo Silberschatz et al. (2001), a estrutura de um BD é definida por um modelo de dados, que consiste de uma coleção de ferramentas conceituais para descrever os dados, o relacionamento entre eles, sua semântica e restrições de consistência. Dentre os diversos modelos de BD existentes, podemos citar o hierárquico, de rede, relacional, orientado a objetos e objeto-relacional. Destacaremos o modelo relacional por ser o mais utilizado e por ter sido usado para montar a base de dados deste trabalho.

4.1.1. Modelo de dados relacional

O modelo relacional utiliza uma coleção de tabelas nomeadas para representar tanto os dados quanto os relacionamentos entre esses dados. Cada tabela é composta por múltiplos atributos nomeados e existe um conjunto de valores possíveis para cada atributo (domínio do atributo). Assim, se tivermos uma tabela T com n atributos e os domínios de cada atributo forem

representados por D_1, D_2, ... D_n, então *T* deve ser um subconjunto do produto cartesiano dos domínios de seus atributos, representado por:

$$D_1 \times D_2 \times ... \times D_n$$

Cada linha da tabela *T* corresponde a um conjunto de valores v_1, v_2, ... v_n (um valor para cada atributo), de modo que cada valor está restrito ao domínio de seu atributo correspondente.

Como um subconjunto de um produto cartesiano de uma lista de domínios é definida na Matemática como uma relação, então pode-se utilizar os termos matemáticos relação e tupla como substitutos para os termos tabela e linha. A Figura 4.1 apresenta alguns dos conceitos do modelo de dados relacional.

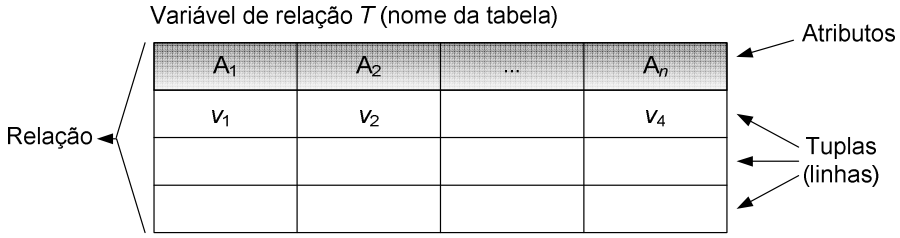

Figura 4.1: Conceitos de modelo relacional (adaptada de Date, 2004).

A consistência dos dados é definida através da declaração de restrições (*constraints*) aos valores das tuplas. Duas tuplas distintas não podem ter o mesmo valor para todos os seus atributos e as restrições de chave (*key constraints*) definem um subconjunto de atributos para os quais não há valores repetidos em nenhuma tupla da relação. Ou seja, uma chave *K* de um esquema de relação *T* é um subconjunto dos atributos de *T* para o qual, para quaisquer duas tuplas t_1 e t_2 da relação, $t_1[K] \neq t_2[K]$. Ainda, a retirada de qualquer atributo de *K* retira de *K* o status de chave da relação *T*. A chave utilizada para identificar as tuplas de uma relação é chamada de chave primária (*primary key*) e é escolhida entre as possíveis chaves candidatas de uma relação (Elmasri e Navathe, 2010).

Segundo Elmasri e Navathe (2010), as restrições de integridade referencial são especificadas para manter a consistência entre tuplas de dois esquemas de relação interligados logicamente. Um subconjunto *FK* dos atributos de um esquema de relação R_1 é uma chave estrangeira de R_1 que referencia uma relação R_2 se satisfaz às condições:

- os atributos de *FK* possuem os mesmos domínios que os atributos da chave primária *PK* de R_2;

- toda tupla em R_1 deve possuir valores para *FK* que sejam nulos ou que estejam presentes em alguma tupla em *PK* de R_2.

As pesquisas em um modelo relacional teórico são realizadas utilizando uma linguagem de busca procedural, que consiste de um conjunto de operações que recebem uma ou mais relações como entrada e produzem uma nova relação como resultado (Silberschatz et al., 2001). As operações fundamentais são seleção, projeção, união, diferença entre conjuntos, renomeação e produto cartesiano, mas existem outras operações, como interseção de conjuntos, divisão e junção natural. Os fundamentos do modelo relacional teórico tem como base álgebra relacional.

Na prática, a linguagem para manipulação de dados utilizada pelos principais SGBDs é a *Structured Query Language* (SQL). A SQL foi baseada em álgebra e cálculo relacionais, mas pode realizar mais do que apenas buscas por dados em um BD. Ela pode ser utilizada para definir estrutura de dados, modificar dados no BD e especificar restrições de segurança. A SQL é composta por diversas partes (Silberschatz et al., 2001), dentre as quais:

- Linguagem de Definição de Dados (*Data Definition Language* – DDL);
- Linguagem de Manipulação de Dados (*Data Manipulation Language* – DML);
- Definição de visões;
- Controle de transações;
- Definição de restrições de integridade;
- Especificação de autorizações de acesso.

4.1.2. Modelo Entidade-Relacionamento

Em um nível de abstração acima do modelo relacional, os projetos de BD são feitos usando o modelo Entidade-Relacionamento (E-R), que são traduzidos posteriormente para o modelo relacional (Silberschatz et al., 2001). O modelo de dados E-R é um modelo semântico que entende o mundo real como sendo composto por objetos básicos (entidades) e relacionamentos entre esses objetos. A construção de diagramas E-R utiliza três noções básicas: conjuntos de entidades, conjuntos de relacionamentos e atributos, que são exemplificados na

Figura 4.2. Note que os retângulos representam as entidades, losangos representam relacionamentos e as elipses representam atributos (atributos de chaves primárias estão sublinhados).

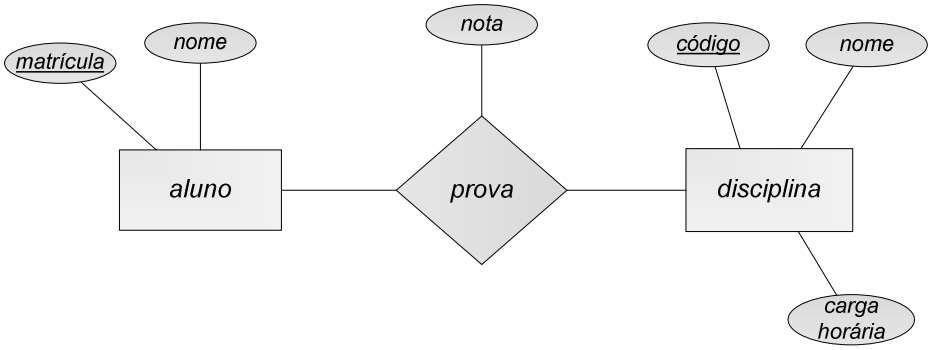

Figura 4.2: Exemplo de diagrama E-R (adaptada de Silberschatz et al., 2001).

4.1.3. Gerenciamento de Banco de Dados *versus* Mineração de Dados

O gerenciamento de BD faz parte dos campos com os quais a descoberta de conhecimento em bases de dados geralmente lida, assim como aprendizagem de máquina, análise estatística, e descoberta científica (Frawley et al., 1992). A entrada fundamental para um sistema de descoberta são os dados armazenados em BD (*raw*), mas estes apresentam algumas características que devem ser tratadas adequadamente por sistemas de descoberta, pois BD são dinâmicos, incompletos, possuem ruídos e são geralmente grandes (Frawley et al., 1992).

A Tabela 4.1 e a Tabela 4.2 apresentam um paralelo entre os termos utilizados nas áreas de gerenciamento de BD e de MD, assim como alguns pontos de vista conflitantes entre elas.

Tabela 4.1: Tradução de termos entre Gerenciamento de Banco de Dados e Mineração de Dados (adaptada de Frawley et al., 1992).

Gerenciamento de Banco de Dados	Mineração de Dados
banco de dados: uma coleção logicamente integrada de arquivos dinâmicos	um conjunto fixo de exemplos
arquivo	banco de dados, conjunto de dados, conjunto de instâncias
tupla, registro	instância, exemplo, vetor de característica
campo, atributo	característica, atributo
domínio de campo	valores de campo possíveis
dicionário de dados	tipo de campo e informação de domínio
dados relacionais	um conjunto de instâncias
dados estruturados, orientados a objeto	dados relacionais
condição lógica	descrição de conceito

Tabela 4.2: Pontos de vista conflitantes entre Gerencimento de Banco de Dados e Mineração de Dados (adaptada de Frawley et al., 1992).

Gerenciamento de Banco de Dados	Mineração de Dados
banco de dados é uma entidade ativa e evolutiva	banco de dados é uma coleção estática de dados
registros podem conter informações erradas ou incompletas	instâncias são geralmente completas e livres de ruídos
um campo típico é numérico	uma característica típica é binária
um banco de dados tipicamente contém milhões de registros	conjuntos de dados tipicamente contêm várias centenas de instâncias

4.2. DESCOBERTA DE CONHECIMENTO E MINERAÇÃO DE DADOS

O processo completo de descoberta de conhecimento útil a partir de dados é conhecido pelo termo Descoberta de Conhecimento em Bases de Dados (DCBD). De acordo com Fayyad, Piatetsky-Shapiro e Smyth (1996), DCBD é um processo não trivial de identificação de padrões válidos, novos, potencialmente úteis e compreensíveis nos dados analisados. Os padrões identificados devem, portanto, ser válidos com algum grau de certeza quando aplicados a novos dados, devem apresentar alguma novidade para o sistema ou para o usuário e devem trazer algum benefício ou vantagem para o usuário ou para a tarefa. Além disso, os

padrões devem ser identificados através de buscas ou inferências e não através da simples computação de quantidades pré-definidas, como a média de um conjunto de números.

Conforme Fayyad, Piatetsky-Shapiro e Smyth (1996), o processo de DCBD é interativo e iterativo, envolvendo várias etapas que vão resultar na descoberta de conhecimento. Algumas dessas etapas são:

1. Compreensão do domínio da aplicação e descoberta dos objetivos do processo de DCBD;
2. Criação de um conjunto de dados alvo: seleção do conjunto ou subconjunto de dados ou variáveis que serão analisados;
3. Limpeza dos dados e pré-processamento: consiste em realizar ações de ajuste nos dados para corrigir possíveis inconsistências ou ruídos e definir estratégias para lidar com falta de dados;
4. Redução e projeção dos dados: o número de variáveis a serem consideradas pode ser diminuído com a redução de dimensionalidade ou através de métodos de transformação que levem em consideração as características úteis para representar os dados no domínio da aplicação;
5. Escolha de um método de MD adequado para os objetivos do processo de DCBD: existem diversos método de MD possíveis de serem utilizados, como sumarização, classificação, regressão, clusterização, dentre outros;
6. Análise exploratória e seleção do modelo e hipótese: decisão a respeito de quais modelos e parâmetros são apropriados para serem utilizadas na busca por padrões nos dados e quais se enquadram nos objetivos gerais do processo de DCBD desejado;
7. MD: a procura propriamente dita por padrões de interesse em uma ou algumas das formas de representação, como regras ou árvores de classificação, regressão ou clusterização;
8. Interpretação dos padrões minerados: possivelmente a interpretação dos padrões pode resultar em iterações com quaisquer dos passos anteriores. Pode envolver a visualização dos padrões e modelos extraídos ou a visualização dos dados obtidos dos modelos extraídos;
9. Utilização do conhecimento descoberto: envolve a utilização direta do conhecimento, sua incorporação em outro sistema ou simplesmente a apresentação do conhecimento para as partes interessadas. Esse passo envolve

também a verificação em relação ao conhecimento anteriormente existente e a resolução de conflitos potenciais.

A Figura 4.3 ilustra os passos do processo de DCBD de modo que todas as etapas anteriormente descritas estão representadas, sendo que as etapas de 5 a 7 estão englobadas no passo de MD.

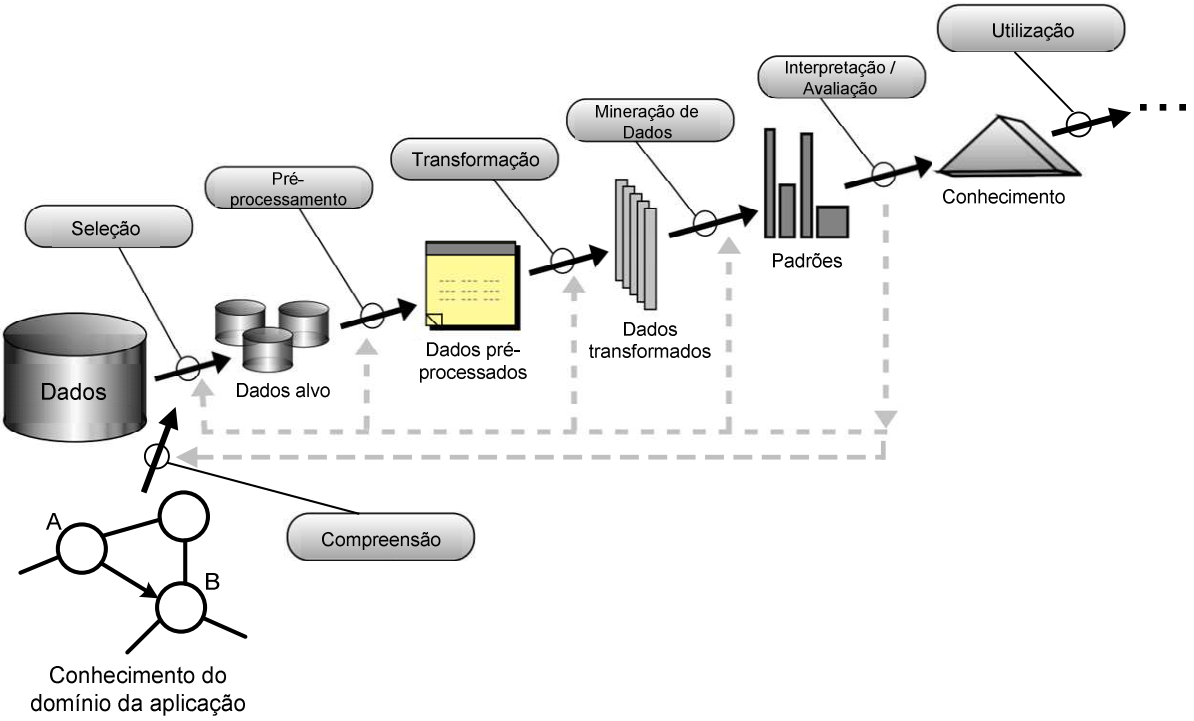

Figura 4.3: Visão geral dos passos de um processo de DCBD (adaptada de Fayyad Piatetsky-Shapiro e Smyth, 1996).

CRISP-DM

Com o crescimento do volume de dados e a necessidade do uso de MD, alguns modelos foram definidos, sendo o *Cross Industry Standard Process for Data Mining* (CRISP-DM) um dos mais utilizados. O CRISP-DM divide o processo de MD (MD é utilizado aqui em seu conceito *lato sensu*, como sinônimo de DCBD) em seis fases e define, para cada fase, quais são as tarefas que devem ser executadas para realizar com sucesso um trabalho de MD e quais artefatos devem ser produzidos (Larose, 2005). A Figura 4.4 apresenta um resumo do modelo CRISP-DM com os relacionamentos entre as fases, assim como as tarefas a serem realizadas e os artefatos a serem produzidos em cada uma delas:

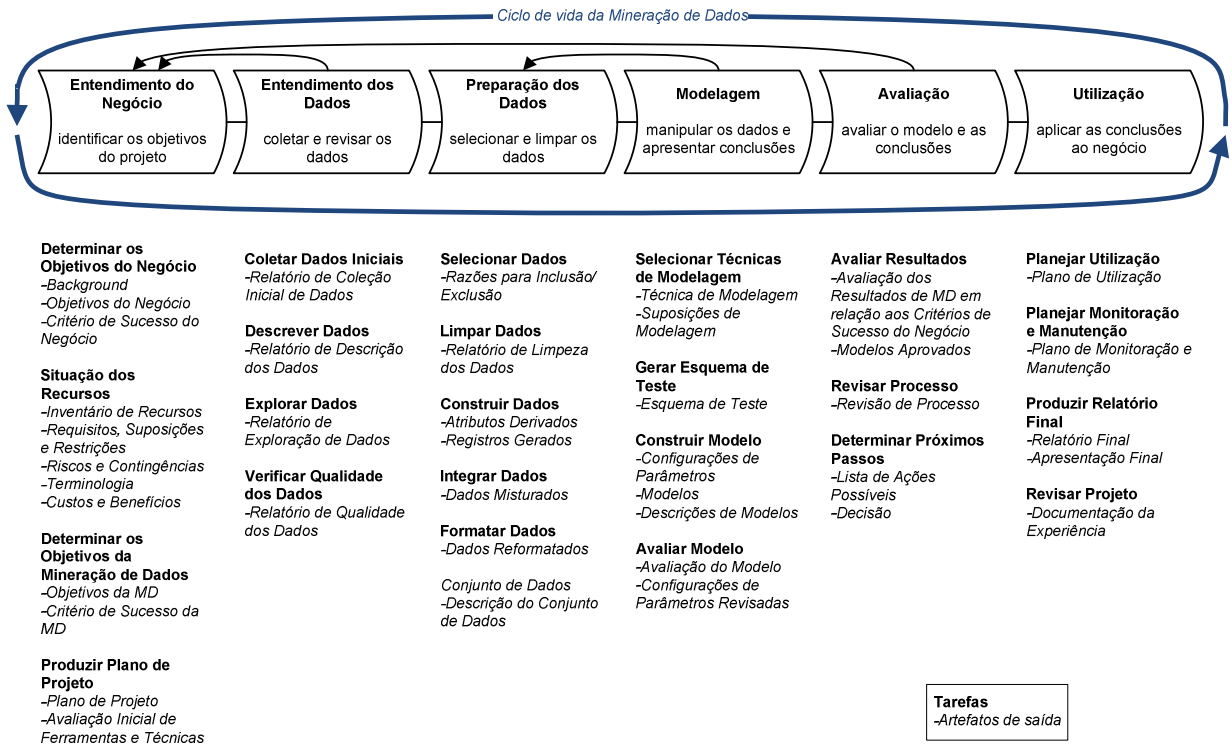

Figura 4.4: Guia visual do CRISP-DM, onde são apresentadas as fases do processo e seus relacionamentos, assim como as tarefas e artefatos de saída de cada fase (adaptada de Leaper, 2009).

4.2.1. Mineração de Dados

Por ser um passo central no processo de DCBD, o termo MD é utilizado por muitos pesquisadores como um sinônimo daquele (Rokach e Maimon, 2008). Neste trabalho, entretanto, vamos tratar a MD como uma das etapas da DCBD em que são adotados algoritmos específicos para extrair padrões dos dados analisados (Fayyad, Piatetsky-Shapiro e Smyth, 1996). Outra definição, de acordo com o Gartner Group, "*Mineração de dados é o processo de descoberta de novas correlações, padrões e tendências significativas presentes em grandes quantidades de dados armazenados em repositórios, usando tecnologias de reconhecimento de padrões assim como técnicas estatísticas e matemáticas.*" (Larose, 2005).

Fayyad, Piatetsky-Shapiro e Smyth (1996) classificam os objetivos de um DCBD em dois tipos principais que, por consequência, requerem dois tipos diferentes de métodos de MD:

- Verificação: métodos de verificação servem para avaliar a hipótese proposta por uma entidade externa, como um especialista. Esses métodos estão mais associados a técnicas de estatística tradicional do que a técnicas de MD (Rokach e Maimon, 2008);

- Descoberta: métodos de descoberta encontram novos padrões de maneira autônoma. A maioria dos métodos de descoberta são baseados em aprendizagem indutiva (Mitchell, 1997), em que um modelo é construído através de generalização a partir de uma quantidade suficiente de exemplos de treinamento, assumindo-se que o modelo treinado será aplicável a exemplos futuros ainda desconhecidos (Rokach e Maimon, 2008).

O objetivo de descoberta em um DCBD pode ainda ser subdividido em:

- Descrição: o sistema busca compreender como os dados operam e encontrar padrões para serem apresentados a um usuário em um formato compreensível;
- Predição: o sistema busca por padrões nas amostras que lhe são apresentadas com o objetivo de construir um modelo para predizer o comportamento futuro de novas entidades. Alguns métodos de predição, como árvores de decisão, também ajudam a fornecer um entendimento dos dados (Rokach e Maimon, 2008).

Métodos de predição são geralmente referenciados na literatura como aprendizagem supervisionada. Os métodos supervisionados buscam descobrir a relação entre atributos de entrada (chamados de variáveis independentes) e um atributo alvo (ou variável dependente) e representar essa relação por meio de um modelo (Rokach e Maimon, 2008). Existem dois tipos principais de modelos supervisionados:

- Modelos de regressão: fazem um mapeamento dos dados de entrada para um conjunto contínuo de valores reais. Como exemplo, podem predizer a demanda esperada para um certo produto (variável dependente) a partir das suas características (variáveis independentes);
- Modelos de classificação (classificadores): fazem um mapeamento do espaço de entrada para um conjunto discreto de classes. Se existirem apenas duas classes possíveis de saída, então a classificação é chamada binária ou booleana. Como exemplo, podem ser usados para classificar possíveis clientes para um determinado novo produto lançado no mercado como potenciais consumidores ou não (variável dependente), com base no padrão de compras anterior desses consumidores (variáveis independentes).

Neste trabalho estamos interessados nos modelos de classificação, pois temos como objetivo classificar arquivos de acordo com sua potencial relevância para análises periciais

(variável dependente) utilizando como variáveis independentes os metadados desses arquivos e seu correlacionamento com outros arquivos da base.

A Figura 4.5 ilustra as classificações anteriormente descritas para os diversos métodos utilizados em MD:

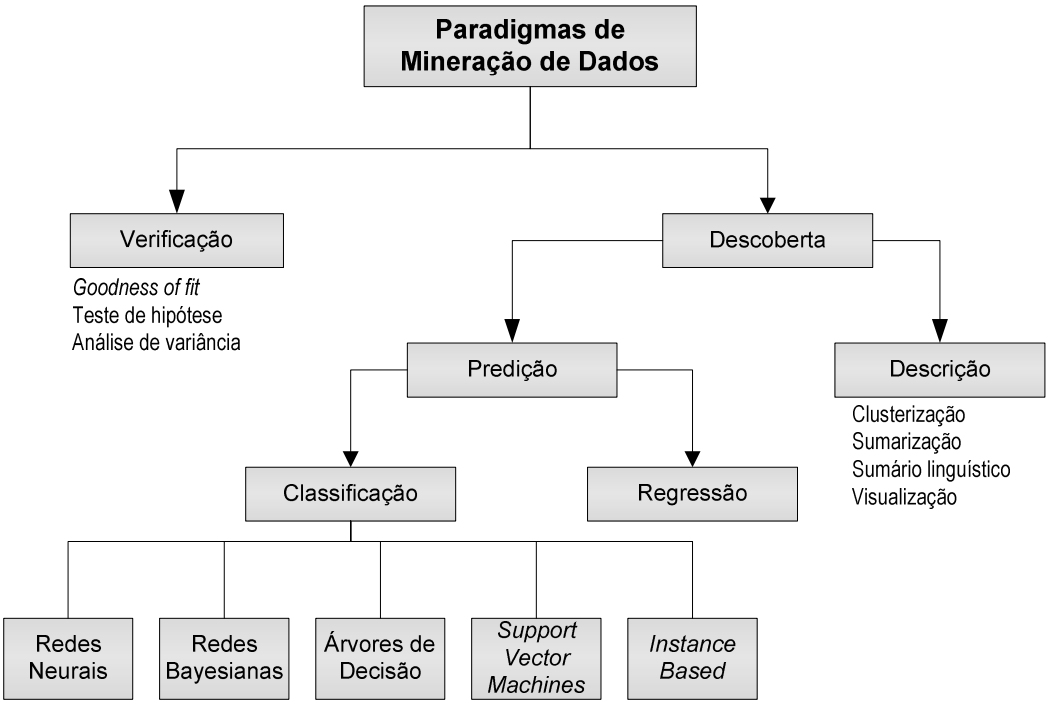

Figura 4.5: Taxonomia de métodos de mineração de dados (adaptada de Rokach e Maimon, 2008).

4.2.2. Classificação de Dados

Na área de MD, a classificação pode ser definida como a tarefa de aprendizagem de uma função alvo *f* que mapeia cada conjunto de atributos **x** para um dos rótulos de classe *y* pré-definidos. A função f é também referenciada como um modelo de classificação (Tan, Steinbach e Kumar, 2005). A Figura 4.6 ilustra esse conceito.

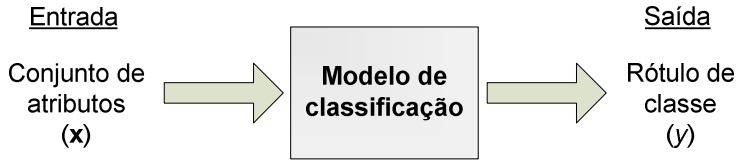

Figura 4.6: Classificação como tarefa de mapeamento de uma entrada de conjunto de atributos x para seu rótulo de classe *y* (adaptada de Tan, Steinbach e Kumar, 2005).

O algoritmo de aprendizagem é o responsável por identificar o modelo de classificação que melhor descreve o relacionamento entre o conjunto de atributos e o rótulo de classe resultante (Tan, Steinbach e Kumar, 2005). A aprendizagem pode ser descrita como a procura, no espaço de hipóteses (modelos) possíveis, por aquela que terá o melhor desempenho, mesmo se aplicada a novos exemplos além daqueles existentes no conjunto de treinamento (Russel e Norvig, 2010).

Normalmente, o conjunto de dados de entrada é dividido em duas partes:

- Conjunto de treinamento: é utilizado como entrada do algoritmo de aprendizagem para construir o modelo de classificação;
- Conjunto de teste: é utilizado para avaliar a acurácia e validar o modelo gerado com o uso do conjunto de treinamento.

Existem diversos métodos para escolha do tamanho dos conjuntos de treinamento e de teste e a forma como esses conjuntos serão selecionados e utilizados. Podemos citar os métodos *holdout*, sub-amostragem aleatória, validação cruzada e *bootstrap* (Tan, Steinbach e Kumar, 2005), que serão descritos posteriormente.

A Figura 4.7 ilustra a tarefa de classificação com a utilização dos dois conjuntos de dados (treinamento e teste). O conjunto de treinamento é utilizado pelo algoritmo de aprendizagem para induzir um modelo de classificação e esse modelo é aplicado posteriormente ao conjunto de teste para validação.

As técnicas de classificação são mais adequadas para trabalhar com conjuntos de dados com categorias binárias ou nominais e são menos efetivas para categorias ordinais, pois não consideram a ordem implícita entre as categorias (Tan, Steinbach e Kumar, 2005). Os atributos contínuos também são melhor processados se for realizada uma etapa prévia de discretização dos dados. A discretização antes da indução pode melhorar significativamente a acurácia do algoritmo de indução (Dougherty, Kohavi, e Sahami, 2005).

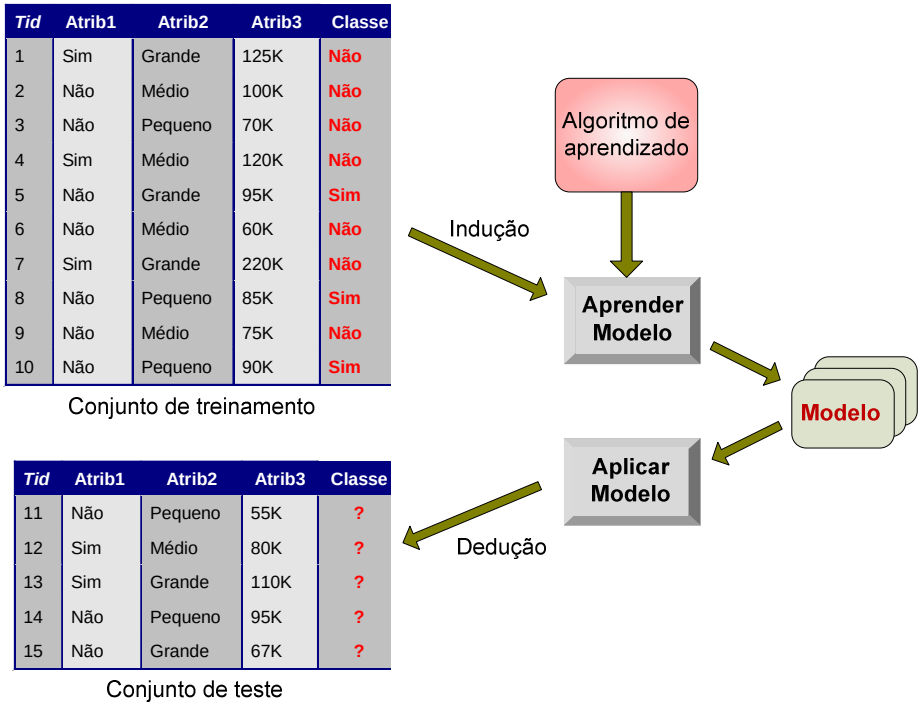

Figura 4.7: Ilustração da tarefa de classificação (adaptada de Tan, Steinbach e Kumar, 2005).

As contagens de erros e acertos do modelo de classificação podem ser tabulados em uma tabela conhecida como matriz de confusão. A Tabela 4.3 mostra como seria uma matriz de confusão para um problema binário (duas classes). Cada entrada f_{ij} representa o número de exemplos da classe *i* previstas pelo modelo como sendo da classe *j*. Podemos perceber que o número de predições corretas são dadas pelas entradas f_{11} e f_{00}, enquanto as predições erradas são dadas por f_{10} e f_{01}.

Tabela 4.3: Matriz de confusão para um problema de duas classes (binário) (adaptada de Tan, Steinbach e Kumar, 2005).

Matriz de Confusão		Classe correta	
		Classe = 1	Classe = 0
Classe prevista	Classe = 1	f_{11}	f_{01}
	Classe = 0	f_{10}	f_{00}

Para avaliar o desempenho de um modelo de classificação, podemos utilizar os dados da matriz de confusão para determinar as métricas de acurácia, taxa de erro, precisão e *recall*. Essas métricas de desempenho são definidas pelas seguintes fórmulas, considerando-se uma classificação binária e a matriz de confusão da Tabela 4.3:

$$Acurácia = \frac{Número\ de\ predições\ corretas}{Número\ total\ de\ predições} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}}$$

$$Taxa\ de\ erro = \frac{Número\ de\ predições\ incorretas}{Número\ total\ de\ predições} = \frac{f_{10} + f_{01}}{f_{11} + f_{10} + f_{01} + f_{00}}$$

$$Precisão\ (Classe\ 1) = \frac{Número\ de\ predições\ corretas\ com\ Classe = 1}{Número\ total\ de\ predições\ com\ Classe = 1} = \frac{f_{11}}{f_{11} + f_{01}}$$

$$Recall\ (Classe\ 1) = \frac{Número\ de\ predições\ corretas\ com\ Classe = 1}{Número\ total\ de\ exemplos\ em\ que\ Classe = 1} = \frac{f_{11}}{f_{11} + f_{10}}$$

Dessa forma, podemos perceber que a *acurácia* e a *taxa de erro* representam medidas relacionadas ao modelo como um todo, enquanto a *precisão* e o *recall* são medidos em relação a cada classe. A *acurácia* calcula o percentual de predições corretas realizadas em relação ao total de predições feitas pelo modelo. A *taxa de erro*, de maneira contrária, representa o percentual de predições erradas em relação ao total de predições realizadas. O cálculo de *precisão* para uma classe *C* representa, dentre o total de predições feitas pelo modelo em que classe é igual a *C*, qual é o percentual de predições realizadas corretamente. O *recall* para uma classe *C*, por sua vez, representa o percentual de predições realizadas corretamente em relação ao total de exemplos em que classe é igual a *C*.

4.2.3. Algoritmos de classificação

Existem diversos algoritmos para realizar a aprendizagem de modelos de classificação, dentre os quais podemos destacar:

Redes neurais

A inspiração para as redes neurais vem do cérebro dos animais e sua complexa rede de neurônios interconectados. Apesar de um único neurônio ter estrutura relativamente simples, as densas redes de neurônicos interconectados podem executar tarefas de aprendizagem bastante complexas, como classificação e reconhecimento de padrões (Larose, 2005). Uma rede neural artificial consiste de um conjunto de unidades de entrada/saída conectadas na qual cada conexão tem um peso associado. Durante a fase de aprendizagem, o algoritmo de aprendizagem ajusta os pesos das conexões de forma a conseguir predizer o rótulo de classe correto para os exemplos apresentados. Redes neurais demandam um longo período de treinamento, possuem alta tolerância a ruídos nos dados e são conhecidas pela habilidade de

classificar padrões para os quais ainda não foram treinadas, mas oferecem pouca capacidade de interpretação do modelo aprendido (Han e Kamber, 2006).

Bayesianos

Classificadores bayesianos são classificadores estatísticos que se baseiam no teorema de Bayes para predizer a probabilidade de um determinado exemplo pertencer a uma classe em particular. Estudos comparativos de algoritmos de classificação encontraram um algoritmo Bayesiano simples – conhecido como *naive Bayes* – com desempenho comparável à de classificadores baseados em árvores de decisão e redes neurais. Para simplificar o custo computacional envolvido, classificadores *naive Bayes* consideram que o efeito do valor de um atributo sobre uma dada classe é independente dos valores dos outros atributos. Essa característica é conhecida como independência condicional de classe. Redes bayesianas, ao contrário de classificadores *naive Bayes*, consideram possível a existência de dependências entre conjuntos de atributos (Han e Kamber, 2006).

Support Vector Machines

Support Vector Machines (SVMs) usam um mapeamento não-linear para transformar os dados do conjunto de treinamento para uma dimensão superior. Depois é feita uma busca pelo hiperplano que otimiza a separação linear dos exemplos de cada classe. SVMs utilizam *support vectors* (exemplos de treinamento "essenciais") e margens (definidas pelos *support vectors*) para encontrar o hiperplano. SVMs são altamente precisas, menos propensas à ocorrência de *overfitting* que outros métodos e oferecem uma descrição compacta do modelo aprendido, mas o tempo de treinamento é lento (Han e Kamber, 2006).

Baseados em exemplos

Ao receber um conjunto de treinamento, um classificador baseado em exemplos (*instance based*) simplesmente armazena essas instâncias recebidas – ou executa apenas um pequeno processamento – e espera até receber um exemplo de teste. Somente quando recebe um exemplo de teste é que o algoritmo realiza uma generalização de modo a classificá-lo baseado na sua similaridade em relação a alguma instância de treinamento armazenada. Por essa razão, são também chamados de *learners* preguiçosos. Eles demandam grande esforço computacional quando precisam fazer uma classificação e oferecem pouca explicação sobre a estrutura dos dados, mas suportam naturalmente aprendizagem incremental (Han e Kamber, 2006).

Árvores de decisão

Escolheu-se neste trabalho utilizar um classificador baseado em árvores de decisão por ser um método que pode lidar com dados com alta dimensionalidade, é apropriado para descoberta exploratória de conhecimento, sua representação de conhecimento em forma de árvore é intuitiva e de fácil assimilação, em geral apresenta boa acurácia e as fases de aprendizagem e classificação são simples e rápidas (Han e Kamber, 2006). Uma descrição detalhada de classificadores baseados em árvores de decisão é apresentada na próxima Seção.

4.3. CLASSIFICADORES BASEADOS EM ÁRVORES DE DECISÃO

Classificadores baseados em árvores de decisão buscam construir o modelo de classificação em forma de árvore direcionada, na qual os nodos não-terminais – incluindo a raiz e outros nodos internos intermediários – contêm as condições de teste das variáveis independentes para separar os exemplos que possuem características diferentes. Os nodos terminais (folhas da árvore) assinalam o rótulo de classificação (variável dependente). A árvore de decisão é, então, uma estrutura hierárquica consistindo de nodos e arestas direcionadas (Tan, Steinbach e Kumar, 2005).

Como exemplo, a Figura 4.8 ilustra uma árvore de decisão que classifica um dia como adequado ou não para se jogar tênis de acordo com as condições climáticas (Mitchell, 1997).

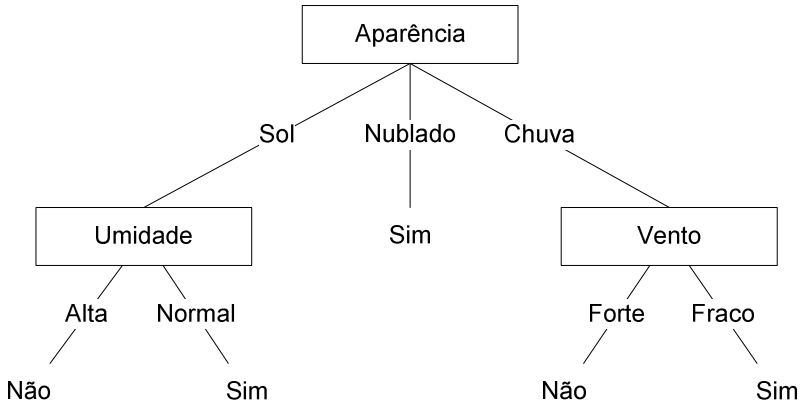

Figura 4.8: Exemplo de árvore de decisão para o conceito *Jogar Tênis* (adaptada de Mitchell, 1997).

Árvores de decisão também podem ser representadas como regras “se-então” (*if-then*) para facilitar a capacidade de leitura, principalmente quando a árvore é muito grande e complexa. Esse método de aprendizagem está entre os algoritmos de inferência indutiva mais populares e foi aplicado com sucesso em áreas que vão desde a aprendizagem de diagnósticos médicos a análise de risco de crédito (Mitchell, 1997).

Uma grande vantagem de utilizar um algoritmo de classificação baseado em árvore de decisão é a capacidade de o modelo gerado apresentar regras ainda desconhecidas para a classificação dos dados. Essa característica é especialmente útil para o presente trabalho por ter o potencial de apresentar novas possibilidades de regras para classificação de arquivos quanto à sua relevância em exames periciais a partir de seus metadados.

4.3.1. Algoritmos de indução de árvores de decisão

Para construir um modelo de árvore de decisão a partir de um conjunto de dados de treinamento, é preciso utilizar um algoritmo que realize a indução com boa acurácia e em um tempo razoável, levando em consideração que a escolha da árvore ótima é computacionalmente inviável devido ao tamanho exponencial do espaço de busca. Em geral, os algoritmos utilizam uma estratégia gananciosa que constrói o modelo através de uma série de decisões otimizadas a respeito de qual atributo usar para particionar os dados. O algoritmo de Hunt utiliza essa abordagem e é a base para diversos outros algoritmos conhecidos para indução de árvores de decisão, como ID3, C4.5 e CART (Tan, Steinbach e Kumar, 2005).

Um exemplo de construção de uma árvore de classificação é ilustrado na Figura 4.9 a partir dos dados de treinamento expostos na Tabela 4.4. Esse exemplo busca predizer se tomadores de empréstimo se tornarão inadimplentes ou não. A árvore começa com um único nodo com rótulo de classe “Inadimplente=Não”, pois a maioria dos tomadores de empréstimo do conjunto de treinamento é adimplente. Novos nodos vão sendo criados recursivamente utilizando atributos selecionados para realizar os testes de condição e, assim, os exemplos vão sendo subdivididos em conjuntos menores em cada ramificação da árvore. Se todos os exemplos de um nodo tiverem o mesmo rótulo de classe ou tiverem os mesmos valores para todos os outros atributos, então teremos um nodo folha (Tan, Steinbach e Kumar, 2005).

Duas decisões importantes devem ser tomadas ao se projetar um algoritmo de indução: como selecionar o atributo a ser utilizado como condição de teste para dividir os registros (qual será o atributo de teste nos novos nodos criados) e quais devem ser as condições de parada do processo de separação (definir se serão criados novos nodos na árvore ou se o nodo será um nodo folha).

Tabela 4.4: Conjunto de treinamento para predizer quais tomadores de empréstimos ficarão inadimplentes no pagamento de empréstimos (adaptada de Tan, Steinbach e Kumar, 2005).

	Binário	*Nominal*	*Contínuo*	*Classe*
ID	Proprietário da casa	Estado civil	Rendimento anual	Inadimplente
1	Sim	Solteiro	125K	Não
2	Não	Casado	100K	Não
3	Não	Solteiro	70K	Não
4	Sim	Casado	120K	Não
5	Não	Divorciado	95K	Sim
6	Não	Casado	60K	Não
7	Sim	Divorciado	220K	Não
8	Não	Solteiro	85K	Sim
9	Não	Casado	75K	Não
10	Não	Solteiro	90K	Sim

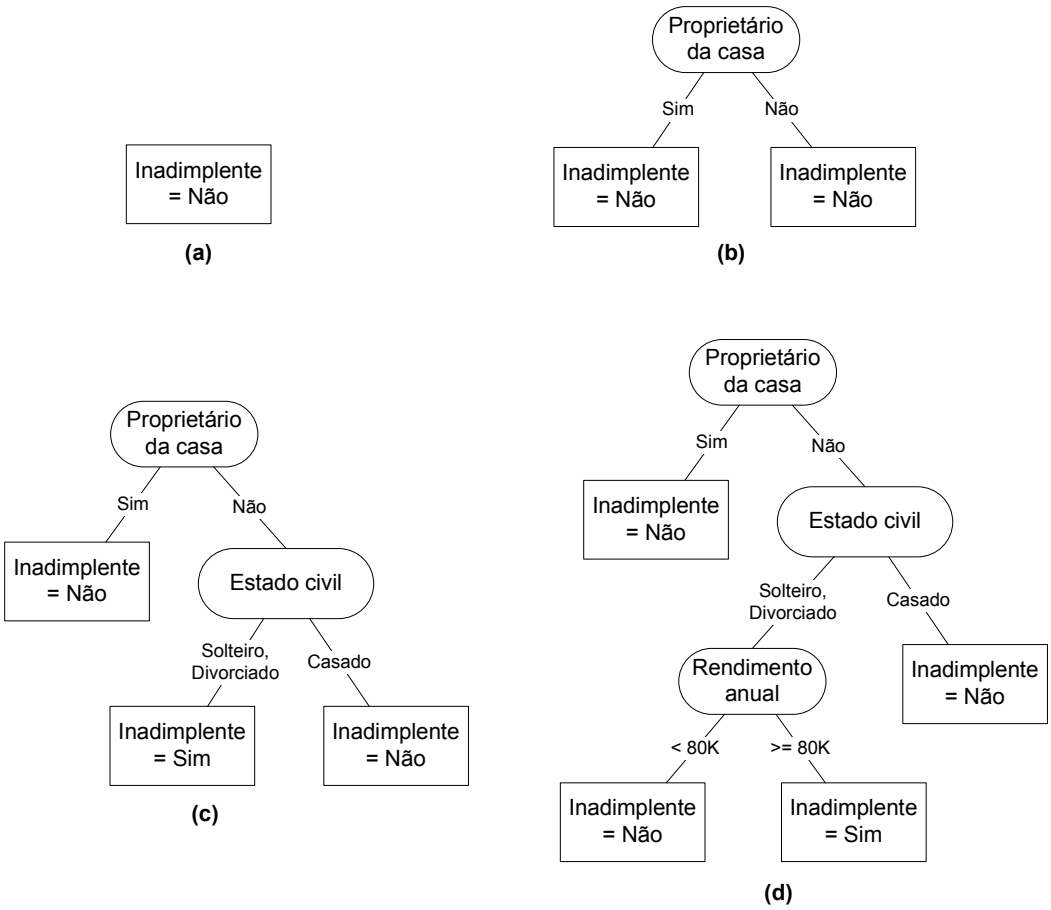

Figura 4.9: Exemplo do algoritmo de Hunt para indução de árvores de decisão, utilizando como conjunto de treinamento os dados presentes na Tabela 4.4 (adaptada de Tan, Steinbach e Kumar, 2005).

Algumas medidas para decidir qual nodo produz a melhor separação dos registros são baseadas no grau de impureza dos nodos filhos, ou seja, no grau de separação dos registros em classes diferentes. Assim, em uma classificação binária, um nodo que produza distribuição de 50% dos registros tem o mais alto nível de impureza, enquanto um que resulte em 100% dos registros em uma mesma classe tem impureza zero (Tan, Steinbach e Kumar, 2005).

Considerando *p(i|t)* como sendo a proporção de exemplos que pertencem à classe *i* em um dado nodo *t* e considerando *c* igual ao número de classes possíveis, podemos calcular as medidas de impureza conforme Equações 4 e 5.

$$Entropia(t) = -\sum_{i=0}^{c-1} p(i|t)\ \log_2 p(i|t) \tag{4}$$

$$Gini(t) = 1 - \sum_{i=0}^{c-1} [p(i|t)]^2 \tag{5}$$

Os cálculos de entropia e índice de Gini atingem seu valor máximo (respectivamente, 1 e 0,5 em uma classificação binária) quando há uma distribuição uniforme entre as classes, conforme pode ser visto na Figura 4.10.

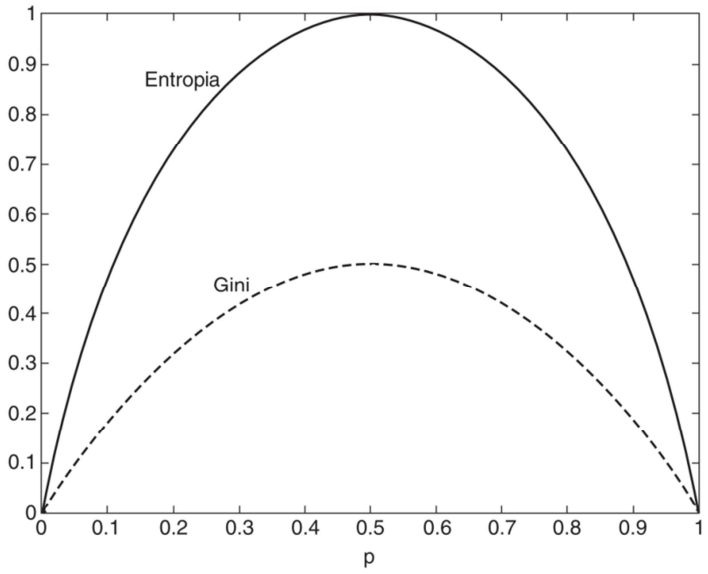

Figura 4.10 - Gráfico comparativo de medidas de entropia e índice de gini para uma classificação binária (adaptada de Tan, Steinbach e Kumar, 2005).

Para medir o desempenho de uma condição de teste é preciso comparar o grau de impureza do nodo pai antes da separação com o grau de impureza dos nodos filhos depois da separação. Quanto maior a diferença, melhor é a condição de teste. Essa diferença é conhecida

como *ganho* (Δ) e pode ser calculada pela Equação 6, onde *I()* é a medida de impureza de um nodo, *N* é o número de exemplos no nodo pai, *k* é o número de nodos filhos e $N(v_j)$ é o número de registros associados ao nodo filho v_j.

$$\Delta = I(pai) - \sum_{j=1}^{k} \frac{N(v_j)}{N} I(v_j) \tag{6}$$

Os algoritmos de indução de árvore de decisão geralmente escolhem condições de teste que maximizam o *ganho* Δ. Quando a medida de impureza utilizada é a entropia, então o *ganho* é conhecido como *ganho de informação*, Δ_{info} (Tan, Steinbach e Kumar, 2005).

A medida de *ganho de informação* possui a característica de preferir atributos com um grande número de valores distintos, o que pode não ser desejável porque o número de exemplos em cada nodo será muito pequeno para que seja possível fazer qualquer predição confiável (Tan, Steinbach e Kumar, 2005). Uma solução para contornar esse problema é utilizar uma adaptação da medida de *ganho de informação* aplicando um tipo de normalização com o uso de um valor de *informação de separação* que leva em consideração o número de saídas produzidas por uma condição de teste, conforme Equação 7 (Tan, Steinbach e Kumar, 2005), onde $P(v_i)$ representa o percentual de registros associados ao nodo filho v_i em relação ao total de registros associados ao nodo pai.

$$Info\ de\ Separação = -\sum_{i=1}^{k} P(v_i)\ \log_2 P(v_i) \tag{7}$$

A medida de *taxa de ganho* é calculada a partir dos valores de *ganho de informação* e *informação de separação*, conforme Equação 8, de modo que se um atributo produzir um grande número de separações (*ganho de informação* alto), sua *informação de separação* também será grande, o que irá reduzir a *taxa de ganho* (Tan, Steinbach e Kumar, 2005).

$$Taxa\ de\ ganho\ = \frac{\Delta_{info}}{Info\ de\ Separação} \tag{8}$$

Após a construção da árvore de decisão, o algoritmo de classificação pode ainda realizar um processamento de poda da árvore para reduzir seu tamanho, de forma a melhorar sua capacidade de generalização e evitar a ocorrência de *overfitting*.

Implementações conhecidas de algoritmos de indução

A maioria dos trabalhos sobre árvores de decisão em aprendizagem de máquina são derivações dos algoritmos ID3 e CART (Murthy, 1998).

O algoritmo ID3 foi proposto por Quinlan (1986) e é considerado um algoritmo simples de indução de árvores de decisão. Ele utiliza a medida de *ganho de informação* (Δ_{info}) como critério de separação e a árvore para de crescer quando todas as instâncias relacionadas a um nodo pertencerem à mesma classe ou quando a melhor medida de ganho de informação não for maior que zero. O ID3 não lida com atributos numéricos ou falta de valores e não realiza nenhum procedimento de poda (Rokach e Maimon, 2008).

Como uma evolução do ID3, Quinlan (1993) propôs o algoritmo C4.5, que utiliza a medida de *taxa de ganho* como critério de separação ao invés da medida de *ganho de informação*. A separação para quando o número de exemplos a serem repartidos é menor que um limite especificado e uma poda baseada em erro é realizada após a fase de expansão da árvore. O C4.5 pode lidar com atributos numéricos e falta de valores (Rokach e Maimon, 2008). Quinlan realizou ainda uma série de melhorias no algoritmo C4.5 e as incorporou no algoritmo comercial C5.0.

O algoritmo *Classification and Regression Trees* (CART) foi proposto por Breiman et al. (1984) e, conforme o próprio nome diz, é utilizado para gerar árvores de classificação ou de regressão, dependendo do tipo da variável dependente. O algoritmo CART gera árvores binárias e, quando utilizado para classificação, pode utilizar como critério de separação: índice de Gini, Twoing ou Twoing ordenado. Na árvore obtida é realizada Poda por Custo-Complexidade. O CART também permite que o usuário forneça uma distribuição de probabilidade prévia (Rokach e Maimon, 2008).

O algoritmo *Chi-squared-Automatic-Interaction-Detector* (CHAID) foi originalmente proposto por Kass (1980) e consiste basicamente de três etapas: fusão (*merging*), separação (*splitting*) e parada (*stopping*). A árvore vai crescendo através do uso repetido dessas etapas em todos os nodos a partir do nodo raiz. O algoritmo aceita atributos qualitativos nominais ou ordinais. Se houver um atributo contínuo, é necessário transformá-lo em ordinal. O CHAID busca identificar interações entre atributos em um conjunto de dados e avaliar qual é o relacionamento desses atributos com o atributo alvo. Os atributos são avaliados e, para cada um deles, é realizada uma busca por pares de categorias que são mais similares com respeito à variável dependente (para problemas de classificação a similaridade é realizada utilizando o teste Chi-quadrado). Se um par de categorias de atributos for considerado similar, é realizada

a fusão dessas categorias. Após a fusão das categorias de atributos similares, é realizada a etapa de separação dos nodos da árvore com a escolha da variável independente que resultará na melhor separação. O algoritmo não realiza poda da árvore (Rokach e Maimon, 2008).

4.3.2. Avaliação de Modelos de Classificação

O comportamento esperado de um modelo de classificação não está relacionado apenas à capacidade de classificar corretamente o conjunto de dados de treinamento, mas espera-se que ele também seja capaz de classificar corretamente exemplos nunca antes vistos por ele.

Os erros cometidos por um modelo de classificação podem ser geralmente divididos em dois tipos: erros de treinamento (também conhecidos como de resubstituição ou aparentes) e erros de generalização (Tan, Steinbach e Kumar, 2005). Os erros de treinamento são classificações incorretas realizadas pelo modelo no conjunto de dados de treinamento, enquanto os erros de generalização são as falhas de classificação esperadas quando o modelo for aplicado a exemplos desconhecidos. Em geral, modelos que se encaixam muito bem ao conjunto de dados de treinamento tendem a apresentar uma capacidade de generalização pior do que modelos com uma taxa de erro de treinamento maior (Tan, Steinbach e Kumar, 2005). Por essa razão, é necessário haver um equilíbrio na criação do modelo em relação aos dois erros citados.

Quando o modelo é exageradamente especializado e ajustado para se encaixar aos dados de treinamento, ocorre o problema conhecido como *model overfitting*. No caso de árvores de decisão, um modelo pode estender ramificações e gerar uma árvore complexa que se encaixa perfeitamente aos dados de treinamento, mas essa árvore será exageradamente especializada e será pouco efetiva para realizar generalizações em instâncias de teste. Por outro lado, quando o modelo ainda não consegue representar minimamente as correlações do conjunto de dados, principalmente devido a uma amostra pouco representativa, ocorre o problema de *model underfitting*. Neste caso, a árvore de decisão tende a ser muito pequena e não consegue efetuar as classificações corretamente nem nos exemplos de treinamento, nem nos de teste.

As causas mais comuns para a ocorrência de *overfitting* são (Tan, Steinbach e Kumar, 2005):

- presença de ruídos: a existência de classificações erradas no conjunto de dados de treinamento dificultam que o algoritmo de indução encontre padrões nos relacionamentos entre as variáveis independentes e o atributo alvo, o que pode levar à criação de um excesso de ramificações para descrever esses exemplos

fora do padrão. Casos de exceção também levam à ocorrência de *model overfitting* e a erros em que exemplos de teste contradizem o padrão dos exemplos padrão;

- falta de amostras representativas: a existência de poucas amostras de treinamento também dificultam que o algoritmo de indução encontre padrões que se repetem nos exemplos. Assim, para explicar o relacionamento dos dados, o algoritmo acaba criando um modelo mais complexo do que seria necessário.

É possível identificar a ocorrência de *overfitting* se houver um modelo mais simples com menor taxa de erro de classificação para o conjunto de dados de treinamento.

Duas estratégias para evitar a ocorrência de *overfitting* em árvores de decisão são conhecidas como *prepruning* (pré-poda) e *postpruning* (pós-poda). O *prepruning* ocorre com o uso de estratégias de parada mais restritivas, que previnem o crescimento da árvore de decisão excessivamente ainda na fase de expansão. Já o procedimento de *postpruning* é realizado após o algoritmo ter feito a expansão completa da árvore de decisão e, através de uma estratégia *bottom-up*, os nodos são avaliados e sub-árvores podem ser substituídas por nodos folha contendo a classificação majoritária ou pela ramificação mais frequentemente utilizada da sub-árvore (Tan, Steinbach e Kumar, 2005). De acordo com Witten, Frank e Hall (2011), o procedimento de *postpruning* apresenta vantagens em relação ao *prepruning* e a maioria dos algoritmos de AD realizam *postpruning*. *Prepruning* pode ser uma alternativa interessante se o tempo de execução do algoritmo for uma preocupação.

Depois que o modelo de classificação é construído ele pode ser aplicado ao conjunto de teste para avaliar sua capacidade de generalização. Existem alguns métodos conhecidos para fazer a escolha dos conjuntos de dados de treinamento e de teste (Tan, Steinbach e Kumar, 2005):

- *holdout* – é o método mais comum, em que os dados originais são separados em dois conjuntos (treinamento e teste). A definição da proporção de cada grupo fica a cargo do analista, sendo comum proporções 50% - 50% ou dois terços para o treinamento e um terço para teste. Como menos exemplos estão disponíveis para fazer a indução do modelo, este pode vir a não ser tão bom como se todos os registros fossem usados para treinamento. Assim, a definição do percentual dos conjuntos de treinamento e teste pode impactar significativamente o resultado do algoritmo;

- sub-amostragem aleatória – este método consiste em aplicar o método *holdout* diversas vezes. Calcula-se então o desempenho do modelo pela média da acurácia obtida em cada iteração. Encontra os mesmos problemas do método *holdout* porque não são utilizados todos os dados possíveis para treinamento.
- validação cruzada – neste método também são realizadas várias iterações nas quais cada exemplo é escolhido o mesmo número de vezes para treinamento e apenas uma vez para teste. O conjunto de dados é dividido em *t* segmentos de igual tamanho e, a cada rodada, um desses segmentos é utilizado para teste enquanto os outros são usados para treinamento. São realizadas *t* rodadas, de modo que cada segmento seja utilizado apenas uma vez como conjunto de teste. O erro total é igual à soma dos erros em todas as *t* iterações.
- *bootstrap* – enquanto nos métodos anteriores não há registros duplicados nos conjuntos de treinamento e de teste, no método *bootstrap* os exemplos de treinamento são escolhidos com reposição (um registro já escolhido para treinamento é colocado de volta no conjunto de registros original e pode ser novamente escolhido). Os exemplos que restarem após a escolha do conjunto de treinamento farão parte do conjunto de teste. Esse procedimento de amostragem é então repetido várias vezes.

Foram apresentados, neste capítulo, os conceitos relacionados à criação de um BD que possa servir de base para a criação de um processo de MD para predição de classificação com o uso de algoritmos de indução de AD. Esses conceitos são utilizados na implementação da solução proposta, apresentada no Capítulo 5.

5. PROPOSTA DE SOLUÇÃO

A proposta deste trabalho consiste em aperfeiçoar a criação e manutenção de BHACs através da implementação de duas soluções principais:

1. aplicação de técnicas de MD com o objetivo de identificar novos arquivos potencialmente irrelevantes para exames periciais a partir de uma amostra de computadores (AC) de uma determinada região ou país;
2. seleção de conjuntos de *hashes*, a partir das BHACs convencionais, que efetivamente representem o parque de softwares utilizados em um dado mercado.

A primeira solução tem por objetivo aumentar a quantidade de arquivos filtrados por BHACs, enquanto a segunda busca diminuir o custo computacional do processo pericial de filtragem de arquivos ao excluir da BHAC os *hashes* pouco efetivos.

Para identificar arquivos considerados ignoráveis, o projeto baseia-se na premissa de que se um arquivo é encontrado de maneira reiterada em diversos computadores não relacionados, então é grande a possibilidade de tratar-se de um arquivo comum e que não acrescenta informação útil para uma análise criminal. O termo “arquivo comum” refere-se, nesse contexto, a arquivos que fazem parte da instalação padrão de aplicativos comerciais ou arquivos que, de alguma forma, sejam normalmente encontrados em computadores de usuários não diretamente relacionados. Se, além de ser encontrado em computadores distintos, os metadados de um arquivo corroborarem a hipótese de sua irrelevância, então ele se torna um forte candidato a fazer parte da base de arquivos ignoráveis.

Uma vez que as diversas mídias de armazenamento computacional que são objeto de exames periciais contêm uma amostra significativa do universo de softwares mais utilizados em um país ou região, elas podem ser utilizadas para montar a AC necessária.

A AC será utilizada, então, tanto para identificar novos arquivos irrelevantes como para auxiliar a filtrar os conjuntos de *hashes* convencionais referentes a softwares obsoletos ou raramente utilizados. Contudo, deve ser tomado cuidado para não excluir conjuntos de *hashes* relacionados a softwares recentemente lançados, pois eles podem ainda não estar suficientemente representados no mercado e, assim, podem não estar presentes de maneira significativa na AC.

O que se propõe neste trabalho, portanto, é adotar uma solução que possa ser adotada de maneira uniforme em todos os países e que não necessite de acompanhamento constante sobre todos os softwares que estejam sendo disponibilizados dia-a-dia em cada região ou país.

Por ter como objetivo auxiliar o trabalho de identificação e separação de arquivos sem interesse para exames periciais, este projeto foi denominado de Joio[9].

Este trabalho não tem por objetivo substituir as BHACs atualmente disponíveis, mas sim complementá-las e utilizá-las como fonte adicional de informação. A consolidação das fontes de *hashes* em uma base centralizada também permitirá a adoção de procedimentos padronizados para sua utilização e atualização pelos órgãos periciais, servindo de referência para todos os peritos desses órgãos.

O armazenamento dos metadados de arquivos analisados em exames periciais também permitirá a formação de uma base de memória dos casos analisados que pode ser utilizada posteriormente em outros estudos de descoberta de conhecimento.

As seções seguintes deste capítulo apresentam o estudo preliminar realizado para avaliar a viabilidade da proposta, o processo de DCBD utilizando AD, os detalhes da arquitetura do sistema e o protótipo implementado.

5.1. ESTUDO PRELIMINAR

Para avaliar a premissa exposta no início deste capítulo a respeito da repetição de arquivos em computadores distintos, foi realizado um estudo preliminar para verificar a quantidade de arquivos presentes em casos reais de exames periciais que apresentavam potencial para integrar uma base de arquivos ignoráveis. Nesse estudo foram avaliadas as informações de arquivos presentes em discos rígidos analisados em 19 exames periciais de casos reais da PF do ano de 2010 escolhidos aleatoriamente. Foi utilizada uma BHAC composta pelo RDS 2.31 da NSRL (de dezembro de 2010) e KFF da empresa AccessData (de junho de 2007) para classificar os arquivos como *ignorável*, *alerta* ou *desconhecido*.

Os dados deste estudo estão expostos na Tabela 5.1, na qual é possível verificar que, de um total de 58.541 arquivos distintos que estavam presentes em mais de uma evidência (alínea b), apenas 18.735 (alíneas b.1 e b.2) foram reconhecidos pelas bases da NSRL e KFF como *ignorável* ou *alerta*. Um total de 39.806 arquivos válidos (alíneas b.3.i e b.3.ii) estavam presentes em mais de uma evidência e não foram filtrados. Esses arquivos exemplificam o objeto de estudo do presente trabalho e representam, neste estudo preliminar, o potencial máximo de arquivos que podem vir a ser acrescidos à base de arquivos classificados como *ignorável*. Considerando que esse potencial de acréscimo de 39.806 arquivos representam um

[9] O nome é uma referência à passagem bíblica do Evangelho de São Mateus, capítulo 13:24-30, onde Jesus propõe a parábola sobre como separar o joio do trigo.

aumento de cerca de 50% em relação ao total de 79.473 arquivos distintos (alínea a.1) identificados pelas BHAC tradicionais, o estudo preliminar indicou boas perspectivas de melhoria potencial a ser obtida com o uso da solução proposta.

Tabela 5.1: Estudo preliminar realizado com informações de arquivos extraídos de discos rígidos analisados em 19 exames periciais de casos reais da Polícia Federal do ano de 2010.

Análise preliminar de arquivos potencialmente ignoráveis

Descrição	**Total**
a) Total de arquivos únicos	871.590
a.1) Filtrados como *ignorável*	**79.473**
b) Total de arquivos únicos presentes em mais de uma evidência	**58.541**
b.1) Filtrados como *ignorável*	18.692
b.2) Filtrados como *alerta*	43
b.3) Desconhecidos	39.806
b.3.i) Temporários de Internet	**4.078**
b.3.ii) Outros arquivos	**35.728**

5.2. PROCESSO DE DCBD

Para encontrar padrões nos dados que levem à criação de um modelo para classificar os arquivos da AC de acordo com o potencial de relevância para exames periciais, é necessário implementar um processo de DCBD. Para gerar o modelo de classificação, a solução proposta utiliza algoritmo de aprendizagem baseado em AD. O resultado final esperado da aplicação do modelo de AD é a identificação de novos arquivos ignoráveis a serem adicionados à BHP.

A utilização de conhecimento de especialistas em perícias em computadores é muito importante na implementação do processo de DCBD, especialmente para auxiliar a identificação de atributos relevantes que possam ser usados pelo modelo de classificação – notadamente informações a respeito do funcionamento de sistemas operacionais e aplicativos mais comuns. O conhecimento de especialistas será utilizado, também, para classificar a amostra de arquivos que servirá como conjunto de treinamento e de teste para o algoritmo de indução de AD.

A Figura 5.1 ilustra o processo de DCBD utilizado para gerar o modelo de AD que irá classificar a relevância dos arquivos da AC. Ele consiste de seis passos baseados nas fases de DCBD descritas na Seção 4.2. Os passos numerados 1 e 2 correspondem à seleção, pré-

processamento, transformação e consolidação dos dados, respectivamente, enquanto os passos 3, 4, 5 e 6 são relacionados ao processo de MD propriamente dito, que cobre a amostragem, classificação, indução e aplicação do modelo de AD.

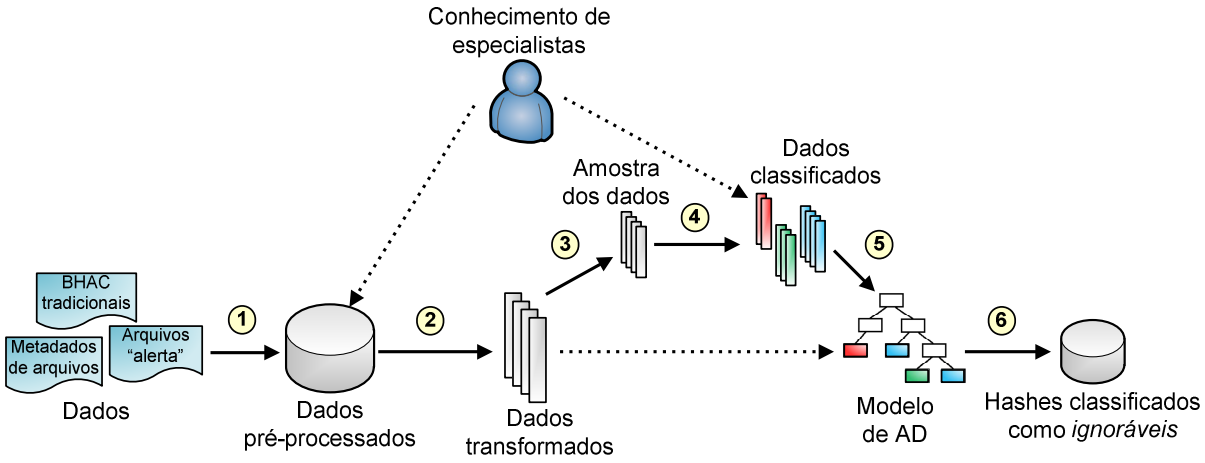

Figura 5.1 - Passos do processo de DCBD com o uso de AD.

Os seis passos numerados do processo de DCBD mostrados na Figura 5.1 estão detalhados a seguir:

1 – Seleção e Pré-processamento

O primeiro passo do processo de DCBD utilizado é a coleta, extração, transformação e carga de todos os dados para o banco de dados previamente modelado. Os dados dos arquivos presentes na AC podem ser coletados com a ajuda de ferramentas forenses, que extraem os metadados do sistema de arquivos e incluem outras informações relevantes sobre o conteúdo de um arquivo, como seu valor de *hash*, tipo baseado na assinatura, correspondência entre o tipo de assinatura e extensão esperada e propriedades como compressão e criptografia. Todas essas informações são importadas para a BDAE. Os conjuntos de *hashes* de BHAC tradicionais também fazem parte dos dados a serem coletados e pré-processados.

2 – Transformação

O segundo passo consiste em transformar e consolidar os dados pré-processados. O processo de DCBD será usado para classificar a relevância de cada valor de *hash* correspondente a um ou mais arquivos da BDAE. Portanto, os dados devem ser consolidados e agrupados para cada valor de *hash* e todos os atributos devem ser expressos com um percentual do total de arquivos que compartilham o mesmo valor de *hash* na base de dados.

3 – Amostragem

A BDAE pode conter centenas de milhares ou mesmo milhões de valores de *hash* distintos. Esses valores devem ser classificados por um especialista para que seja possível utilizar um algoritmo de indução de AD. Para que esse processo de classificação seja viável, deve ser utilizada uma amostra do BDAE.

4 – Classificação dos dados amostrados

As entradas dos dados amostrados devem, então, ser manualmente classificados por especialistas quanto à sua relevância para exames periciais.

5 – Indução do modelo de AD

Os dados amostrados e classificados serão divididos em conjuntos de treinamento e de teste, de modo que o algoritmo de indução de AD possa derivar e avaliar o modelo de classificação apropriado.

6 – Aplicação do modelo de AD

Uma vez que o modelo de AD para classificar a relevância de arquivos – identificados pelos seus valores de *hash* – esteja criado, ele poderá ser aplicado à BDAE completa para identificar os arquivos ignoráveis a serem incluídos na BHP.

5.3. ARQUITETURA PROPOSTA

A arquitetura da solução proposta é composta por:

- um *framework* para criar e manter uma base de dados consolidada (Base Joio) formada por dados extraídos da AC, do Sistema Criminalística[10] da PF (SISCRIM), das BHAC tradicionais e por informações obtidas a partir de conhecimento de especialistas;
- processos de MD que, a partir dos dados da Base Joio, utilizam algoritmos de aprendizagem baseados em AD para classificar os arquivos da AC quanto à sua relevância para exames periciais.

O resultado da aplicação do modelo de AD sobre os dados da AC é a identificação de arquivos considerados irrelevantes e que, portanto, podem ser utilizados em conjunto com BHAC tradicionais no procedimento pericial de filtragem de arquivos conhecidos.

[10] Sistema de gestão de fluxo de documentos, materiais e atividades dentro do âmbito da Criminalística no Departamento de Polícia Federal.

A Figura 5.2 apresenta uma visão geral da solução proposta, na qual é possível verificar que a AC utilizada é composta por computadores a serem periciados em casos reais da PF (A) onde, após a análise pericial, os metadados dos arquivos são extraídos e armazenados em duas bases de dados: (1) dados de todos os arquivos examinados (BDAE) e (2) dados dos arquivos classificados pelo perito como *alerta*. À BDAE são acrescidas informações do SISCRIM (B) sobre a evidência e o caso de análise pericial referentes a cada computador da AC. Uma terceira base de dados (3) é formada pelas BHACs obtidas de fontes externas (C). O processo de MD é baseado, então, nos dados dessas três bases (1, 2 e 3) e nas informações obtidas do conhecimento de especialistas (D) e fornece como resultado um modelo de AD (E) para classificar os arquivos da BDAE quanto a sua relevância para exames periciais. A aplicação do modelo de AD sobre os dados dos arquivos extraídos da AC permite a identificação de novos *hashes* de arquivos classificados como *ignoráveis* (4).

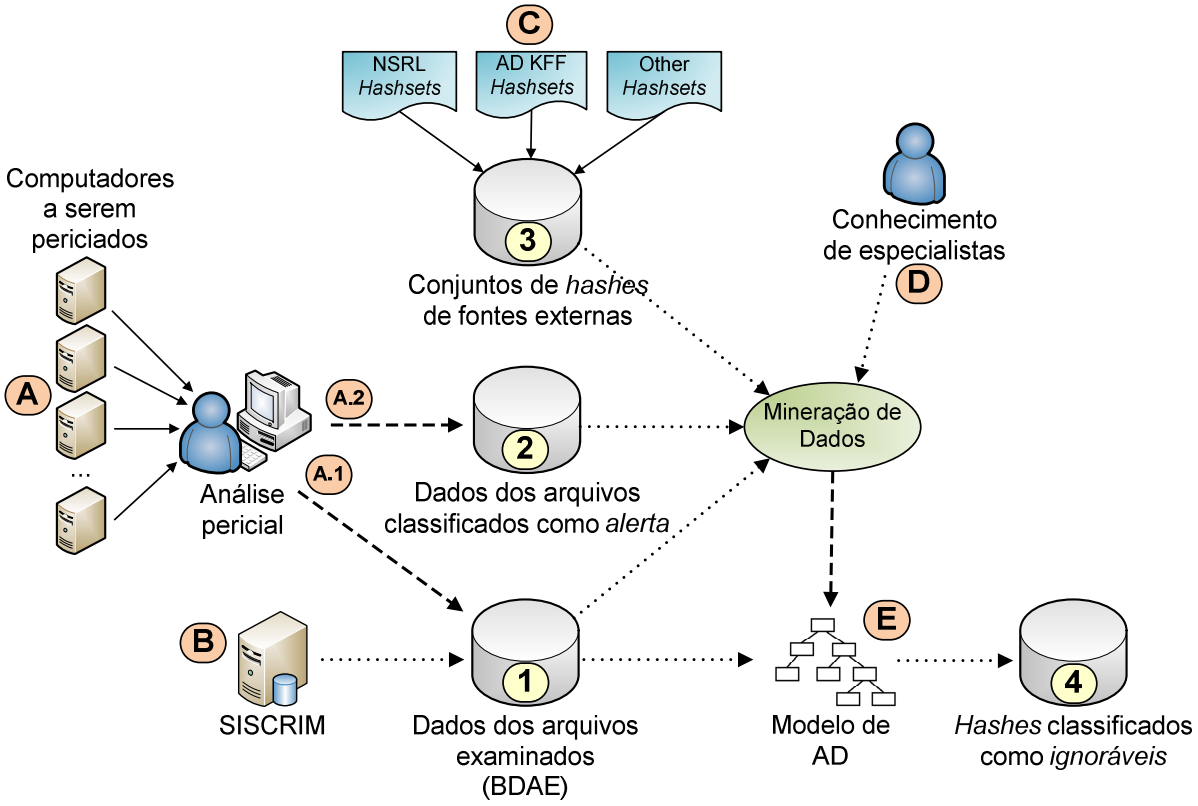

Figura 5.2: Visão geral da solução proposta para identificar arquivos ignoráveis em uma amostra de computadores.

As bases de dados numeradas como 1, 2 e 3 na Figura 5.2 e as informações introduzidas a partir do conhecimento de especialistas (D) formam a Base Joio e contêm todos os dados utilizados para implementar a solução proposta.

Os arquivos classificados como *ignoráveis* (4) terão seus valores de *hash* acrescidos à BHP, que será composta ainda de uma seleção de conjuntos de *hashes* efetivamente utilizados – identificados com o auxílio da AC – extraída de BHACs tradicionais.

A BHP pode, então, ser utilizada na fase de análise pericial (em laboratório forense) ou em análises de triagem de materiais a serem apreendidos (trabalho de campo) para identificar de forma automática arquivos já conhecidos.

A arquitetura da solução proposta foi implementada em três módulos: (i) *framework* para montagem da Base Joio; (ii) processo de MD para classificação de arquivos com o uso de AD e (iii) método para seleção de conjuntos de *hashes* efetivos das BHACs tradicionais.

5.3.1. *Framework* para montagem da Base Joio

O *framework* para montagem da Base Joio deve implementar tarefas de coleta de dados, modelagem dos processos de ETL, modelagem da base de dados e inclusão de conhecimento de especialistas, consolidação e correlacionamento dos dados.

Coleta de dados

São quatro as fontes de dados a serem coletados que servirão de base para este trabalho, os quais estão identificados conforme a Figura 5.2:

- informações de arquivos extraídas de exames periciais (A.1) – o formato desses dados é definido pela ferramenta pericial utilizada para sua extração, sendo mais comuns os formatos CSV (*comma separated values*) e MDB (BD da ferramenta *Microsoft Access*). Os atributos extraídos também variam de acordo com a ferramenta pericial utilizada, mas geralmente incluem os metadados do sistema de arquivos, a assinatura que identifica o tipo do arquivo, categorização com base no tipo do arquivo, informações sobre compressão, criptografia e, claro, seu valor de *hash*;
- *hashes* de arquivos classificados como *alerta* em exames periciais (A.2) – os arquivos de interesse para inclusão na base são aqueles que potencialmente podem ser encontrados com o conteúdo inalterado em computadores distintos e que servem para evidenciar a prática de crimes. Como exemplo temos imagens e vídeos de abuso sexual a criança ou adolescente, racismo e correlatos, *malwares*, aplicativos de criptografia e esteganografia. É necessário realizar o cálculo dos *hashes* desses arquivos, extrair seus metadados e salvar esses dados em arquivo, que posteriormente serão incluídos na base;

- SISCRIM (B) – o SISCRIM armazena as informações sobre todos os pedido de análises periciais da PF, como o código de identificação da evidência, local e data de apreensão, assunto de investigação e operação policial vinculada. Sua base de dados é implementada com o SGBD PostgreSQL. As informações extraídas do SISCRIM devem ser importadas para a Base Joio para permitir o mapeamento das evidências que foram periciadas ao caso e assunto de investigação relacionados. Esse mapeamento é importante na fase de MD, pois permitirá identificar arquivos que aparecem em evidências de casos distintos;
- conjuntos de *hashes* de arquivos conhecidos obtidos de fontes externas (C) – podem ter formatos variados, sendo mais utilizados os formatos *Hashkeeper* e CSV. Podem incluir informações sobre o fabricante, tipo de aplicação e nome do produto do qual o arquivo faz parte.

Modelagem do processo de ETL

Como serão coletados dados de diversas fontes, com formatos próprios e atributos específicos, faz-se necessário definir e modelar processos automatizados para realizar as tarefas de ETL para inclusão de dados na base. Esses processos irão realizar as tarefas de seleção e limpeza dos dados (tratamento de valores nulos, inconsistências e ruídos), agregação (integração dos dados de várias fontes), transformação e normalização dos dados, enquadramento às restrições do BD e, finalmente, a importação dos dados tratados.

Modelagem da base de dados

Conforme apresentado na Figura 5.2, é necessário modelar as três bases (1, 2 e 3) que vão receber os dados obtidos na etapa de coleta de dados. A modelagem será baseada nos atributos específicos fornecidos pelas fontes de dados coletados. Adicionalmente, a etapa de MD também vai requerer estruturas de banco de dados próprias para que sejam introduzidas informações com conhecimento de especialistas e para fazer o correlacionamento e a consolidação dos dados que irão formar o conjunto de treinamento e de teste para o algoritmo de indução de ADs.

Inclusão de conhecimento de especialistas, consolidação e correlacionamento dos dados

Nesta etapa é realizada a inclusão de conhecimento de especialistas, a organização e a consolidação das informações para utilização durante o processo de classificação. Devem ser realizadas pesquisas na literatura e consultas a especialistas em análises periciais a respeito de

quais informações podem ser utilizadas para auxiliar a tarefa de classificação de um arquivo quanto à sua potencial relevância para exames periciais. Os resultados desses estudos devem ser modelados e incorporados à base de dados. Essas informações são utilizadas para auxiliar o processo de indução dos modelos de AD.

Após a realização de entrevistas com cinco peritos criminais da área de Computação Forense da Polícia Federal – com tempo de experiência em perícias digitais variando entre 5 e 7 anos – e pesquisas na literatura a respeito de perícias em computadores (Steel, 2006; Carvey, 2007; Bunting, 2007; Anson e Bunting, 2007), foram identificadas informações adicionais que podem vir a auxiliar o processo de classificação de arquivos quanto à sua relevância para exames periciais. Essas informações foram agregadas à base e consistem dos seguintes itens:

- identificação da evidência à qual pertence o arquivo;
- identificação do caso de investigação ao qual pertence o arquivo;
- identificação do assunto de investigação ao qual pertence o arquivo (por exemplo: pedofilia, *malware*, crime financeiro, entre outros);
- classificação do tipo da pasta na qual se encontra um arquivo quanto ao seu propósito para o Sistema Operacional (SO). Essa informação é independente do metadado definido pelo sistema de arquivos e deve levar em consideração as características de organização de pastas do SO. Foram definidas quatro classificações:
    1. pasta de sistema;
    2. pasta de usuário;
    3. *cache* de Internet;
    4. outras.
- identificação dos tipos de arquivos que são comumente encontrados em cada uma das principais pastas do sistema (por exemplo, arquivos do tipo *planilha eletrônica* são comumente encontrados na pasta “Meus documentos” de um usuário, enquanto arquivos do tipo *executável* são normalmente encontrados abaixo da pasta “Windows”);
- identificação dos tipos de arquivos que <u>não</u> são comumente encontrados em cada uma das pastas principais do sistema;
- identificação das extensões nos nomes dos arquivos que são comumente encontradas em cada uma das pastas principais do sistema;

- identificação das extensões nos nomes dos arquivos que não são comumente encontradas em cada uma das pastas principais do sistema.

Essas informações adicionais acrescentadas buscam fornecer ao processo de MD os atributos que são geralmente utilizados por especialistas para classificar um arquivo quanto à sua relevância. Por exemplo, é esperado que haja maior chance de serem classificados como *ignorável* aqueles arquivos que:

- são encontrados em várias evidências/casos/assuntos de investigação distintos;
- estão presentes apenas em pastas de sistema;
- cujo tipo e extensão estejam de acordo com os padrões esperados;
- cujos arquivos irmãos em uma pasta tenham um perfil típico de arquivos ignoráveis.

Mas estas suposições deverão ser validadas para serem confirmadas ou refutadas a partir do modelo de AD induzido a partir dos dados reais.

Após a incorporação e modelagem dessas novas informações, deve ser feito o correlacionamento e consolidação dos dados dos arquivos pelo valor de *hash*. Assim, cada valor de *hash* irá indicar um conteúdo único de arquivo na Base Joio e é necessário correlacionar e consolidar todos os dados referentes a esses valores de *hash*, os quais serão identificados pelo processo de MD como *ignorável* ou não.

Para a definição desse processo também foram utilizados conhecimentos de especialistas, que identificaram as seguintes consolidações a serem implementadas para cada valor de *hash* que representa um conteúdo digital único:

- perfil consolidado dos arquivos que estão presentes na mesma pasta – *sibling files* – do arquivo em análise (por exemplo, a quantidade de arquivos que foram classificados como *alerta* ou *ignorável* pelas BHACs tradicionais ou quantos arquivos eram de um tipo não esperado para aquela pasta);
- caminho no sistema de arquivos mais comumente utilizado;
- nome de arquivo mais frequente;
- tipo de arquivo mais frequente;
- categoria de arquivo mais frequente;
- extensão de arquivo mais frequente.

Os quarenta e sete atributos resultantes do processo de consolidação dos dados pelo valor de *hash* estão descritos no Apêndice A, onde é possível verificar que – para cada valor de *hash* distinto presente na base – será realizada uma consolidação dos dados indicando,

percentualmente, quantas vezes o arquivo referente àquele valor de *hash* apareceu em evidências distintas, em casos distintos, em pastas de sistema, em pastas de usuário, em pastas de cache de Internet, qual o nome de arquivo mais comum, qual o caminho no sistema de arquivos mais comum e assim por diante. Essas informações serão utilizadas como variáveis independentes para que o algoritmo de indução de AD possa inferir o modelo adequado para realizar a classificação dos *hashes* de acordo com a sua relevância para exames periciais.

A organização do *framework* para montagem da Base Joio está ilustrada na Figura 5.3. Processos de ETL são responsáveis por extrair, tratar e carregar os dados coletados de diversas fontes para tabelas da Base Joio:

- ETL 1: busca informações no SISCRIM a respeito das evidências e dos casos de investigação – conforme nomes dos arquivos de metadados – e as armazena em arquivos XML;
- ETL 2: lê os arquivos XML criados pelo processo ETL 1 e os arquivos contendo os metadados dos arquivos da AC (que podem estar em diferentes formatos, como MDB ou CSV) e insere nas tabelas **`cases`**, **`evidences`** e **`case_files`**;
- ETL 3: extrai os dados dos arquivos contendo os conjuntos de *hashes* obtidos de fontes externas e insere nas tabelas **`external_sources`** e **`external_hashsets`**. Os conjuntos de *hashes* de interesse são copiados para a tabela **`selected_hashsets`**.

As informações obtidas de especialistas são introduzidas nas tabelas **`folder_types`**, **`expected_folder_files_extension`** e **`expected_folder_files_types`** e correlacionadas com os dados das tabelas **`case_files`** e **`selected_hashsets`**. O resultado é inserido na tabela **`files_metadata`**, que servirá de base para montar as tabelas com as consolidações intermediárias dos dados. Os dados finais consolidados por *hash* e que serão utilizados pelo processo de MD são armazenados na tabela **`final_instances`**, cujos campos estão descritos no Apêndice A.

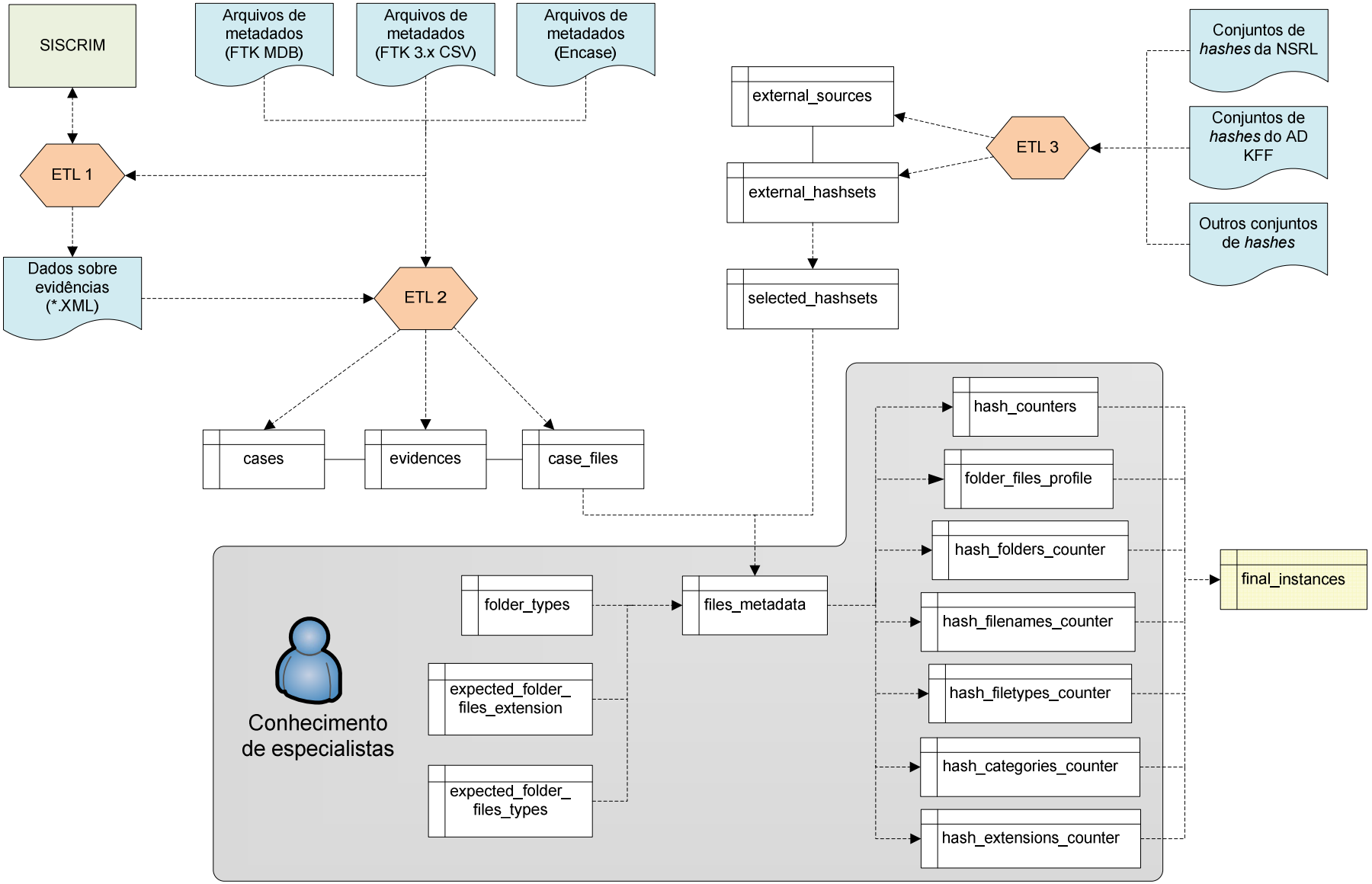

Figura 5.3: Organização do *framework* para montagem da Base Joio.

5.3.2. Classificação de arquivos com o uso de AD

Uma vez que a Base Joio esteja montada e os dados tenham sido tratados e carregados, as informações dos arquivos da AC – consolidadas por *hash* – estarão disponíveis na tabela **`final_instances`**. Conforme mostrado na Figura 5.4, a partir dos dados da tabela **`final_instances`** serão obtidos os conjuntos de treinamento e de teste que serão utilizados pelo algoritmo de indução de AD para inferir o modelo adequado para classificar os arquivos quanto à sua relevância. Devem ser realizados testes de parametrização dos algoritmos de indução de AD e avaliações dos modelos obtidos – com a ajuda dos resultados da aplicação do modelo no conjunto de teste. Ao final dos testes, um modelo de AD deve ser escolhido para ser aplicado à BDAE com o objetivo de identificar novos arquivos ignoráveis a serem adicionados à BHP.

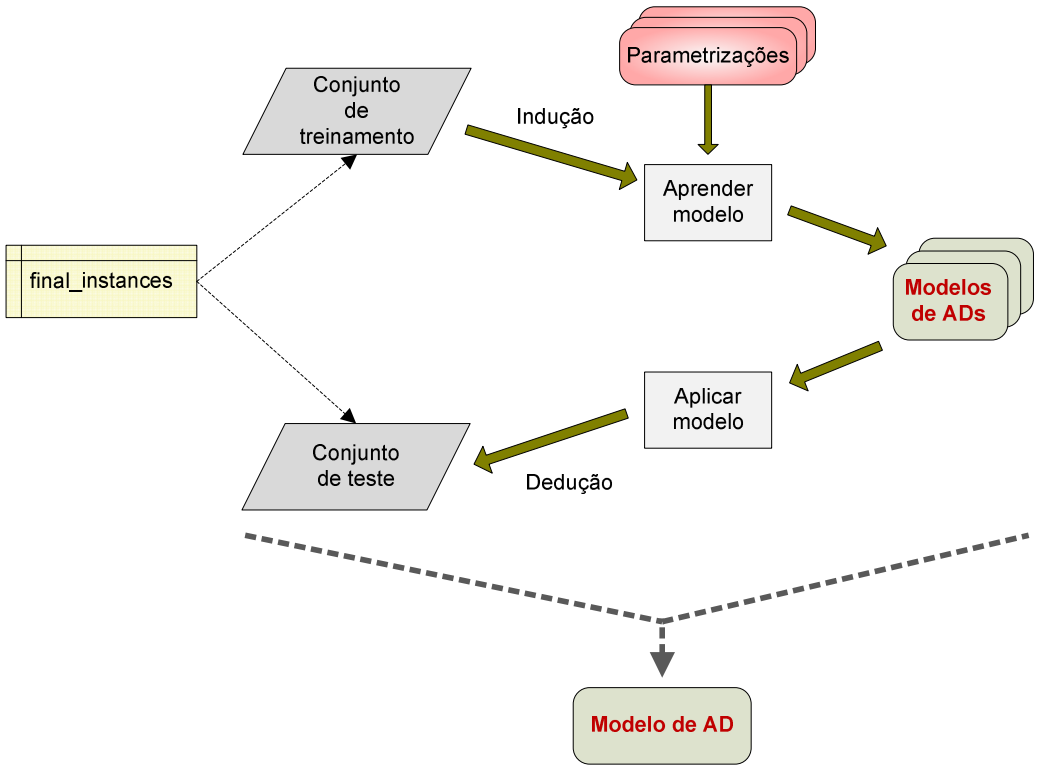

Figura 5.4: Esquema para a definição do modelo de AD a ser utilizado, tendo por base as ADs geradas por algoritmos de indução com o uso de várias opções de parametrização (adaptada de Tan, Steinbach e Kumar, 2005).

Para a realização do processo de indução dos modelos de AD é necessário realizar tarefas de amostragem, classificação manual dos arquivos amostrados, escolha dos conjuntos de treinamento e de teste, parametrização dos algoritmos de indução e análise dos resultados de classificação obtidos pelos modelos de AD.

Amostragem e classificação manual de arquivos para uso como conjunto de treinamento e de teste

O algoritmo de indução de AD necessita que os valores da variável dependente sejam previamente definidos antes de gerar um modelo de AD para um determinado conjunto de treinamento. Neste trabalho, a variável dependente é a *relevância potencial* de um arquivo para exames periciais. Essa variável, entretanto, não está definida nos dados dos arquivos submetidos a exame pericial. É possível apenas determinar a *relevância potencial* dos arquivos identificados por uma BHAC.

Entretanto, a premissa exposta neste trabalho é que existem outros arquivos irrelevantes para exames periciais que também deveriam fazer parte da BHAC. Assim, os dados dos arquivos devem ser analisados por um ou mais especialistas e classificados de acordo com sua relevância potencial. Porém, como a quantidade de arquivos presentes na BDAE é da ordem de milhões de arquivos, é inviável fazer essa classificação manual para todos os arquivos da base. É necessário, então, realizar uma amostragem dos dados e executar o processo de classificação manual a partir dos dados amostrados. Essa amostra classificada manualmente servirá de conjunto de treinamento e de teste para o algoritmo de indução de AD.

Para realizar a amostragem de forma adequadamente aleatória, a solução adotada é utilizar a própria propriedade de pseudo-aleatoriedade das funções criptográficas de *hash* (conforme exposto na Seção 3.1). Dessa forma, a escolha aleatória dos arquivos pode ser alcançada simplesmente selecionando-se uma sequência qualquer de valores de *hash* contendo a quantidade desejada de itens. Esses valores de *hash*, por sua vez, irão representar arquivos da base de forma pseudo-aleatória.

A amostragem deve ser feita apenas sobre arquivos que se repetem ao menos uma vez em mídias computacionais diferentes, pois a premissa deste trabalho é a de que esses arquivos são os que apresentam suficiente potencial de irrelevância para exames periciais. Assume-se que arquivos que não se repetem em mais de uma mídia não possuem indicações fortes o suficiente para serem considerados em uma busca por arquivos irrelevantes.

De acordo com Barbetta (2006), considerando uma população N suficientemente grande e uma amostragem aleatória simples, na qual a proporção de classes na amostra não deva diferir da proporção de classes na população em mais do que o valor do erro amostral tolerável E_0 (com alto nível de confiança, tipicamente 95%), podemos utilizar a Equação 9 para determinar o tamanho da amostra n.

$$n = \frac{1}{{E_0}^2} \tag{9}$$

Portanto, o tamanho da amostra necessário para que o erro amostral tolerável seja de 1% em uma população de tamanho elevado é de $1/(0{,}01)^2 = 10.000$.

Uma vez selecionada a amostra, deve-se fazer a classificação dos arquivos de acordo com 8 níveis de potencial relevância para exames periciais, conforme a Tabela 5.2.

Tabela 5.2 - Classificações sugeridas para a relevância de arquivos para exames periciais.

Sigla	**Descrição**
R	Relevante
PI	Provavelmente ignorável
I+	Ignorável - baixo grau de confiança
I++	Ignorável - médio grau de confiança
I+++	Ignorável - alto grau de confiança
IC	Ignorável conhecido (identificado automaticamente por BHACs tradicionais)
PT	Provavelmente temporário de Internet ignorável
T	Temporário de Internet ignorável

A utilização de vários níveis de classificação se justifica para: (i) permitir uma maior flexibilidade nos ajustes de parametrização do processo de MD e (ii) permitir uma classificação com maior granularidade. Durante o processo de MD os níveis de relevância podem ser reagrupados para ajustar o nível de complexidade e de granularidade de classificação da AD resultante. A maior granularidade da classificação pode permitir utilizações diferentes dos arquivos classificados posteriormente com o uso da AD, a critério do usuário do sistema. Como exemplo, arquivos classificados como “PI” ou “I+” podem ser opcionalmente submetidos a inspeção manual antes de serem incluídos diretamente na base de arquivos ignoráveis.

As classificações “PI”, “I+”, “I++” e “I+++” representam, nessa ordem, uma escala crescente de confiança por parte do especialista na definição de um arquivo como *ignorável*. Foi feita a opção pela utilização desses quatro níveis de confiança porque os critérios para definir a relevância de um arquivo não são únicos e exatos. Cada especialista utiliza seus próprios critérios e espera-se que o uso de uma classificação em níveis, ao invés de uma definição simplesmente binária (sim/não), amenize as distorções de classificação realizadas por especialistas distintos.

Optou-se também por diferenciar na classificação os arquivos advindos de cache de Internet, por sua característica especial para exames periciais. Dependendo do caso de

investigação em curso, pode ser desejável ou não filtrar esses arquivos, uma vez que eles podem indicar o perfil de uso da Internet de um determinado usuário.

Parametrização e análise de resultados de classificadores usando ADs

Para fazer a classificação dos arquivos quanto à sua relevância, foi proposta neste trabalho a utilização de algoritmos classificadores baseados em AD. Foi feita a escolha de algoritmos baseados em ADs por serem apropriados para descoberta exploratória de conhecimento e sua representação de conhecimento em forma de árvore ser intuitiva e de fácil assimilação.

Tendo por base os dados consolidados dos *hashes* amostrados e classificados manualmente, devem ser realizados testes de parametrização em algoritmos de indução de AD para gerar um modelo que classifique os *hashes* de acordo com sua relevância para exames periciais. A partir da análise dos critérios de classificação utilizados pelos modelos de AD gerados será escolhida uma proposta final de AD para ser utilizada na Base Joio.

Os testes com os algoritmos de indução de AD incluem:

- procedimentos de análise exploratória dos dados;
- a utilização de diferentes algoritmos (como C4.5 e CHAID) e parâmetros de separação e de poda utilizados pelos algoritmos;
- diferentes parâmetros de discretização de valores;
- agrupamentos e filtragens de classes;
- exclusão de variáveis independentes irrelevantes (redução de dimensionalidade).

Para realizar a escolha dos conjuntos de dados de treinamento e de teste, propõe-se a realização de testes utilizando os métodos (i) *holdout* na proporção 70% - 30%; (ii) validação cruzada com 10 subconjuntos; e (iii) *bootstrapping* com 10 validações e amostragem de 100% para o conjunto de treinamento. Segundo Rokach e Maimon (2008), geralmente são escolhidos dois terços dos dados para o conjunto de treinamento e um terço para o conjunto de teste no método *holdout*, enquanto Murthy (1998) cita que a validação cruzada com 10 subconjuntos é geralmente reconhecida como uma boa avaliação da qualidade preditiva de uma AD.

Aplicação do modelo de AD

De acordo com a avaliação dos resultados dos modelos de AD realizados durante a MD, um modelo de AD deve ser escolhido e aplicado à BDAE completa para identificar os *hashes* de

arquivos potencialmente irrelevantes. Esses *hashes* irão compor a base de *hashes* de arquivos classificados como ignoráveis e serão adicionados à BHP.

5.3.3. Seleção de conjuntos de *hashes* efetivos das BHAC tradicionais

O crescimento do número de registros em BHAC podem impactar significativamente o custo computacional do procedimento pericial de filtragem de arquivos conhecidos pelo valor de *hash*, conforme destacado por Roussev, Richard III e Marziale (2008): "*Em perícias digitais, tem sido bem-sucedido o uso de bancos de dados de hashes de arquivos conhecidos de sistemas e aplicativos, tais como aqueles mantidos pelo NIST. Mas é discutível se essa abordagem irá funcionar quando os bancos de dados contiverem bilhões de valores de hash – seria necessário utilizar clusters de computadores apenas para realizar buscas por hash?*".

Para evitar o aumento do custo computacional advindo do uso de BHAC muito grandes, propõe-se adotar um método para identificar os conjuntos de *hashes* mais efetivamente utilizados das BHAC tradicionais, excluindo da BHP aqueles que não se mostrarem potencialmente efetivos. Como medida de efetividade, sugere-se utilizar os *hashes* dos arquivos da AC, pois esses arquivos contêm uma indicação a respeito do parque de *softwares* mais utilizados em um determinado país ou região e, por esse motivo, podem identificar quais conjuntos de *hashes* da BHAC possuem maior efetividade.

Dessa forma, a proposta para a montagem de uma BHAC mais enxuta é utilizar apenas os conjuntos de *hashes* das BHAC tradicionais que atendam a pelo menos um dos seguintes critérios:

1. ao menos um valor de *hash* contido no conjunto de *hashes* deve estar presente na BDAE;
2. o conjunto de *hashes* deve ser referente a um *software* lançado recentemente.

O primeiro critério, ilustrado na Figura 5.5, busca identificar os conjuntos de *hashes* referentes a *softwares* efetivamente utilizados em um dado mercado. Note que os conjuntos de *hash* referenciados ao menos uma vez pela AC (destacados em vermelho) são inseridos na base de *hashes* efetivamente usados.

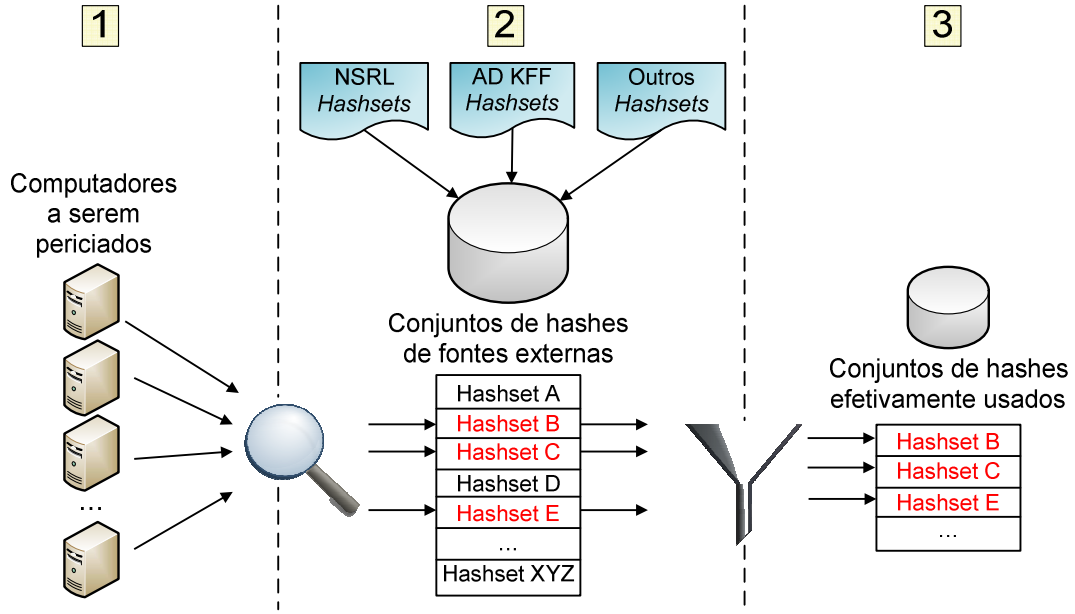

Figura 5.5 - Extração dos conjuntos de *hashes* efetivos das BHACs tradicionais com o auxílio dos dados dos arquivos presentes nos computadores a serem periciados.

Para satisfazer o segundo critério, propõe-se incluir os conjuntos de *hashes* adicionados recentemente às BHAC tradicionais – ver Figura 5.6. Como esses conjuntos de *hashes* normalmente representam lançamentos novos de *softwares*, eles podem ainda não estar presentes na AC. Esse critério, portanto, serve para complementar o primeiro e evitar que conjuntos de *hashes* potencialmente efetivos não sejam identificados pela AC por serem muito recentes. Dessa forma, é preciso avaliar quanto tempo é necessário para que um *software* recentemente lançado seja representado na AC utilizada e, assim, identificar quão recente deve ser um conjunto de *hash* para satisfazer o segundo critério. Essa tarefa pode ser realizada empiricamente considerando a experiência do perito.

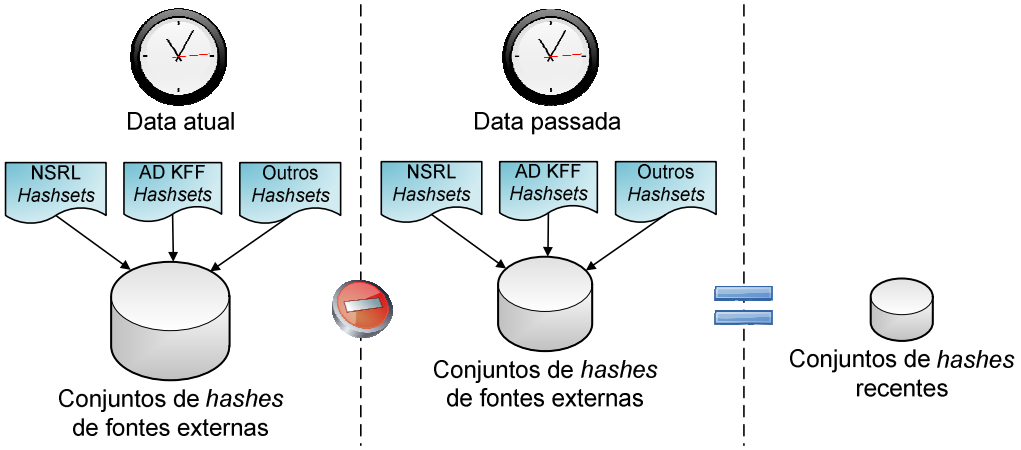

Figura 5.6 - Extração dos conjuntos de *hashes* mais recentes das BHACs tradicionais.

Dessa forma, a SCH será formada por (i) conjuntos de *hashes* de BHACs tradicionais identificados como efetivamente usados pela BDAE; e (ii) conjuntos de *hashes* recentemente adicionados a BHACs tradicionais.

Conforme ilustrado na Figura 5.7, a BHP será então composta pela SCH e pelos *hashes* dos arquivos identificados como *ignorável* pelo processo de MD.

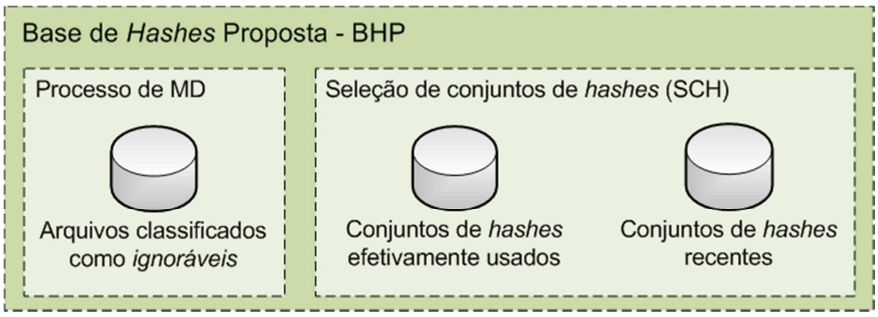

Figura 5.7 - Composição da Base de *Hashes* Proposta.

5.4. PROTÓTIPO

Um protótipo foi implementado para avaliar a viabilidade da solução proposta. Os conjuntos de dados e ferramentas utilizados serão descritos, assim como os detalhes de projeto e implementação.

5.4.1. Conjuntos de dados e ferramentas

A AC utilizada foi composta por 100 computadores de casos reais de exames periciais da Polícia Federal, contendo 4.045.808 arquivos, cujos dados foram extraídos com o uso da ferramenta forense *AccessData FTK* versão 1.81. A Tabela 5.3 apresenta em detalhes as características da AC. Os computadores faziam parte de quatro casos de investigação, relacionados a três assuntos de investigação: desvio de dinheiro público/corrupção, crime ambiental e exploração de jogos de azar. Apenas computadores contendo o SO *Microsoft Windows* foram utilizados, uma vez que esse SO representa a grande maioria dos casos de exames periciais na PF.

Tabela 5.3 - Descrição da AC utilizada.

Característica da AC	Quantidade
Computadores	100
Arquivos	4.045.808
Arquivos únicos	1.866.121
Casos de investigação	4
Assuntos de investigação	3

Foram utilizados os conjuntos de *hash* da NSRL nas versões RDS 2.32 (contendo 19.255.196 *hashes* únicos) e RDS 2.31 (18.840.521 *hashes* únicos) para identificar arquivos conhecidos na AC e também como referencial de comparação para a BHP. Foram usadas duas versões da NSRL para que fosse possível identificar os conjuntos de *hashes* mais recentes introduzidos na versão 2.32.

A implementação do *framework* para montagem da Base Joio foi realizada por meio de três módulos:

- o banco de dados foi implementado com o SGBD *PostgreSQL*[11] versão 9.0.3;
- as operações de ETL foram realizadas com o uso da ferramenta *Pentaho Data Integration Community Edition*[12] (*Kettle*) versão 4.1.0 (Pentaho, 2011);
- as operações de agrupamento e consolidação dos dados foram implementadas através de comandos e consultas SQL e de códigos PL/pgSQL.

O processo de MD com AD foi implementado com o uso da ferramenta *Rapid-I RapidMiner*[13] versão 5.1. Todos os softwares utilizados, com exceção da ferramenta pericial *AccessData FTK*, são ferramentas de código-aberto referência em suas áreas de aplicação.

5.4.2. Projeto e implementação

O projeto e a implementação do protótipo foram realizados de acordo com as especificações apresentadas na Seção 5.3 para a construção do *framework* para montagem da Base Joio, classificação de arquivos com o uso de AD e seleção de conjuntos de *hashes* efetivos de BHAC tradicionais.

[11] http://www.postgresql.org/
[12] http://kettle.pentaho.com/
[13] http://rapid-i.com/

Coleta de dados

Inicialmente, foi realizada a coleta dos dados dos arquivos da AC e dos RDS da NSRL. Conforme exposto na Tabela 5.4, cada versão de RDS da NSRL consiste de quatro arquivos CSV inter-relacionados contendo dados sobre os produtos e os respectivos arquivos, SO e fabricantes aos quais se referem.

Tabela 5.4 - Conteúdo dos arquivos de um RDS da NSRL.

Arquivo	Conteúdo
NSRLFile.txt	Dados dos arquivos
NSRLProd.txt	Dados dos produtos aos quais referem-se os arquivos
NSRLOS.txt	Dados dos SO aos quais referem-se os produtos
NSRLMfg.txt	Dados dos fabricantes aos quais referem-se os produtos ou SO

O relacionamento lógico dos registros de um RDS da NSRL está ilustrado na Figura 5.8.

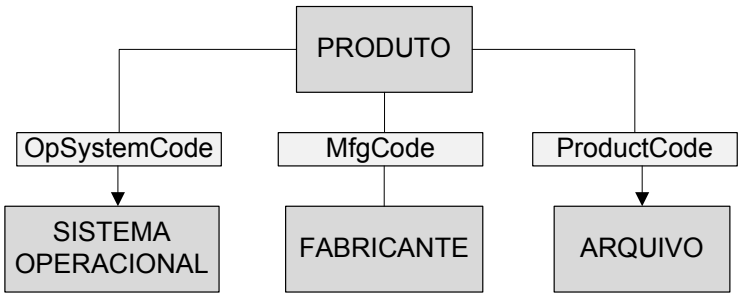

Figura 5.8 - Relacionamento lógico dos registros de um RDS da NSRL (adaptada de National Institute of Standards and Technology, 2009).

Os dados da AC, por sua vez, são armazenados em arquivos do tipo MDB nomeados de acordo com o número e ano do laudo pericial ao qual estão relacionados. O leiaute de nome utilizado é formado por *LaudoXXXX_AAAA.mdb*, onde *XXXX* representa o número do laudo e *AAAA* representa o ano. Um exemplo de metadados de um arquivo presente em um arquivo MDB da AC está exposto na Tabela 5.5.

Tabela 5.5 - Exemplo de metadados de um arquivo extraídos com a ferramenta forense FTK.

Atributo	**Valor**
Nome do arquivo	igfxress.dll
Caminho do arquivo	Disco 765.001\Part_1\NONAME-NTFS\WINDOWS\NLDRV\008\
Tamanho lógico	15.032.320
Data de criação	2008-07-06 07:10:49
Tipo de arquivo	Executable File
Categoria	Executable
Extensão	dll
Extensão confere com tipo do arquivo?	Sim
Deletado?	Não
Somente leitura?	Não
Sistema?	Não
Oculto?	Não
Compactado?	Não
Criptografado?	Não
Cabeçalho	4D5A900003000000
Hash MD5	1ED9073BB40AAED3B4BE6708ABECEE57
Hash SHA-1	FE5A2F48C8C90ED9FC0EE4F427FB7DEDE2445FEB

Operações de ETL

Para realizar as operações de ETL foram implementadas transformações e *jobs* com o uso da ferramenta Kettle. As transformações são utilizadas para transformar e mover os dados da fonte de origem para o destino, enquanto os *jobs* estão relacionados a um nível mais alto de controle de fluxo – executando as transformações e promovendo verificações e notificações ao usuário (Pentaho, 2011).

A Figura 5.9 apresenta os *jobs* e transformações implementados para executar os processos ETL 1 e ETL 2 mostrados na Figura 5.3. O Item 1 da figura representa o *job* que utiliza a transformação definida no Item 2 para extrair os números dos laudos dos nomes dos arquivos MDB, acessar as informações correspondentes sobre os casos e evidências no SISCRIM e salvá-las em arquivos XML. O Item 3 mostra a transformação Kettle que utiliza os metadados gravados nos arquivos MDB e as informações correspondentes salvas nos arquivos XML pela transformação descrita no Item 2 para inserir na base de dados as informações sobre os casos (Tabela **`cases`**), evidências (Tabela **`evidences`**) e metadados

dos arquivos (Tabela **`case_files`**) da AC. Portanto, os Itens 1 e 2 implementam o processo ETL 1, enquanto o Item 3 implementa o processo ETL 2 da Figura 5.3.

O motivo para salvar as informações sobre os casos e evidências em arquivos XML através de um processo de ETL separado (ETL 1) deve-se ao fato de que nem sempre esses dados estarão disponíveis no SISCRIM - seja porque o sistema está inacessível ou porque os dados foram obtidos de computadores não cadastrados no sistema. Assim, o usuário tem a opção de criar manualmente arquivos XML contendo as informações necessárias para a execução do processo ETL 2.

A implementação do processo ETL 3 – que importa e trata os dados dos conjuntos de *hashes* obtidos de fontes externas – foi feita através de uma transformação Kettle que carrega os dados dos quatro arquivos que compõem um RDS da NSRL – apresentados na Tabela 5.4 – para quatro tabelas intermediárias, uma para cada arquivo, e depois realiza o *join* dos dados – conforme os relacionamentos ilustrados na Figura 5.8 – e os insere na Tabela **`external_hashsets`**. Também é realizada a inserção das informações sobre a fonte de origem dos dados na Tabela **`external_sources`**.

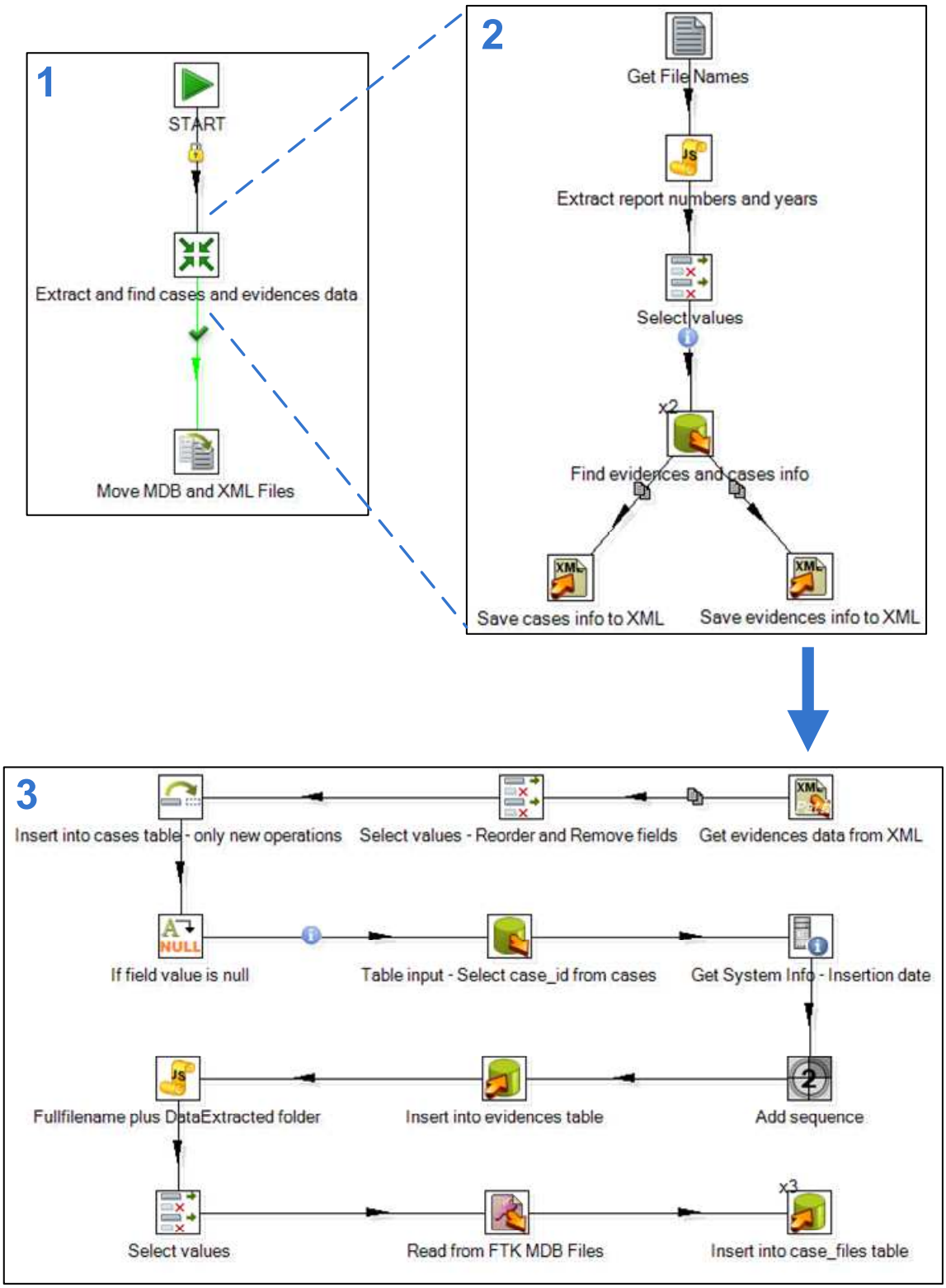

Figura 5.9: Procedimento de ETL utilizando a ferramenta Kettle para inserir na base de dados as informações de arquivos extraídos com o uso da ferramenta pericial FTK.

A Tabela 5.6 ilustra os atributos disponíveis na NSRL relacionados a um arquivo, os quais são importados para a Tabela **`external_hashsets`**.

Tabela 5.6 - Exemplo de atributos disponíveis na NSRL a respeito de um arquivo.

Atributo	**Valor**
Nome do arquivo	01402007.gif
Hash MD5	354D691FF38648791AA01052BCF478DE
Hash SHA-1	D6ADCC23E78E57902D191B68C9160672353576F3
Código CRC32	97171725
Nome do produto	Gallery
Fabricante do produto	-
Idioma do produto	English
Tipo de aplicação	Graphic/Drawing
Nome do SO	Windows
Versão do SO	Generic
Fabricante do SO	-

Modelagem do BD

O banco de dados, por sua vez, foi modelado para que seja possível:

- receber os dados coletados;
- implementar as ideias e informações obtidas a partir de entrevistas realizadas com peritos criminais federais a respeito de atributos que possam auxiliar a tomada de decisão sobre a relevância de arquivos para exames periciais;
- consolidar as informações por valor de *hash*, de modo a serem utilizadas pelo processo de MD.

O modelo E-R do banco de dados está ilustrado no Apêndice B, o qual foi elaborado utilizando-se a versão 3.6 *freeware* da ferramenta *Toad Data Modeler*[14] da empresa *Quest Software*.

As principais ideias e informações apresentadas por peritos criminais federais foram descritas na Seção 5.3.1. A proposta de classificar as pastas como sendo de sistema, usuário ou cache de Internet foi implementada através da tabela **`folder_types`**, cujo conteúdo está exposto na Tabela 1 do Apêndice C. Essa tabela contém expressões regulares para identificar as pastas mais conhecidas de um SO Windows e campos para classificá-las de acordo com o tipo correspondente.

A identificação dos tipos e extensões de arquivos esperados e inesperados para as principais pastas de um SO Windows foi implementada através das Tabelas **`expected_folder_files_types`** e **`expected_folder_files_extensions`**,

[14] http://www.quest.com/toad-data-modeler/

respectivamente, cujos conteúdos estão expostos nas Tabelas 2 e 3 do Apêndice C. Essas tabelas também utilizam expressões regulares para identificar as pastas mais conhecidas de um SO Windows, além de campos para identificar os tipos ou extensões de arquivos e classificá-los como "esperado" ou "inesperado" de acordo com a pasta. Para identificar os tipos e extensões mais comuns esperados ou inesperados para as principais pastas de um SO Windows, foram realizadas consultas bibliográficas (conforme descrito na Seção 5.3.1) e análises estatísticas nos dados da AC.

O resultado do correlacionamento dos dados das Tabelas **`case_files`**, **`folder_types`**, **`expected_folder_files_types`** e **`expected_folder_files_extensions`** é inserido na Tabela **`files_metadata`**, que também utiliza os dados dos conjuntos de *hashes* de BHAC tradicionais (Tabela **`selected_hashsets`**) para identificar – pelo valor de *hash* – os arquivos conhecidos.

Dessa forma, a Tabela **`files_metadata`** irá conter novos atributos a respeito de um arquivo além daqueles extraídos originalmente com o uso da ferramenta forense. A Tabela 5.7 apresenta um exemplo desses atributos adicionais para o arquivo apresentado na Tabela 5.5. Pode-se perceber que também foi criado um atributo contendo o caminho no sistema de arquivos apenas com os nomes das pastas onde se encontra o arquivo – excluindo nomes do disco, número e rótulo da partição.

Tabela 5.7 - Exemplo de atributos adicionais de um arquivo implementados a partir de ideias apresentadas por especialistas.

Atributo	**Valor**
Caminho "limpo"	\WINDOWS\NLDRV\008\
Ignorável	-
Alerta	-
Pasta de sistema?	Sim
Pasta de usuário?	-
Pasta de cache de Internet?	-
Extensão do arquivo era esperada para a pasta?	Sim
Extensão do arquivo era inesperada para a pasta?	-
Tipo do arquivo era esperado para a pasta?	Sim
Tipo do arquivo era inesperado para a pasta?	-
Evidência	A
Caso de investigação	Operação X *<nome real omitido por sigilo>*
Assunto de investigação	Corrupção

Entretanto, para implementar o processo de MD que utilizará algoritmos de indução de AD para identificar novos arquivos potencialmente ignoráveis, é necessário consolidar todos os dados por valor de *hash*, uma vez que é o *hash* que identifica unicamente o conteúdo de um arquivo – que será classificado como *ignorável* ou não.

Dessa forma, foram criadas tabelas para fazer consolidações intermediárias a partir da Tabela **`files_metadata`**, conforme descrito a seguir:

- **`hash_counters`**: tabela que apresenta a contagem dos atributos de todos os arquivos que compartilham um mesmo valor de *hash*: o total de arquivos; quantos estavam presentes em pastas de sistema, usuário ou cache de Internet; a quantidade de evidências, assuntos de investigação e casos distintos onde os arquivos foram encontrados; o total de arquivos com atributos no sistema de arquivos iguais a *deletado*, *somente-leitura* ou *oculto*; quantos arquivos continham a extensão do nome esperada ou inesperada para a pasta em que estavam; quantos arquivos eram de um tipo esperado ou inesperado para a pasta onde estavam;
- **`folder_files_profile`**: consolida o perfil dos arquivos presentes em cada pasta de todas as evidências da AC: o total de arquivos; quantos são reconhecidos por BHAC como *ignorável* ou *alerta*; o total de arquivos deletados, somente-leitura, ocultos, compactados ou criptografados; quantos tinham a extensão ou o tipo esperado ou inesperado para a pasta; e também os atributos da pasta no sistema de arquivos: se era *deletada*, *somente-leitura*, *oculta* ou de *sistema*;
- **`hash_folders_counter`**: identifica, para cada valor de *hash h*, todos os nomes de pasta (caminhos completos) onde *h* foi encontrado, calculando quantas vezes *h* era encontrado na mesma pasta em evidências distintas;
- **`hash_filenames_counter`**: identifica, para cada valor de *hash h*, todos os nomes de arquivo com os quais *h* foi encontrado, assim como o respectivo total de arquivos com valor de *hash h* compartilhando o mesmo nome;
- **`hash_filetypes_counter`**: identifica, para cada valor de *hash h*, todos os tipos de arquivo com os quais *h* foi encontrado, assim como o respectivo total de arquivos identificados com o mesmo tipo de arquivo e valor de *hash*;
- **`hash_categories_counter`**: identifica, para cada valor de *hash h*, todas as categorias de tipos de arquivo com as quais *h* foi classificado, assim como o respectivo total de arquivos com valor de *hash h* identificados com a mesma categoria;

- **`hash_extensions_counter`**: identifica, para cada valor de *hash h*, todas as extensões de nomes de arquivo com as quais *h* foi encontrado, assim como o respectivo total de arquivos com valor de *hash h* compartilhando a mesma extensão de nome.

A consolidação final por valor de *hash* – agregando os dados de todas as tabelas de consolidação intermediárias – foi implementada por meio da Tabela **`final_instances`**. A consolidação foi feita atribuindo valores percentuais – ao invés de nominais – para facilitar a identificação de regras pelo processo de MD que possam ser aplicadas em AC de tamanhos variados. Dependendo do campo, a consolidação percentual foi feita em relação ao total de arquivos, de evidências, de casos ou de assuntos de investigação Os campos da tabela **`final_instances`** consistem dos atributos que serão utilizados pelo processo de MD para classificar os arquivos quanto à sua relevância potencial para exames periciais. Esses atributos e, portanto, os campos da Tabela **`final_instances`**, estão descritos no Apêndice A. A Tabela 5.8 apresenta um exemplo com alguns campos de um registro da Tabela **`final_instances`**.

Conforme detalhado na Seção 5.3.2, somente foram incluídos na Tabela **`final_instances`** os registros referentes a valores de *hash* encontrados em pelo menos duas evidências diferentes. Assim, um total de 242.557 valores de *hash* distintos foram consolidados na Tabela **`final_instances`** a partir dos dados coletados.

Com a consolidação dos dados na Tabela **`final_instances`**, o *framework* para montagem da Base Joio – descrito na Seção 5.3.1 – estava, então, projetado, implementado e pronto para ser utilizado pelo processo de MD.

Tabela 5.8 – Exemplo de informações consolidadas por *hash*, que serão utilizadas pelo algoritmo de indução de AD.

Atributo	**Valor**
Valor de *hash* (MD5)	FFF5D2AA3A829AA5151873F2E986CEF3
Valor de *hash* (SHA-1)	6705BDCB7DE2CAC14055CF09084AA5DBA42E3F85
percent_distinct_evidences	7%
percent_distinct_cases	75%
percent_distinct_case_subjects	66%
percent_systemfolders	87%
percent_userfolders	0%
percent_internetcachefolders	0%
percent_badxt	0%
percent_deleted	0%
percent_readonly	0%
percent_hidden	0%
percent_filetype_expected	87%
percent_filetype_not_expected	0%
percent_extension_expected	87%
percent_extension_not_expected	0%
avg_percent_folder_files_ignorable	0,578035%
avg_percent_folder_files_alert	0%
avg_percent_folder_files_badxt	0%
avg_percent_folder_files_readonly	0,385356%
avg_percent_folder_files_hidden	0%
avg_percent_folder_files_duplicated	31,02119%
avg_percent_folder_files_filetype_expected	87,66859%
avg_percent_folder_files_filetype_not_expected	0%
avg_percent_folder_files_extension_expected	87,66859%
avg_percent_folder_files_extension_not_expected	0%
most_common_cleanpath	\Arquivos de programas\HP\Digital Imaging\Skins\oov1\sc\img\
percent_max_path_occurrences	62%
most_common_filename	sc-rotateRight.png
percent_max_filename_occurrences	100%
most_common_filetype	PiNG File (Portable Network Graphics)
percent_max_filetype_occurrences	100%
most_common_category	Graphic
percent_max_category_occurrences	100%
most_common_extension	png
percent_max_extension_occurrences	100%

Classificação manual de amostra

Conforme detalhado nas Seções 5.2 e 5.3.2, faz-se necessário extrair uma amostra dos registros da Tabela **`final_instances`** para serem utilizados como conjunto de treinamento e de teste para o algoritmo de indução de AD.

Uma amostragem satisfatoriamente aleatória foi obtida utilizando a propriedade de pseudo-aleatoriedade das funções criptográficas de *hash*. Conforme exposto na Seção 5.3.2, o tamanho desejável da amostra é de cerca de 10.000 registros. Assim, para selecionar uma amostra com tamanho próximo ao desejado, foram escolhidos os registros cujos valores de *hash* MD5 iniciavam com o valor “F”, exceto “F1”, “F2” e “F3”. Através dessa abordagem, foi possível selecionar 12.528 registros referentes a conteúdos de arquivos escolhidos de maneira pseudo-aleatória para compor a amostra. Cabe ressaltar que poderia ter sido utilizado qualquer outro intervalo de valores de *hash* que resultasse em um tamanho de amostra próximo ao desejado.

Os registros da amostra foram, então, classificados manualmente de acordo com os níveis de relevância expostos na Tabela 5.2. O resultado dessa classificação é apresentado na Tabela 5.9.

Tabela 5.9 – Resultado da classificação da amostra de registros.

Classificação de relevância	Quantidade de registros
R	3.128
PI	884
I+	1.182
I++	2.677
I+++	849
IC	2.929
PT	826
T	53

Processo de MD

Para realizar a indução de modelos de AD para classificar os arquivos quanto à sua relevância para exames periciais, foram implementados processos por meio da ferramenta RapidMiner. Os processos são projetados através de operadores que realizam tarefas de: acesso ao repositório de dados; transformação dos dados (limpeza, filtragem, redução de dimensionalidade, conversão de tipos, entre outras); importação/exportação de dados; escolha

do algoritmo de indução de AD; escolha do método de definição dos conjuntos de treinamento e de teste; e avaliação do desempenho do modelo induzido.

A Figura 5.10 apresenta o processo implementado para realizar a indução do modelo de AD pelo protótipo. Os dados da amostra classificada são obtidos do repositório de dados e, em seguida, são utilizados diversos operadores que atuam sobre os dados e permitem que sejam parametrizadas as tarefas de redução de dimensionalidade, definição dos papéis (*roles*) dos atributos, filtragem de exemplos, mapeamento e discretização de valores e substituição de valores nulos. Os dados tratados são então submetidos a um operador de validação que divide os exemplos entre os conjuntos de treinamento e de teste, realiza a indução do modelo de AD e avalia o desempenho do modelo quando aplicado ao conjunto de teste. É possível escolher entre operadores de validação que utilizam os métodos *holdout*, validação cruzada e *bootstrapping* para definição dos conjuntos de treinamento e de teste e avaliação do modelo gerado, conforme apresentado na Seção 5.3.2 - *Parametrização e análise de resultados de classificadores usando ADs*.

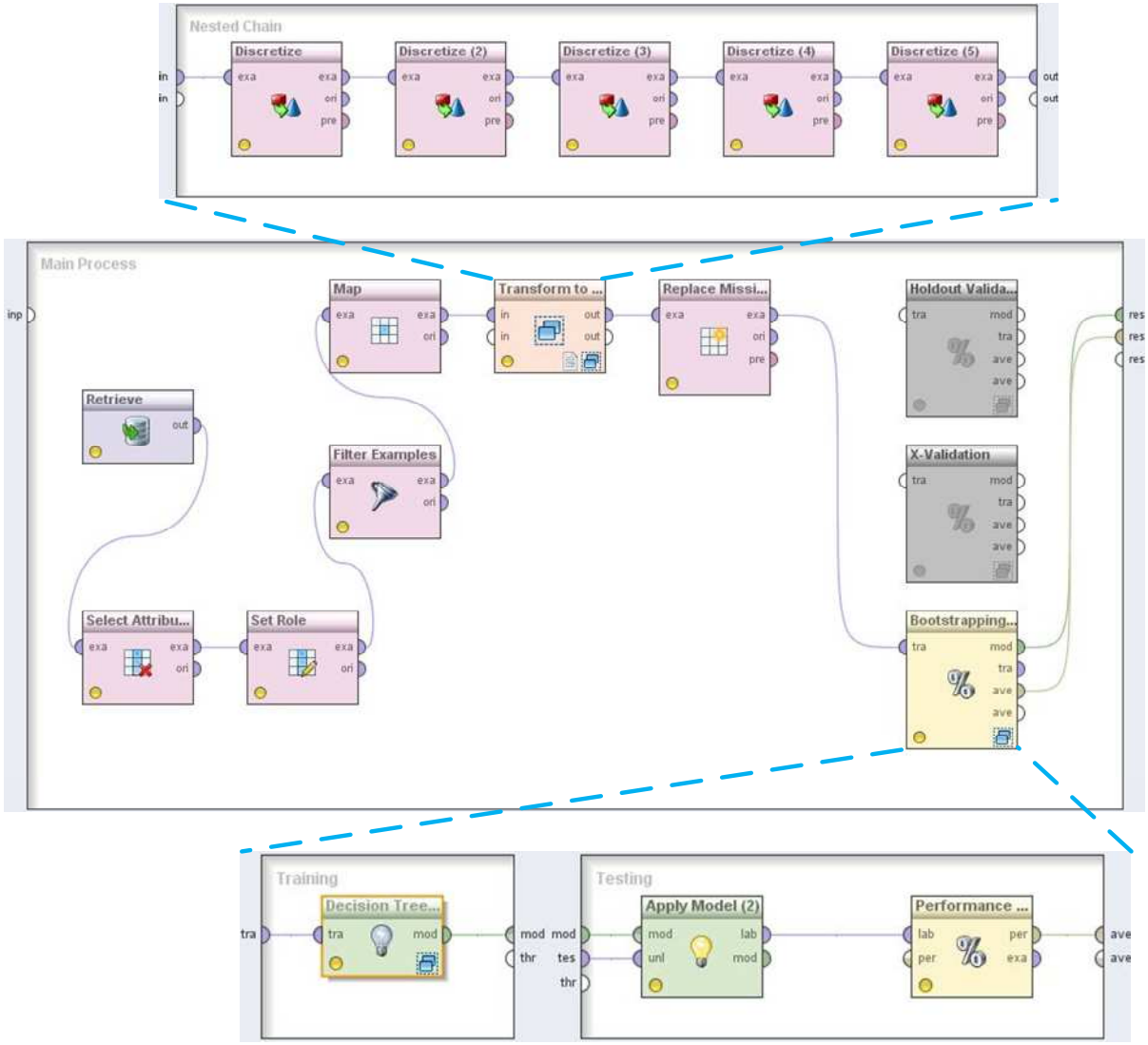

Figura 5.10 - Processo de MD para indução de AD implementado com o uso da ferramenta RapidMiner.

Também foi implementado um processo *RapidMiner* para aplicar o modelo de AD em todos os registros da Tabela **`final_instances`**, obtendo, assim, a classificação da relevância de todos os registros da tabela e, consequentemente, a identificação de todos os valores de *hash* da AC considerados ignoráveis pelo modelo de AD utilizado. Esses valores de *hash* classificados como *ignorável* irão compor a BHP.

Seleção de conjuntos de *hashes* de BHAC tradicionais

Como o *framework* para montagem da Base Joio do protótipo já estava implementado, a criação da SCH, conforme defindo na Seção 5.3.3, consistiu da execução de comandos SQL na Base Joio. A BHAC utilizada foi composta pelo RDS 2.32 da NSRL.

Para identificar os conjuntos de *hashes* efetivos da BHAC, conforme ilustrado na Figura 5.5, foi utilizado o código SQL apresentado a seguir:

```
create table effective_hashsets as (
     select *
     from selected_hashsets
     where hashset_name IN (
          select distinct s.hashset_name
          from case_files c LEFT OUTER JOIN selected_hashsets s
ON (c.md5 = s.md5)
          where s.hashset_name IS NOT NULL
     )
)
```

Esse código identifica os nomes dos conjuntos de *hashes* que são referenciados ao menos uma vez por algum arquivo da AC e seleciona todos os registros na BHAC referentes a esses conjuntos de *hashes* identificados. O resultado é armazenado na Tabela **effective_hashsets**, que é criada com estrutura idêntica à da Tabela **selected_hashsets**.

A execução do código SQL na base de dados do protótipo resultou na identificação de um total de 9.699.684 valores de *hash*.

Para identificar os conjuntos de *hashes* mais recentes da BHAC, conforme ilustrado na Figura 5.6, optou-se por utilizar os *hashes* adicionais acrescentados à NSRL pela versão de RDS 2.32 (de abril de 2011) em relação àqueles existentes no RDS 2.31 (de janeiro de 2011). Caso desejado, esse período de diferença entre versões pode ser aumentado de modo a ampliar o prazo no qual novos *hashes* serão considerados recentes.

Foi utilizado o seguinte código SQL para encontrar os *hashes* mais recentes da BHAC:

```
CREATE TABLE most_recent_hashsets AS (
     SELECT newer.*
     FROM      (select * from external_hashsets
                where source_id = 2) newer
               LEFT OUTER JOIN
               (select * from external_hashsets
                where source_id = 1) older
               ON (newer.sha1 = older.sha1)
     WHERE older.filename IS NULL
)
```

Esse comando SQL realiza um *outer join* entre os registros da Tabela **external_hashsets** referentes ao RDS 2.32 da NSRL com aqueles referentes ao RDS 2.31 e seleciona os registros que ocorrem apenas no RDS 2.32. O resultado é gravado na

Tabela **`most_recent_hashsets`**, que é criada com estrutura idêntica à da Tabela **`external_hashsets`**.

A execução do comando SQL na base de dados do protótipo resultou na identificação de um total de 414.675 valores de *hash* novos no RDS 2.32. Desse total, 245.218 valores de *hash* também estão presentes na tabela **`effective_hashsets`**, de modo que apenas 169.457 novos valores de *hash* foram identificados.

Dessa forma, conforme apresentado na Tabela 5.10, a SCH do protótipo, formada pelos *hashes* presentes nas Tabelas **`effective_hashsets`** e **`most_recent_hashsets`** foi composta de um total de 9.869.141 valores de *hash*, o equivalente a cerca de 51,25% do total de valores de *hash* presentes no RDS 2.32 da NSRL.

Tabela 5.10 - Composição da SCH do protótipo

Conjunto de *hashes*	**Tabela**	**Total de *hashes* únicos**
Hashes efetivos	**`effective_hashsets`**	9.699.684
Hashes mais recentes	**`most_recent_hashsets`**	169.457
SCH – Total	**–**	**9.869.141**

Portanto, conforme ilustrado na Figura 5.7, a BHP implementada pelo protótipo será composta pela SCH, além dos *hashes* identificados como ignoráveis pelo processo de MD.

A proposta de solução que foi apresentada neste capítulo é composta por três módulos: (i) *framework* para montagem da Base Joio; (ii) processo de MD para classificação de arquivos com o uso de AD e (iii) seleção de conjuntos de *hashes* efetivos das BHAC tradicionais. A arquitetura definida pela solução proposta foi implementada por meio de um protótipo. Os experimentos relacionados à indução de modelos de AD para classificação dos arquivos e os estudos de caso comparando os resultados de filtragem da BHP com os da NSRL serão descritos no Capítulo 6.

6. EXPERIMENTAÇÃO E RESULTADOS

Para a definição da AD a ser utilizada no protótipo para classificar os arquivos quanto à sua relevância para exames periciais, foram conduzidos diversos experimentos com diferentes opções de agrupamento de classes, discretizações dos valores dos atributos, redução de dimensionalidade e parametrizações de algoritmos de indução de AD. Foram selecionados três desses experimentos para serem apresentados em detalhes neste capítulo. Também foram realizados estudos de caso com arquivos de casos reais para avaliar os resultados da utilização da BHP em comparação com o uso dos conjuntos de *hashes* de arquivos conhecidos da NSRL, os quais serão apresentados nas próximas seções.

6.1. EXPERIMENTOS – CONSIDERAÇÕES INICIAIS

Os testes iniciais de parametrização do processo de MD mostraram que o número de classes utilizadas no protótipo (oito classes: “R”, “PI”, “I+”, “I++”, “I+++”, “IC”, “PT” e “T” – conforme Tabela 5.2) estava resultando em modelos de AD muito complexos e levando à possível ocorrência de *overfitting*. Como, para os objetivos experimentais do protótipo, é desnecessário utilizar tantos níveis de relevância, os experimentos foram conduzidos sempre com algum tipo de agrupamento de classes para simplificar a AD resultante, sem prejudicar o objetivo final de identificar novos arquivos potencialmente ignoráveis.

Conforme já destacado nas Seções 5.3.2 e 5.4.2, somente são utilizados no processo de MD os registros a respeito de arquivos que se repetem ao menos uma vez em mídias computacionais diferentes na AC. Portanto, todas as regras estabelecidas pelas AD induzidas nos experimentos têm essa premissa como regra base.

Os parâmetros possíveis de serem utilizados no *RapidMiner* para configurar o operador que implementa o algoritmo de indução de AD são os seguintes:

- *critério* – define qual é a medida a ser utilizada para a escolha dos atributos que resultam na melhor separação dos registros. A escolha do critério *taxa de ganho* resulta na utilização do Algoritmo C4.5;
- *tamanho mínimo para separação* – define qual é a quantidade mínima de registros em um nodo para permitir uma separação;
- *tamanho mínimo de folha* – especifica a quantidade mínima de registros classificados por cada folha da árvore;

- *ganho mínimo* – define o ganho mínimo que deve ser obtido (de acordo com o critério escolhido) para que seja realizada uma separação de um nodo;
- *profundidade máxima* – estabelece a altura máxima que a árvore pode alcançar;
- *confiança* – especifica o nível de confiança utilizado para o cálculo de erro pessimista na poda;
- *número de alternativas de pré-poda* – quantidade de nodos alternativos testados quando a pré-proda for evitar uma separação.

O conjunto de valores desses parâmetros utilizado em cada experimento foi definido após a realização de vários testes, por meio dos quais foram escolhidos os parâmetros que melhor atenderam os seguintes critérios:

- *acurácia*, *precisão* e *recall* do modelo de AD – quanto maiores esses valores, mensurados pela matriz de confusão, melhor avaliado é o modelo;
- tamanho e complexidade da AD – a AD deve apresentar tamanho e complexidade razoáveis, de modo a evitar a ocorrência de *overfitting* ou *underfitting*;
- regras de classificação da AD – as regras da AD devem ser avaliadas por um especialista em perícias digitais para determinar se são factíveis ou não.

O primeiro critério é objetivo e obtido com o uso de operadores específicos da ferramenta *RapidMiner*, enquanto a avaliação dos dois últimos critérios é subjetiva e dependente da percepção do avaliador nos testes empíricos realizados.

Até serem definidos os modelos apresentados nos experimentos a seguir, foram realizadas diversas rodadas de testes preliminares, com redução de dimensionalidade, mapeamento de classes, filtragem de exemplos, discretizações de valores e variações nos parâmetros dos algoritmos de indução de AD. A análise dos modelos induzidos nestes testes preliminares ajudou a aperfeiçoar as etapas de processamento, transformação e consolidação dos dados, que foram modificadas e resultaram em novas rodadas de testes do processo de MD e indução de modelos de AD.

6.1.1. Experimento 1

O primeiro experimento foi realizado com o intuito de se obter um modelo de AD que permita inferir regras para identificar os arquivos reconhecidos pela NSRL. Para atingir esse objetivo, foram utilizadas apenas duas classificações: “IC” para os arquivos identificados pela NSRL e

“R” para os demais (os quais foram considerados relevantes), conforme agrupamento de classes mostrado na Tabela 6.1.

Não foi realizada nenhuma discretização de valores e foi utilizado o método *holdout* com amostragem aleatória na proporção 70%-30% para definição dos conjuntos de treinamento e de teste, respectivamente, conforme exposto na Seção 5.3.2.

Tabela 6.1 - Mapeamento de classes utilizado no Experimento 1.

Classe original	Classe utilizada
R	R
PI	
I+	
I++	
I+++	
PT	
T	
IC	IC

Os parâmetros utilizados para o algoritmo de indução de AD estão expostos na Figura 6.1.

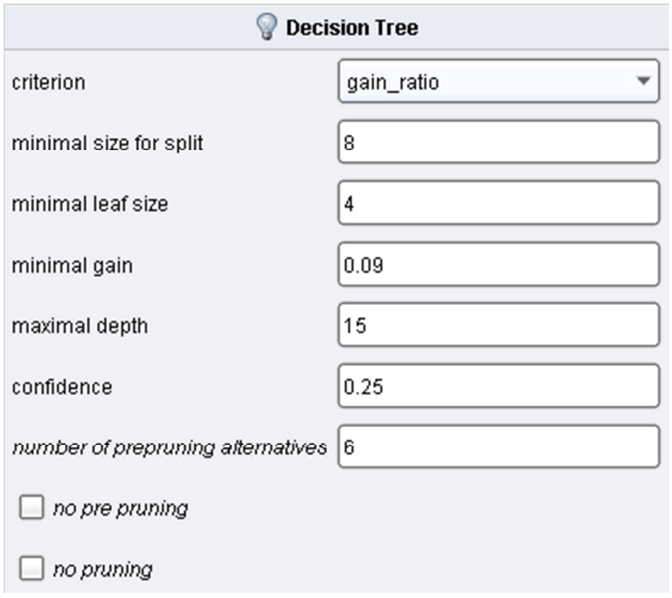

Figura 6.1 - Parâmetros do algoritmo de indução de AD usados no Experimento 1.

O modelo de AD induzido está ilustrado na Figura 6.2, onde também estão destacados em vermelho as classificações divergentes das que seriam normalmente esperadas. De acordo

com avaliação de especialistas, era de se esperar que os seguintes comportamentos fossem observados na implementação das regras com o uso dos atributos a seguir:

- *avg_percent_folder_files_filetype_expected* e *avg_percent_folder_files_extension_expected* – arquivos ignoráveis são provavelmente encontrados em pastas contendo arquivos com padrão uniforme para o tipo e extensão de arquivo (arquivos de ajuda, DLLs, executáveis, entre outros) e os tipos e extensões esperados para cada pasta principal deveriam seguir um padrão que resultaria na classificação "IC" quanto maior fosse o valor destes atributos;
- *avg_percent_folder_files_ignorable* – arquivos ignoráveis são provavelmente encontrados em pastas contendo outros arquivos ignoráveis. Quanto mais arquivos ignoráveis houver em uma pasta, maior seria a chance de que os demais também fossem ignoráveis;
- *percent_userfolders* – espera-se que arquivos presentes em pastas de usuários sejam relevantes.

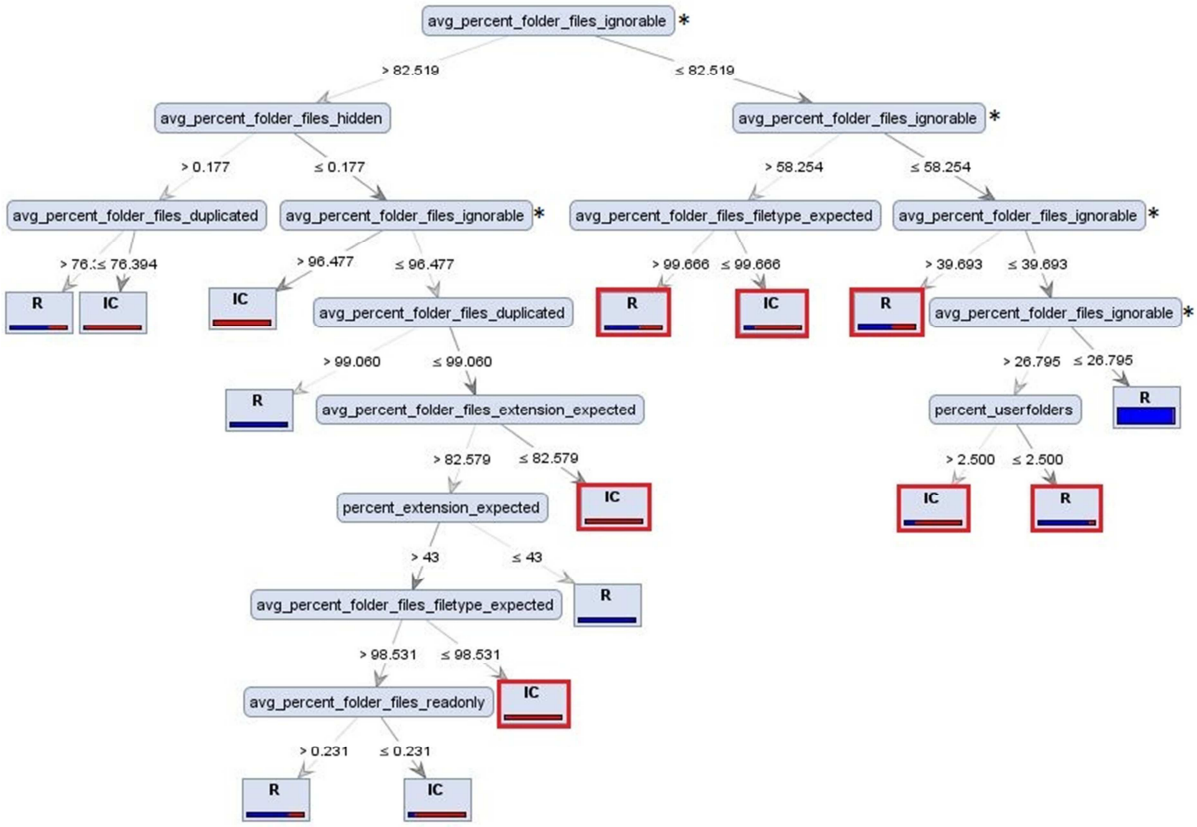

Figura 6.2 - Modelo de AD induzido no Experimento 1.

A partir dos resultados do Experimento 1 e de análises posteriores em uma amostra de registros da AC composta por arquivos identificados pela NSRL, foi possível perceber que vários desses arquivos ignoráveis conhecidos (“IC”) são arquivos que não apresentam um padrão definido de comportamento no sistema de arquivos. Eles estão presentes em diversos *softwares* distintos e com atributos muitas vezes diferentes – conforme pode ser visto na Tabela 6.2, em que é mostrado um exemplo de arquivo ignorável conhecido com atributos de nome e caminho variáveis no sistema de arquivos.

Entretanto, o objetivo do processo de MD apresentado neste trabalho é identificar arquivos ignoráveis em uma AC que apresentem um comportamento sistematicamente uniforme no sistema de arquivos, cujo padrão possa ser captado pelo algoritmo de indução de AD. Arquivos que não apresentam um padrão de comportamento uniforme não fazem parte do alvo deste trabalho – mesmo que sejam arquivos reconhecidamente ignoráveis – e não podem ser identificados com o método proposto, além de introduzirem ruídos que distorcem o modelo de AD induzido. Por esse motivo, os experimentos seguintes foram realizados sem esses registros, para tentar gerar um modelo que consiga captar as regras utilizadas por especialistas para classificar a relevância de um registro, sejam elas explícitas ou implícitas.

Tabela 6.2 - Exemplo de arquivo identificado pela NSRL sem padrão de comportamento no sistema de arquivos.

Nome do arquivo	Caminho completo
spacer.gif	\Arquivos de programas\HP\Digital Imaging\Skins\hp1\sc\img\
spacer.gif	\Arquivos de programas\Hewlett-Packard\Digital Imaging\bbfe\scan\img\
spacer.gif	\Program Files\HP\Digital Imaging\{68550918-63B5-4762-85CB-3C160AA4B213}\setup\hpqadobeui\images\
branco.gif	\Arquivos de programas\Positivo\Aurelio\manual\verbete\imagens\
branco.gif	\Arquivos de programas\Positivo\Aurelio\
shim[1].gif	\[orphan]\
teclado_spacer[1].gif	\[orphan]\
spacer.gif	\IBGE\aeb\images\images\
spacer.gif	\IBGE\images\
transp[3].gif	\WINDOWS\Temporary Internet Files\Content.IE5\KH8JAP8V\
spacer.gif	\WINDOWS\PCHEALTH\HELPCTR\Vendors\CN=Microsoft Corporation,L=Redmond,S=Washington,C=US\

Outra observação em relação ao modelo de AD obtido neste primeiro experimento refere-se às consequências da não realização de discretização prévia dos atributos númericos. Conforme destacado por Witten, Frank e Hall (2011): “*Enquanto um atributo nominal só pode ser testado uma vez em cada caminho a partir da raiz de uma árvore até uma folha, um atributo numérico pode ser testado várias vezes. Isso pode resultar em árvores que são*

confusas e difíceis de compreender porque os testes para cada atributo numérico não estão localizados juntos, mas podem estar espalhados ao longo do caminho". Esse fato pôde ser observado no modelo induzido, onde o atributo *avg_percent_folder_files_ignorable* (*), por exemplo, foi testado diversas vezes no galho direito da árvore. Portanto, para evitar os problemas apontados, os experimentos subsequentes realizados implementaram operadores para realizar a discretização prévia de todos os valores numéricos.

6.1.2. Experimento 2

O segundo experimento foi realizado com o intuito de se obter um modelo de AD que permita identificar os arquivos relevantes, os potencialmente ignoráveis e os potencialmente ignoráveis de cache de Internet classificados manualmente, sem utilizar nos conjuntos de treinamento e de teste os registros classificados como "IC". Para atingir esse objetivo, foi realizado um mapeamento de classes conforme exposto na Tabela 6.3.

Tabela 6.3 - Mapeamento de classes utilizado no Experimento 2.

Classe original	Classe utilizada
R	R
PI	I
I+	
I++	
I+++	
PT	T
T	
IC	–

Para definição dos conjuntos de treinamento e de teste foi utilizado o método *bootstrapping* com dez validações e percentual de amostragem de 100% para o conjunto de treinamento – a cada validação o método escolhe, com reposição, os itens que irão compor o conjunto de treinamento e os itens restantes farão parte do conjunto de teste.

Foram utilizados operadores para realizar a discretização prévia de valores de atributos numéricos, conforme apresentado na Tabela 6.4.

Tabela 6.4 - Discretizações de atributos numéricos realizadas no Experimento 2.

Atributo	**Nome da classe**	**Regra para discretização**	**Tipo do operador**
percent_system_folders, percent_user_folders, percent_internetcachefolders	0	= 0	Limites definidos pelo usuário
	1-30	> 0 e <= 30	
	30-70	> 30 e <= 70	
	70-99	> 70 e <= 99	
	100	> 99	
percent_distinct_evidences	2	<= 2	Limites definidos pelo usuário
	3-4	> 2 e <= 4	
	5-10	>4 e <= 10	
	11-15	> 10 e <= 15	
	15-*	> 15	
percent_distinct_cases	range1	< 43,75	*Binning*
	range2	>= 43,75 e < 62,5	
	range3	>= 62,5 e < 81,25	
	range4	>= 81,25	
percent_distinct_case_subjects	range1	< 55,33	*Binning*
	range2	>= 55,33 e <77,67	
	range3	>= 77,67	
avg_percent_folder_files_*, percent_max_*_occurrences, percent_*	0	= 0	Limites definidos pelo usuário
	0.1-10	> 0 e <= 10	
	10-50	> 10 e <= 50	
	50-80	> 50 e <= 80	
	80-100	> 80	

Após várias experimentações de parâmetros para o algoritmo de indução de AD, foram selecionados aqueles apresentados na Figura 6.3. Foi utilizado um operador de AD no *RapidMiner* baseado em teste de relevância dos atributos por *information gain ratio* (operador *Decision Tree (Weight-Based)* / *Weight by Information Gain Ratio*). Esse operador trabalha apenas com atributos nominais e, da forma como foi configurado, implementa o Algoritmo C4.5.

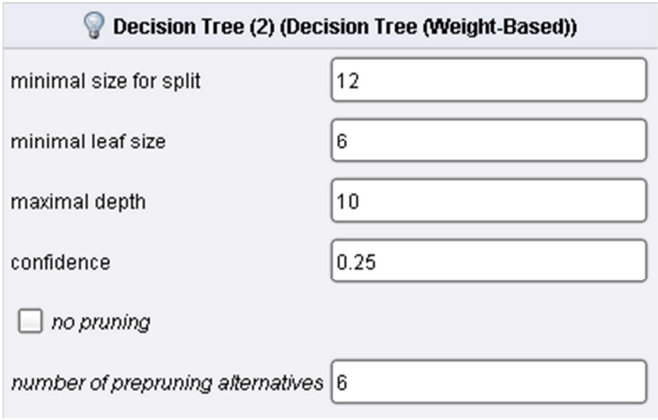

Figura 6.3 - Parâmetros do algoritmo de indução de AD utilizados no Experimento 2.

A AD obtida neste segundo experimento está ilustrada na Figura 6.4 e na Figura 6.5 (a divisão foi feita para facilitar a visualização). O modelo induzido separou as classificações de modo que as classes “I” ocorrem apenas na Figura 6.4 e as classes “T” aparecem apenas nos galhos exibidos na Figura 6.5 – as classes “R” estão espalhadas por toda a árvore. Isso deve-se ao fato de que o nodo raiz escolhido pelo algoritmo foi o atributo *percent_internetcachefolders* e todos os exemplos com valores diferentes de zero para esse atributo foram considerados *relevantes* ou *ignoráveis de cache de Internet*; enquanto aqueles com valor igual a zero são considerados *relevantes* ou *ignoráveis*.

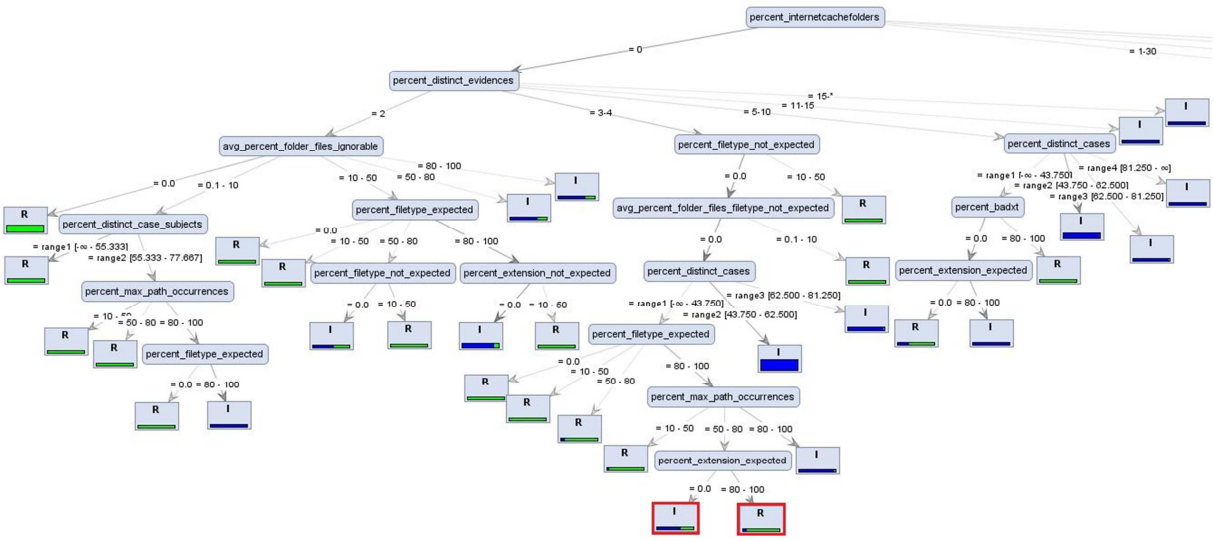

Figura 6.4 - AD obtida no Experimento 2 (galho superior esquerdo).

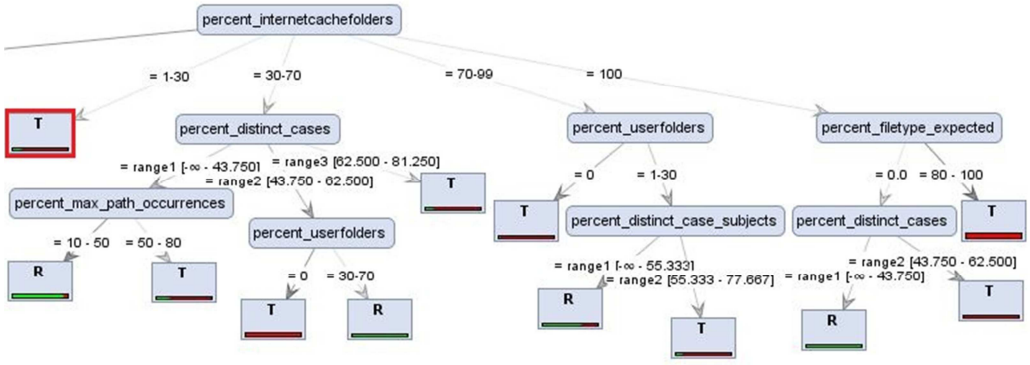

Figura 6.5 - AD obtida no Experimento 2 (galhos superiores direitos).

As regras de classificação expostas no modelo de AD induzido neste experimento foram consideradas viáveis e factíveis por especialistas que as analisaram, com exceção dos dois nodos destacados em vermelho na Figura 6.4 e do nodo destacado em vermelho na Figura 6.5. Os primeiros dois nodos implementam uma regra de verificação do atributo *percent_extension_expected* de maneira oposta à esperada por especialistas. As classificações com o uso desse atributo nesse caso específico ("I" e "R") estão invertidas em relação ao que seria previsto. O motivo pode ser a falta de especificação, na Tabela **`expected_folder_files_extensions`** de uma extensão esperada para alguma pasta do sistema de arquivos, o que ocasionou essa inconsistência. Já o nodo destacado em vermelho na Figura 6.5 classifica os exemplos como "T" simplesmente se eles forem encontrados em pastas de cache de Internet de 1% a 30% do total de suas ocorrências – o que representa uma regra muito simples e pouco factível. A razão para essa regra de classificação excessivamente simples pode ser a falta de uma amostra significativa de exemplos que se enquadrem nesse cenário. Por esse motivo, os exemplos classificados como "T" ou como "I" pelos nodos destacados em vermelho anteriormente descritos foram desconsiderados durante a montagem da BHP nos estudos de caso.

A matriz de confusão resultante deste segundo experimento está exposta na Figura 6.6. A *acurácia* do modelo foi de 98,25%, a *precisão* variou de 98,04% a 98,63% e o *recall* ficou entre 96,03% e 99,45%. Os significados dessas medidas foram apresentados na Seção 4.2.2.

accuracy: 98.25% +/- 0.23% (mikro: 98.25%)

	true I	true R	true T	class precision
pred. I	20506	404	7	98.04%
pred. R	113	11113	41	98.63%
pred. T	0	55	3181	98.30%
class recall	99.45%	96.03%	98.51%	

Figura 6.6 - Matriz de confusão do modelo de AD do Experimento 2.

O modelo de AD obtido neste segundo experimento foi utilizado para a realização do primeiro estudo de caso, detalhado na Seção 6.2.1.

6.1.3. Experimento 3

O terceiro experimento, de modo semelhante ao segundo, também teve o objetivo de obter um modelo de AD que permita identificar os arquivos relevantes, os potencialmente ignoráveis e os potencialmente ignoráveis de cache de Internet classificados manualmente, sem utilizar nos conjuntos de treinamento e de teste os registros classificados como "IC". Entretanto, foi utilizado um mapeamento de classes mais restritivo, em que as classes originais "PI" e "I+" foram mapeadas para a classe "R" para que a análise dos resultados fosse mais diversificada. Ou seja, apenas os arquivos considerados ignoráveis com maior grau de certeza ("I++" e "I+++") foram utilizados no processo de MD como exemplos de arquivos ignoráveis (classificação "I").

Tabela 6.5 - Mapeamento de classes utilizado no Experimento 2.

Classe original	Classe utilizada
R	R
PI	
I+	
I++	I
I+++	
PT	T
T	
IC	–

Foi utilizado o método de validação cruzada com dez iterações e amostragem aleatória para definição dos conjuntos de treinamento e de teste. Também foram utilizados os mesmos operadores de discretização de atributos numéricos e parâmetros para o algoritmo de indução usados no segundo experimento (Tabela 6.4 e Figura 6.3, respectivamente).

O modelo de AD induzido neste experimento está ilustrado na Figura 6.7 e na Figura 6.8. Os galhos superiores direitos são idênticos aos da AD obtida no Experimento 2, uma vez que não houve mudanças significativas nos parâmetros que afetassem essa parte da árvore. Mas a parte da AD relativa aos exemplos cuja verificação da regra do nodo raiz são referentes a "*percent_internetcachefolders* = 0" (Figura 6.7) foram completamente modificados em

relação ao segundo experimento. Conforme esperado, essas mudanças decorreram do diferente mapeamento de classes implementado. Nenhuma regra dessa parte da AD foi considerada inesperada por especialistas, sendo todas viáveis e factíveis.

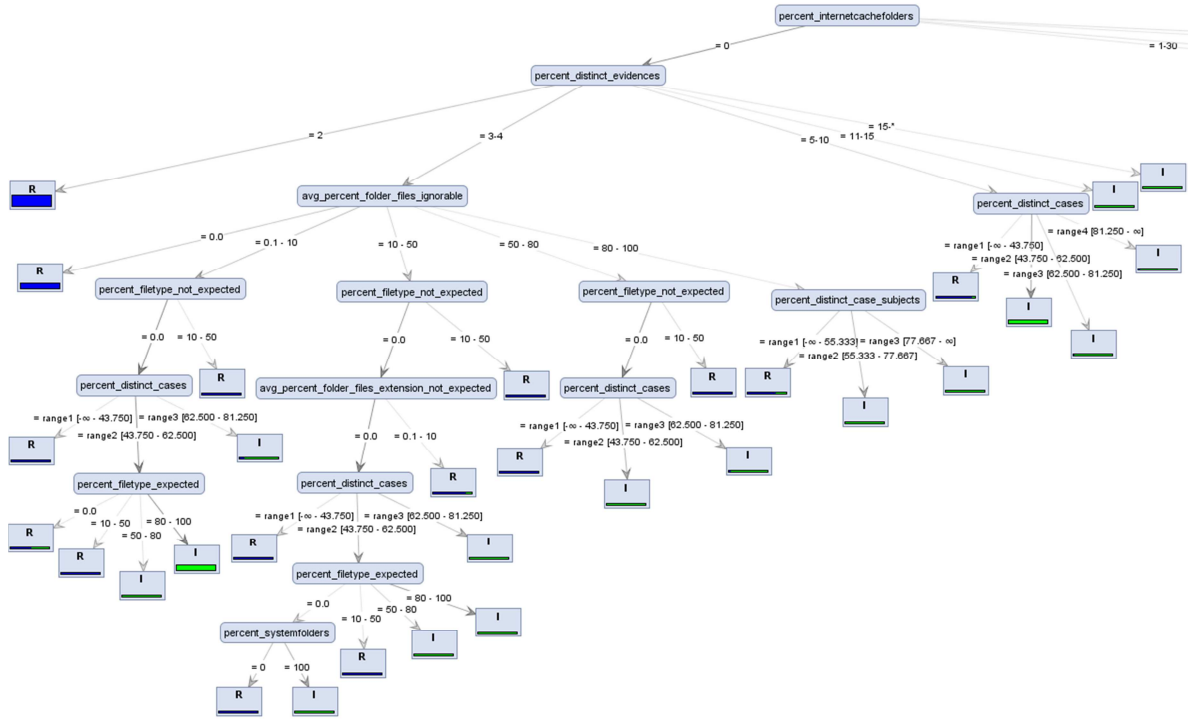

Figura 6.7 - AD obtida no Experimento 3 (galho superior esquerdo).

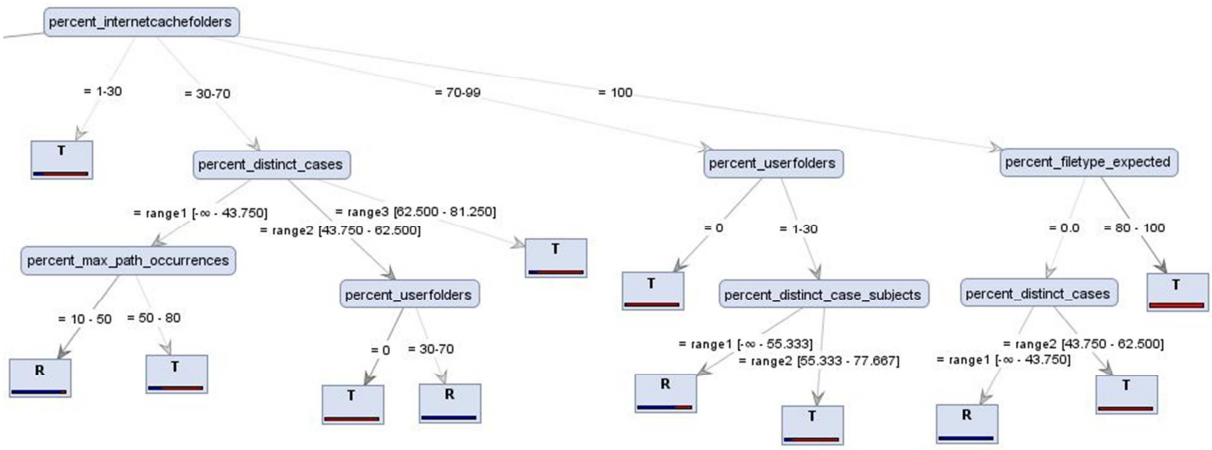

Figura 6.8 - AD obtida no Experimento 3 (galhos superiores direitos).

A matriz de confusão resultante deste terceiro experimento está exposta na Figura 6.9. A *acurácia* do modelo foi de 99,14%, a *precisão* variou de 98,63% a 99,38% e o *recall* ficou entre 98,52% e 99,46%.

accuracy: 99.14% +/- 0.35% (mikro: 99.14%)

	true R	true I	true T	class precision
pred. R	5143	19	13	99.38%
pred. I	39	3507	0	98.90%
pred. T	12	0	866	98.63%
class recall	99.02%	99.46%	98.52%	

Figura 6.9 - Matriz de confusão do modelo de AD do Experimento 3.

O modelo de AD obtido no Experimento 3 foi utilizado para a realização do segundo estudo de caso, detalhado na Seção 6.2.2.

6.2. ESTUDOS DE CASO

Os estudos de caso foram conduzidos por meio da aplicação dos modelos de AD obtidos nos Experimento 2 e 3 para identificar novos *hashes* de arquivos ignoráveis na AC e utilizá-los para montar a BHP. A BHP – formada por esses novos *hashes* de arquivos ignoráveis, acrescida de uma SCH retirada de BHAC tradicionais, conforme detalhado na Seção 5.4.2 – foi então aplicada a novos casos reais de perícia.

Os casos reais utilizados nos estudos de caso – que chamaremos de *amostra de teste* – foram escolhidos aleatoriamente entre computadores apreendidos no começo do ano de 2011 a serem analisados pelo Serviço de Perícias em Informática da Polícia Federal. Os detalhes dessa amostra estão expostos na Tabela 6.6. Os dados de 962.419 arquivos foram extraídos de nove computadores com o uso da ferramenta pericial *AccessData FTK*. Somente foram considerados os arquivos extraídos pelo FTK com valor de *hash* válido. Os computadores da amostra de teste faziam parte de três operações distintas, cada qual lidando com um assunto de investigação diferente. Os nomes reais das operações das quais cada computador fazia parte foram omitidos por questões de sigilo.

Tabela 6.6 - Informações sobre os computadores utilizados nos estudos de caso.

Computador	Operação	Assunto	Total de arquivos
A	X	Desvio de dinheiro público	54.770
B			71.570
C			197.874
D			95.118
E			189.867
F			80.394
G	Y	Crime financeiro	144.219
H			82.709
I	Z	Fraudes em licitações	45.898
Total			**962.419**

Para servir de comparação nos estudos de caso, foi utilizado o RDS 2.32 da NSRL para formar a BHAC. A quantidade de arquivos identificados pela BHAC em cada computador da amostra de teste pode ser vista na Tabela 6.7, na qual é possível perceber que o percentual de arquivos identificados em cada computador variou de 2,2% a 43,1% e que a média de identificação foi de aproximadamente 22,3% do total de arquivos.

Tabela 6.7 - Quantidade de arquivos identificados pelo RDS 2.32 da NSRL na amostra de teste.

Computador	Total de arquivos (A)	NSRL (B)	% NSRL (B)/(A)
A	54.770	**12.794**	23,4%
B	71.570	**19.394**	27,1%
C	197.874	**22.445**	11,3%
D	95.118	**31.814**	33,4%
E	189.867	**81.780**	43,1%
F	80.394	**11.938**	14,8%
G	144.219	**12.747**	8,8%
H	82.709	**20.693**	25,0%
I	45.898	**1.018**	2,2%
Total	**962.419**	**214.623**	**22,3%**

Como o computador “I” apresentou um percentual de identificação pela NSRL muito menor do que os demais computadores da AC, foi realizada uma verificação de seus atributos. Entretanto, não foi identificada nenhuma anormalidade ou falha. Pôde-se identificar alguns *softwares* com arquivos não identificados pela NSRL, como *Adobe Reader*, *HP Digital Imaging*, *Microsoft Works*, *VIVO Internet*, *Skype*, *Windows Live Bar*, *Windows Sidebar*, além

de vários arquivos relacionados a licenças de *softwares* e uma grande quantidade de documentos XML de configuração do SO *Windows* (notadamente arquivos com a extensão "manifest"). Essa variação de resultados de filtragem por BHACs não é incomum em análises periciais, onde os computadores examinados costumam apresentar diferenças consideráveis de configurações de uso e *softwares* instalados.

Também foi feita a verificação do desempenho de identificação na amostra de teste obtido por uma SCH retirada da BHAC. Essa SCH é composta (i) por conjuntos de *hashes* efetivamente utilizados e (ii) pelos *hashes* recentemente adicionados à BHAC. Conforme descrito na Seção 5.4.2, a SCH do protótipo foi composta por um total de 9.869.141 *hashes* distintos, o equivalente a cerca de 51,25% do total de registros do RDS 2.32 da NSRL. O desempenho de identificação da SCH do protótipo está exposto na Tabela 6.8.

Tabela 6.8 - Quantidade de arquivos identificados pela SCH na amostra de teste.

Computador	Total de arquivos (A)	Conjuntos de *hashes* efetivamente usados	*Hashes* mais recentes (*)	Total SCH (B)	% SCH (B)/(A)
A	54.770	12.578	5	**12.578**	23,0%
B	71.570	19.195	28	**19.195**	26,8%
C	197.874	22.443	29	**22.443**	11,3%
D	95.118	31.740	1.665	**31.740**	33,4%
E	189.867	77.018	1.666	**77.018**	40,6%
F	80.394	11.933	10	**11.933**	14,8%
G	144.219	12.702	23	**12.702**	8,8%
H	82.709	20.690	17	**20.690**	25,0%
I	45.898	1.018	282	**1.018**	2,2%
Total	**962.419**	**209.317**	**3.725**	**209.317**	**21,7%**

(*) Todos os hashes identificados pelos hashes mais recentes já haviam sido identificados pelos conjuntos de hashes efetivamente usados.

Percebe-se, comparando os resultados da Tabela 6.7 e da Tabela 6.8, que a quantidade de arquivos identificados pelo RDS 2.32 completo da NSRL foi muito próxima da quantidade identificada com o uso da SCH do protótipo. O desempenho total de identificação diminuiu de 22,3% para 21,7% e o total de arquivos identificados caiu de 214.623 para 209.317. Ou seja, a seleção composta por cerca de metade dos hashes da NSRL resultou em uma queda de apenas 2,47% no número total de arquivos identificados.

6.2.1. Estudo de Caso 1

O primeiro estudo de caso foi realizado com o objetivo de aplicar o modelo de AD obtido no Experimento 2 sobre todos os registros da tabela **`final_instances`** ainda não identificados pelas BHAC tradicionais, de modo a identificar novos *hashes* de arquivos ignoráveis para serem utilizados na BHP. Como resultado, foram identificados (i) 93.832 *hashes* de arquivos classificados como ignoráveis e (ii) 16.557 *hashes* de arquivos classificados como temporários de Internet ignoráveis. A BHP foi ainda acrescida da SCH do protótipo.

Conforme ilustrado na Figura 6.10, a BHP deste estudo de caso – identificada como *BHP 1* – foi então composta por 9.979.530 *hashes* únicos, correspondentes a aproximadamente 51,83% do total de *hashes* únicos do RDS 2.32 da NSRL.

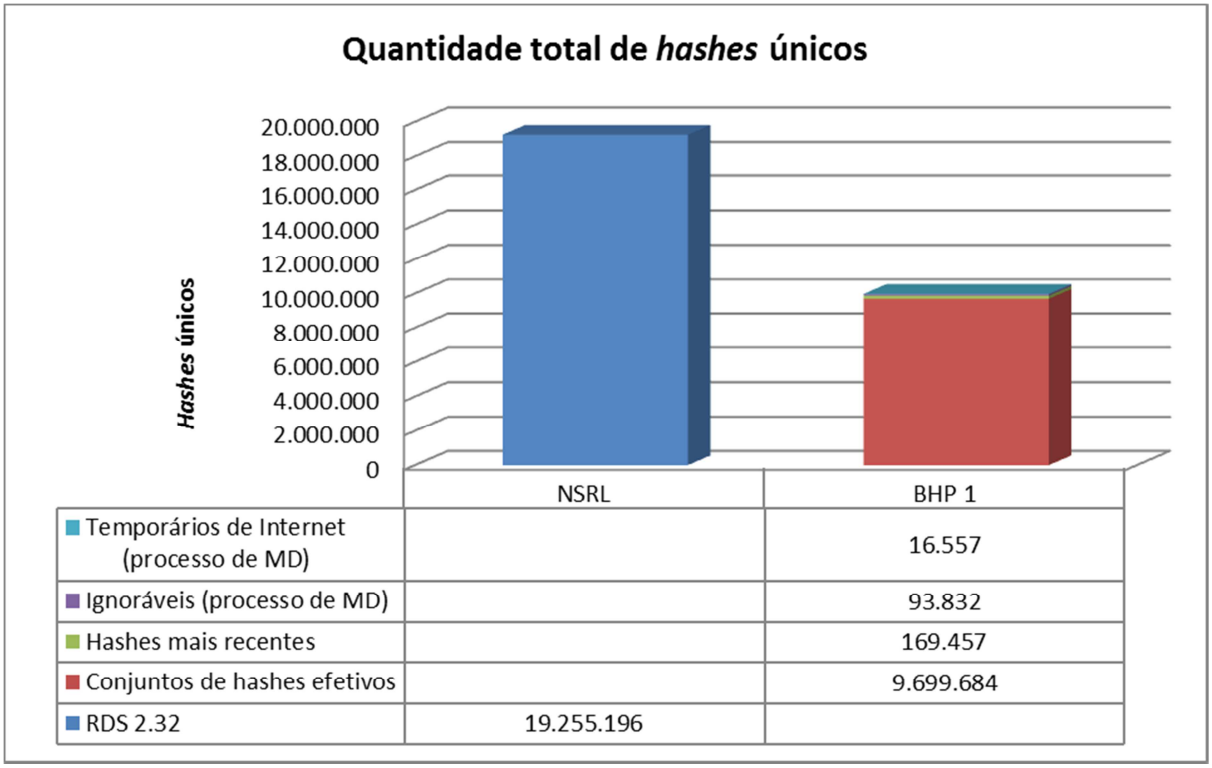

Figura 6.10 - Composição da BHP 1 e da BHAC do Estudo de Caso 1.

Para avaliar seu potencial de filtragem, a BHP 1 foi utilizada para identificar arquivos ignoráveis na amostra de teste. O resultado está detalhado na Tabela 6.9, onde pode-se verificar que o percentual de arquivos identificados em cada computador da amostra variou de um mínimo de 13,3% a um máximo de 51,4% e que a média de identificação foi de cerca de 29,1% do total de arquivos.

Tabela 6.9 - Quantidade de arquivos identificados pela BHP 1 na amostra de teste.

Computador	Total de arquivos (A)	SCH	Ignoráveis (processo de MD)	Temporários de Internet (processo de MD)	BHP 1 (B)	% BHP 1 (B)/(A)
A	54.770	12.578	309	1.206	**14.093**	25,7%
B	71.570	19.195	4.163	1.210	**24.568**	34,3%
C	197.874	22.443	2.072	1.889	**26.404**	13,3%
D	95.118	31.740	13.090	1.074	**45.904**	48,3%
E	189.867	77.018	20.142	507	**97.667**	51,4%
F	80.394	11.933	4.369	307	**16.609**	20,7%
G	144.219	12.702	7.064	1.071	**20.837**	14,4%
H	82.709	20.690	5.762	783	**27.235**	32,9%
I	45.898	1.018	5.687	464	**7.169**	15,6%
Total	**962.419**	**209.317**	**62.658**	**8.511**	**280.486**	**29,1%**

As informações dos arquivos adicionais identificados como ignoráveis pelo processo de MD foram avaliadas para determinar a sua real potencialidade como arquivos irrelevantes para exames periciais. Através de análise visual das informações dos arquivos, não foi identificado nenhum classificado erroneamente como *ignorável*. Uma amostra dos arquivos identificados está disponível no Apêndice D.

A Tabela 6.10 apresenta a quantidade de arquivos identificados como *ignoráveis* (com classificação “I”) na amostra de teste pela BHP 1 usando os *hashes* classificados como *ignoráveis* pelo processo de MD, distribuídos de acordo com o percentual de evidências distintas na AC nos quais tais *hashes* foram encontrados. É possível perceber que os *hashes* mais efetivos na filtragem são aqueles encontrados de 3% a 4% das evidências da AC. Portanto, se fosse definido um percentual mínimo de 3% de ocorrência em evidências distintas para que os *hashes* fossem incluídos na BHP, o impacto na efetividade de filtragem seria mínimo. No entanto, se esse percentual fosse elevado para mais de 3%, o resultado da filtragem de arquivos seria mais fortemente prejudicado. Por exemplo, se fosse estabelecido um percentual mínimo de 5% de ocorrência na AC para os *hashes* da BHP, a quantidade de arquivos da amostra de teste identificados pela BHP 1 cairia para 247.134 (equivalentes a 25,7% do total de arquivos), o que ainda representa uma melhoria considerável em relação aos 214.623 arquivos identificados pelo RDS 2.32 da NSRL (conforme apresentado na Tabela 6.7).

Tabela 6.10 – Quantidade de arquivos identificados na amostra de teste utilizando os *hashes* classificados como *ignoráveis* pelo processo de MD – distribuídos pelo percentual de evidências distintas na AC.

	Distribuição por percentual de evidências distintas na AC dos *hashes* classificados como *ignoráveis* pelo processo de MD						
Computador	**2%**	**3%**	**4%**	**5%**	**6%**	**7%**	**>= 8%**
A	1	43	59	37	22	22	125
B	12	501	366	306	176	794	2.008
C	5	93	167	218	93	98	1.398
D	52	4.892	5.613	392	189	70	1.882
E	78	9.013	6.863	946	54	464	2.383
F	2	56	227	82	395	40	3.908
G	60	528	409	328	372	419	4.948
H	75	542	499	410	207	490	3.539
I	2	724	2.470	331	802	238	1.120
Total	**287**	**16.392**	**16.673**	**3.050**	**2.310**	**2.635**	**21.311**

A ocorrência em pelo menos dois computadores distintos é a exigência mínima imposta para que um arquivo seja avaliado como potencialmente ignorável. Entretanto, conforme pode ser visto na Tabela 6.10, as restrições impostas pelas regras da AD para que os *hashes* que ocorrem em apenas 2% da AC sejam classificados como ignoráveis fazem com que uma pequena quantidade de arquivos seja identificada por esses *hashes* na amostra de teste.

6.2.2. Estudo de Caso 2

O segundo estudo de caso teve como objetivo a aplicação do modelo de AD resultante do Experimento 3 sobre os registros da tabela **`final_instances`** que não haviam sido identificados pelas BHAC tradicionais. Como resultado, foram identificados novos *hashes* de arquivos ignoráveis para serem utilizados na BHP. Por meio desse processo foram identificados (i) 26.947 *hashes* de arquivos classificados como ignoráveis e (ii) 16.557 *hashes* de arquivos classificados como temporários de Internet ignoráveis. Conforme esperado, houve uma redução de cerca de 71,28% do total de *hashes* identificados como ignoráveis em relação àqueles identificados no primeiro estudo de caso. Esse resultado decorreu do maior rigor utilizado no Experimento 3 para classificar um arquivo como ignorável. A BHP foi também formada pela SCH do protótipo.

Conforme ilustrado na Figura 6.11, a BHP deste segundo estudo de caso – identificada como *BHP 2* – foi então composta por 9.912.645 *hashes* únicos, correspondentes a aproximadamente 51,48% do total de *hashes* únicos do RDS 2.32 da NSRL.

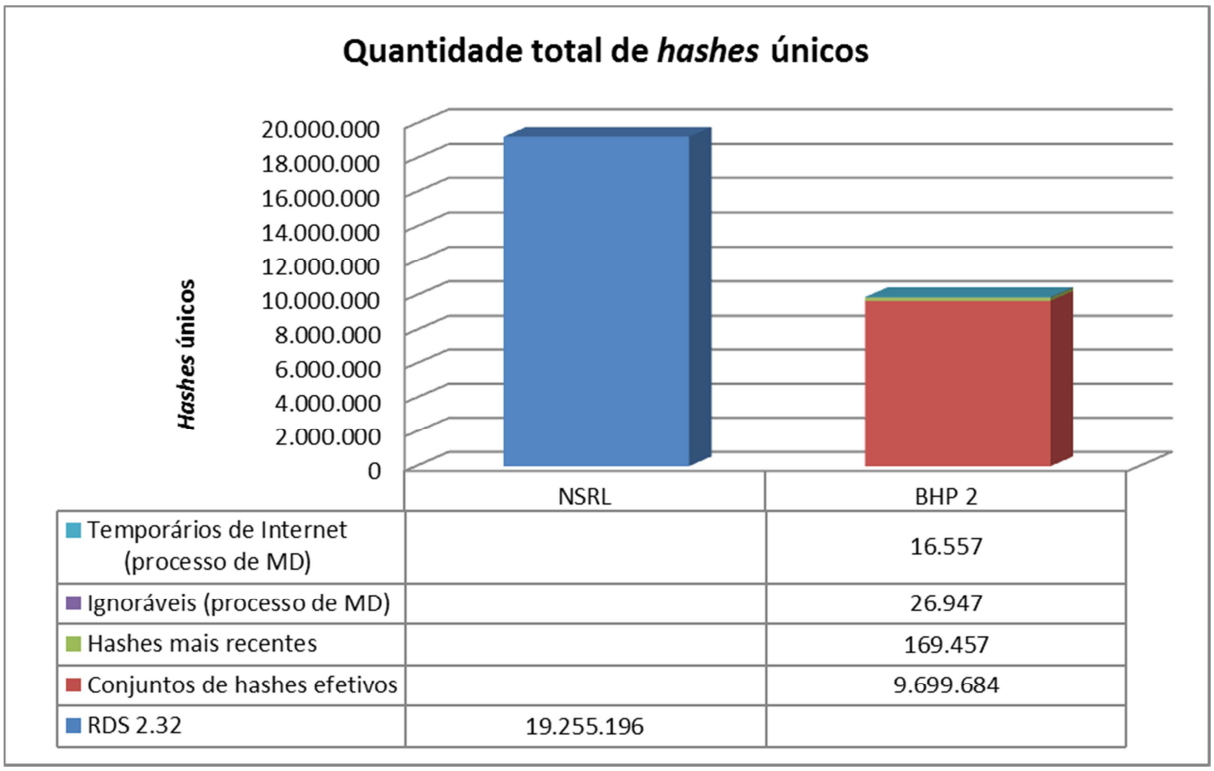

Figura 6.11 - Composição da BHP 2 e da BHAC do Estudo de Caso 2.

De forma semelhante à realizada no primeiro estudo de caso, a BHP 2 foi submetida a um teste de desempenho de filtragem dos arquivos da amostra de teste. Conforme esperado, o resultado foi inferior ao obtido no primeiro estudo de caso, mas – mesmo com o uso de uma quantidade significativamente menor de *hashes* ignoráveis oriundos do processo de MD – ainda assim foram obtidos índices de filtragem significativos. O resultado está exposto na Tabela 6.11, sendo que o percentual de arquivos identificados em cada item da amostra teve variação entre 8,6% e 43,1% e o percentual médio de identificação foi de aproximadamente 25,7% do total de arquivos.

Tabela 6.11 - Quantidade de arquivos identificados pela BHP 2 na amostra de teste.

Computador	Total de arquivos (A)	SCH	Ignoráveis (processo de MD)	Temporários de Internet (processo de MD)	BHP 2 (B)	% BHP 2 (B)/(A)
A	54.770	12.578	170	1.206	**13.954**	25,5%
B	71.570	19.195	3.297	1.210	**23.702**	33,1%
C	197.874	22.443	1.779	1.889	**26.111**	13,2%
D	95.118	31.740	2.617	1.074	**35.431**	37,2%
E	189.867	77.018	4.241	507	**81.766**	43,1%
F	80.394	11.933	4.076	307	**16.316**	20,3%
G	144.219	12.702	6.054	1.071	**19.827**	13,7%
H	82.709	20.690	4.667	783	**26.140**	31,6%
I	45.898	1.018	2.477	464	**3.959**	8,6%
Total	**962.419**	**209.317**	**29.378**	**8.511**	**247.206**	**25,7%**

6.3. ANÁLISE DE RESULTADOS

De posse dos dados sobre os desempenhos de filtragem de arquivos na amostra de teste obtidos pela BHP 1 (Estudo de Caso 1) e BHP 2 (Estudo de Caso 2), foi realizada uma análise dos resultados em comparação com o desempenho de filtragem obtido com o uso do RDS 2.32 da NSRL. A Tabela 6.12 apresenta os resultados desse estudo comparativo, onde é possível perceber que o ganho total de filtragem da BHP 1 em relação à NSRL foi de cerca de 30,7%; enquanto o ganho da BHP 2 foi de aproximadamente 15,2%. Os ganhos mínimos e máximos por computador obtidos pela BHP 1 em relação à NSRL foram de 10,2% e 604,2%; enquanto a BHP 2 obteve ganhos entre 0,0% e 288,9%.

Os resultados obtidos, ilustrados também na Figura 6.12, mostram que houve um ganho significativo de filtragem, principalmente se levarmos em consideração que tanto a BHP 1 quanto a BHP 2 possuem cerca de metade do número de *hashes* do RDS 2.32 da NSRL. Ou seja, com a utilização da solução proposta foram obtidos melhores resultados de filtragem de arquivos com um menor custo computacional.

Tabela 6.12 - Comparação de resultados de filtragem de arquivos com o uso da BHP 1, BHP 2 e o RDS 2.32 da NSRL.

Computador	Total de arquivos	% NSRL (A)	% BHP 1 (B)	Ganho BHP 1 (B)/(A)	% BHP 2 (C)	Ganho BHP 2 (C)/(A)
A	54.770	23,4%	25,7%	10,2%	25,5%	9,1%
B	71.570	27,1%	34,3%	26,7%	33,1%	22,2%
C	197.874	11,3%	13,3%	17,6%	13,2%	16,3%
D	95.118	33,4%	48,3%	44,3%	37,2%	11,4%
E	189.867	43,1%	51,4%	19,4%	43,1%	0,0%
F	80.394	14,8%	20,7%	39,1%	20,3%	36,7%
G	144.219	8,8%	14,4%	63,5%	13,7%	55,5%
H	82.709	25,0%	32,9%	31,6%	31,6%	26,3%
I	45.898	2,2%	15,6%	604,2%	8,6%	288,9%
Total	**962.419**	**22,3%**	**29,1%**	**30,7%**	**25,7%**	**15,2%**

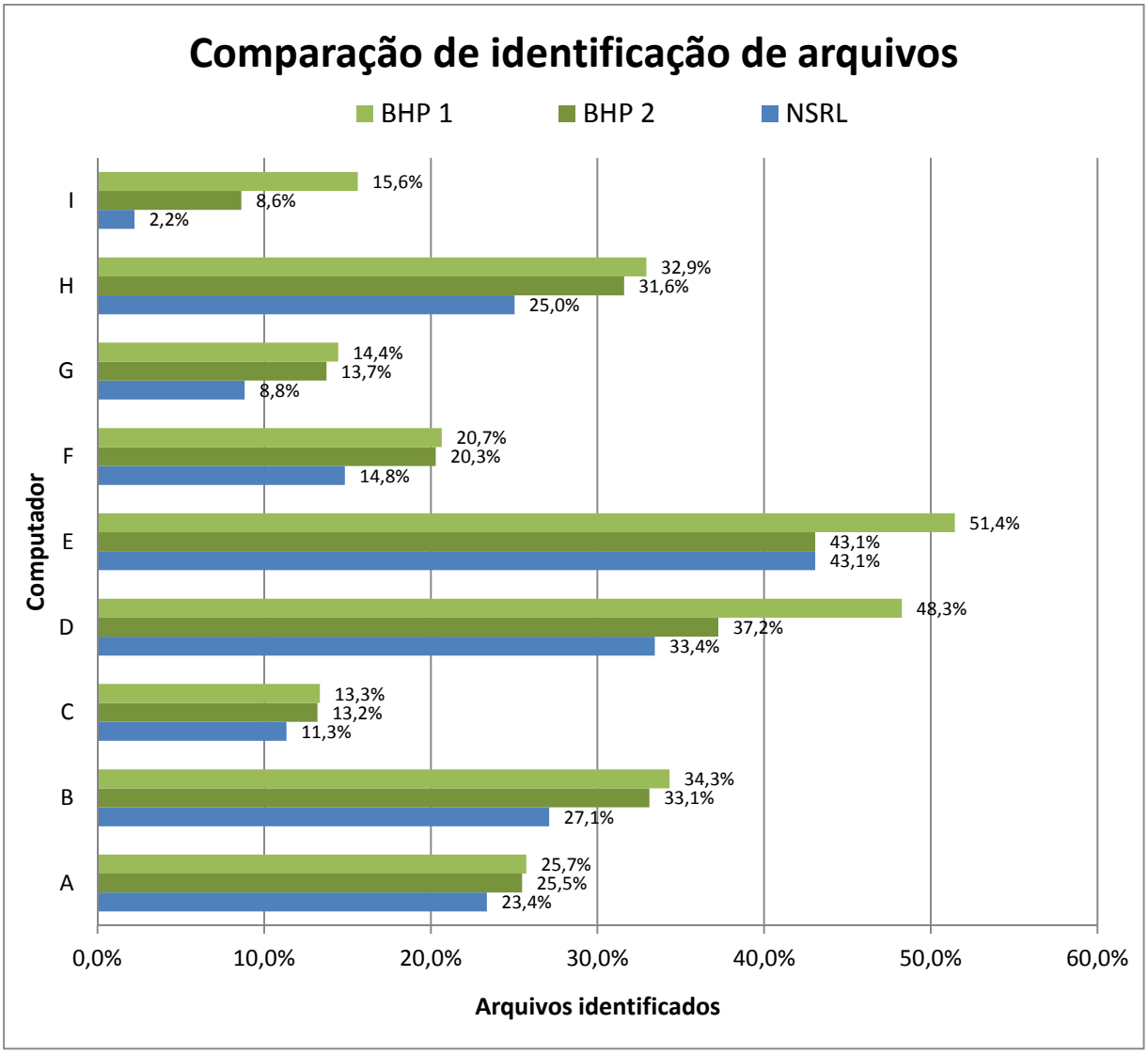

Figura 6.12 - Comparação dos resultados de filtragem de arquivos com o uso da BHP 1, BHP 2 e RDS 2.32 da NSRL.

Os experimentos realizados com o uso do protótipo buscaram avaliar a viabilidade da solução proposta, identificando os *hashes* de arquivos que, de acordo com as hipóteses utilizadas, possuem maior potencial de filtragem de arquivos ignoráveis. Esses *hashes* foram selecionados para compor a BHP de forma a se obter uma maior efetividade de filtragem a um menor custo computacional.

As BHPs dos experimentos foram utilizadas em estudos de caso compostos por computadores oriundos de exames periciais reais da PF e os resultados de filtragem foram comparados com os obtidos por meio do uso da NSRL, a mais conhecida e utilizada BHAC disponível. Os resultados obtidos apresentaram melhorias substanciais na quantidade de arquivos identificados – mesmo utilizando um número menor de *hashes* na BHP. As conclusões a respeito do trabalho realizado e as propostas de melhorias e trabalhos futuros serão apresentados no Capítulo 7.

7. CONCLUSÕES

Este trabalho apresentou uma nova abordagem para a criação, manutenção e uso de bases de *hashes* para realizar o processo de filtragem automática de arquivos irrelevantes em exames periciais. A solução proposta envolve a implementação de um *framework* para realizar a montagem de uma base de dados formada por (i) metadados de arquivos extraídos de uma AC, (ii) informações do SISCRIM sobre evidências e casos de análise pericial, (iii) conjuntos de *hashes* obtidos de BHACs diversas e (iv) informações auxiliares obtidas de especialistas em Computação Forense a respeito do comportamento padrão esperado de organização do sistema de arquivos em SO atuais. Por ter a função de auxiliar o processo de identificação e separação de arquivos sem interesse para exames periciais, essa base de dados foi denominada Base Joio.

A Base Joio é utilizada para a implementação de um processo de MD que utiliza algoritmos de aprendizagem baseados em AD com o objetivo de identificar um modelo de classificação a respeito da relevância de arquivos para exames periciais. Esse modelo de classificação é utilizado para identificar novos arquivos ignoráveis na AC para compor a BHP. A BHP é formada, ainda, por conjuntos de *hashes* de BHAC tradicionais selecionados com o auxílio dos dados da Base Joio.

O objetivo deste trabalho foi melhorar a eficácia e diminuir o custo computacional do processo de filtragem automática de arquivos irrelevantes em exames periciais. A identificação desses arquivos evita um esforço desnecessário por parte do perito durante o processo de análise de conteúdo e também reduz a quantidade de dados a serem processados em procedimentos periciais automatizados ou semi-automatizados realizados por ferramentas forenses.

A solução apresentada pode ser utilizada tanto na fase de análise pericial realizada em laboratório quanto para facilitar o trabalho de campo durante o procedimento de triagem de computadores e mídias de armazenamento computacional a serem apreendidos.

O protótipo implementado utilizou dados de arquivos de uma AC obtida de casos reais de investigação para montar a BHP. Duas BHPs distintas, com diferentes níveis para os critérios de relevância dos arquivos, foram criadas e testadas para identificar arquivos ignoráveis em uma amostra de teste formada também por computadores oriundos de investigações reais. Os resultados mostraram que as BHPs criadas obtiveram uma melhoria de desempenho de 15,2% e 30,7% em relação ao RDS 2.32 da NSRL na filtragem dos arquivos, mesmo contendo apenas cerca de metade de sua quantidade de *hashes*.

A maneira tradicional de criar e manter BHACs poderia ser mais eficaz, especialmente em países cujo idioma não seja o inglês. Este trabalho mostra que há outras alternativas para identificar arquivos irrelevantes que dificultam o trabalho de análise pericial. Os resultados obtidos demonstraram boas perspectivas para a melhoria do processo pericial de filtragem de arquivos, resultando em um grande potencial para auxiliar o examinador forense a realizar uma análise mais rápida e confiável.

7.1. TRABALHOS FUTUROS

Existem diversas possibilidades de trabalhos futuros e oportunidades para aperfeiçoar a solução proposta, assim como oportunidades de aplicação na área de Computação Forense.

7.1.1. Melhorias no modelo

O processo de MD obteve resultados promissores, com a indução de ADs que implementam regras factíveis de classificação. Mas esse processo pode ainda ser melhor desenvolvido e aperfeiçoado. Por exemplo, um percentual mínimo de repetições de um arquivo em mídias distintas para que ele seja considerado ignorável pode ser deixado como parâmetro opcional para o usuário, possibilitando um ajuste na classificação de arquivos com maior ou menor grau de confiança quanto à sua relevância. Outras ações que podem representar um melhor ajuste na classificação feita pelo modelo de AD são:

- aperfeiçoar a validação dos resultados da classificação através da inspeção por vários especialistas;
- realizar mais testes de parametrização do processo de MD, ampliando os estudos de análise exploratória dos dados;
- aperfeiçoar a preparação dos dados, com a inclusão de novas ideias obtidas a partir do conhecimento de especialistas.

A estrutura da Base Joio também pode ser utilizada para armazenar valores de *hash* de similaridade, por meio dos quais arquivos com conteúdo parcialmente semelhantes podem ser identificados. Conforme detalhado na Seção 3.4, propostas de *hashing* de similaridade vêm sendo apresentadas recentemente para uso em computação forense e tanto a NSRL quanto ferramentas forenses vêm implementando funcionalidades relacionadas. Sua aplicação está voltada principalmente para identificação de arquivos ou fragmentos de arquivos relacionados à prática de crimes (classificação do tipo *alerta* em BHACs).

Em investigações criminais é usual haver equipes especializadas em tipos de crimes específicos (financeiros, abuso sexual de criança ou adolescente, ataques no ciberespaço, entre outros). A análise pericial relacionada a cada um desses tipos de investigação também possui características específicas. A filtragem de arquivos é um processo que poderia ser customizado para melhor se adequar às especificidades de cada caso. Por meio da Base Joio é possível desenvolver uma solução para classificar os conjuntos de *hashes* que compõem a BHAC consolidada quanto ao seu interesse para cada tipo de investigação. Dessa forma, arquivos que compõem um aplicativo antivírus podem ser classificados como *ignorável* para investigações contábeis, mas serão *alerta* para investigações de *malware* e, de maneira oposta, arquivos de instalação de programas da Receita Federal do Brasil podem ser classificados como *alerta* para investigações contábeis e como *ignorável* para investigações de *malware*.

7.1.2. Melhorias no protótipo

O protótipo implementado foi definido com foco em arquivos ignoráveis. Entretanto, informações a respeito de arquivos do tipo *alerta* podem também auxiliar o processo de identificação da relevância de arquivos. Por esse motivo, a utilização de bases de *hashes* de arquivos *alerta* pode ajudar a aperfeiçoar o processo de MD.

A qualidade dos resultados do processo de MD depende diretamente dos dados da AC, razão pela qual devem ser buscadas formas para aumentar seu tamanho e representatividade. Também devem ser tomados cuidados para evitar a inclusão de dados inconsistentes na AC, como, por exemplo, utilizar programas antivírus para garantir que os arquivos que irão compor a AC não são relacionados a *malwares* que possam ser confundidos com arquivos ignoráveis.

7.1.3. Aplicações adicionais

As ferramentas forenses podem também implementar novos níveis de relevância para os arquivos identificados a partir de BHACs, ao invés de simplesmente identificá-los como *ignorável* ou *alerta*. Para cada nível, diferentes ações podem estar disponíveis. Arquivos identificados por BHACs tradicionais como *ignorável* podem ser diretamente excluídos da análise pericial, enquanto arquivos identificados por métodos alternativos – como o proposto neste trabalho – podem ser simplesmente marcados como *possivelmente ignoráveis,* ajudando o examinador forense a encontrar mais rapidamente os arquivos de interesse para o caso em análise.

As classificações propostas pelo processo de MD podem ser utilizadas para auxiliar sistemas mais sofisticados de suporte a decisão voltados para a identificação da potencial relevância de arquivos em análises forenses, como, por exemplo, o sistema proposto por Hoelz, Ralha e Geeverghese (2009).

Outros estudos de descoberta de conhecimento podem ser desenvolvidos futuramente aproveitando as informações armazenadas na Base Joio, uma vez que ela pode ser utilizada como uma base de memória de casos analisados. Esses estudos podem estar relacionados com a descoberta de padrão de comportamento de usuários e o *modus operandi* de práticas criminosas. As informações e conhecimentos adquiridos podem oferecer subsídios para a implementação de diversas soluções na área de Computação Forense, podendo ser citado o trabalho sobre análise *live* com o uso de Raciocínio Baseado em Casos apresentado por Hoelz, Ralha e Mesquita (2011).

Além disso, a estrutura do *framework* proposto pode ser utilizada para alimentar uma base de dados com metadados de evidências relacionadas a um determinado caso de investigação e, assim, fornecer meios para que seja possível encontrar relacionamentos entre arquivos presentes nessas evidências que auxiliem a elucidação da prática criminosa.

7.1.4. Aspectos Finais

A evolução do uso de sistemas computacionais e o aumento da capacidade de armazenamento das mídias digitais exigem soluções efetivas por parte da comunidade de Computação Forense, sob o risco de que as perícias digitais, em meio a um universo tão grande de informações a serem analisadas, sejam inviabilizadas em um futuro próximo devido à incapacidade dos peritos de processar tanta informação. Por esse motivo, procedimentos excessivamente rigorosos historicamente utilizados na área de Computação Forense precisam ser revistos e atualizados, conforme destacado na Seção 2.3.

A construção de bases de *hashes* com a finalidade de filtragem de arquivos em exames periciais ainda tem sido realizada, até hoje, apenas através da extração de *hashes* de arquivos diretamente dos pacotes de instalação dos softwares correspondentes, seguindo a visão purista de que a base de *hashes* deve ser composta apenas por arquivos que possam ser rastreados e cuja fonte possa ser verificada. Entretanto, conforme verificado nos experimentos realizados, uma grande quantidade de arquivos que são encontrados em vários computadores não relacionados entre si e que não apresentam nenhuma característica de interesse para exames periciais não são filtrados utilizando as BHACs tradicionais. Além disso, são diversos os cenários de exames periciais em mídias de armazenamento e o perito deve ter a capacidade de

avaliar cada um desses cenários e identificar em quais situações o uso de *hashes* de arquivos não diretamente rastreáveis é adequado, ou mesmo utilizar a base de *hashes* não como uma ferramenta para exclusão dos arquivos do caso em análise, mas sim para fornecer um indicativo, uma sugestão, a respeito da relevância dos arquivos.

A centralização de dados sobre casos analisados e *hashes* de arquivos conhecidos em uma base única apresenta claros benefícios de padronização e disponibilidade de informações para os peritos que atuam na área de informática. Além disso, segundo pesquisa realizada e revisão do artigo apresentado à *IFIP 2012 WG 11.9* (Ruback, Hoelz e Ralha, 2012), a solução proposta neste trabalho é inovadora na área de Computação Forense e, conforme detalhado neste capítulo, há várias possibilidades de melhorias a serem implementadas no modelo proposto, visando aumentar a qualidade e confiabilidade dos resultados obtidos e acrescentar novas funcionalidades que possam contribuir para o aperfeiçoamento da análise pericial.

REFERÊNCIAS BIBLIOGRÁFICAS

Anson, S. e Bunting, S. (2007). *Mastering Windows Network Forensics and Investigation* (1ª ed., p. 552). Sybex. ISBN 978-0470097625.

Barbetta, P. A. (2006). Técnicas de Amostragem. *Estatística Aplicada às Ciências Sociais* (6ª ed.). Editora da UFSC. ISBN: 978-8532803962.

Beebe, N. e Clark, J. (2005). Dealing with Terabyte Data Sets in Digital Investigations. In M. Pollitt & S. Shenoi (Eds.), *Advances in Digital Forensics* (Vol. 194, pp. 3-16). Boston: Springer. doi: 10.1007/0-387-31163-7.

Beebe, N. (2009). Digital Forensic Research : The Good, The Bad and The Unaddressed. *Advances in Digital Forensics* (pp. 17-36). Springer. doi: 10.1007/978-3-642-04155-6_2.

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, *13*(7), 422-426. ACM. doi:10.1145/362686.362692

Boer, B. D. e Bosselaers, A. (1993). Collisions for the compression function of MD5. Springer.

Breiman, L.; Friedman, J.; Stone, C. J. e Olshen, R. A. (1984). *Classification and Regression Trees* (p. 368). Chapman and Hall/CRC.

Bunting, S. (2007). *EnCase Computer Forensics - The Official EnCE: EnCase Certified Examiner Study Guide* (2ª ed., p. 648). Sybex. ISBN 978-0470181454.

Carrier, B. (2005). *File System Forensic Analysis* (1ª ed., p. 600). Addison-Wesley. ISBN 978-0321268174.

Carroll, O. L.; Brannon, S. K. e Song, T. (2008). Computer Forensics: Digital Forensic Analysis Methodology. *The United States Attorneys' Bulletin*, *56*, 1-8. Disponível em: http://www.justice.gov/usao/eousa/foia_reading_room/usab5601.pdf. Acesso: 08 nov. 2011.

Carvey, H. (2007). *Windows Forensic Analysis*. (D. Kleiman, Ed.) (p. 416). Syngress. ISBN 978-1597491563.

Damasceno, C. T. M.; Zacca, J. J. e Nogueira, J. H. M. (2006). *Criminalística* (3ª ed., p. 55). Brasília: Academia Nacional de Polícia do Departamento de Polícia Federal.

Damgard, I. (1989). A Design Principle for Hash Functions. *Proceedings, CRYPTO'89*, (Published by Springer-Verlag).

Date, C. J. (2004). *Introdução a Sistemas de Bancos de Dados* (8ª ed.). Campus. ISBN 8535212736.

Dobbertin, H. (1996). *Cryptanalysis of MD5 Compress*. German Information Security Agency.

Dougherty, J.; Kohavi, R. e Sahami, M. (2005). Supervised and Unsupervised Discretization of Continuous Features. *International Conference on Machine Learning*.

Eleutério, P. e Machado, M. (2011). *Desvendando a Computação Forense*. Novatec. ISBN 978-85-7522-260-7.

Elmasri, R. e Navathe, S. B. (2010). The Relational Data Model and Relational Database Constraints. *Fundamentals of Database Systems* (6ª ed.). Addison Wesley. ISBN 978-0136086208.

Fayyad, U.; Piatetsky-shapiro, G. e Smyth, P. (1996). From Data Mining to Knowledge Discovery in Databases. *AI Magazine*, 37-54.

Frawley, W. J.; Piatetsky-shapiro, G. e Matheus, C. J. (1992). Knowledge Discovery in Databases : An Overview. *AI Magazine*, *13*(3), 57-70.

Garfinkel, S. L. (2010). Digital forensics research: The next 10 years. *Digital Investigation*, *7*, S64-S73. Elsevier Ltd. doi: 10.1016/j.diin.2010.05.009.

Han, J., e Kamber, M. (2006). *Data Mining: Concepts and Techniques* (2ª edição). Morgan Kaufmann.

Hinshaw, F. D. (2004). Data Warehouse Appliances: Driving the Business Intelligence Revolution. *Information Management Magazine*. Disponível em: http://www.information-management.com/issues/20040901/1009168-1.html. Acesso: 08 nov. 2011.

Hoelz, B. W. P. (2009). *MADIK: Uma Abordagem Multiagente para o Exame Pericial de Sistemas Computacionais*. Dissertação de Mestrado, Departamento de Ciência da Computação, Universidade de Brasília.

Hoelz, B. W. P.; Ralha, C. G. e Geeverghese, R. (2009). Artificial intelligence applied to computer forensics. *Proceedings of the 2009 ACM symposium on Applied Computing - SAC '09* (p. 883). New York, New York, USA: ACM Press. doi:10.1145/1529282.1529471.

Hoelz, B. W. P.; Ralha, C. G. e Mesquita, F. I. (2011). A Case-Based Reasoning Framework for Live Forensics. *In: Mark Pollitt, editor, Advances in Digital Forensics VII, Seventh Annual IFIP WG 11.9 International Conference on Digital Forensics*, pp 1–20. Springer. ISBN: 978-3642155055.

Hoelz, B. W. P.; Ruback, M. C.; Nogueira, J. H. M. e Campello, R. S. (2007). *Informática Forense* (2ª ed., p. 110). Brasília: Academia Nacional de Polícia do Departamento de Polícia Federal.

Hurlbut, D. (2009). Fuzzy Hashing for Digital Forensic Investigators. AccessData. Disponível em: http://accessdata.com/downloads/media/Fuzzy_Hashing_for_Investigators.pdf. Acesso: 08 nov. 2011.

Kaminsky, A. (2004a). Cryptographic One-Way Hash Functions. Disponível em: http://www.cs.rit.edu/~ark/lectures/onewayhash/onewayhash.shtml. Acesso: 08 nov. 2011.

Kaminsky, D. (2004b). MD5 To Be Considered Harmful Someday. *Cryptology ePrint Archive*, (Report 2004/357). Disponível em: http://eprint.iacr.org/2004/357. Acesso: 08 nov. 2011.

Kass, G. V. (1980). An Exploratory Technique for Investigating Large Quantities of Categorical Data. *Applied Statistics*, *29*:119-127. Royal Statistical Society.

Kim, K.; Park, S.; Chang, T.; Lee, C. e Baek, S. (2009). Lessons learned from the construction of a Korean software reference data set for digital forensics. *Digital Investigation*, *6*, S108-S113. Elsevier Ltd. doi: 10.1016/j.diin.2009.06.005.

Kornblum, J. (2006). Identifying almost identical files using context triggered piecewise hashing. *Digital Investigation*, 3S:S91-S97. Elsevier.

Kornblum, J. (2011). Ssdeep. Disponível em: http://ssdeep.sourceforge.net. Acesso: 08 nov. 2011.

Larose, D. T. (2005). *Discovering Knowledge in Data: An Introduction to Data Mining*. Wiley-Interscience.

Leaper, N. (2009). A Visual Guide to CRISP-DM Methodology. Disponível em: http://exde.wordpress.com/2009/03/13/a-visual-guide-to-crisp-dm-methodology/. Acesso: 08 nov. 2011.

Mandia, K.; Prosise, C. e Pepe, M. (2003). *Incident Response & Computer Forensics* (2nd ed., p. 507). McGraw-Hill/Osborne. ISBN: 978-0072226966.

Manuel, S. (2008). Classification and Generation of Disturbance Vectors for Collision Attacks against SHA-1. *Cryptology ePrint Archive*, Report 2008/469.

Mead, S. (2006). Unique file identification in the National Software Reference Library. *Digital Investigation*, *3*:138-150. Elsevier.

Menezes, A.; Oorschot, P. van e Vanstone, S. (1996). *Handbook of Applied Cryptography* (1st ed., p. 780). CRC Press. ISBN 978-0849385230.

Merkle, R. C. (1979). *Secrecy, Authentication, and Public Key Systems*. Ph.D. Thesis, Stanford University.

Merkle, R. C. (1989). One Way Hash Functions and DES. *Proceedings, CRYPTO'89*, (Published by Springer-Verlag).

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill. ISBN: 978-0070428072.

Moore, G. E. (1965). Cramming more components onto integrated circuits. *Electronics Magazine* (vol. 38, número 8). Disponível em:

ftp://download.intel.com/museum/Moores_Law/Articles-Press_Releases/Gordon_Moore_1965_Article.pdf. Acesso: 08 nov. 2011.

Murthy, S. K. (1998). Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey. *Data Mining and Knowledge Discovery*. Kluwer Academic Publishers.

National Institute of Standards and Technology (1995). Secure Hash Standard. *Federal Information Processing Standards Publication 180-1*. Disponível em: http://www.itl.nist.gov/fipspubs/fip180-1.htm. Acesso: 08 nov. 2011.

National Institute of Standards and Technology (2002). Secure Hash Standard. *Federal Information Processing Standards Publication 180-2*. Disponível em: http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf. Acesso: 08 nov. 2011.

National Institute of Standards and Technology (2008). Secure Hash Standard (SHS). *Federal Information Processing Standards Publication 180-3*, (October). Disponível em: http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf. Acesso: 08 nov. 2011.

National Institute of Standards and Technology (2009). *Data Formats of the NSRL Reference Data Set (RDS) Distribution*. Disponível em: http://www.nsrl.nist.gov/Documents/Data-Formats-of-the-NSRL-Reference-Data-Set-16.pdf. Acesso: 08 nov. 2011.

National Institute of Standards and Technology (2011). *National Software Reference Library*. Disponível em: http://www.nsrl.nist.gov. Acesso: 08 nov. 2011.

Nicholas, H. (2006). Dcfldd. *Defense Computer Forensics Lab*. Disponível em: http://dcfldd.sourceforge.net. Acesso: 08 nov. 2011.

Palmer, G. (2001). A Road Map for Digital Forensic Research. *Report from the First Digital Forensic Research Workshop (DFRWS)*. Utica, New York. Disponível em: http://www.dfrws.org/2001/dfrws-rm-final.pdf. Acesso: 08 nov. 2011.

Pentaho (2011). Pentaho Data Integration Documentation. Disponível em: http://wiki.pentaho.com/display/EAI/Latest+Pentaho+Data+Integration+(aka+Kettle)+Documentation. Acesso: 08 nov. 2011.

Preneel, B. (1993). *Analysis and Design of Cryptographic Hash Functions*. Ph.D. Thesis. Katholieke Universiteit Leuven.

Quinlan, J. R. (1986). Induction of Decision Trees. *Machine Learning*, *1*(1), 81-106.

Quinlan, J. R. (1993). *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc.

Rijmen, V. e Barreto, P. S. L. M. (2001). The WHIRLPOOL Hash Function. Disponível em: http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html. Acesso: 08 nov. 2011.

Rijmen, V. e Oswald, E. (2005). Update on SHA-1. In Menezes, A. J. (Ed.), *CT-RSA 2005*. LNCS, vol. 3376, pp. 58-71. Springer.

Rivest, R. (1992a). The MD4 Message-Digest Algorithm. *RFC 1320*. Disponível em: http://tools.ietf.org/html/rfc1320. Acesso: 08 nov. 2011.

Rivest, R. (1992b). The MD5 Message-Digest Algorithm. *RFC 1321*. Disponível em: http://tools.ietf.org/html/rfc1321. Acesso: 08 nov. 2011.

Rokach, L. e Maimon, O. (2008). *Data Mining With Decision Trees: Theory and Applications*. World Scientific.

Roussev, V.; Richard III, G. e Marziale, L. (2007). Multi-resolution similarity hashing. *Digital Investigation, 4*, pp. 105-113. Elsevier.

Roussev, V.; Richard III, G. e Marziale, L. (2008). Class-Aware Similarity Hashing for Data Classification. In I. Ray & S. Shenoi (Eds.), *Advances in Digital Forensics IV* (Vol. 285, pp. 101-113). Boston, MA: Springer. doi:10.1007/978-0-387-84927-0

Ruback, M. C.; Hoelz, B. W. P. e Ralha, C. G. (2012). Improving Hashsets in Computer Forensics. *Eigth Annual IFIP WG 11.9 International Conference on Digital Forensics*, Pretoria, África do Sul, Janeiro, 2012.

Rukhin, A.; Soto, J.; Nechvatal, J.; Smid, M.; Barker, E.; Leigh, S.; Levenson, M.; et al. (2001). A Statistical Test Suite For Random and Pseudorandom Number Generators For Cryptographic Applications. *NIST Special Publication*.

Russell, S. e Norvig, P. (2010). *Artificial Intelligence, A Modern Approach* (3ª ed.). Prentice Hall. ISBN 978-0-13-604259-4.

Sasaki, Y. e Aoki, K. (2009). Finding Preimages in Full MD5 Faster Than Exhaustive Search. *Advances in Cryptology - EUROCRYPT*, volume *5479*, pp. 134-152. doi: 10.1007/978-3-642-01001-9_8.

Schneier, B. (1995). *One-way hash functions*. In: Applied Cryptography. 2ª ed. John Wiley & Sons. ISBN 0-471-11709-9. Cap. 18.

Schweitzer, D. (2003). *Incident Response: Computer Forensic Toolkit* (p. 360). Wiley. ISBN 978-0764526367

Silberschatz, A.; Korth, H. e Sudarshan, S. (2001). *Database System Concepts*. *System* (4ª ed.). McGraw-Hill.

Solomon, M. G.; Barrett, D. e Broom, N. (2005). *Computer Forensics JumpStart* (p. 304). Sybex. ISBN 978-0782143751

Sommer, P. (2004). The challenges of large computer evidence cases. *Digital Investigation, 1*(1), 16-17. doi: 10.1016/j.diin.2004.01.005.

Stallings, W. (2010). *Cryptography and Network Security: Principles and Practice* (5ª ed., p. 744). Prentice Hall. ISBN 978-0136097044.

Steel, C. (2006). *Windows Forensics: The Field Guide for Corporate Computer Investigations* (1ª ed., p. 408). Wiley. ISBN 978-0470038628.

Stevens, M.; Lenstra, A. e Weger, B. de. (2009). HashClash Project. Disponível em: http://www.win.tue.nl/hashclash/. Acesso: 08 nov. 2011.

Stevens, M.; Lenstra, A. e Weger, B. de. (2007). Chosen-prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities. *Advances in Cryptology – Eurocrypt 2007, vol. 4515* (Lecture Notes in Computer Science), pp. 1-22.

Tan, P.-N.; Steinbach, M. e Kumar, V. (2005). Classification: Basic Concepts, Decision Trees, and Model Evaluation. *Introduction to Data Mining*. Addison-Wesley.

Thompson, E. (2005). MD5 collisions and the impact on computer forensics. *Digital Investigation*, *2*(1), 36-40. doi:10.1016/j.diin.2005.01.004

Tridgell, A. (2002). Spamsum README. Disponível em: http://www.samba.org/ftp/unpacked/junkcode/spamsum/README. Acesso: 08 nov. 2011.

US-CERT (2008).Computer Forensics.US-CERT. Disponível em http://www.us-cert.gov/reading_room/forensics.pdf. Acesso: 08 nov. 2011.

Wang, X.; Feng, D.; Lai, X. e Yu, H. (2004). Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD. *Cryptology ePrint Archive*, Report 2004/199.

Wang, X. e Yu, H. (2005). How to Break MD5 and Other Hash Functions. *EUROCRYPT 2005*, LNCS.

Wang, X.; Yin, Y. L. e Yu, H. (2005). Finding Collisions in the Full SHA-1. *Advances in Cryptology – CRYPTO 2005*, Pág.17-36. doi: 10.1007/11535218_2.

Wang, X.; Yao, A. e Yao, F. (2005). New Collision Search for SHA-1. *Crypto 2005 Rump Session*. Disponível em: http://www.iacr.org/conferences/crypto2005/r/2.pdf. Acesso: 08 nov. 2011.

Witten, I. H.; Frank, E. e Hall, M. A. (2011). Advanced Data Mining. *Data Mining: Practical Machine Learning Tools and Techniques* (3ª ed., p. 629). Elsevier. ISBN: 978-0123748560.

APÊNDICES

A – ATRIBUTOS UTILIZADOS NO PROCESSO DE MD

A Tabela A.1 apresenta os 47 atributos consolidados por valor de *hash* identificados pelo *framework* para a montagem da Base Joio e o significado de cada atributo. Esses atributos compõem os campos da Tabela **`final_instances`** da Base Joio.

Tabela A.1 - Atributos utilizados no processo de MD e respectivos significados.

	Atributo	**Significado**
1	md5	Valor de *hash* MD5
2	sha1	Valor de *hash* SHA1
3	ignorable	Arquivo identificado como ignorável por BHAC
4	alert	Arquivo identificado como *alerta* por BHAC
5	percent_distinct_evidences	Percentual de evidências distintas nas quais o arquivo foi encontrado
6	percent_distinct_cases	Percentual de casos distintos em que o arquivo foi encontrado
7	percent_distinct_case_subjects	Percentual de assuntos de investigação distintos em que o arquivo foi encontrado
8	percent_systemfolders	Percentual de arquivos encontrados em pastas de sistema
9	percent_userfolders	Percentual de arquivos encontrados em pastas de usuários
10	percent_internetcachefolders	Percentual de arquivos encontrados em cache de Internet
11	percent_badxt	Percentual de arquivos com extensão não condizente com a sua assinatura
12	percent_deleted	Percentual de arquivos deletados do sistema de arquivos
13	percent_recycled	Percentual de arquivos presentes na Lixeira do sistema operacional
14	percent_readonly	Percentual de arquivos com status "somente leitura" no sistema de arquivos
15	percent_system	Percentual de arquivos com status "sistema" no sistema de arquivos
16	percent_hidden	Percentual de arquivos com status "oculto" no sistema de arquivos
17	percent_emailed	Percentual de arquivos enviados por email
18	percent_encoded	Percentual de arquivos identificados como encriptados
19	percent_duplicated	Percentual de arquivos duplicados em uma mesma evidência

20	percent_filetype_expected	Percentual de arquivos em que o tipo do arquivo correspondia ao esperado para a pasta
21	percent_filetype_not_expected	Percentual de arquivos em que o tipo do arquivo não correspondia ao esperado para a pasta
22	percent_extension_ expected	Percentual de arquivos em que a extensão do arquivo correspondia ao esperado para a pasta
23	percent_extension_ not_expected	Percentual de arquivos em que a extensão do arquivo não correspondia ao esperado para a pasta
24	avg_percent_folder_files_ ignorable	Percentual médio dos arquivos presentes nas mesmas pastas do arquivo em questão que foram identificados como ignoráveis por BHAC
25	avg_percent_folder_files_ alert	Percentual médio dos arquivos presentes nas mesmas pastas do arquivo em questão que foram identificados como *alerta* por BHAC
26	avg_percent_folder_files_ deleted	Percentual médio dos arquivos presentes nas mesmas pastas do arquivo em questão que foram deletados do sistema de arquivos
27	avg_percent_folder_files_ badxt	Percentual médio dos arquivos presentes nas mesmas pastas do arquivo em questão que possuíam extensão não condizente com a sua assinatura
28	avg_percent_folder_files_ readonly	Percentual médio dos arquivos presentes nas mesmas pastas do arquivo em questão que possuíam status "somente leitura" no sistema de arquivos
29	avg_percent_folder_files_ system	Percentual médio dos arquivos presentes nas mesmas pastas do arquivo em questão que possuíam status "sistema" no sistema de arquivos
30	avg_percent_folder_files_ hidden	Percentual médio dos arquivos presentes nas mesmas pastas do arquivo em questão que possuíam status "oculto" no sistema de arquivos
31	avg_percent_folder_files_ emailed	Percentual médio dos arquivos presentes nas mesmas pastas do arquivo em questão que foram enviados por email
32	avg_percent_folder_files_ encoded	Percentual médio dos arquivos presentes nas mesmas pastas do arquivo em questão que foram identificados como encriptados
33	avg_percent_folder_files_ duplicated	Percentual médio dos arquivos presentes nas mesmas pastas do arquivo em questão que estavam duplicados em uma mesma evidência
34	avg_percent_folder_files_ filetype_expected	Percentual médio dos arquivos presentes nas mesmas pastas do arquivo em questão cujo tipo de arquivo correspondia ao esperado para a pasta

35	avg_percent_folder_files_ filetype_not_expected	Percentual médio dos arquivos presentes nas mesmas pastas do arquivo em questão cujo tipo de arquivo não correspondia ao esperado para a pasta
36	avg_percent_folder_files_ extension_expected	Percentual médio dos arquivos presentes nas mesmas pastas do arquivo em questão cuja extensão correspondia ao esperado para a pasta
37	avg_percent_folder_files_ extension_not_expected	Percentual médio dos arquivos presentes nas mesmas pastas do arquivo em questão cuja extensão não correspondia ao esperado para a pasta
38	most_common_cleanpath	Pasta no sistema de arquivos em que o arquivo foi encontrado mais vezes
39	percent_max_path_ occurrences	Percentual de vezes em que o arquivo foi encontrado na pasta definida pelo atributo "most_common_cleanpath"
40	most_common_filename	Nome de arquivo mais comum com o qual o arquivo foi encontrado
41	percent_max_filename_ occurrences	Percentual de vezes em que o arquivo possuía o nome definido pelo atributo "most_common_filename"
42	most_common_filetype	Tipo de arquivo mais comum identificado para o arquivo
43	percent_max_filetype_ occurrences	Percentual de vezes em que o arquivo foi identificado com o tipo definido pelo atributo "most_common_filetype"
44	most_common_category	Categoria mais comum identificada para o arquivo
45	percent_max_category_ occurrences	Percentual de vezes em que o arquivo foi identificado com a categoria definida pelo atributo "most_common_category"
46	most_common_extension	Extensão de arquivo mais comum com a qual o arquivo foi encontrado
47	percent_max_extension_ occurrences	Percentual de vezes em que o arquivo foi encontrado com a extensão definida pelo atributo "most_common_extension"

B – MODELO E-R DA BASE JOIO

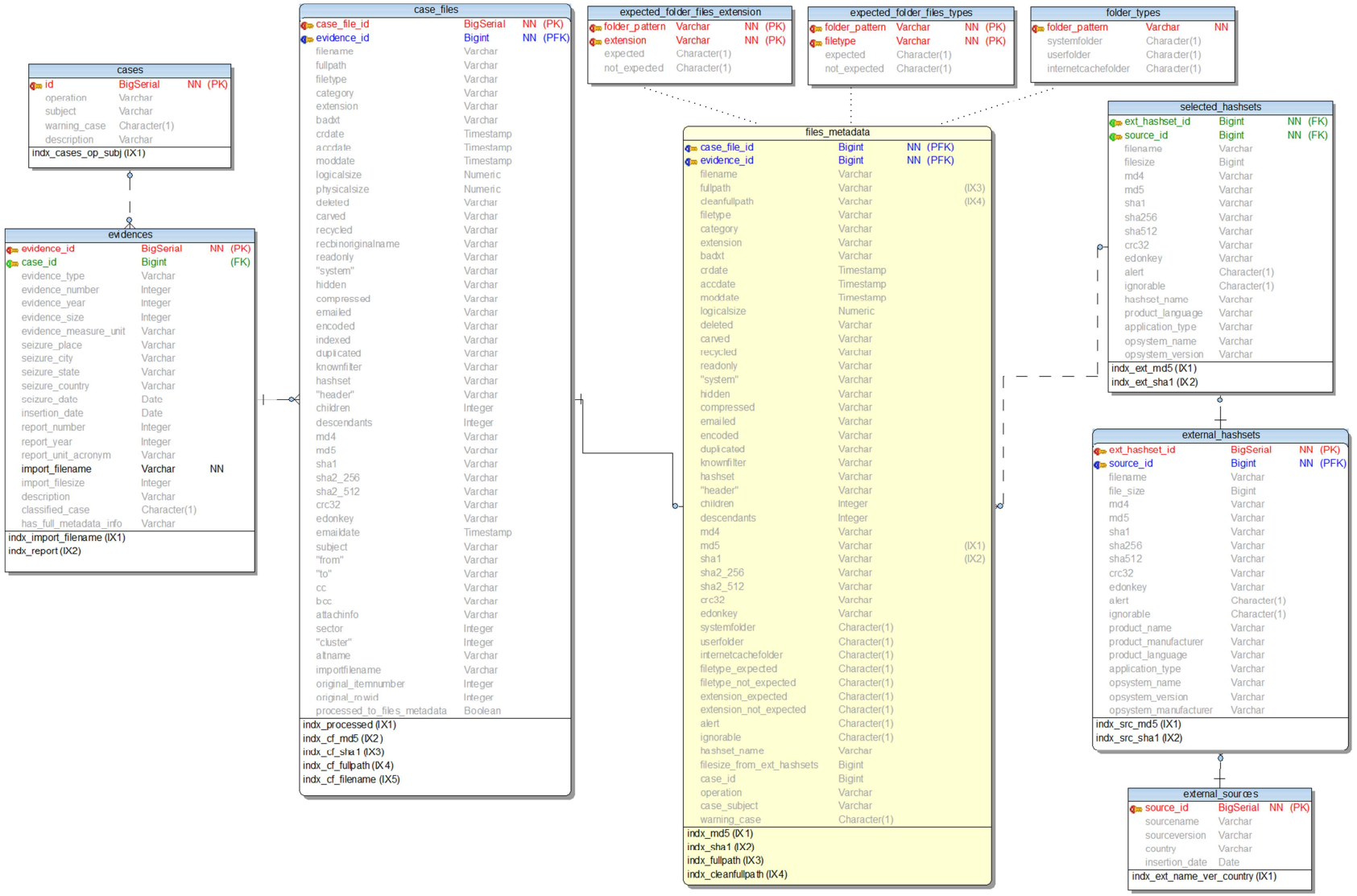

Figura B.0.1: Modelo E-R da Base JOIO

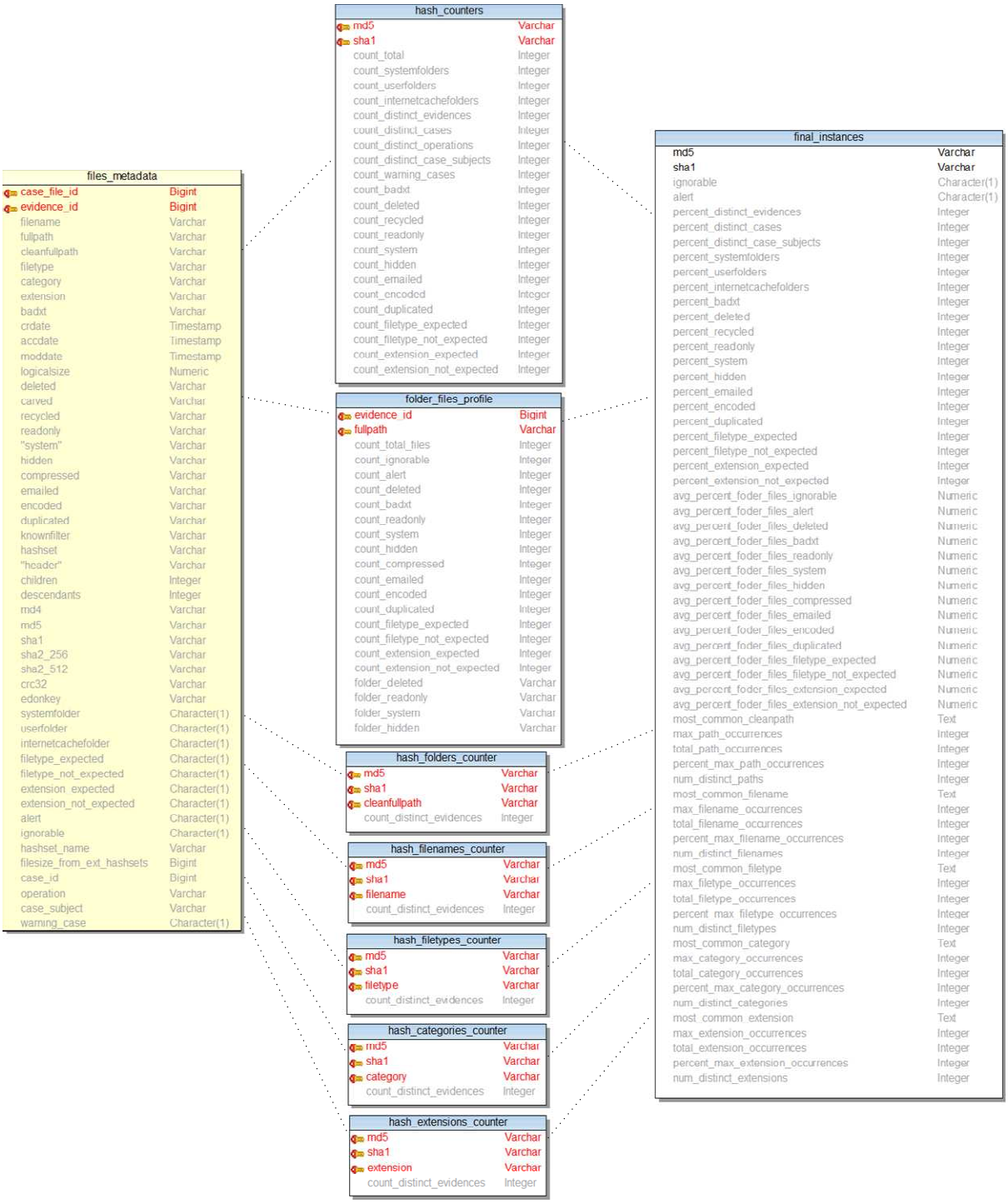

Figura B.0.2: Definição da tabela "final_instances" contendo as instâncias a serem utilizadas no processo de MD. Os dados são obtidos a partir de tabelas intermediárias que fazem a consolidação das informações presentes na tabela "files_metadata".

C – CONCEITOS IMPLEMENTADOS NO PROTÓTIPO A PARTIR DO CONHECIMENTO DE ESPECIALISTAS

A Tabela C.1 apresenta os parâmetros utilizados para padronizar os caminhos no sistema de arquivo relativos a pastas correlatas, de modo a permitir a identificação de arquivos que são encontrados nas mesmas pastas no sistema de arquivos.
A Tabela C.2 é utilizada no protótipo para classificar pastas entre os tipos "sistema", "usuário" e "cache de Internet", enquanto a Tabela C.3 e a Tabela C.4 identificam os tipos e extensões de arquivos esperados, respectivamente, para as principais pastas de um SO Windows.

Tabela C.1 - Parâmetros para padronizar caminhos no sistema de arquivos relativos a pastas correlatas.

Padrão de busca (SQL)	Nome de pasta real (expressão regular)	Nome de pasta padronizado
\\Documents and Settings\\%\\%	\\Documents and Settings\\[^\\]+\\	\\Documents and Settings\\XXXUSERXXX\\
\\Users\\%\\%	\\Users\\[^\\]+\\	\\Users\\XXXUSERXXX\\
\\Usuários\\%\\%	\\Usuários\\[^\\]+\\	\\Usuários\\XXXUSERXXX\\
\\Windows\\Profiles\\%\\%	\\Windows\\Profiles\\[^\\]+\\	\\Windows\\Profiles\\XXXUSERXXX\\
\\Winnt\\Profiles\\%\\%	\\Winnt\\Profiles\\[^\\]+\\	\\Winnt\\Profiles\\XXXUSERXXX\\
\\Documents and Settings\\%\\Configurações locais\\Temporary Internet Files\\Content.IE5\\%\\%	\\Temporary Internet Files\\Content.IE5\\[^\\]+\\	\\Temporary Internet Files\\Content.IE5\\XXXCACHEXXX\\
\\Documents and Settings\\%\\Local Settings\\Temporary Internet Files\\Content.IE5\\%\\%	\\Temporary Internet Files\\Content.IE5\\[^\\]+\\	\\Temporary Internet Files\\Content.IE5\\XXXCACHEXXX\\
\\Users\\%\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\Content.IE5\\%\\%	\\Temporary Internet Files\\Content.IE5\\[^\\]+\\	\\Temporary Internet Files\\Content.IE5\\XXXCACHEXXX\\
\\Usuários\\%\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\Content.IE5\\%\\%	\\Temporary Internet Files\\Content.IE5\\[^\\]+\\	\\Temporary Internet Files\\Content.IE5\\XXXCACHEXXX\\
\\Windows\\Profiles\\%\\Temporary Internet Files\\Content.IE5\\%\\	\\Temporary Internet Files\\Content.IE5\\[^\\]+\\	\\Temporary Internet Files\\Content.IE5\\XXXCACHEXXX\\
\\Winnt\\Profiles\\%\\Temporary Internet Files\\Content.IE5\\%\\	\\Temporary Internet Files\\Content.IE5\\[^\\]+\\	\\Temporary Internet Files\\Content.IE5\\XXXCACHEXXX\\
\\Winnt\\Temporary Internet Files\\Content.IE5\\%\\	\\Temporary Internet Files\\Content.IE5\\[^\\]+\\	\\Temporary Internet Files\\Content.IE5\\XXXCACHEXXX\\
\\Windows\\Temporary Internet Files\\Content.IE5\\%\\	\\Temporary Internet Files\\Content.IE5\\[^\\]+\\	\\Temporary Internet Files\\Content.IE5\\XXXCACHEXXX\\

\\Documents and Settings\\%\\Configurações locais\\Dados de aplicativos\\Mozilla\\Firefox\\Profiles\\%\\%	\\Mozilla\\Firefox\\Profiles\\[^\\]+\\	\\Mozilla\\Firefox\\Profiles\\XXXPROFILEXXX\\
\\Documents and Settings\\%\\Local Settings\\Application Data\\Mozilla\\Firefox\\Profiles\\%\\%	\\Mozilla\\Firefox\\Profiles\\[^\\]+\\	\\Mozilla\\Firefox\\Profiles\\XXXPROFILEXXX\\
\\Users\\%\\AppData\\Local\\Mozilla\\Firefox\\Profiles\\%\\%	\\Mozilla\\Firefox\\Profiles\\[^\\]+\\	\\Mozilla\\Firefox\\Profiles\\XXXPROFILEXXX\\
\\Usuários\\%\\AppData\\Local\\Mozilla\\Firefox\\Profiles\\%\\%	\\Mozilla\\Firefox\\Profiles\\[^\\]+\\	\\Mozilla\\Firefox\\Profiles\\XXXPROFILEXXX\\

Tabela C.2 - Conteúdo da tabela `folder_types`, utilizada no protótipo para classificar pastas entre os tipos "sistema", "usuário" e "cache de Internet".

folder_pattern	system folder	user folder	internet cache folder
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents)|(Desktop))\\		Y	
^\\((Users)|(Usuários))\\[^\\]+\\((Documents)|(Desktop)|(Music)|(Pictures)|(Videos))\\		Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\			Y
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\			Y
^\\((Windows)|(Winnt))\\Temporary Internet Files\\			Y
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\			Y
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\((Dados de aplicativos)|(Application Data))\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\			Y
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\			Y
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	Y		
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	Y		
^\\ProgramData\\	Y		
^\\System Volume Information\\	Y		
^\\MSOCache\\	Y		

Tabela C.3 - Conteúdo da tabela `expected_folder_files_types`, utilizada no protótipo para identificar os tipos de arquivos esperados para as pastas principais de um SO Windows.

folder_pattern	filetype	expected	not_ expected
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	MS Office%	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	Microsoft Word%	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	Microsoft Excel%	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	%PowerPoint%	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	Acrobat Portable Document Format (PDF)	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	Plain Text Document	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	MP3%	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	MPEG%	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	JPEG%	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	GIF File	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	Windows Media Audio/Video	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	Unknown File Type	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	Executable File		Y
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	XML		Y
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	Unicode Text Document		Y
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	CAB Archive		Y
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	MS Office%	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	Microsoft Word%	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	Microsoft Excel%	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	%PowerPoint%	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	Plain Text Document	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	Acrobat Portable Document Format (PDF)	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	Zip Archive	Y	

^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	Unknown File Type	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	Executable File		Y
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	Unicode Text Document		Y
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	XML		Y
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	CAB Archive		Y
^\\((Users)|(Usuários))\\[^\\]+\\Music\\	MP3%	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Music\\	MPEG%	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Music\\	Windows Media Audio/Video	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Pictures\\	JPEG%	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Pictures\\	GIF File	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Videos\\	MP3%	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Videos\\	MPEG%	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Videos\\	Windows Media Audio/Video	Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	Unknown File Type	Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	JPEG%	Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	GIF File	Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	P_NG File (Portable Network Graphics)	Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	Hypertext Document	Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	Plain Text Document	Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	Audio Flash	Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	GZIP Archive	Y	

^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	MS Office%		Y
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	Microsoft Word%		Y
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	Microsoft Excel%		Y
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	Executable File		Y
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	Unknown File Type	Y	
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	JPEG%	Y	
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	GIF File	Y	
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	P_NG File (Portable Network Graphics)	Y	
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	Hypertext Document	Y	
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	Plain Text Document	Y	
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	Audio Flash	Y	
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	GZIP Archive	Y	
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	MS Office%		Y
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	Microsoft Word%		Y
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	Microsoft Excel%		Y
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	Executable File		Y
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	Unknown File Type	Y	
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	JPEG%	Y	
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	GIF File	Y	
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	P_NG File (Portable Network Graphics)	Y	
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	Hypertext Document	Y	
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	Plain Text Document	Y	
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	Audio Flash	Y	
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	GZIP Archive	Y	

^\\((Windows)|(Winnt))\\Temporary Internet Files\\	MS Office%		Y
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	Microsoft Word%		Y
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	Microsoft Excel%		Y
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	Executable File		Y
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	Unknown File Type	Y	
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	JPEG%	Y	
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	GIF File	Y	
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	P_NG File (Portable Network Graphics)	Y	
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	Hypertext Document	Y	
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	Plain Text Document	Y	
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	Audio Flash	Y	
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	GZIP Archive	Y	
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	MS Office%		Y
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	Microsoft Word%		Y
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	Microsoft Excel%		Y
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	Executable File		Y
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\((Dados de aplicativos)|(Application Data))\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\	Unknown File Type	Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\((Dados de aplicativos)|(Application Data))\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\	JPEG%	Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\((Dados de aplicativos)|(Application Data))\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\	GIF File	Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\((Dados de aplicativos)|(Application Data))\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\	P_NG File (Portable Network Graphics)	Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\((Dados de aplicativos)|(Application Data))\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\	Hypertext Document	Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\((Dados de aplicativos)|(Application Data))\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\	Plain Text Document	Y	

^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\((Dados de aplicativos)|(Application Data))\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\	Audio Flash	Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\((Dados de aplicativos)|(Application Data))\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\	GZIP Archive	Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\((Dados de aplicativos)|(Application Data))\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\	MS Office%		Y
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\((Dados de aplicativos)|(Application Data))\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\	Microsoft Word%		Y
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\((Dados de aplicativos)|(Application Data))\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\	Microsoft Excel%		Y
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\((Dados de aplicativos)|(Application Data))\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\	Executable File		Y
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\	Unknown File Type	Y	
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\	JPEG%	Y	
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\	GIF File	Y	
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\	P_NG File (Portable Network Graphics)	Y	
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\	Hypertext Document	Y	
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\	Plain Text Document	Y	
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\	Audio Flash	Y	
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\	GZIP Archive	Y	
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\	MS Office%		Y
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\	Microsoft Word%		Y
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\	Microsoft Excel%		Y
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Mozilla\\Firefox\\Profiles\\[^\\]+\\Cache\\	Executable File		Y
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	Executable File	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	Unknown File Type	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	XML	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	Unicode Text Document	Y	

^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	Windows NT/2000/XP Registry File	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	Hypertext Document	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	Help File	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	Cursor	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	P_NG File (Portable Network Graphics)	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	CAB Archive	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	Plain Text Document	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	EFS Master Key _ile	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	Bitmap File	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	MS Office%		Y
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	Microsoft Word%		Y
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	Microsoft Excel%		Y
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	%PowerPoint%		Y
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	Acrobat Portable Document Format (PDF)		Y
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	Unknown File Type	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	Executable File	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	GIF File	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	P_NG File (Portable Network Graphics)	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	Hypertext Document	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	Windows Metafile	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	XML	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	Unicode Text Document	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	Plain Text Document	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	MS Office%		Y
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	Microsoft Word%		Y
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	Microsoft Excel%		Y

^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	%PowerPoint%		Y
^\\ProgramData\\	Unknown File Type	Y	
^\\ProgramData\\	Executable File	Y	
^\\ProgramData\\	XML	Y	
^\\ProgramData\\	GIF File	Y	
^\\ProgramData\\	P_NG File (Portable Network Graphics)	Y	
^\\ProgramData\\	Unicode Text Document	Y	
^\\ProgramData\\	MS Office%		Y
^\\ProgramData\\	Microsoft Word%		Y
^\\ProgramData\\	Microsoft Excel%		Y
^\\ProgramData\\	%PowerPoint%		Y
^\\ProgramData\\	Acrobat Portable Document Format (PDF)		Y
^\\System Volume Information\\	Unknown File Type	Y	
^\\System Volume Information\\	Windows NT/2000/XP Registry File	Y	
^\\System Volume Information\\	Executable File	Y	
^\\System Volume Information\\	Unicode Text Document	Y	
^\\System Volume Information\\	Plain Text Document	Y	
^\\System Volume Information\\	MS Office%		Y
^\\System Volume Information\\	Microsoft Word%		Y
^\\System Volume Information\\	Microsoft Excel%		Y
^\\System Volume Information\\	%PowerPoint%		Y
^\\System Volume Information\\	Acrobat Portable Document Format (PDF)		Y
^\\System Volume Information\\	JPEG%		Y
^\\MSOCache\\	CAB Archive	Y	
^\\MSOCache\\	XML	Y	
^\\MSOCache\\	Executable File	Y	

^\\MSOCache\\	OLE Archive	Y	
^\\MSOCache\\	Unknown File Type	Y	
^\\MSOCache\\	MS Office%		Y
^\\MSOCache\\	Microsoft Word%		Y
^\\MSOCache\\	Microsoft Excel%		Y
^\\MSOCache\\	%PowerPoint%		Y
^\\MSOCache\\	Acrobat Portable Document Format (PDF)		Y
^\\MSOCache\\	JPEG%		Y

Tabela C.4 - Conteúdo da tabela `expected_folder_files_extensions`, utilizada no protótipo para identificar as extensões de arquivos esperadas para as pastas principais de um SO Windows.

folder_pattern	extension	expected	not_expected
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	doc	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	docx	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	rtf	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	tmp	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	pdf	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	txt	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	xls	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	xlsx	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	ppt	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	pptx	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	zip	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	jpg	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	bmp	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	gif	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	mp3	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	m4a	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	wav	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	wma	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	wmv	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	avi	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	mpg	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	mpeg	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	mkv	Y	
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	dll		Y
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	sys		Y

^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	dat		Y
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	inf		Y
^\\Documents And Settings\\[^\\]+\\((Meus documentos)|(My Documents))\\	cat		Y
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	doc	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	docx	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	rtf	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	tmp	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	pdf	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	txt	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	xls	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	xlsx	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	ppt	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	pptx	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	zip	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	dll		Y
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	sys		Y
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	dat		Y
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	inf		Y
^\\((Users)|(Usuários))\\[^\\]+\\Documents\\	cat		Y
^\\((Users)|(Usuários))\\[^\\]+\\Music\\	mp3	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Music\\	m4a	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Music\\	wav	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Music\\	wma	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Pictures\\	jpg	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Pictures\\	bmp	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Pictures\\	gif	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Videos\\	wmv	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Videos\\	avi	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Videos\\	mpg	Y	

^\\((Users)|(Usuários))\\[^\\]+\\Videos\\	mpeg	Y	
^\\((Users)|(Usuários))\\[^\\]+\\Videos\\	mkv	Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	gif	Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	png	Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	jpg	Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	js	Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	css	Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	htm	Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	html	Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	swf	Y	
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	xls		Y
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	xlsx		Y
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	doc		Y
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	docx		Y
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	rtf		Y
^\\Documents And Settings\\[^\\]+\\((Configurações locais)|(Local Settings))\\Temporary Internet Files\\	exe		Y
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	gif	Y	
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	png	Y	
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	jpg	Y	
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	js	Y	
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	css	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	ppt		Y
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	htm	Y	
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	html	Y	
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	swf	Y	
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	xls		Y
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	xlsx		Y
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	doc		Y
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	docx		Y

^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	rtf		Y
^\\((Users)|(Usuários))\\[^\\]+\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\	exe		Y
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	gif	Y	
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	png	Y	
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	jpg	Y	
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	js	Y	
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	css	Y	
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	htm	Y	
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	html	Y	
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	swf	Y	
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	xls		Y
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	xlsx		Y
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	doc		Y
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	docx		Y
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	rtf		Y
^\\((Windows)|(Winnt))\\Temporary Internet Files\\	exe		Y
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	gif	Y	
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	png	Y	
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	jpg	Y	
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	js	Y	
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	css	Y	
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	htm	Y	
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	html	Y	
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	swf	Y	
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	xls		Y
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	xlsx		Y
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	doc		Y
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	docx		Y
^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	rtf		Y

^\\((Windows)|(Winnt))\\Profiles\\[^\\]+\\Temporary Internet Files\\	exe		Y
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	dll	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	exe	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	inf	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	manifest	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	dat	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	sys	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	chm	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	hlp	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	ttf	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	fon	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	pnf	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	lnk	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	log	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	xml	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	ini	Y	
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	xls		Y
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	xlsx		Y
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	doc		Y
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	docx		Y
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	ppt		Y
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	pptx		Y
^\\((Windows)|(Winnt))\\((?!Temporary Internet Files).)*$	pdf		Y
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	exe	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	dll	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	sys	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	dat	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	inf	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	gif	Y	

^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	jpg	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	bmp	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	png	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	jar	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	java	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	xml	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	htm	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	html	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	chm	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	ico	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	ini	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	wmf	Y	
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	doc		Y
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	docx		Y
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	xls		Y
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	xlsx		Y
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	pptx		Y
^\\((Arquivos de Programas)|(Program Files))(\(x86\))?\\	rtf		Y
^\\ProgramData\\	dll	Y	
^\\ProgramData\\	gif	Y	
^\\ProgramData\\	png	Y	
^\\ProgramData\\	log	Y	
^\\ProgramData\\	xml	Y	
^\\ProgramData\\	doc		Y
^\\ProgramData\\	docx		Y
^\\ProgramData\\	xls		Y
^\\ProgramData\\	xlsx		Y
^\\ProgramData\\	pdf		Y
^\\System Volume Information\\	dll	Y	

^\\System Volume Information\\	ini	Y	
^\\System Volume Information\\	exe	Y	
^\\System Volume Information\\	log	Y	
^\\System Volume Information\\	doc		Y
^\\System Volume Information\\	docx		Y
^\\System Volume Information\\	xls		Y
^\\System Volume Information\\	xlsx		Y
^\\System Volume Information\\	pdf		Y
^\\MSOCache\\	cab	Y	
^\\MSOCache\\	xml	Y	
^\\MSOCache\\	msi	Y	
^\\MSOCache\\	exe	Y	
^\\MSOCache\\	dll	Y	
^\\MSOCache\\	chm	Y	
^\\MSOCache\\	doc		Y
^\\MSOCache\\	docx		Y
^\\MSOCache\\	xls		Y
^\\MSOCache\\	xlsx		Y
^\\MSOCache\\	pdf		Y
^\\MSOCache\\	rtf		Y

D – EXEMPLOS DE ARQUIVOS IDENTIFICADOS COMO *IGNORÁVEL* PELO PROCESSO DE MD NO ESTUDO DE CASO 1

A título de exemplo, estão expostos na Tabela D.1 todos os 57 arquivos da amostra de teste identificados como *ignorável* pelo processo de MD no Estudo de Caso 1 que possuem valor de *hash* SHA-1 começados por “000”, “001” ou “002”.

Tabela D.1 - Exemplos de arquivos identificados como *ignorável* pelo processo de MD no Estudo de Caso 1.

Item	SHA-1	Nome do arquivo	Caminho completo	Tipo	Classe
B	002BC480B2C58CFC3E41F3ACC9B96023085573B5	branches.inf	Disco 351-11.001\Part_1\NONAME-NTFS\WINDOWS\SoftwareDistribution\Download\880c198cf1f0ca01c4a0719276bb5d2e\update\branches.inf	Unknown File Type	I
B	002BC480B2C58CFC3E41F3ACC9B96023085573B5	branches.inf	Disco 351-11.001\Part_1\NONAME-NTFS\WINDOWS\hf_mig\KB961501\update\branches.inf	Unknown File Type	I
C	002FD1969684319DC6C8E0561A3FBAD4D8D2E644	ua[1].gif	Disco 039-11.001\Part_1\SISTEMA-NTFS\Documents and Settings\usuario01\Configurações locais\Temporary Internet Files\Content.IE5\KH2NCXAB\ua[1].gif	GIF File	T
C	0009CA8C7FC32BC29D22FF1647B78B94B1DD93DC	menu-main_r3_c10[1].gif	Disco 039-11.001\Part_1\SISTEMA-NTFS\Documents and Settings\usuario01\Configurações locais\Temporary Internet Files\Content.IE5\731Z3P0W\menu-main_r3_c10[1].gif	GIF File	T
D	002D7EEFFE7DCE4E7CA66F92259A78D0EEFFEDA8	winmobil.inf	2099-10\Part_1\NONAME-NTFS\Windows\winsxs\x86_winmobil.inf_31bf3856ad364e35_6.0.6001.18000_none_9f841f055f7e71c2\winmobil.inf	Unicode Text Document	I
D	002D7EEFFE7DCE4E7CA66F92259A78D0EEFFEDA8	winmobil.inf	2099-10\Part_1\NONAME-NTFS\Windows\System32\DriverStore\FileRepository\winmobil.inf_a7c8ce31\winmobil.inf	Unicode Text Document	I
D	002D7EEFFE7DCE4E7CA66F92259A78D0EEFFEDA8	winmobil.inf	2099-10\Part_1\NONAME-NTFS\Windows\inf\winmobil.inf	Unicode Text Document	I
D	0014CBBBF334B092104A0	iscsied.dll	2099-10\Part_1\NONAME-NTFS\Windows\System32\iscsied.dll	Executable	I

	C5EEBC4F3FD2C819A9D			File	
D	0014CBBBF334B092104A0C5EEBC4F3FD2C819A9D	x86_microsoft-windows-i..i_initiator_service_31bf3856ad364e35_6.0.6001.18000_none_da73ab3e1517f045_iscsied.dll_e933fb0e	2099-10\Part_1\NONAME-NTFS\Windows\winsxs\Backup\x86_microsoft-windows-i..i_initiator_service_31bf3856ad364e35_6.0.6001.18000_none_da73ab3e1517f045_iscsied.dll_e933fb0e	Executable File	I
D	0029FA489121D46DAF226EA6FAF286F953455C5F	vdmredir.dll	2099-10\Part_1\NONAME-NTFS\Windows\System32\vdmredir.dll	Executable File	I
D	000B078DC31D4762F6FA601A95343ABBD1DB9BDA	WsmCl.dll	2099-10\Part_1\NONAME-NTFS\Windows\winsxs\x86_microsoft-windows-w..for-management-core_31bf3856ad364e35_6.0.6002.18005_none_cc50ee6baa2997a1\WsmCl.dll	Executable File	I
D	001723CBEFEB922274E169BEEE7A388AD34DA66D	parentalcontrols-ppdlic.xrm-ms	2099-10\Part_1\NONAME-NTFS\Windows\System32\licensing\ppdlic\parentalcontrols-ppdlic.xrm-ms	XML	I
D	001F0650F5069A30EA719EE4C5CC55307B263CCB	x86_microsoft-windows-w..mediadeliveryengine_31bf3856ad364e35_6.0.6002.18005_none_1f5b99e45ab5a251.manifest	2099-10\Part_1\NONAME-NTFS\Windows\winsxs\Manifests\x86_microsoft-windows-w..mediadeliveryengine_31bf3856ad364e35_6.0.6002.18005_none_1f5b99e45ab5a251.manifest	XML	I
D	001433860913DDE3B6D6C64949DE0A6003F2B2DC	x86_wcf-m_svc_mon_sup_dll_31bf3856ad364e35_6.0.6002.18005_none_a7c97e5948c03629.manifest	2099-10\Part_1\NONAME-NTFS\Windows\winsxs\Manifests\x86_wcf-m_svc_mon_sup_dll_31bf3856ad364e35_6.0.6002.18005_none_a7c97e5948c03629.manifest	XML	I
D	0007B93EB4A0E8C85B588D663930B03194181A19	x86_netfx-shfusion_dll_b03f5f7f11d50a3a_6.0.6002.18005_none_5ab3d7c5cc906c6a.manifest	2099-10\Part_1\NONAME-NTFS\Windows\winsxs\Manifests\x86_netfx-shfusion_dll_b03f5f7f11d50a3a_6.0.6002.18005_none_5ab3d7c5cc906c6a.manifest	XML	I
D	0025AD96B62DD4FAAC13D337A86AE1C9D6CB3A53	__AssemblyInfo__.ini	2099-10\Part_1\NONAME-NTFS\Windows\assembly\GAC\mscomctl\10.0.4504.0__31bf3856ad364e35__AssemblyInfo__.ini	Unknown File Type	I

D	0023C30F0EC752FADD990B0E26AB766163C84E54	unstall.bmp	2099-10\Part_1\NONAME-NTFS\Users\DMP\AppData\Local\Temp\unstall.bmp	Bitmap File	I
D	0023CC7FADD2A8722DFD3931AC43F4A474F7338F	x86_617c8c0a27ff91ce825462168dc26a2d_31bf3856ad364e35_6.0.6001.18495_none_89b41df3afd10e27.manifest	2099-10\Part_1\NONAME-NTFS\Windows\winsxs\Manifests\x86_617c8c0a27ff91ce825462168dc26a2d_31bf3856ad364e35_6.0.6001.18495_none_89b41df3afd10e27.manifest	XML	I
D	0023CC7FADD2A8722DFD3931AC43F4A474F7338F	x86_617c8c0a27ff91ce825462168dc26a2d_31bf3856ad364e35_6.0.6001.18495_none_89b41df3afd10e27.manifest	2099-10\Part_1\NONAME-NTFS\Windows\SoftwareDistribution\Download\5a5d6de9a0efe94dbaf9cda22559ce8a\x86_617c8c0a27ff91ce825462168dc26a2d_31bf3856ad364e35_6.0.6001.18495_none_89b41df3afd10e27.manifest	XML	I
D	0029FA489121D46DAF226EA6FAF286F953455C5F	x86_microsoft-windows-ntvdm-system32_31bf3856ad364e35_6.0.6002.18005_none_fff8f2266fafa2e8_vdmredir.dll_6eee2d39	2099-10\Part_1\NONAME-NTFS\Windows\winsxs\Backup\x86_microsoft-windows-ntvdm-system32_31bf3856ad364e35_6.0.6002.18005_none_fff8f2266fafa2e8_vdmredir.dll_6eee2d39	Executable File	I
E	000ACE363FA7DAAA0798CC783FF8D41507B9593A	drvmain.sdb	1688_10\Part_1\NONAME-NTFS\Windows\winsxs\x86_microsoft-windows-a..ence-mitigations-c5_31bf3856ad364e35_6.0.6001.18165_none_0bea5d21f274f4a9\drvmain.sdb	Unknown File Type	I
E	002D7EEFFE7DCE4E7CA66F92259A78D0EEFFEDA8	winmobil.inf	1688_10\Part_1\NONAME-NTFS\Windows\winsxs\x86_winmobil.inf_31bf3856ad364e35_6.0.6001.18000_none_9f841f055f7e71c2\winmobil.inf	Unicode Text Document	I
E	002D7EEFFE7DCE4E7CA66F92259A78D0EEFFEDA8	winmobil.inf	1688_10\Part_1\NONAME-NTFS\Windows\System32\DriverStore\FileRepository\winmobil.inf_a7c8ce31\winmobil.inf	Unicode Text Document	I
E	002D7EEFFE7DCE4E7CA66F92259A78D0EEFFEDA8	winmobil.inf	1688_10\Part_1\NONAME-NTFS\Windows\inf\winmobil.inf	Unicode Text Document	I
E	0014CBBBF334B092104A0C5EEBC4F3FD2C819A9D	iscsied.dll	1688_10\Part_1\NONAME-NTFS\Windows\System32\iscsied.dll	Executable File	I
E	0014CBBBF334B092104A0	x86_microsoft-windows-	1688_10\Part_1\NONAME-	Executable	I

	C5EEBC4F3FD2C819A9D	i..i_initiator_service_31bf3856ad364e35_6.0.6001.18000_none_da73ab3e1517f045_iscsied.dll_e933fb0e	NTFS\Windows\winsxs\Backup\x86_microsoft-windows-i..i_initiator_service_31bf3856ad364e35_6.0.6001.18000_none_da73ab3e1517f045_iscsied.dll_e933fb0e	File	
E	0029FA489121D46DAF226EA6FAF286F953455C5F	vdmredir.dll	1688_10\Part_1\NONAME-NTFS\Windows\System32\vdmredir.dll	Executable File	I
E	000B078DC31D4762F6FA601A95343ABBD1DB9BDA	WsmCl.dll	1688_10\Part_1\NONAME-NTFS\Windows\System32\WsmCl.dll	Executable File	I
E	001723CBEFEB922274E169BEEE7A388AD34DA66D	parentalcontrols-ppdlic.xrm-ms	1688_10\Part_1\NONAME-NTFS\Windows\System32\licensing\ppdlic\parentalcontrols-ppdlic.xrm-ms	XML	I
E	001B008B0138E8DA8CBBB422050B1D5AA53422A6	x86_7cbc72c3e9f08aecb6fdb5eef0e0ef52_31bf3856ad364e35_6.0.6001.18226_none_cc251e80b4ad19be.manifest	1688_10\Part_1\NONAME-NTFS\Windows\winsxs\Manifests\x86_7cbc72c3e9f08aecb6fdb5eef0e0ef52_31bf3856ad364e35_6.0.6001.18226_none_cc251e80b4ad19be.manifest	XML	I
E	000E521EC48035A031DAC56FF650F048DFA8D256	NlsLexicons0414.dll	1688_10\Part_1\NONAME-NTFS\Windows\winsxs\x86_microsoft-windows-naturallanguage6_31bf3856ad364e35_6.0.6000.16710_none_9be9c78e2d9d5d54\NlsLexicons0414.dll	Executable File	I
E	0025AD96B62DD4FAAC13D337A86AE1C9D6CB3A53	__AssemblyInfo__.ini	1688_10\Part_1\NONAME-NTFS\Windows\assembly\GAC\mscomctl\10.0.4504.0__31bf3856ad364e35__AssemblyInfo__.ini	Unknown File Type	I
E	000850D02F54B07B6F8E5E7B9ECBFDF95974CAB6	mstscax.dll	1688_10\Part_1\NONAME-NTFS\Windows\winsxs\x86_microsoft-windows-t..s-clientactivexcore_31bf3856ad364e35_6.0.6000.21061_none_2e516291e1cf33e3\mstscax.dll	Executable File	I
E	0007B93EB4A0E8C85B588D663930B03194181A19	x86_netfx-shfusion_dll_b03f5f7f11d50a3a_6.0.6002.18005_none_5ab3d7c5cc906c6a.manifest	1688_10\Part_1\NONAME-NTFS\Windows\SoftwareDistribution\Download\829570003409c6fd8183f615cab83b0b\x86_netfx-shfusion_dll_b03f5f7f11d50a3a_6.0.6002.18005_none_5ab3d7c5cc906c6a.manifest	XML	I
E	001433860913DDE3B6D6C64949DE0A6003F2B2DC	x86_wcf-m_svc_mon_sup_dll_31bf385	1688_10\Part_1\NONAME-NTFS\Windows\SoftwareDistribution\Download\829570003409	XML	I

		6ad364e35_6.0.6002.18005_none_a7c97e5948c03629.manifest	c6fd8183f615cab83b0b\x86_wcf-m_svc_mon_sup_dll_31bf3856ad364e35_6.0.6002.18005_none_a7c97e5948c03629.manifest		
E	001F0650F5069A30EA719EE4C5CC55307B263CCB	x86_microsoft-windows-w..mediadeliveryengine_31bf3856ad364e35_6.0.6002.18005_none_1f5b99e45ab5a251.manifest	1688_10\Part_1\NONAME-NTFS\Windows\SoftwareDistribution\Download\829570003409c6fd8183f615cab83b0b\x86_microsoft-windows-w..mediadeliveryengine_31bf3856ad364e35_6.0.6002.18005_none_1f5b99e45ab5a251.manifest	XML	I
E	0007B93EB4A0E8C85B588D663930B03194181A19	x86_netfx-shfusion_dll_b03f5f7f11d50a3a_6.0.6002.18005_none_5ab3d7c5cc906c6a.manifest	1688_10\Part_1\NONAME-NTFS\Windows\winsxs\Manifests\x86_netfx-shfusion_dll_b03f5f7f11d50a3a_6.0.6002.18005_none_5ab3d7c5cc906c6a.manifest	XML	I
E	001433860913DDE3B6D6C64949DE0A6003F2B2DC	x86_wcf-m_svc_mon_sup_dll_31bf3856ad364e35_6.0.6002.18005_none_a7c97e5948c03629.manifest	1688_10\Part_1\NONAME-NTFS\Windows\winsxs\Manifests\x86_wcf-m_svc_mon_sup_dll_31bf3856ad364e35_6.0.6002.18005_none_a7c97e5948c03629.manifest	XML	I
E	001F0650F5069A30EA719EE4C5CC55307B263CCB	x86_microsoft-windows-w..mediadeliveryengine_31bf3856ad364e35_6.0.6002.18005_none_1f5b99e45ab5a251.manifest	1688_10\Part_1\NONAME-NTFS\Windows\winsxs\Manifests\x86_microsoft-windows-w..mediadeliveryengine_31bf3856ad364e35_6.0.6002.18005_none_1f5b99e45ab5a251.manifest	XML	I
E	0029FA489121D46DAF226EA6FAF286F953455C5F	x86_microsoft-windows-ntvdm-system32_31bf3856ad364e35_6.0.6002.18005_none_fff8f2266fafa2e8_vdmredir.dll_6eee2d39	1688_10\Part_1\NONAME-NTFS\Windows\winsxs\Backup\x86_microsoft-windows-ntvdm-system32_31bf3856ad364e35_6.0.6002.18005_none_fff8f2266fafa2e8_vdmredir.dll_6eee2d39	Executable File	I
E	002593369C08B705E73597360DE6034D2A2BB095	ACWZUSR12.ACCDU	1688_10\Part_1\NONAME-NTFS\Windows\Installer\$PatchCache$\Managed\00002109030000000000000F01FEC\12.0.6425\ACWZUSR12.ACCDU	MS Jet Database	I
E	000FE0BD00DD7E2FA8A9B6E05F426162B885FF12	MMSUIPlugin_001.ico	1688_10\Part_1\NONAME-NTFS\Program Files\TIM Web Banda Larga\plugins\MMSUIPlugin\MMSUIPlugin_001.ico	Unknown File Type	I

E	002106DAEC430987EBDCF6EE0EF54B9B70DA6C47	Icon_06.ico	1688_10\Part_1\NONAME-NTFS\Program Files\TIM Web Banda Larga\plugins\NetInfoUIExPlugin\Icon_06.ico	Unknown File Type	I
E	000FE0BD00DD7E2FA8A9B6E05F426162B885FF12	SMSUIPlugin_001.ico	1688_10\Part_1\NONAME-NTFS\Program Files\TIM Web Banda Larga\plugins\SMSUIPlugin\SMSUIPlugin_001.ico	Unknown File Type	I
E	0023CC7FADD2A8722DFD3931AC43F4A474F7338F	x86_617c8c0a27ff91ce825462168dc26a2d_31bf3856ad364e35_6.0.6001.18495_none_89b41df3afd10e27.manifest	1688_10\Part_1\NONAME-NTFS\Windows\winsxs\Manifests\x86_617c8c0a27ff91ce825462168dc26a2d_31bf3856ad364e35_6.0.6001.18495_none_89b41df3afd10e27.manifest	XML	I
G	0022A8F84379A4003D033EE2AE4A69D1A3C74C82	dbnetlib.dll	Part_1\NONAME-NTFS\WINDOWS\system32\dbnetlib.dll	Executable File	I
G	0022A8F84379A4003D033EE2AE4A69D1A3C74C82	dbnetlib.dll	Part_1\NONAME-NTFS\WINDOWS\system32\dllcache\dbnetlib.dll	Executable File	I
G	002EF92F97EF6DA6A4881E09E8C3CF40B5DD1280	aumente-vendas[1].gif	Part_1\NONAME-NTFS\Documents and Settings\AMAZON GOLD\Configurações locais\Temporary Internet Files\Content.IE5\JIGV3H89\aumente-vendas[1].gif	GIF File	T
G	0012FDC01604D684BC8BD3757F1EE75C208CA312	SOA.DLL	Part_1\NONAME-NTFS\Arquivos de programas\Microsoft Office\OFFICE11\SOA.DLL	Executable File	I
H	0022A8F84379A4003D033EE2AE4A69D1A3C74C82	dbnetlib.dll	Part_1\NONAME-NTFS\WINDOWS\system32\dbnetlib.dll	Executable File	I
H	002A637999A864937E14B4E404E6078D417431AB	msoe.dll	Part_1\NONAME-NTFS\WINDOWS\ServicePackFiles\i386\msoe.dll	Executable File	I
H	0025933 69C08B705E73597360DE6034D2A2BB095	ACWZUSR12.ACCDU	Part_1\NONAME-NTFS\WINDOWS\Installer\$PatchCache$\Managed\00002109110000000000000000F01FEC\12.0.6425\ACWZUSR12.ACCDU	MS Jet Database	I
H	0001AB38A34C5A495328807605953B0FF15FCC63	WudfCustom.dll	Part_1\NONAME-NTFS\WINDOWS\$NtUninstallWudf01000$\spuninst\WudfCustom.dll	Executable File	I
H	002BC480B2C58CFC3E41F3ACC9B96023085573B5	A0000364.dll	Part_1\NONAME-NTFS\System Volume Information_restore{4C7AA497-5525-4136-903F-706D851A6113}\RP9\A0000364.dll	Unknown File Type	I
I	002251F921F99E9F62A1B25C482D336600D0EA7A	Package_1_for_KB950125~31bf3856ad364e35~x86~~6.0.1.0.mum	784\Part_3\OS-NTFS\Windows\servicing\Packages\Package_1_for_KB950125~31bf3856ad364e35~x86~~6.0.1.0.mum	XML	I

I	002D7EEFFE7DCE4E7CA66F92259A78D0EEFFEDA8	winmobil.inf	784\Part_3\OS-NTFS\Windows\SoftwareDistribution\Download\40f104edd8fff18ebca7c9e5389c3391\x86_winmobil.inf_31bf3856ad364e35_6.0.6001.18000_none_9f841f055f7e71c2\winmobil.inf	Unicode Text Document	I
I	001723CBEFEB922274E169BEEE7A388AD34DA66D	parentalcontrols-ppdlic.xrm-ms	784\Part_3\OS-NTFS\Windows\SoftwareDistribution\Download\40f104edd8fff18ebca7c9e5389c3391\x86_microsoft-windows-parentalcontrols_31bf3856ad364e35_6.0.6001.18000_none_9b3de6b69ca85905\parentalcontrols-ppdlic.xrm-ms	XML	I